Foundations of Secure Computation: Perfect Security and Fairness

Gilad Asharov

Department of Computer Science

Ph.D. Thesis

Submitted to the Senate of Bar-Ilan University

Ramat-Gan, Israel March, 2014

Approved: January, 2015

This work was carried out under the supervision of

Prof. Yehuda Lindell

Department of Computer Science, Bar-Ilan University

Acknowledgements

Ph.D. studies are full of challenges, personal achievements, friendships and sometimes even failures. As this period is reaching its end, it is time to conclude by thanking the many people who made these past few years some of the most enjoyable in my life.

First and foremost, I would like to thank my advisor, Prof. Yehuda Lindell. Yehuda introduced me the field of Cryptography and to the research world. Yehuda taught me so many important lessons during my studies and I am sure that he will continue to inspire me in the future. His endless demand for excellence, his immense knowledge, his unwillingness to compromise, along with his drive, enthusiasm and guidance are the major reasons for the achievements in these studies. Yehuda has put a lot of time and effort in so that I could succeed and become an independent researcher and I am deeply grateful for it. Aside for his outstanding professional achievements, Yehuda also has a great personality and I feel greatly privileged that I had the opportunity to work closely with him.

I would like also to thank our crypto group in Bar-Ilan: Benny Pinkas, Ran Cohen, Eran Omri, Hila Zarosim, Claudio Orlandi, Carmit Hazay, Ben Riva, Rafi Chen, Yael Ejgenberg, Moriya Farbstein, Meital Levy, Tali Oberman, Eli Oxman and Asaf Cohen. To our visitors: Tal Malkin and Kobi Nissim, and to our own Yonit Homburger.

During my studies, I joined a very warm research community, and had the privilege to meet so many talented people. Among them, I would like to mention some that influenced my work. First, I would like to thank Ran Canetti and Tal Rabin, both who guided and inspired me over the course of many hours of research, discussions and fun. I gratitude to Tal from my Summer Internship at the cryptography group at IBM T.J. Watson in 2011. I am also grateful to Oded Goldreich for the significant time and effort that he dedicated to improving this work. A special note of thanks must also be given to Benny Pinkas and Yuval Ishai for some fruitful discussions.

This project would have never reached its ending without the help of my very talented coauthors: Ran Canetti, Carmit Hazay, Abhishek Jain, Adriana Lopez-Alt, Claudio Orlandi, Tal Rabin, Thomas Schneider, Eran Tromer, Vinod Vaikuntanathan, Daniel Wichs, Hila Zarosim and Michael Zohner. Thank you all.

I am leaving Bar-Ilan University after many years of a great social atmosphere, and I am sure that I will miss this place. I would like to thank my closest friends here who made these years so wonderful. During the studies, I was fortuitous to also be a teaching assistant in the Computer Science department at Bar-Ilan, and I would like to thank my many students, who taught me some important lessons for life and gave me a lot of personal gratification.

Most importantly, I would like to thank my family. I especially thank my parents, Nina and Yigal, for their endless support, love, concern, strength, patience, encouragement, care and attention for all these years. To my brothers, my grandmother and grandfather - there are no words to express how grateful I am, and how much I love them.

Gilad Asharov March, 2014

Contents

Abstract

In the setting of secure multiparty computation, several distrustful parties wish to carry out a distributed computing task on their local private data while satisfying several security properties such as correctness, privacy, independence of inputs and fairness. The aim of secure multiparty computation (MPC) is to enable the parties to carry out the computation in a secure manner, eliminating any attempt of an adversarial entity to harm the execution. The concept of secure computation is very general and fundamental in cryptography, and models any distributed computing task, including simple computations as coin-tossing and broadcast, as well as more complex tasks such as electronic auctions and anonymous transactions. In this thesis, we study two foundational aspects of secure computation: *perfect security* and *fairness*.

In the first part of this thesis, we study perfect security in secure computation. A protocol that is perfectly secure cannot be broken even when the adversary has unlimited computational power and its security holds unconditionally.

One of the most fundamental results of secure computation was presented by Ben-Or, Goldwasser and Wigderson (BGW) in 1988. They demonstrated that any n-party functionality can be computed with *perfect security*, in the private channels model. When the adversary is semi-honest this holds as long as $t < n/2$ parties are corrupted, and when the adversary is malicious this holds as long as $t < n/3$ parties are corrupted. Unfortunately, a full proof of these results was never published. In this thesis, we remedy this situation and provide a full proof of security of the BGW protocol. This also includes a full description of the protocol for the malicious setting.

In addition to the above, we observe that by some simple and natural modifications, the BGW protocol can be significantly simplified and one of its expensive (and perhaps most complicated) subprotocols can be saved. We present a new multiplication protocol that is based on the original construction, but is simpler and achieves higher efficiency.

In the second part of this thesis, we study fairness in secure *two* party computation. Informally, the fairness property guarantees that if one party receives its output, then the other party does too.

The well-known impossibility result of Cleve (STOC 1986) implies that, in general, it is impossible to securely compute a function with *complete fairness* without an honest majority. Until recently, the accepted belief has been that *nothing* non-trivial can be computed with complete fairness in the two party setting. The surprising work of Gordon, Hazay, Katz and Lindell (STOC 2008) shows that this belief is false, and that there exist *some* non-trivial (deterministic, finite-domain) Boolean functions that can be computed fairly. This raises the fundamental question of characterizing complete fairness in secure two-party computation.

We focus on this question and give both positive and negative results. We first characterize which functions cannot be computed with complete fairness, since they are already ruled out by Cleve's impossibility. We define a simple property and show that any function that satisfies the property implies fair coin-tossing (i.e., the existence of a fair protocol for computing the function implies the existence of a fair coin-tossing protocol), and therefore is ruled out by Cleve's impossibility. On the other hand, we show that any function that does not satisfy the property cannot be used to construct a fair coin-tossing protocol (in the information theoretic setting). This extends our knowledge of what cannot be fairly computed, and provides a focus on which functions may potentially be computed fairly.

In addition, we define another property and show that any function that satisfies this property can be computed with complete fairness. Surprisingly, our results show not only that *some* or *few* functions can be computed fairly, but rather an enormous number of functions can be computed fairly, including very non-trivial functionalities like set-membership, private evaluation of Boolean function, private matchmaking, set-disjointness and more. Finally, we demonstrate that fairness can also be obtained in some cases in the class of asymmetric Boolean functions (where the output of the parties is not necessarily the same), and in the class of non-binary outputs.

Chapter 1

Introduction

1.1 Secure Computation

In the setting of secure multiparty computation, a set of n parties with possibly private inputs wish to securely compute some function of their inputs in the presence of adversarial behavior. In a nutshell, the computation should be such that each party receives its correct output (*correctness*), and none of the parties learn anything beyond their prescribed output (*privacy*). The concept of secure computation is very general and fundamental in cryptography. It was introduced by Yao in 1982 [99], and has been studied extensively since then. Several examples for such computation tasks may be:

- *Joint database computation.* Assume that two hospitals hold medical databases on their patients, including some sensitive information on the diseases of their patients, their family background, habits, etc. These two hospitals wish to perform research, including analysis and data-mining on the *union* of their databases. However, each one of the parties is committed to the privacy of its patients, and the information of each patient is *confidential.* Any unauthorized disclosure of private information can make the perpetrator subject to criminal penalties. Secure computation enables performing research on the *joint* databases without revealing any unnecessary information on the individuals.
- *Private database access.* We all ask "Google" for a name, term or a keyword, and receive a response. This operation is, in fact, an example of database access: One party ("Google") holds a huge database and an individual person queries this database. Sometimes, the queries may include some sensitive information, such as a personal question on a disease, disability, education or employment. Using secure computation, one can query the database without the database knowing what has been asked.
- *Electronic auctions.* Consider the task of an auction, where several bidders bid for a work contract over the Internet, and the lowest bid gets the contract. The bids should remain private, and each bidder should choose its bid independently from the others.

The above are just a few examples of tasks that are possible due to secure computation, and reflects the power that is concealed in this field.

Security in multiparty computation. In order to claim and prove that a protocol is secure, or, alternatively, to claim that the adversary cannot manipulate the execution, a definition for security of protocols is required. The security requirements from a secure protocol are that nothing is learned from the protocol other than the output (*privacy*), that the output is distributed according to the prescribed functionality (*correctness*), that parties cannot choose their inputs as a function of the others' inputs (*independence of inputs*), and that the corrupted parties should receive their outputs if and only if the honest parties also receive their outputs (*fairness*).

The actual definition [55, 85, 15, 27, 53] formalizes this by comparing the result of a real protocol execution with the result of an execution in an ideal model where an incorruptible trusted party carries out the computation for the parties. This definition has come to be known as the "ideal/real simulation paradigm". In more detail, in the ideal world execution, the parties simply send their inputs to the trusted party who computes the desired function and passes each party its prescribed output. Notice that all security requirements mentioned above are ensured in this execution. In contrast, in the real world execution a real protocol is run by the parties (with no trusted party). We define "security" by comparing the outcomes of these two executions. Specifically, a protocol is secure if for any adversary in the real execution, there exists an adversary in the ideal model such that the input/output distributions of the parties in the real and ideal executions are the same. Since the adversary in the ideal model has no ability to make any damage, it implies that the real world adversary cannot break the security in the real model, and therefore security is guaranteed.

It is known that there exist functionalities that cannot be computed securely under the above (informally stated) definition, where the property that is being breached is fairness [34]. In particular, under certain circumstances when there is no honest majority, honest parties may not receive their prescribed output, and fairness is not always guaranteed. This imperfection is *inherent*. However, protocols may still guarantee *all* security properties defined above, *except* for fairness. Security in this case is formalized by relaxing the ideal model and "asking" the adversary whether to give the output to the honest parties. Usually, full security (including fairness) is considered when there is an honest majority, whereas security with no fairness ("security-with-abort") is considered when there is no honest majority. However, for some particular functions, full security is possible also when there is no honest majority [58]. Jumping ahead, characterizing for which functions fairness is possible is one of the goals of this thesis.

Feasibility of secure multiparty computation. There are many different settings within which secure computation has been considered. Regarding the adversary, one can consider semi-honest adversaries (who follow the protocol specification but try to learn more than they should by inspecting the protocol transcript) or malicious adversaries (who may follow an arbitrary strategy). In addition, an adversary may be limited to polynomial-time (as in the computational setting) or unbounded (as in the information-theoretic setting). Finally, the adversary may be static (meaning that the set of corrupted parties is fixed before the protocol execution begins) or adaptive (meaning that the adversary can adaptively choose to corrupt throughout the protocol execution).

The first feasibility result for secure computation was in the computational setting for the two-party case with respect to semi-honest adversary, and was provided by [100]. Other feasibility results were presented in the mid to late 1980's. The most central of these are as follows, where the adversary is assumed to be malicious: (n denotes the number of parties, t denotes the number of corrupted parties):

1. For $t < n/3$, secure multiparty computation (with fairness) can be achieved for any function in a point-to-point network. This can be achieved both in the computational setting [54] (under suitable cryptographic assumptions), and in the information theoretic setting [22, 32].
2. For $t < n/2$, secure multiparty computation (with fairness) can be achieved for any function assuming that the parties have access to a broadcast channel, both in the computational setting [54] (under suitable cryptographic assumptions), and the information theoretic setting [92] (with statistical security).
3. For $t \leq n$, secure multiparty computation with *no* fairness can be achieved for any function in the computational setting, under suitable cryptographic assumptions, and assuming access to a broadcast channel [54, 53].

1.2 Perfect Security

In the first part of this thesis, we focus on perfect security in secure computation. In this setting, the adversary is not bound to any complexity class (and in particular, is not assumed to be polynomial-time). Results in this model require no complexity or cryptographic hardness assumption and security holds unconditionally.

1.2.1 A Full Proof of the BGW Protocol (Chapter 2)

Our focus is on the results of Ben-Or, Goldwasser and Wigderson (BGW) [22], who showed that every functionality can be computed with *perfect security* in the presence of semi-honest adversaries controlling a minority of parties, and in the presence of malicious adversaries controlling less than a third of the parties. The discovery that secure computation can be carried out information theoretically, and the techniques used by BGW, were highly influential. In addition, as we shall see, the fact that security is *perfect* – informally meaning that there is a *zero probability* of cheating by the adversary – provides real security advantages over protocols that have a negligible probability of failure (cf. [75]). For this reason, we focus on the BGW protocol [22] rather than on the protocol of Chaum et al. [32].

Our Results

The BGW protocol is widely regarded as a classic result and one of the fundamental theorems of the field. It had a huge impact on the field of cryptography, and many papers and results are built upon it. Despite the importance of this result, a full proof of its security has never appeared. In addition, a full description of the protocol in the malicious setting was also never published. In this thesis, we remedy this situation and provide a full description and proof of the BGW protocol, for both the semi-honest and malicious settings. We prove security relative to the ideal/real definition of security for multiparty computation. This also involves carefully defining the functionalities and sub-functionalities that are used in order to achieve the result, as needed for presenting a modular proof. Providing a full proof for the security of the BGW protocol fulfills an important missing cornerstone in the field of secure computation.

Our main result is a proof of the following informally stated theorem:

Theorem 1 (basic security of the BGW protocol – informally stated): *Consider a synchronous network with pairwise private channels and a broadcast channel. Then:*

1. Semi-honest: *For every n-ary functionality f, there exists a protocol for computing f with perfect security in the presence of a static semi-honest adversary controlling up to $t < n/2$ parties;*
2. Malicious: *For every n-ary functionality f, there exists a protocol for computing f with perfect security in the presence of a static malicious adversary controlling up to $t < n/3$ parties.*[1]

Theorem 1 is proven in the classic setting of a static adversary and stand-alone computation (where the latter means that security is proven for the case that only a single protocol execution takes place at a time). However, using theorems stating that perfect security under certain conditions derives security in some other more powerful adversarial models, we derive security "for free" in these models. First, we show that the protocol is also secure under universal composability (UC) [28], meaning that security is guaranteed to hold when many arbitrary protocols are run concurrently with the secure protocol. Second, our proof refers to information-theoretic security in the ideal private channels model (namely, the adversary cannot tap the communication between two honest parties). We also derive a corollary to the computational model with authenticated channels only (where the adversary can tap the communication), assuming semantically secure encryption [56, 99]. Finally, we also derive security in the presence of an adaptive adversary, alas with inefficient simulator.[2]

1.2.2 Efficient Perfectly-Secure Multiplication Protocol (Chapter 3)

We observe that by some simple and natural modifications, the protocol of BGW can be significantly simplified and one of its expensive subprotocols can be saved.

Before going into further details, we first provide a high-level overview of the BGW protocol. An important building block for the BGW protocol is Shamir's secret sharing scheme [94]. In a t-out-of-n secret sharing, a dealer can divide a secret into n shares, such that any subset of t shares does not provide any information about the secret, and any subset of more than t shares uniquely defines the secret, and the latter can be efficiently reconstructed from these shares.

The BGW protocol works by having the parties compute the desired function f (without loss of generality, from n inputs to n outputs) by securely emulating the computation of an arithmetic circuit computing f. In this computation, the parties compute shares of the output

[1]We remark that $t < n/3$ is not merely a limitation of the way the BGW protocol works. In particular, the fact that at most $t < n/3$ corruptions can be tolerated in the malicious model follows immediately from the fact that at most $t < n/3$ corruptions can be tolerated for Byzantine agreement [87]. In contrast, we recall that given a broadcast channel, it *is* possible to securely compute any functionality with information-theoretic (statistical) security for any $t < n/2$ [92, 14].

[2]In [29], it was shown that any protocol that is proven perfectly secure under the security definition of [41] is also secure in the presence of adaptive adversaries, alas with inefficient simulation. We use this to derive security in the presence of adaptive adversaries, albeit with the weaker guarantee provided by inefficient simulation (in particular, this does not imply adaptive security in the computational setting).

of a circuit gate given shares of the input wires of that gate. To be more exact, the parties first share their inputs with each other using Shamir's secret sharing (in the case of malicious adversaries, a *verifiable* secret sharing protocol (cf. [33, 54]) is used). The parties then emulate the computation of each gate of the circuit, computing Shamir shares of the gate's output from the Shamir shares of the gate's inputs. As we shall see, this secret sharing has the property that addition gates in the circuit can be emulated using local computation only. Thus, the parties only interact in order to emulate the computation of multiplication gates; this step is the most involved part of the protocol. Finally, the parties reconstruct the secrets from the shares of the output wires of the circuit in order to obtain their output.

We show that the multiplication protocol of BGW (where the parties emulate the computation of multiplication gate) can be significantly simplified and one of its expensive (and perhaps most complicated) subprotocols can be saved. We propose a new protocol that is based on the multiplication protocol of BGW, but utilizes the additional information given in a *bivariate* polynomial sharing (as used in the verifiable secret sharing of BGW and Feldman [22, 45]) in order to significantly improve the efficiency of each multiplication. We provide a full specification and full proof of security for this new protocol.

Our protocol achieves *perfect security*, as in the original work of BGW. We stress that perfect security is not just a question of aesthetics, but rather provides a substantive advantage over protocols that are only proven statistically secure. Using the fact that our protocol is perfectly secure, we also derive concurrent security and even adaptive security (albeit with the weaker guarantee provided by inefficient simulator), exactly as with our proof of the BGW protocol. We also derive a corollary to the computational model with authenticated channels (instead of perfect security with ideal private channels).

1.3 Complete Fairness in Secure Two-Party Computation

In the second part of this thesis, we study complete fairness in secure two-party computation (in the computational setting). Recall that *complete fairness* means, intuitively, that the adversary learns the output if and only if the honest parties learn the output.

As we have mentioned, a basic feasibility result of secure computation states that when the majority of the parties are honest, it is possible to securely compute any functionality with complete fairness [54, 22, 32, 92, 53]. In the case when an honest majority is not guaranteed, including the important case of the two-party settings where one may be corrupted, it is possible to securely compute any function while satisfying *all* security properties *except* for fairness [100, 54, 53]. The deficiency of fairness is not just an imperfection of these constructions, but rather a result of inherent limitation. The well-known impossibility result of Cleve [34] shows that there exist functions that cannot be computed by two parties with complete fairness, and thus, fairness cannot be achieved *in general*.

Specifically, Cleve showed that the coin-tossing functionality, where two parties toss an unbiased fair coin, cannot be computed with complete fairness. This implies that any function that can be used to toss a fair coin (like, for instance, the Boolean XOR function) cannot be computed fairly as well. From Cleve's result and until 2008, the accepted belief was that *only trivial functions*[3] can be computed with complete fairness. This belief is not just a hasty conclusion, and is based on a solid and substantiate intuition. In any protocol computing any interesting function, the parties move from a state of no knowledge about the output to full knowledge about it. Protocols proceed in rounds and the parties cannot exchange information simultaneously, therefore, apparently, there must be a point in the execution where one party has more knowledge about the output than the other. Aborting at that round yields the unfair situation where one party learns the output alone.

Questions regarding fairness have been studied extensively since the early days of secure computation. These questions do not deal with the notion of complete fairness, but rather on several relaxations of fairness, possibly because of Cleve's result. In 2008, the work of [58] was published and showed very interesting *feasibility* results for complete fairness in the two-party setting, and changed our perception regarding fairness. We will mainly focus on this work and its ramifications. However, before describing the work of [58] and our goals in more detail, we first give a high level overview of the history of fairness in secure computation. The enormous amount of works on this topic reflects the importance of this topic and its significant to the theory of secure computation.

1.3.1 Fairness in Secure Computation – Related Work

Impossibility of Complete Fairness

The first impossibility result of fairness appeared in 1980, even before secure computation was introduced, in the work of Even and Yacobi [44]. The paper dealt with several applications of digital signatures, and one of the studied questions was the task of exchanging signatures. The paper showed impossibility of this latter task, by arguing that one party must have enough information to efficiently produce a verifiable signature before its opponent. This was the first impossibility for the "fair exchange" functionality (the first party holds secret input x, the second party holds y, and the parties wish to exchange their secrets such that the first learns y and the second learns x).

This work was later formalized and strengthened by the impossibility result of Cleve [34]. As mentioned above, Cleve's theorem showed that there does not exist a completely fair protocol for enabling two parties to agree on an unbiased coin. In particular, Cleve showed that in any r-round coin-tossing protocol, there exists an efficient adversary that can bias the output of the honest party by $\Omega(1/r)$. In fact, the adversary that is constructed in the proof is not even a malicious adversary, but rather is only "fail-stop", meaning that it behaves honestly throughout the interaction with the only exception that it may halt the execution prematurely.

The relation between coin-tossing and fair exchange was also studied, and it was shown that the task of coin-tossing is strictly "weaker" than the task of fair-exchange. That is, the impossibility of coin-flipping implies the impossibility of fair-exchange, while the inverse does not. In particular, impossibility of coin-flipping implies impossibility of the most degenerated fair-exchange task (the Boolean XOR function), since the existence of a fair protocol for the XOR function implies the existence of a fair protocol for coin-tossing. On the other hand, even if coin-tossing would have been possible, the Boolean XOR function is still impossible to compute fairly [2].

[3]In our context, the term "trivial functions" refers to constant functions, i.e., functions that depend on only one party's input and functions where only one party receives output. It is easy to see that these functions can be computed fairly.

Other impossibility results. Among other things, the work of [60] strengthened the difficulty for achieving fairness, by showing the existence of a "fairness hierarchy". Intuitively, they showed that even given an using ideal box for exchanging "short" strings of length $\ell = O(\log \kappa)$-bits, no protocol can exchange $\ell + 1$ bits. In fact, the result is even more general and enables the parties to use *any* short ℓ-bit primitive. The work of [2] proves impossibility for some other randomized functions where the parties have no inputs (i.e., sampling outputs from some distribution), where coin-tossing and the impossibility result of Cleve may be interpreted as a special case of this result. This strengthens our understanding of when fairness is impossible.

Possibility of Partial Fairness

As we have seen, there exist functionalities that cannot be securely computed with complete fairness when there is no honest majority. Despite this impossibility, some partial fairness is still possible to achieve. A number of different approaches have been considered, with the aim of finding "relaxed" models of fairness which can be achieved.

Gradual release. The extreme case of breaching fairness occurs when one party learns the *whole* output alone, while the other party does not gain any information regarding the output. This phenomenon can be reduced, using the notion of gradual release. The idea is that instead of learning the output in "one shot", the parties learn the output gradually, that is, a party cannot learn its next piece of information before sending the other party its own piece. At any point one of the parties has an advantage over the other, but the protocol keeps this advantage relatively small (but this advantage cannot disappear). There are many papers that are based on the notion of gradual release [23, 42, 52, 91, 43, 100, 25, 67, 38, 24, 50, 89, 49], and we refer the reader to [57] for a nice survey.

Probabilistic fairness. In a nutshell, the works of [80, 95, 16, 35, 55, 21] use similar approach to gradual release, but conceptually a bit different. Here the players' confidence in the solution (correct output) increases over time, as opposed to the gradual release approach in which the solution becomes cryptographically easier to find after each round.

For instance, the work of [80] considers the tasks of exchanging one bit. Here, in each round the parties exchange random coins, where the distributions from which the coins are chosen are getting closer to the true input with each round. That is, in the first round the coins are uniform and independent from the actual inputs, whereas at the end of the executions the coins are the true inputs (i.e., the bits that they transmit). In each round in-between, the distributions are biased towards the actual inputs. By applying this approach, in any round the parties do not have certainty regarding the correct output, but their confidence about it is increased with each round. Beaver and Goldwasser [16] formalized the notion of fairness that is provided by this protocol, and show how any Boolean function can be computed with this notion of fairness.

Optimistic exchange. A different notion of fairness involves some limited trusted party. This trusted party remains offline during the computation, and is involved only when there is some misbehaviour during the interaction (i.e., a "judge"). The approach was suggested by Micali [83] and Asokan et al. [10] and is followed in the works of [11, 13, 88, 12, 48, 26, 84, 40, 77, 74, 73].

$1/p$-security. As mentioned above, the two common security definitions in the ideal-real paradigm consider either full security (with complete fairness), or security with-abort (with no fairness at all). Although many works achieve some sort of fairness, to the best of our knowledge, there are few works that formalize partial fairness in the ideal-real simulation paradigm [49, 61, 4]. The work of [61] formalizes partial fairness in this paradigm, and suggests the notion of $1/p$-security. Informally, instead of requiring computationally indistinguishability between the ideal world and the real world (i.e., no efficient adversary can distinguish between an output of the real execution and an output of the ideal execution with some non-negligible probability), the requirement is relaxed such that the outputs cannot be efficiently distinguished with probability greater than $1/p$, for some specified polynomial p. This work also shows how to achieve this notion for two-party functions, and the work of [19] shows possibility for this notion for the multiparty case.

Optimal coin-tossing. As we mentioned above, Cleve's impossibility shows that for any r-round coin-tossing protocol, there exists an efficient adversary that succeeds in biasing the output of the honest party by $\Omega(1/r)$. This raises the question of whether this lower bound is tight, i.e., whether there exists a protocol that guarantees maximal bias of $O(1/r)$. This question addresses the "power" of the adversary in the real world, and how it can influence the output of the honest party and has direct ramifications to fairness.

Cleve [34] gave a protocol for coin-tossing that guarantees maximal bias of $O(1/\sqrt{r})$, when assuming the existence of one-way functions. In 1993, Cleve and Impagliazzo [36] showed that for any r-round coin-tossing protocol there exists an *inefficient* adversary that succeeds in biasing the output of the honest party by $\Omega(1/\sqrt{r})$, which suggests that Cleve's protocol is optimal. However, a recent result [86] shows that, assuming the existence of Oblivious Transfer [43, 53], there exists a protocol for coin-tossing that guarantees maximal bias of $O(1/r)$. This implies that Cleve's lower bound is optimal. This result was extended in the multiparty case, in which less than 2/3 of the parties are corrupted [20].

Fairness with rational players. The work of Halpern and Teague introduced the notion of "rational secret sharing and multiparty computation" [63]. In this model, we do not divide the players into "good" (honest) and "bad" (malicious), but rather all are rational players (in the game-theoretic sense) that wish to maximize their utilities. In particular, it is assumed that each party prefers to learn the output rather then not learning it, and prefers to be the only party that learns the output. This approach is incomparable to the standard notion of fairness in secure computation: On the one hand, fairness is harder to achieve since *all* parties are rational and there are no honest parties; and on the other hand, fairness is easier to achieve since the "adversary" is rational and therefore we can predict its behavior. It has been shown that fairness in this model can sometimes be achieved [1, 81, 71, 70, 5, 47].

1.3.2 The Work of Gordon, Hazay, Katz and Lindell [58]

Our understanding regarding fairness was recently changed by the surprising work of Gordon, Hazay, Katz and Lindell (GHKL) [58]. This work focuses on complete fairness, and shows that there *exist* some non-trivial (deterministic, finite-domain) Boolean functions that can be computed in the malicious settings with *complete fairness*. The work re-opens the research on this subject. The fact that *some* functions can be computed fairly, while some others were proven to be impossible to compute fairly, raises the following fundamental question:

Which functions can be computed with complete fairness?

Since the publication of [58], there have been no other works that further our understanding regarding which (Boolean) functions can be computed fairly without an honest majority in the two party setting. Specifically, Cleve's impossibility result is the only known function that cannot be computed fairly, and the functions for which [58] shows possibility are the only known possible functions. There is therefore a large class of functions for which we have no idea as to whether or not they can be securely computed with complete fairness.

To elaborate further, the work of [58] shows that any (polynomial-size domain) function that does not contain an embedded XOR (i.e., inputs x_1, x_2, y_1, y_2 such that $f(x_1, y_1) = f(x_2, y_2) \neq f(x_1, y_2) = f(x_2, y_1)$) can be computed fairly. Examples of functions without an embedded XOR include the Boolean OR / AND functions and the greater-than function. Given the fact that Cleve's impossibility result rules out completely fair computation of Boolean XOR, a natural conjecture is that any function that does contain an embedded XOR is impossible to compute fairly. However, [58] shows also that this conclusion is incorrect. It considers a *specific* function that does contain an embedded XOR, and constructs a protocol that securely computes this function with complete fairness. Furthermore, the paper presents a generalization of this protocol that may potentially compute a large class of functions. It also shows how to construct a (rather involved) set of equations for a given function, that indicates whether the function can be computed fairly using this protocol. We later refer to this protocol as "the GHKL protocol".

The results in [58] completely change our perception regarding fairness. The fact that *something* non-trivial can be computed fairly is surprising in that it contradicts the aforementioned natural intuition and common belief, and raises many interesting questions. For instance, are there many functions that can be computed fairly, or only a few? Which functions can be computed fairly? Which functions can be computed using the generalized GHKL protocol? What property distinguishes these functions from the functions that are impossible to compute fairly?

Our Work

1.3.3 A Full Characterization of Coin-Tossing (Chapter 4)

Motivated by the fundamental question of characterizing which functions can be computed with complete fairness, in Chapter 4, we analyze which functions *cannot* be computed fairly since they are already ruled out by Cleve's original result. That is, we show which finite-domain Boolean functions "imply" the coin-tossing functionality. We provide a simple property (criterion) on the truth table of a given Boolean function. We then show that for every function that satisfies this property, it holds that the existence of a protocol that fairly computes the given function implies the existence of a protocol for fair coin-tossing in the presence of a fail-stop adversary, in contradiction to Cleve's impossibility result. This implies that the functions that satisfy the property cannot be computed fairly. The property is very simple, clean and general.

In a nutshell, the property that we define over the function's truth table relates to the question of whether or not it is possible for one party to singlehandedly change the probability that the output of the function is 1 (or 0) based on how it chooses its input. That is, assume that there exists a fair protocol that fairly computes the function and assume that each party has some distribution over its inputs, such that once it chooses its input according to this distribution in the execution of the protocol, the other party cannot bias the result of the execution. Once these input distributions exist for both parties, it is easy to see that the function implies a fair-coin tossing functionality. This is because a fair protocol for the function implies the existence of a fair protocol for some δ-coin. In order to obtain a fair coin, we can apply the method of von-Neumann [97], and to conclude that this function implies fair coin-tossing.

The more challenging and technically interesting part of our work is a proof that the property is *tight.* Namely, we show that a function f that does not satisfy the property *cannot* be used to construct a fair coin-tossing protocol (in the information theoretic setting). More precisely, we show that it is impossible to construct a fair two-party coin-tossing protocol, even if the parties are given access to a trusted party that computes f *fairly* for them. We prove this impossibility by showing the existence of an (inefficient) adversary that can bias the outcome with non-negligible probability. Thus, we prove that it is not possible to toss a coin with information-theoretic security, when given access to fair computations of f. We stress that this "impossibility" result is actually a source of optimism, since it *may* be possible to securely compute such functions with complete fairness. Indeed, the fair protocols presented in [58] are for functions for which the property does not hold,[4] and in Chapter 5 we show possibility for many more functions that do not satisfy this property.

It is important to note that our proof that functions that do not satisfy the property do not imply coin tossing is very different to the proof of impossibility by Cleve. Specifically, the intuition behind the proof by Cleve is that since the parties exchange messages in turn, there must be a point where one party has more information than the other about the outcome of the coin-tossing protocol. If that party aborts at this point, then this results in bias. This argument holds since the parties cannot exchange information simultaneously. In contrast, in our setting, the parties *can* exchange information simultaneously via the computation of f. Thus, our proof is conceptually very different to that of Cleve, and in particular, is not a reduction to the proof by Cleve.

1.3.4 Towards Characterizing Complete Fairness (Chapter 5)

Finally, in Chapter 5 we study which functions *can* be computed with complete fairness. We find an interesting connection between a geometric representation of the function and the possibility of fairness. We show that any function that defines a full-dimensional geometric object *can be computed with complete fairness.* That is, we present a simple property on the truth table of the function, and show that that for every function that satisfies this property, the function can be computed fairly. This extends our knowledge of what can be computed fairly, and is another important step towards a full characterization for fairness.

Our results deepen our understanding of fairness and show that many more functions can be computed fairly than what has been thought previously. Using results from combinatorics, we show that a random Boolean function with distinct domain sizes (i.e., functions $f : X \times Y \to \{0,1\}$ where $|X| \neq |Y|$) defines a full-dimensional geometric object with overwhelming probability. Therefore, surprisingly, *almost all* functions with distinct domain sizes can be

[4]We remark that since our impossibility result is information theoretic, there is the possibility that some of the functions for which the property does not hold do imply coin tossing computationally. In such a case, the impossibility result of Cleve still applies to them. See further discussion in the body of the thesis.

computed with complete fairness. Although only one bit of information is revealed by output, the class of Boolean functions that define full-dimensional geometric objects (and therefore can be computed with complete fairness) is very rich, and includes a fortune of interesting and non-trivial tasks, like *set-membership*, *private evaluation of Boolean function*, *private matchmaking*, *set-disjointness* and more.

Furthermore, we provide an additional property that indicates that a function *cannot* be computed using the protocol of GHKL (with the particular simulation strategy described in [58]). This property is almost always satisfied in the case where $|X| = |Y|$. Thus, at least at the intuitive level, almost all functions with $|X| \neq |Y|$ can be computed fairly, whereas almost all functions with $|X| = |Y|$ cannot be computed using the only known possibility result that we currently have for fairness. We emphasize that this negative result does not rule out the possibility of these functions using some other protocol or even using the protocol of [58] itself using some other simulation strategy. Moreover, we remark that when incorporating these results with the full characterization of coin-tossing, our characterization for fairness is not tight, and there exists a class of functions that do not imply fair coin-tossing (and therefore are not ruled out by Cleve's impossibility), and also cannot be computed using the protocol of [58] (with that particular simulation strategy).

In addition to the above, we also consider larger families of functions rather than the symmetric Boolean functions with finite domain, and show that fairness is sometimes possible in these classes. We consider the class of asymmetric functions where the parties do not necessarily get the same output, as well as the class of functions with non-binary outputs. This is the first time that fairness is shown to be possible in these families of functions, and shows that the fairness property can be obtained in a much larger and wider class of functions than previously known.

1.4 Organization

The thesis consists of two parts, both of which deal with aspects of secure computation. Part I deals with perfect security in secure-computation, and Part II deals with the question of complete fairness. In spite of the risk of a small amount of repetition, each part is self-contained and includes the necessary definitions and preliminaries. Each one of the parts also consists of two closely related chapters. Chapter 3 is based on Chapter 2, and Chapter 5 is based on the definitions that appear in Chapter 4. Chapters 2,3,4,5 were published in [6, 8, 9, 3], respectively.

Part I

Perfect Security

Chapter 2

A Full Proof of the BGW Protocol

One of the most fundamental results of secure computation was presented by Ben-Or, Goldwasser and Wigderson (BGW) in 1988. They demonstrated that any n-party functionality can be computed with *perfect security*, in the private channels model. When the adversary is semi-honest this holds as long as $t < n/2$ parties are corrupted, and when the adversary is malicious this holds as long as $t < n/3$ parties are corrupted. Unfortunately, a full proof of these results was never published. In this chapter, we provide a full proof of security of the BGW protocol. This also includes a full description of the protocol for the malicious setting.

2.1 Introduction

2.1.1 The BGW Protocol

Our focus is on the results of Ben-Or, Goldwasser and Wigderson (BGW) [22], who showed that every functionality can be computed with *perfect security* in the presence of semi-honest adversaries controlling a minority of parties, and in the presence of malicious adversaries controlling less than a third of the parties. We have already discussed the importance of this protocol and its historical context in the introduction, and thus we start with a high-level overview of the protocol.

The BGW construction – an overview. The BGW protocol works by having the parties compute the desired function f (from n inputs to n outputs) by securely emulating the computation of an arithmetic circuit computing f. In this computation, the parties compute shares of the output of a circuit gate given shares of the input wires of that gate. To be more exact, the parties first share their inputs with each other using Shamir's secret sharing [94]; in the case of malicious adversaries, a *verifiable* secret sharing protocol (cf. [33, 54]) is used. The parties then emulate the computation of each gate of the circuit, computing Shamir shares of the gate's output from the Shamir shares of the gate's inputs. As we shall see, this secret sharing has the property that addition gates in the circuit can be emulated using local computation only. Thus, the parties only interact in order to emulate the computation of multiplication gates; this step is the most involved part of the protocol. Finally, the parties reconstruct the secrets from the shares of the output wires of the circuit in order to obtain their output.

We proceed to describe the protocol in a bit more detail. Shamir's secret sharing enables the sharing of a secret s amongst n parties, so that any subset of $t+1$ or more parties can efficiently reconstruct the secret, and any subset of t or less parties learn no information whatsoever about the secret. Let $\mathbb{F}$ be a finite field of size greater than n, let $\alpha_1, \ldots, \alpha_n$ be n distinct non-zero field elements, and let $s \in \mathbb{F}$. Then, in order to share s, a polynomial $p(x) \in \mathbb{F}[x]$ of degree t with constant term s is randomly chosen, and the share of the ith party P_i is set to $p(\alpha_i)$. By interpolation, given any $t+1$ points it is possible to reconstruct p and compute $s = p(0)$. Furthermore, since p is random, its values at any t or less of the α_i's give no information about s.

Now, let n denote the number of parties participating in the multiparty computation, and let t be a bound on the number of corrupted parties. The first step of the BGW protocol is for all parties to share their inputs using Shamir's secret sharing scheme. In the case of semi-honest adversaries, plain Shamir sharing with a threshold $t < n/2$ is used, and in the case of malicious adversaries verifiable secret sharing (VSS) with a threshold $t < n/3$ is used. A verifiable secret sharing protocol is needed for the case of malicious adversaries in order to prevent cheating, and the BGW paper was also the first to construct a *perfect* VSS protocol.

Next, the parties emulate the computation of the gates of the circuit. The first observation is that addition gates can be computed locally. That is, given shares $p(\alpha_i)$ and $q(\alpha_i)$ of the two input wires to an addition gate, it holds that $r(\alpha_i) = p(\alpha_i) + q(\alpha_i)$ is a valid sharing of the output wire. This is due to the fact that the polynomial $r(x)$ defined by the sum of the shares has the same degree as both $p(x)$ and $q(x)$, and $r(0) = p(0) + q(0)$.

Regarding multiplication gates, observe that by computing $r(\alpha_i) = p(\alpha_i) \cdot q(\alpha_i)$ the parties obtain shares of a polynomial $r(x)$ with constant term $p(0) \cdot q(0)$ as desired. However, the degree of $r(x)$ is $2t$, since the degrees of $p(x)$ and $q(x)$ are both t. Since reconstruction works as long as the polynomial used for the sharing is of degree t, this causes a problem. Thus, the multiplication protocol works by *reducing the degree* of the polynomial $r(x)$ back to t. In the case of semi-honest parties, the degree reduction can be carried out as long as $t < n/2$ (it is required that $t < n/2$ since otherwise the degree of $r(x) = p(x) \cdot q(x)$ will be greater than or equal to n, which is not fully defined by the n parties' shares). In the case of malicious parties, the degree reduction is much more complex and works as long as $t < n/3$. In order to obtain some intuition as to why $t < n/3$ is needed, observe that a Shamir secret sharing can also be viewed as a Reed-Solomon code of the polynomial [82]. With a polynomial of degree t, it is possible to correct up $(n-t-1)/2$ errors. Setting $t < n/3$, we have that $n \geq 3t+1$ and so $(n-t-1)/2 \geq t$ errors can be corrected. This means that if up to t malicious parties send incorrect values, the honest parties can use error correction and recover. Indeed, the BGW protocol in the case of malicious adversaries relies heavily on the use of error correction in order to prevent the adversary from cheating.

2.1.2 Our Results

Despite the importance of the BGW result, a full proof of its security has never appeared, and a full description of the protocol in the malicious setting was also never published. We remedy this situation, and our main result is a proof of the following informally stated theorem:

Theorem 2 (basic security of the BGW protocol – informally stated): *Consider a synchronous network with pairwise private channels and a broadcast channel. Then:*

1. Semi-honest: *For every n-ary functionality f, there exists a protocol for computing f with perfect security in the presence of a static semi-honest adversary controlling up to $t < n/2$ parties;*

2. Malicious: *For every n-ary functionality f, there exists a protocol for computing f with perfect security in the presence of a static malicious adversary controlling up to $t < n/3$ parties.*

The communication complexity of the protocol is $O(\mathsf{poly}(n) \cdot |C|)$ where C is an arithmetic circuit computing f, and the round complexity is linear in the depth of the circuit C.

All of our protocols are presented in a model with pairwise private channels *and* secure broadcast. Since we only consider the case of $t < n/3$ malicious corruptions, secure broadcast can be achieved in a synchronous network with pairwise channels by running Byzantine Generals [87, 76, 46]. In order to obtain (expected) round complexity linear in the depth of $|C|$, an expected constant-round Byzantine Generals protocol of [46] (with composition as in [79, 14]) is used.

Security under composition. Theorem 2 is proven in the classic setting of a static adversary and stand-alone computation, where the latter means that security is proven for the case that only a single protocol execution takes place at a time. Fortunately, it was shown in [75] that any protocol that is *perfectly* secure and has a black-box non-rewinding simulator, is also secure under universal composability [28] (meaning that security is guaranteed to hold when many arbitrary protocols are run concurrently with the secure protocol). Since our proof of security satisfies this condition, we obtain the following corollary, which relates to a far more powerful adversarial setting:

Corollary 3 (UC information-theoretic security of the BGW protocol): *Consider a synchronous network with private channels. Then, for every n-ary functionality f, there exists a protocol for computing f with perfect* universally composable security *in the presence of an static semi-honest adversary controlling up to $t < n/2$ parties, and there exists a protocol for computing f with perfect* universally composable security *in the presence of a static malicious adversary controlling up to $t < n/3$ parties.*

Corollary 3 refers to information-theoretic security in the ideal private channels model. We now derive a corollary to the computational model with authenticated channels only. In order to derive this corollary, we first observe that information-theoretic security implies security in the presence of polynomial-time adversaries (this holds as long as the simulator is required to run in time that is polynomial in the running time of the adversary, as advocated in [53, Sec. 7.6.1]). Furthermore, the ideal private channels of the information-theoretic setting can be replaced with computationally secure channels that can be constructed over authenticated channels using semantically secure public-key encryption [56, 99]. We have:

Corollary 4 (UC computational security of the BGW protocol): *Consider a synchronous network with authenticated channels. Assuming the existence of semantically secure public-key encryption, for every n-ary functionality f, there exists a protocol for computing f with* universally composable security *in the presence of a static malicious adversary controlling up to $t < n/3$ parties.*

We stress that unlike the UC-secure computational protocols of [31] (that are secure for any $t < n$), the protocols of Corollary 4 are in the *plain model*, with authenticated channels but with no other trusted setup (in particular, no common reference string). Although well-accepted folklore, Corollaries 3 and 4 have never been proved. Thus, our work also constitutes the first full proof that universally composable protocols exist in the plain model (with authenticated channels) for any functionality, in the presence of static malicious adversaries controlling any $t < n/3$ parties.

Adaptive security with inefficient simulation. In [29] it was shown that any protocol that is proven perfectly secure under the security definition of [41] is also secure in the presence of adaptive adversaries, alas with inefficient simulation. We use this to derive security in the presence of adaptive adversaries, albeit with the weaker guarantee provided by inefficient simulation (in particular, this does not imply adaptive security in the computational setting). See Section 2.8 for more details.

Organization. In Section 2.2, we present a brief overview of the standard definitions of perfectly secure multiparty computation and of the modular sequential composition theorem that is used throughout in our proofs. Then, in Section 2.3, we describe Shamir's secret sharing scheme and rigorously prove a number of useful properties of this scheme. In Section 2.4 we present the BGW protocol for the case of semi-honest adversaries. An overview of the overall construction appears in Section 2.4.1, and an overview of the multiplication protocol appears at the beginning of Section 2.4.3.

The BGW protocol for the case of malicious adversaries is presented in Sections 2.5, 2.6, 2.7. In Section 2.5 we present the BGW verifiable secret sharing (VSS) protocol that uses bivariate polynomials. This section includes background on Reed-Solomon encoding and properties of bivariate polynomials that are needed for proving the security of the VSS protocol. Next, in Section 2.6 we present the most involved part of the protocol – the multiplication protocol for computing shares of the product of shares. This involves a number of steps and subprotocols, some of which are new. The main tool for the BGW multiplication protocol is a subprotocol for verifiably sharing the product of a party's shares. This subprotocol, along with a detailed discussion and overview, is presented in Section 2.6.6. Our aim has been to prove the security of the original BGW protocol. However, where necessary, some changes were made to the multiplication protocol as described originally in [22]. Finally, in Section 2.7, the final protocol for secure multiparty computation is presented. The protocol is proven secure for any VSS and multiplication protocols that securely realize the VSS and multiplication functionalities that we define in Sections 2.5 and 2.6, respectively. In addition, an exact count of the communication complexity of the BGW protocol for malicious adversaries is given. We conclude in Section 2.8 by showing how to derive security in other settings (adaptive adversaries, composition, and the computational setting).

The specification of the protocol is full of detailed and contain full proofs. As a result, the specification of the protocol is not consecutive. A reader who may find it beneficial and more convenient to read the full specification continuously may refer to Appendix A.

2.2 Preliminaries and Definitions

In this section, we review the definition of perfect security in the presence of semi-honest and malicious adversaries. We refer the reader to [53, Sec. 7.6.1] and [27] for more details and discussion.

In the definitions below, we consider the *stand-alone* setting with a *synchronous* network, and perfectly *private channels* between all parties. For simplicity, we will also assume that the parties have a broadcast channel; as is standard, this can be implemented using an appropriate Byzantine Generals protocol [87, 76]. Since we consider synchronous channels and the computation takes place in clearly defined rounds, if a message is not received in a given round, then this fact is immediately known to the party who is supposed to receive the message. Thus, we can write "if a message is not received" or "if the adversary does not send a message" and this is well-defined. We consider *static corruptions* meaning that the set of corrupted parties is fixed ahead of time, and the *stand-alone setting* meaning that only a single protocol execution takes place; extensions to the case of adaptive corruptions and composition are considered in Section 2.8.

Basic notation. For a set A, we write $a \in_R A$ when a is chosen uniformly from A. We denote the number of parties by n, and a bound on the number of corrupted parties by t. Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a possibly probabilistic n-ary functionality, where $f_i(x_1, \ldots, x_n)$ denotes the ith element of $f(x_1, \ldots, x_n)$. We denote by $I = \{i_1, \ldots i_\ell\} \subset [n]$ the indices of the corrupted parties, where $[n]$ denotes the set $\{1, \ldots, n\}$. By the above, $|I| \leq t$. Let $\vec{x} = (x_1, \ldots, x_n)$, and let $\vec{x}_I$ and $f_I(\vec{x})$ denote projections of the corresponding n-ary sequence on the coordinates in I; that is, $\vec{x}_I = (x_{i_1}, \ldots, x_{i_\ell})$ and $f_I(\vec{x}) = (f_{i_1}(\vec{x}), \ldots, f_{i_\ell}(\vec{x}))$. Finally, to ease the notation, we omit the index i when we write the set $\{(i, a_i)\}_{i=1}^n$ and simply write $\{a_i\}_{i=1}^n$. Thus, for instance, the set of shares $\{(i_1, f(\alpha_{i_1})), \ldots, (i_\ell, f(\alpha_{i_\ell}))\}$ is denoted as $\{f(\alpha_i)\}_{i \in I}$.

Terminology. In this chapter, we consider security in the presence of both semi-honest and malicious adversaries. As in [53], we call security in the presence of a semi-honest adversary controlling t parties t-privacy, and security in the presence of a malicious adversary controlling t parties t-security. Since we only deal with perfect security in this chapter, we use the terms t-private and t-secure without any additional adjective, with the understanding that the privacy/security is always perfect.

2.2.1 Perfect Security in the Presence of Semi-Honest Adversaries

We are now ready to define security in the presence of semi-honest adversaries. Loosely speaking, the definition states that a protocol is t-private if the view of up to t corrupted parties in a real protocol execution can be generated by a simulator given only the corrupted parties' inputs and outputs.

The view of the ith party P_i during an execution of a protocol π on inputs $\vec{x}$, denoted $\text{VIEW}_i^\pi(\vec{x})$, is defined to be $(x_i, r_i; m_{i_1}, \ldots, m_{i_k})$ where x_i is P_i's private input, r_i is its internal coin tosses, and m_{i_j} is the jth message that was received by P_i in the protocol execution. For every $I = \{i_1, \ldots i_\ell\}$, we denote $\text{VIEW}_I^\pi(\vec{x}) = (\text{VIEW}_{i_1}^\pi(\vec{x}), \ldots \text{VIEW}_{i_\ell}^\pi(\vec{x}))$. The output of all parties from an execution of π on inputs $\vec{x}$ is denoted $\text{OUTPUT}^{\pi}(\vec{x})$; observe that the output of each party can be computed from its own (private) view of the execution.

We first present the definition for deterministic functionalities, since this is simpler than the general case of probabilistic functionalities.

Definition 2.2.1 (t-privacy of n-party protocols – deterministic functionalities):
Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a deterministic *n-ary functionality and let π be a protocol. We say that π is* t-private for f *if for every $\vec{x} \in (\{0,1\}^*)^n$ where $|x_1| = \ldots = |x_n|$,*

$$\text{OUTPUT}^{\pi}(x_1, \ldots, x_n) = f(x_1, \ldots, x_n) \tag{2.2.1}$$

and there exists a probabilistic polynomial-time algorithm $\mathcal{S}$ such that for every $I \subset [n]$ of cardinality at most t, and every $\vec{x} \in (\{0,1\}^*)^n$ where $|x_1| = \ldots = |x_n|$, it holds that:

$$\left\{ S\left(I, \vec{x}_I, f_I\left(\vec{x}\right)\right) \right\} \equiv \left\{ \text{VIEW}_I^{\pi}(\vec{x}) \right\} \ . \tag{2.2.2}$$

The above definition separately considers the issue of output correctness (Eq. (2.2.1)) and privacy (Eq. (2.2.2)), where the latter captures privacy since the ability to generate the corrupted parties' view given only the input and output means that nothing more than the input and output is learned from the protocol execution. However, in the case of probabilistic functionalities, it is necessary to intertwine the requirements of privacy and correctness and consider the *joint* distribution of the output of $\mathcal{S}$ and of the parties; see [27, 53] for discussion. Thus, in the general case of probabilistic functionalities, the following definition of t-privacy is used.

Definition 2.2.2 (t-privacy of n-party protocols – general case): *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a probabilistic n-ary functionality and let π be a protocol. We say that π is* t-private for f *if there exists a probabilistic polynomial-time algorithm $\mathcal{S}$ such that for every $I \subset [n]$ of cardinality at most t, and every $\vec{x} \in (\{0,1\}^*)^n$ where $|x_1| = \ldots = |x_n|$, it holds that:*

$$\left\{ (S(I, \vec{x}_I, f_I(\vec{x})), f(\vec{x})) \right\} \equiv \left\{ (\text{VIEW}_I^{\pi}(\vec{x}), \text{OUTPUT}^{\pi}(\vec{x})) \right\}. \tag{2.2.3}$$

We remark that in the case of deterministic functionalities, the separate requirements of Equations (2.2.1) and (2.2.2) actually imply the joint distribution of Eq. (2.2.3). This is due to the fact that when f is deterministic, $f(\vec{x})$ is a single value and not a distribution.

Our presentation – deterministic functionalities. For the sake of simplicity and clarity, we present the BGW protocol and prove its security for the case of deterministic functionalities only. This enables us to prove the overall BGW protocol using Definition 2.2.1, which makes the proof significantly simpler. Fortunately, this does not limit our result since it has already been shown that it is possible to t-privately compute *any probabilistic functionality* using a general protocol for t-privately computing any deterministic functionality; see [53, Sec. 7.3.1].

2.2.2 Perfect Security in the Presence of Malicious Adversaries

We now consider malicious adversaries that can follow an arbitrary strategy in order to carry out their attack; we stress that the adversary is not required to be efficient in any way. Security is formalized by comparing a real protocol execution to an ideal model where the parties just send their inputs to the trusted party and receive back outputs. See [27, 53] for details on how to define these real and ideal executions; we briefly describe them here.

Real model: In the real model, the parties run the protocol π. We consider a synchronous network with private point-to-point channels, and an authenticated broadcast channel. This means that the computation proceeds in rounds, and in each round parties can send private messages to other parties and can broadcast a message to all other parties. We stress that the adversary cannot read or modify messages sent over the point-to-point channels, and that the broadcast channel is authenticated, meaning that all parties know who sent the message and the adversary cannot tamper with it in any way. Nevertheless, the adversary is assumed to be *rushing*, meaning that in every given round it can see the messages sent by the honest parties before it determines the messages sent by the corrupted parties.

Let π be a n-party protocol, let $\mathcal{A}$ be an arbitrary machine with auxiliary input z, and let $I \subset [n]$ be the set of corrupted parties controlled by $\mathcal{A}$. We denote by $\text{REAL}_{\pi,\mathcal{A}(z),I}(\vec{x})$ the random variable consisting of the view of the adversary $\mathcal{A}$ and the outputs of the honest parties, following a real execution of π in the aforementioned real model, where for every $i \in [n]$, party P_i has input x_i.

Ideal model: In the ideal model for a functionality f, the parties send their inputs to an incorruptible trusted party who computes the output for them. We denote the ideal adversary by $\mathcal{S}$ (since it is a "simulator"), and the set of corrupted parties by I. An execution in the ideal model works as follows:

- **Input stage:** The adversary $\mathcal{S}$ for the ideal model receives auxiliary input z and sees the inputs x_i of the corrupted parties P_i (for all $i \in I$). $\mathcal{S}$ can substitute any x_i with any x'_i of its choice under the condition that $|x'_i| = |x_i|$.
- **Computation:** Each party sends its (possibly modified) input to the trusted party; denote the inputs sent by $x'_1, \ldots, x'_n$. The trusted party computes $(y_1, \ldots, y_n) = f(x'_1, \ldots, x'_n)$ and sends y_j to P_j, for every $j \in [n]$.
- **Outputs:** Each honest party P_j ($j \notin I$) outputs y_j, the corrupted parties output $\perp$, and the adversary $\mathcal{S}$ outputs an arbitrary function of its view.

Throughout the chapter, we will refer to communication between the parties and the functionality. For example, we will often write that a party sends its input to the functionality; this is just shorthand for saying that the input is sent to the trusted party who computes the functionality.

We denote by $\text{IDEAL}_{f,\mathcal{S}(z),I}(\vec{x})$ the outputs of the ideal adversary $\mathcal{S}$ controlling the corrupted parties in I and of the honest parties after an ideal execution with a trusted party computing f, upon inputs $x_1, \ldots, x_n$ for the parties and auxiliary input z for $\mathcal{S}$. We stress that the communication between the trusted party and $P_1, \ldots, P_n$ is over an ideal private channel.

Definition of security. Informally, we say that a protocol is secure if its real-world behavior can be emulated in the ideal model. That is, we require that for every real-model adversary $\mathcal{A}$ there exists an ideal-model adversary $\mathcal{S}$ such that the result of a real execution of the protocol with $\mathcal{A}$ has the same distribution as the result of an ideal execution with $\mathcal{S}$. This means that the adversarial capabilities of $\mathcal{A}$ in a real protocol execution are just what $\mathcal{S}$ can do in the ideal model.

In the definition of security, we require that the ideal-model adversary $\mathcal{S}$ run in time that is polynomial in the running time of $\mathcal{A}$, whatever the latter may be. As argued in [27, 53], this definitional choice is important since it guarantees that information-theoretic security implies computational security. In such a case, we say that $\mathcal{S}$ is of comparable complexity to $\mathcal{A}$.

Definition 2.2.3 *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-ary functionality and let π be a protocol. We say that π is t-secure for f if for every probabilistic adversary $\mathcal{A}$ in the real model, there exists a probabilistic adversary $\mathcal{S}$ of comparable complexity in the ideal model, such that for every $I \subset [n]$ of cardinality at most t, every $\vec{x} \in (\{0,1\}^*)^n$ where $|x_1| = \ldots = |x_n|$, and every $z \in \{0,1\}^*$, it holds that:*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z),I}(\vec{x})\right\} \equiv \left\{\text{REAL}_{\pi,\mathcal{A}(z),I}(\vec{x})\right\}.$$

Reactive functionalities. The above definition refers to functionalities that map inputs to outputs in a single computation. However, some computations take place in stages, and state is preserved between stages. Two examples of such functionalities are mental poker (where cards are dealt and thrown and redealt [54]) and commitment schemes (where there is a separate commitment and decommitment phase; see [28] for a definition of commitments via an ideal functionality). Such functionalities are called reactive, and the definition of security is extended to this case in the straightforward way by allowing the trusted party to obtain inputs and send outputs in phases; see [53, Section 7.7.1.3].

2.2.3 Modular Composition

The sequential modular composition theorem [27] is an important tool for analyzing the security of a protocol in a modular way. Let π_f be a protocol for securely computing f that uses a subprotocol π_g for computing g. Then the theorem states that it suffices to consider the execution of π_f in a hybrid model where a trusted third party is used to ideally compute g (instead of the parties running the real subprotocol π_g). This theorem facilitates a modular analysis of security via the following methodology: First prove the security of π_g, and then prove the security of π_f in a model allowing an ideal party for g. The model in which π_f is analyzed using ideal calls to g, instead of executing π_g, is called the g-hybrid model because it involves both a real protocol execution and an ideal trusted third party computing g.

More formally, in the hybrid model, the parties all have oracle-tapes for some oracle (trusted party) that computes the functionality g. Then, if the real protocol π_f instructs the parties to run the subprotocol π_g using inputs $u_1, \ldots, u_n$, then each party P_i simply writes u_i to its outgoing oracle tape. Then, in the next round, it receives back the output $g_i(u_1, \ldots, u_n)$ on its incoming oracle tape. We denote by $\text{HYBRID}^g_{\pi_f,\mathcal{A}(z),I}(\vec{x})$ an execution of protocol π_f where each call to π_g is carried out using an oracle computing g. See [27, 53] for a formal definition of this model for both the semi-honest and malicious cases, and for proofs that if π_f is t-private (resp., t-secure) for f in the g-hybrid model, and π_g is t-private (resp., t-secure) for g, then π_f when run in the real model using π_g is t-private (resp., t-secure) for f.

2.3 Shamir's Secret Sharing Scheme [94] and Its Properties

2.3.1 The Basic Scheme

A central tool in the BGW protocol is Shamir's secret-sharing scheme [94]. Roughly speaking, a $(t+1)$-out-of-n secret sharing scheme takes as input a secret s from some domain, and outputs n shares, with the property that it is possible to efficiently reconstruct s from every subset of $t+1$ shares, but every subset of t or less shares reveals nothing about the secret s. The value $t+1$ is called the threshold of the scheme. Note that in the context of secure multiparty computation with up to t corrupted parties, the threshold of $t+1$ ensures that the corrupted parties (even when combining all t of their shares) can learn nothing.

A secret sharing scheme consist of two algorithm: the first algorithm, called the sharing algorithm, takes as input the secret s and the parameters $t+1$ and n, and outputs n shares. The second algorithm, called the reconstruction algorithm, takes as input $t+1$ or more shares and outputs a value s. It is required that the reconstruction of shares generated from a value s yields the same value s.

Informally, Shamir's secret-sharing scheme works as follows. Let $\mathbb{F}$ be a finite field of size greater than n and let $s \in \mathbb{F}$. The sharing algorithm defines a polynomial $q(x)$ of degree t in $\mathbb{F}[x]$, such that its constant term is the secret s and all the other coefficients are selected uniformly and independently at random in $\mathbb{F}$.[1] Finally, the shares are defined to be $q(\alpha_i)$ for every $i \in \{1, \ldots, n\}$, where $\alpha_1, \ldots, \alpha_n$ are any n distinct non-zero predetermined values in $\mathbb{F}$. The reconstruction algorithm of this scheme is based on the fact that any $t+1$ points define exactly one polynomial of degree t. Therefore, using interpolation it is possible to efficiently reconstruct the polynomial $q(x)$ given any subset of $t+1$ points $(\alpha_i, q(\alpha_i))$ output by the sharing algorithm. Finally, given $q(x)$ it is possible to simply compute $s = q(0)$. We will actually refer to reconstruction using all n points, even though $t+1$ suffice, since this is the way that we use reconstruction throughout the chapter.

In order to see that any subset of t or less shares reveals nothing about s, observe that for every set of t points $(\alpha_i, q(\alpha_i))$ and every possible secret $s' \in \mathbb{F}$, there exists a unique polynomial $q'(x)$ such that $q'(0) = s'$ and $q'(\alpha_i) = q(\alpha_i)$. Since the polynomial is chosen randomly by the sharing algorithm, there is the same likelihood that the underlying polynomial is $q(x)$ (and so the secret is s) and that the polynomial is $q'(x)$ (and so the secret is s'). We now formally describe the scheme.

Shamir's $(t+1)$-out-of-n secret sharing scheme. Let $\mathbb{F}$ be a finite field of order greater than n, let $\alpha_1, \ldots, \alpha_n$ be any *distinct non-zero* elements of $\mathbb{F}$, and denote $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$. For a polynomial q Let $\mathsf{eval}_{\vec{\alpha}}(q(x)) = (q(\alpha_1), \ldots, q(\alpha_n))$.

- **The sharing algorithm for $\alpha_1, \ldots, \alpha_n$:** Let $\mathsf{share}_{\vec{\alpha}}(s, t{+}1)$ be the algorithm that receives for input s and $t+1$ where $s \in \mathbb{F}$ and $t < n$. Then, $\mathsf{share}_{\vec{\alpha}}$ chooses t random values $q_1, \ldots q_t \in_R \mathbb{F}$, independently and uniformly distributed in $\mathbb{F}$, and defines the polynomial:

$$q(x) = s + q_1 x + \ldots q_t x^t$$

[1]Throughout, when we refer to a polynomial of degree t, we mean of degree *at most* t.

where all calculations are in the field $\mathbb{F}$. Finally, it outputs $\mathsf{eval}_{\vec{\alpha}}(q(x)) = (q(\alpha_1), \ldots, q(\alpha_n))$, where $q(\alpha_i)$ is the share of party P_i.

- **The reconstruction algorithm:** Algorithm $\mathsf{reconstruct}_{\vec{\alpha}}(\beta_1, \ldots, \beta_n)$ finds the unique polynomial $q(x)$ of degree t such that for every $i = 1, \ldots, n$ it holds that $q(\alpha_i) = \beta_i$, when such a polynomial exists (this holds as long as $\beta_1, \ldots, \beta_n$ all lie on a single polynomial). The algorithm then outputs the coefficients of the polynomial $q(x)$ (note that the original secret can be obtained by simply computing $s = q(0)$).

By the above notation, observe that for every polynomial $q(x)$ of degree $t < n$, it holds that

$$\mathsf{reconstruct}_{\vec{\alpha}}(\mathsf{eval}_{\vec{\alpha}}(q(x))) = q(x). \tag{2.3.1}$$

Notation. Let $\mathcal{P}^{s,t}$ be the set of all polynomials with degree less than or equal to t with constant term s. Observe that for every two values $s, s' \in \mathbb{F}$, it holds that $|\mathcal{P}^{s,t}| = |\mathcal{P}^{s',t}| = |\mathbb{F}|^t$.

2.3.2 Basic Properties

In this section, we prove some basic properties of Shamir's secret sharing scheme (the proofs of these claims are standard but appear here for the sake of completeness). We first show that the value of a polynomial chosen at random from $\mathcal{P}^{s,t}$ at any single non-zero point is distributed uniformly at random in $\mathbb{F}$; this can be generalized to hold for any t points.

Claim 2.3.1 *For every $t \geq 1$, and for every $s, \alpha, y \in \mathbb{F}$ with $\alpha \neq 0$, it holds that:*

$$\Pr_{q \in_R \mathcal{P}^{s,t}} [q(\alpha) = y] = \frac{1}{|\mathbb{F}|}.$$

Proof: Fix s, y and α with $\alpha \neq 0$. Denote the ith coefficient of the polynomial $q(x)$ by q_i, for $i = 1, \ldots, t$. Then:

$$\Pr[q(\alpha) = y] \;=\; \Pr\left[y = s + \sum_{i=1}^{t} q_i \alpha^i\right] = \Pr\left[y = s + q_1\alpha + \sum_{i=2}^{t} q_i \alpha^i\right]$$

where the probability is taken over the random choice of $q \in_R \mathcal{P}^{s,t}$, or equivalently of the coefficients $q_1, \ldots, q_t \in_R \mathbb{F}$. Fix $q_2, \ldots, q_t$ and denote $v = \sum_{i=2}^{t} q_i \alpha^i$. Then, for a randomly chosen $q_1 \in_R \mathbb{F}$ we have that

$$\begin{aligned}
\Pr\Big[q(\alpha) = y\Big] &= \Pr\Big[y = s + q_1\alpha + v\Big] \\
&= \Pr\Big[q_1\alpha = y - s - v\Big] \\
&= \Pr\Big[q_1 = \alpha^{-1} \cdot (y - s - v)\Big] \\
&= \frac{1}{|\mathbb{F}|}
\end{aligned}$$

where the third equality holds since $\alpha \in \mathbb{F}$ and $\alpha \neq 0$ implying that α has an inverse, and the last equality is due to the fact that $q_1 \in_R \mathbb{F}$ is randomly chosen. ■

In the protocol for secure computation, a dealer hides a secret s by choosing a polynomial $f(x)$ at random from $\mathcal{P}^{s,t}$, and each party P_i receives a share, which is a point $f(\alpha_i)$. In this context, the adversary controls a subset of at most t parties, and thus receives at most t shares. We now show that any subset of at most t shares does not reveal *any* information about the secret. In Section 2.3.1, we explained intuitively why the above holds. This is formalized in the following claim that states that for every subset $I \subset [n]$ with $|I| \leq t$ and every two secrets s, s', the distribution over the shares seen by the parties P_i $(i \in I)$ when s is shared is identical to when s' is shared.

Claim 2.3.2 *For any set of distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$, any pair of values $s, s' \in \mathbb{F}$, any subset $I \subset [n]$ where $|I| = \ell \leq t$, and every $\vec{y} \in \mathbb{F}^\ell$ it holds that:*

$$\Pr_{f(x) \in_R \mathcal{P}^{s,t}} \left[\vec{y} = \left(\{f(\alpha_i)\}_{i \in I}\right)\right] = \Pr_{g(x) \in_R \mathcal{P}^{s',t}} \left[\vec{y} = \left(\{g(\alpha_i)\}_{i \in I}\right)\right] = \frac{1}{|\mathbb{F}|^\ell}$$

where $f(x)$ and $g(x)$ are chosen uniformly and independently from $\mathcal{P}^{s,t}$ and $\mathcal{P}^{s',t}$, respectively.

Proof: We first prove the claim for the special case that $\ell = t$. Fix $s, s' \in \mathbb{F}$, fix non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$, and fix $I \subset [n]$ with $|I| = t$. Moreover, fix $\vec{y} \in \mathbb{F}^t$. Let y_i be the ith element of the vector $\vec{y}$ for every $i \in \{1, \ldots, t\}$. We now show that:

$$\Pr_{f(x) \in_R \mathcal{P}^{s,t}} \left[\vec{y} = \left(\{f(\alpha_i)\}_{i \in I}\right)\right] = \frac{1}{|\mathcal{P}^{s,t}|}.$$

The values of $\vec{y}$ define a unique polynomial from $\mathcal{P}^{s,t}$. This is because there exists a single polynomial of degree t that passes through the points $(0, s)$ and $\{(\alpha_i, y_i)\}_{i \in I}$. Let $f'(x)$ be this unique polynomial. By definition we have that $f'(x) \in \mathcal{P}^{s,t}$ and so:

$$\Pr_{f(x) \in_R \mathcal{P}^{s,t}} \left[\vec{y} = \left(\{f(\alpha_i)\}_{i \in I}\right)\right] = \Pr\left[f(x) = f'(x)\right] = \frac{1}{|\mathcal{P}^{s,t}|}$$

where the latter is true since $f(x)$ is chosen uniformly at random from $\mathcal{P}^{s,t}$, and $f'(x)$ is a fixed polynomial in $\mathcal{P}^{s,t}$.

Using the same reasoning, and letting $g'(x)$ be the unique polynomial that passes through the points $(0, s')$ and $\{(\alpha_i, y_i)\}_{i \in I}$ we have that:

$$\Pr_{g(x) \in_R \mathcal{P}^{s',t}} \left[\vec{y} = \left(\{g(\alpha_i)\}_{i \in I}\right)\right] = \Pr\left[g(x) = g'(x)\right] = \frac{1}{|\mathcal{P}^{s',t}|}.$$

The proof for the case of $\ell = t$ is concluded by observing that for every s and s' in $\mathbb{F}$, it holds that $|\mathcal{P}^{s,t}| = |\mathcal{P}^{s',t}| = |\mathbb{F}|^t$, and so:

$$\Pr_{f(x) \in_R \mathcal{P}^{s,t}} \left[\vec{y} = \left(\{f(\alpha_i)\}_{i \in I}\right)\right] = \Pr_{g(x) \in_R \mathcal{P}^{s',t}} \left[\vec{y} = \left(\{g(\alpha_i)\}_{i \in I}\right)\right] = \frac{1}{|\mathbb{F}|^t}.$$

For the general case where $|I| = \ell$ may be less than t, fix $J \subset [n]$ with $|J| = t$ and $I \subset J$. Observe that for every vector $\vec{y} \in \mathbb{F}^\ell$:

$$\begin{aligned}\Pr_{f(x) \in_R \mathcal{P}^{s,t}} \left[\vec{y} = \left(\{f(\alpha_i)\}_{i \in I}\right)\right] &= \sum_{\vec{y}' \in \mathbb{F}^{t-\ell}} \Pr\left[(\vec{y}, \vec{y}') = \left(\{f(\alpha_i)\}_{i \in I}, \{f(\alpha_j)\}_{j \in J \setminus I}\right)\right] \\ &= |\mathbb{F}|^{t-\ell} \cdot \frac{1}{|\mathbb{F}|^t} = \frac{1}{|\mathbb{F}|^\ell}.\end{aligned}$$

This holds for both s and s' and so the proof is concluded. ∎

As a corollary, we have that any $\ell \le t$ points on a random polynomial are uniformly distributed in the field $\mathbb{F}$. This follows immediately from Claim 2.3.2 because stating that every $\vec{y}$ appears with probability $1/|\mathbb{F}|^\ell$ is equivalent to stating that the shares are uniformly distributed. That is:

Corollary 2.3.3 *For any secret $s \in \mathbb{F}$, any set of distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$, and any subset $I \subset [n]$ where $|I| = \ell \le t$, it holds that $\left\{ \{f(\alpha_i)\}_{i\in I} \right\} \equiv \left\{ U_{\mathbb{F}}^{(1)}, \ldots, U_{\mathbb{F}}^{(\ell)} \right\}$, where $f(x)$ is chosen uniformly at random from $\mathcal{P}^{s,t}$ and $U_{\mathbb{F}}^{(1)}, \ldots, U_{\mathbb{F}}^{(\ell)}$ are ℓ independent random variables that are uniformly distributed over $\mathbb{F}$.*

Multiple polynomials. In the protocol for secure computation, parties hide secrets and distribute them using Shamir's secret sharing scheme. As a result, the adversary receives $m \cdot |I|$ shares, $\{f_1(\alpha_i), \ldots, f_m(\alpha_i)\}_{i\in I}$, for some value m. The secrets $f_1(0), \ldots, f_m(0)$ may not be independent. We therefore need to show that the shares that the adversary receives for all secrets do not reveal any information about any of the secrets. Intuitively, this follows from the fact that Claim 2.3.2 is stated for *any* two secrets s, s', and in particular for two secrets that are known and may be related. The following claim can be proven using standard facts from probability:

Claim 2.3.4 *For any $m \in \mathbb{N}$, any set of non-zero distinct values $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$, any two sets of secrets $(a_1, \ldots, a_m) \in \mathbb{F}^m$ and $(b_1, \ldots, b_m) \in \mathbb{F}^m$, and any subset $I \subset [n]$ of size $|I| \le t$, it holds that:*

$$\left\{ \{(f_1(\alpha_i), \ldots, f_m(\alpha_i))\}_{i\in I} \right\} \equiv \left\{ \{(g_1(\alpha_i), \ldots, g_m(\alpha_i))\}_{i\in I} \right\}$$

where for every j, $f_j(x)$, $g_j(x)$ are chosen uniformly at random from $\mathcal{P}^{a_j,t}$ and $\mathcal{P}^{b_j,t}$, respectively.

Hiding the leading coefficient. In Shamir's secret sharing scheme, the dealer creates shares by constructing a polynomial of degree t, where its constant term is fixed and all the other coefficients are chosen uniformly at random. In Claim 2.3.2 we showed that any t or fewer points on such a polynomial do not reveal any information about the fixed coefficient which is the constant term.

We now consider this claim when we choose the polynomial differently. In particular, we now fix the *leading* coefficient of the polynomial (i.e., the coefficient of the monomial x^t), and choose all the other coefficients uniformly and independently at random, including the constant term. As in the previous section, it holds that any subset of t or fewer points on such a polynomial do not reveal any information about the fixed coefficient, which in this case is the leading coefficient. We will need this claim for proving the security of one of the sub-protocols for the malicious case (in Section 2.6.6).

Let $\mathcal{P}_{s,t}^{\mathsf{lead}}$ be the set of all the polynomials of degree t with *leading* coefficient s. Namely, the polynomials have the structure: $f(x) = a_0 + a_1x + \ldots a_{t-1}x^{t-1} + sx^t$. The following claim is derived similarly to Corollary 2.3.3.

Claim 2.3.5 *For any secret $s \in \mathbb{F}$, any set of distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$, and any subset $I \subset [n]$ where $|I| = \ell \le t$, it holds that:*

$$\left\{ \{f(\alpha_i)\}_{i\in I} \right\} \equiv \left\{ U_{\mathbb{F}}^{(1)}, \ldots, U_{\mathbb{F}}^{(\ell)} \right\}$$

where $f(x)$ is chosen uniformly at random from $\mathcal{P}^{\mathsf{lead}}_{s,t}$ and $U^{(1)}_{\mathbb{F}}, \ldots, U^{(\ell)}_{\mathbb{F}}$ are ℓ independent random variables that are uniformly distributed over $\mathbb{F}$.

2.3.3 Matrix Representation

In this section we present a useful representation for polynomial evaluation. We being by defining the Vandermonde matrix for the values $\alpha_1, \ldots, \alpha_n$. As is well known, the evaluation of a polynomial at $\alpha_1, \ldots, \alpha_n$ can be obtained by multiplying the associated Vandermonde matrix with the vector containing the polynomial coefficients.

Definition 2.3.6 (Vandermonde matrix for $(\alpha_1, \ldots, \alpha_n)$): *Let $\alpha_1, \ldots, \alpha_n$ be n distinct non-zero elements in $\mathbb{F}$. The Vandermonde matrix $V_{\vec{\alpha}}$ for $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$ is the $n \times n$ matrix over $\mathbb{F}$ defined by $V_{\vec{\alpha}}[i,j] \stackrel{\text{def}}{=} (\alpha_i)^{j-1}$. That is,*

$$V_{\vec{\alpha}} \stackrel{\text{def}}{=} \begin{pmatrix} 1 & \alpha_1 & \ldots & (\alpha_1)^{n-1} \\ 1 & \alpha_2 & \ldots & (\alpha_2)^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & \alpha_n & \ldots & (\alpha_n)^{n-1} \end{pmatrix} \tag{2.3.2}$$

The following fact from linear algebra will be of importance to us:

Fact 2.3.7 *Let $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$, where all α_i are distinct and non-zero. Then, $V_{\vec{\alpha}}$ is invertible.*

Matrix representation of polynomial evaluations. Let $V_{\vec{\alpha}}$ be the Vandermonde matrix for $\vec{\alpha}$ and let $q = q_0 + q_1 x + \cdots + q_t x^t$ be a polynomial where $t < n$. Define the vector $\vec{q}$ of length n as follows: $\vec{q} \stackrel{\text{def}}{=} (q_0, \ldots q_t, 0, \ldots, 0)$. Then, it holds that:

$$V_{\vec{\alpha}} \cdot \vec{q} = \begin{pmatrix} 1 & \alpha_1 & \ldots & (\alpha_1)^{n-1} \\ 1 & \alpha_2 & \ldots & (\alpha_2)^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & \alpha_n & \ldots & (\alpha_n)^{n-1} \end{pmatrix} \cdot \begin{pmatrix} q_0 \\ \vdots \\ q_t \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} q(\alpha_1) \\ \vdots \\ \vdots \\ q(\alpha_n) \end{pmatrix}$$

which is the evaluation of the polynomial $q(x)$ on the points $\alpha_1, \ldots, \alpha_n$.

2.4 The Protocol for Semi-Honest Adversaries

2.4.1 Overview

We now provide a high-level overview of the protocol for t-privately computing any deterministic functionality in the presence of a semi-honest adversary who controls up to at most $t < n/2$

parties. Let $\mathbb{F}$ be a finite field of size greater than n and let $f : \mathbb{F}^n \to \mathbb{F}^n$ be the functionality that the parties wish to compute. Note that we assume that each party's input and output is a *single field element*. This is only for the sake of clarity of exposition, and the modifications to the protocol for the general case are straightforward. Let C be an *arithmetic circuit* with fan-in of 2 that computes f. We assume that all arithmetic operations in the circuit are carried out over $\mathbb{F}$. In addition, we assume that the arithmetic circuit C consists of three types of gates: *addition* gates, *multiplication* gates, and *multiplication-by-a-constant* gates. Recall that since a circuit is acyclic, it is possible to sort the wires so that for every gate the input wires come before the output wires.

The protocol works by having the parties jointly propagate values through the circuit from the input wires to the output wires, so that at each stage of the computation the parties obtain Shamir shares of the value on the wire that is currently being computed. In more detail, the protocol has three phases:

- **The input sharing stage:** In this stage, each party creates shares of its input using Shamir's secret sharing scheme using threshold $t+1$ (for a given $t < n/2$), and distributes the shares among the parties.

- **The circuit emulation stage:** In this stage, the parties jointly emulate the computation of the circuit C, gate by gate. In each step, the parties compute shares of the output of a given gate, based on the shares of the inputs to that gate that they already have. The actions of the parties in this stage depends on the type of gate being computed:

  1. *Addition gate:* Given shares of the input wires to the gate, the output is computed without any interaction by each party simply adding their local shares together. Let the inputs to the gate be a and b and let the shares of the parties be defined by two degree-t polynomials $f_a(x)$ and $f_b(x)$ (meaning that each party P_i holds $f_a(\alpha_i)$ and $f_b(\alpha_i)$ where $f_a(0) = a$ and $f_b(0) = b$). Then the polynomial $f_{a+b}(x)$ defined by shares $f_{a+b}(\alpha_i) = f_a(\alpha_i) + f_b(\alpha_i)$, for every i, is a degree-t polynomial with constant term $a + b$. Thus, each party simply locally adds its own shares $f_a(\alpha_i)$ and $f_b(\alpha_i)$ together, and the result is that the parties hold legal shares of the sum of the inputs, as required.

  2. *Multiplication-by-a-constant gate:* This type of gate can also be computed without any interaction. Let the input to the gate be a and let $f_a(x)$ be the t-degree polynomial defining the shares, as above. The aim of the parties is to obtain shares of the value $c \cdot a$, where c is the constant of the gate. Then, each party P_i holding $f_a(\alpha_i)$ simply defines its output share to be $f_{c \cdot a}(\alpha_i) = c \cdot f_a(\alpha_i)$. It is clear that $f_{c \cdot a}(x)$ is a degree-t polynomial with constant term $c \cdot a$, as required.

  3. *Multiplication gate:* As in (1) above, let the inputs be a and b, and let $f_a(x)$ and $f_b(x)$ be the polynomials defining the shares. Here, as in the case of an addition gate, the parties can just multiply their shares together and define $h(\alpha_i) = f_a(\alpha_i) \cdot f_b(\alpha_i)$. The constant term of this polynomial is $a \cdot b$, as required. However, $h(x)$ will be of degree $2t$ instead of t; after repeated multiplications the degree will be n or greater and the parties' n shares will not determine the polynomial or enable reconstruction. In addition, $h(x)$ generated in this way is not a "random polynomial" but has a specific structure. For example, $h(x)$ is typically not irreducible (since it can be

expressed as the product of $f_a(x)$ and $f_b(x)$), and this may leak information. Thus, local computation does not suffice for computing a multiplication gate. Instead, the parties compute this gate by running an interactive protocol that t-privately computes the multiplication functionality F_{mult}, defined by

$$F_{mult}\Big((f_a(\alpha_1), f_b(\alpha_1)), \ldots, (f_a(\alpha_n), f_b(\alpha_n))\Big) = \Big(f_{ab}(\alpha_1), \ldots, f_{ab}(\alpha_n)\Big) \qquad (2.4.1)$$

where $f_{ab}(x) \in_R \mathcal{P}^{a \cdot b, t}$ is a random degree-t polynomial with constant term $a \cdot b$.[2]

- **The output reconstruction stage:** At the end of the computation stage, the parties hold shares of the output wires. In order to obtain the actual output, the parties send their shares to one another and reconstruct the values of the output wires. Specifically, if a given output wire defines output for party P_i, then all parties send their shares of that wire value to P_i.

Organization of this section. In Section 2.4.2, we fully describe the above protocol and prove its security in the F_{mult}-hybrid model. (Recall that in this model, the parties have access to a trusted party who computes F_{mult} for them, and in addition exchange real protocol messages.) We also derive a corollary for t-privately computing any linear function in the plain model (i.e., without any use of the F_{mult} functionality), that is used later in Section 2.4.3. Then, in Section 2.4.3, we show how to t-privately compute the F_{mult} functionality for any $t < n/2$. This involves specifying and implementing two functionalities F_{rand}^{2t} and F_{reduce}^{deg}; see the beginning of Section 2.4.3 for an overview of the protocol for t-privately computing F_{mult} and for the definition of these functionalities.

2.4.2 Private Computation in the F_{mult}-Hybrid Model

In this section we present a formal description and proof of the protocol for t-privately computing any deterministic functionality f in the F_{mult}-hybrid model. As we have mentioned, it is assumed that each party has a single input in a known field $\mathbb{F}$ of size greater than n, and that the arithmetic circuit C is over $\mathbb{F}$. See Protocol 2.4.1 for the description.

We now prove the security of Protocol 2.4.1. We remark that in the F_{mult}-hybrid model, the protocol is actually t-private for *any* $t < n$. However, as we will see, in order to t-privately compute the F_{mult} functionality, we will need to set $t < n/2$.

Theorem 2.4.2 *Let $\mathbb{F}$ be a finite field, let $f : \mathbb{F}^n \to \mathbb{F}^n$ be an n-ary functionality, and let $t < n$. Then, Protocol 2.4.1 is t-private for f in the F_{mult}-hybrid model, in the presence of a static semi-honest adversary.*

Proof: Intuitively, the protocol is t-private because the only values that the parties see until the output stage are random shares. Since the threshold of the secret sharing scheme used is $t+1$, it holds that no adversary controlling t parties can learn anything. The fact that the view of the adversary can be simulated is due to the fact that t shares of any two possible secrets are identically distributed; see Claim 2.3.2. This implies that the simulator can generate the shares

[2]This definition of the functionality assumes that all of the inputs lie on the polynomials $f_a(x), f_b(x)$ and ignores the case that this does not hold. However, since we are dealing with the semi-honest case here, the inputs are always guaranteed to be correct. This can be formalized using the notion of a partial functionality [53, Sec. 7.2].

PROTOCOL 2.4.1 (t-Private Computation in the F_{mult}-Hybrid Model)

- **Inputs:** Each party P_i has an input $x_i \in \mathbb{F}$.
- **Auxiliary input:** Each party P_i has an arithmetic circuit C over the field $\mathbb{F}$, such that for every $\vec{x} \in \mathbb{F}^n$ it holds that $C(\vec{x}) = f(\vec{x})$, where $f : \mathbb{F}^n \to \mathbb{F}^n$. The parties also have a description of $\mathbb{F}$ and distinct non-zero values $\alpha_1, \ldots, \alpha_n$ in $\mathbb{F}$.
- **The protocol:**
  1. **The input sharing stage:** Each party P_i chooses a polynomial $q_i(x)$ uniformly from the set $\mathcal{P}^{x_i,t}$ of all polynomials of degree t with constant term x_i. For every $j \in \{1, \ldots, n\}$, P_i sends party P_j the value $q_i(\alpha_j)$.

     Each party P_i records the values $q_1(\alpha_i), \ldots, q_n(\alpha_i)$ that it received.
  2. **The circuit emulation stage:** Let $G_1, \ldots, G_\ell$ be a predetermined topological ordering of the gates of the circuit. For $k = 1, \ldots, \ell$ the parties work as follows:
     - *Case 1 – G_k is an addition gate:* Let β_i^k and γ_i^k be the shares of input wires held by party P_i. Then, P_i defines its share of the output wire to be $\delta_i^k = \beta_i^k + \gamma_i^k$.
     - *Case 2 – G_k is a multiplication-by-a-constant gate with constant c:* Let β_i^k be the share of the input wire held by party P_i. Then, P_i defines its share of the output wire to be $\delta_i^k = c \cdot \beta_i^k$.
     - *Case 3 – G_k is a multiplication gate:* Let β_i^k and γ_i^k be the shares of input wires held by party P_i. Then, P_i sends (β_i^k, γ_i^k) to the ideal functionality F_{mult} of Eq. (2.4.1) and receives back a value δ_i^k. Party P_i defines its share of the output wire to be δ_i^k.
  3. **The output reconstruction stage:** Let $o_1, \ldots, o_n$ be the output wires, where party P_i's output is the value on wire o_i. For every $k = 1, \ldots, n$, denote by $\beta_1^k, \ldots, \beta_n^k$ the shares that the parties hold for wire o_k. Then, each P_i sends P_k the share β_i^k.

     Upon receiving all shares, P_k computes $\mathsf{reconstruct}_{\vec{\alpha}}(\beta_1^k, \ldots, \beta_n^k)$ and obtains a polynomial $g_k(x)$ (note that $t+1$ of the n shares suffice). P_k then defines its output to be $g_k(0)$.

based on any arbitrary value, and the resulting view is identical to that of a real execution. Observe that this is true until the output stage where the simulator must make the random shares that were used match the actual output of the corrupted parties. This is not a problem because, by interpolation, any set of t shares can be used to define a t-degree polynomial with its constant term being the actual output.

Since C computes the functionality f, it is immediate that $\text{OUTPUT}^\pi(x_1, \ldots, x_n)$ equals $f(x_1, \ldots, x_n)$, where π denotes Protocol 2.4.1. We now proceed to show the existence of a simulator $\mathcal{S}$ as required by Definition 2.2.1. Before describing the simulator, we present some necessary notation. Our proof works by inductively showing that the partial view of the adversary at every stage is identical in the simulated and real executions. Recall that the view of party P_i is the vector $(x_i, r_i; m_i^1, \ldots, m_i^\ell)$, where x_i is the party's input, r_i its random tape, m_i^k is the kth message that it receives in the execution, and ℓ is the overall number of messages received (in our context here, we let m_i^k equal the series of messages that P_i receives when the parties compute gate G_k). For the sake of clarity, we add to the view of each party the values $\sigma_i^1, \ldots, \sigma_i^\ell$, where σ_i^k equals the shares on the wires that Party P_i holds *after* the parties emulate the computation of gate G_k. That is, we denote

$$\text{VIEW}_i^\pi(\vec{x}) = \left(x_i, r_i; m_i^1, \sigma_i^1, \ldots, m_i^\ell, \sigma_i^\ell\right).$$

We stress that since the σ_i^k values can be efficiently computed from the party's input, random tape and incoming messages, the view including the σ_i^k values is equivalent to the view without them, and this is only a matter of notation.

We are now ready to describe the simulator $\mathcal{S}$. Loosely speaking, $\mathcal{S}$ works by simply sending random shares of arbitrary values until the output stage. Then, in the final output stage $\mathcal{S}$ sends values so that the reconstruction of the shares on the output wires yield the actual output.

The Simulator $\mathcal{S}$:

- **Input:** *The simulator receives the inputs and outputs, $\{x_i\}_{i\in I}$ and $\{y_i\}_{i\in I}$ respectively, of all corrupted parties.*

- **Simulation:**

  1. Simulating the input sharing stage:

  (a) For every $i \in I$, the simulator $\mathcal{S}$ chooses a uniformly distributed random tape for P_i; this random tape and the input x_i fully determines the degree-t polynomial $q_i'(x) \in \mathcal{P}^{x_i,t}$ chosen by P_i in the protocol.

  (b) For every $j \notin I$, the simulator $\mathcal{S}$ chooses a random degree-t polynomial $q_k'(x) \in_R \mathcal{P}^{0,t}$ with constant term 0.

  (c) The view of the corrupted party P_i in this stage is then constructed by $\mathcal{S}$ to be the set of values $\{q_j(\alpha_i)\}_{j\notin I}$ (i.e., the share sent by each honest P_j to P_i). The view of the adversary $\mathcal{A}$ consists of the view of P_i for every $i \in I$.

  2. Simulating the circuit emulation stage: *For every $G_k \in \{G_1, \ldots, G_\ell\}$:*

  (a) G_k is an addition gate: *Let $\{f_a(\alpha_i)\}_{i\in I}$ and $\{f_b(\alpha_i)\}_{i\in I}$ be the shares of the input wires of the corrupted parties that were generated by $\mathcal{S}$ (initially these are input wires and so the shares are defined by $q_k'(x)$ above). For every $i \in I$, the simulator $\mathcal{S}$ computes $f_a(\alpha_i) + f_b(\alpha_i) = (f_a + f_b)(\alpha_i)$ which defines the shares of the output wire of G_k.*

  (b) G_k is a multiplication-with-constant gate: *Let $\{f_a(\alpha_i)\}_{i\in I}$ be the shares of the input wire and let $c \in \mathbb{F}$ be the constant of the gate. $\mathcal{S}$ computes $c \cdot f_a(\alpha_i) = (c \cdot f_a)(\alpha_i)$ for every $i \in I$ which defines the shares of the output wire of G_k.*

  (c) G_k is a multiplication gate: *$\mathcal{S}$ chooses a degree-t polynomial $f_{ab}(x)$ uniformly at random from $\mathcal{P}^{0,t}$ (irrespective of the shares of the input wires), and defines the shares of the corrupted parties of the output wire of G_k to be $\{f_{ab}(\alpha_i)\}_{i\in I}$.*

  $\mathcal{S}$ adds the shares to the corrupted parties' views.

  3. Simulating the output reconstruction stage: *Let $o_1, \ldots, o_n$ be the output wires. We now focus on the output wires of the corrupted parties. For every $k \in I$, the simulator $\mathcal{S}$ has already defined $|I|$ shares $\{\beta_k^i\}_{i\in I}$ for the output wire o_k. $\mathcal{S}$ thus chooses a* random polynomial *$g_k'(x)$ of degree t under the following constraints:*

(a) $g'_k(0) = y_k$, where y_k is the corrupted P_k's output (the polynomial's constant term is the correct output).

(b) For every $i \in I$, $g'_k(\alpha_i) = \beta^i_k$ (i.e., the polynomial is consistent with the shares that have already been defined).

(Note that if $|I| = t$, then the above constraints yield $t + 1$ equations, which in turn fully determine the polynomial $g'_k(x)$. However, if $|I| < t$, then $\mathcal{S}$ can carry out the above by choosing $t - |I|$ additional random points and interpolating.)

Finally, $\mathcal{S}$ adds the shares $\{g'_k(\alpha_1), \ldots, g'_k(\alpha_n)\}$ to the view of the corrupted party P_k.

4. $\mathcal{S}$ outputs the views of the corrupted parties and halts.

Denote by $\widetilde{\text{VIEW}}^\pi_I(\vec{x})$ the VIEW of the corrupted parties up to the output reconstruction stage (and not including that stage). Likewise, we denote by $\tilde{\mathcal{S}}\left(I, \vec{x}_I, f_I\left(\vec{x}\right)\right)$ the view generated by the simulator up to but not including the output reconstruction stage.

We begin by showing that the partial views of the corrupted parties up to the output reconstruction stage in the real execution and simulation are identically distributed.

Claim 2.4.3 *For every $\vec{x} \in \mathbb{F}^n$ and every $I \subset [n]$ with $|I| \leq t$,*

$$\left\{\widetilde{\text{VIEW}}^\pi_I(\vec{x})\right\} \equiv \left\{\tilde{\mathcal{S}}\left(I, \vec{x}_I, f_I\left(\vec{x}\right)\right)\right\}$$

Proof: The only difference between the partial views of the corrupted parties in a real and simulated execution is that the simulator generates the shares in the input-sharing stage and in multiplication gates from random polynomials with constant term 0, instead of with the correct value defined by the actual inputs and circuit. Intuitively, the distributions generated are the same since the shares are distributed identically, for every possible secret.

Formally, we construct an algorithm H that receives as input $n-|I|+\ell$ sets of shares: $n-|I|$ sets of shares $\{(i, \beta^1_i)\}_{i\in I}, \ldots, \{(i, \beta^{n-|I|}_i)\}_{i\in I}$ and ℓ sets of shares $\{(i, \gamma^1_i)\}_{i\in I}, \ldots, \{(i, \gamma^\ell_i)\}_{i\in I}$. Algorithm H generates the partial view of the corrupted parties (up until but not including the output reconstruction stage) as follows:

- H uses the jth set of shares $\{\beta^j_i\}_{i\in I}$ as the shares sent by the jth honest party to the corrupted parties in the input sharing stage (here $j = 1, \ldots, n - |I|$),
- H uses the kth set of shares $\{\gamma^k_i\}_{i\in I}$ are viewed as the shares received by the corrupted parties from F_{mult} in the computation of the k gate G_k, if it is a multiplication gate (here $k = 1, \ldots, \ell$).

Otherwise, H works exactly as the simulator $\mathcal{S}$.

It is immediate that if H receives shares that are generated from random polynomials that all have constant term 0, then the generated view is *exactly* the same as the partial view generated by $\mathcal{S}$. In contrast, if H receives shares that are generated from random polynomials that have constant terms as determined by the inputs and circuit (i.e., the shares β^j_i are generated using the input of the jth honest party, and the shares γ^k_i are generated using the value on the output wire of G_k which is fully determined by the inputs and circuit), then the generated view is *exactly* the same as the partial view in a real execution. This is due to the fact that all shares are generated using the correct values, like in a real execution. By Claim 2.3.2, these two sets of shares are identically distributed and so the two types of views generated by H are identically distribution; that is, the partial views from the simulated and real executions are identically distributed. ■

It remains to show that the output of the simulation after the output reconstruction stage is identical to the view of the corrupted parties in a real execution. For simplicity, we assume that the output wires appear immediately after multiplication gates (otherwise, they are fixed functions of these values).

Before proving this, we prove a claim that describes the processes of the real execution and simulation in a more abstract way. The aim of the claim is to prove that the process carried out by the simulator in the output reconstruction stage yields the same distribution as in a protocol execution. We first describe two processes and prove that they yield the same distribution, and later show how these are related to the real and simulation processes.

Random Variable $X(s)$	**Random Variable $Y(s)$**
(1) Choose $q(x) \in_R \mathcal{P}^{s,t}$	(1) Choose $q'(x) \in_R \mathcal{P}^{0,t}$
(2) $\forall i \in I$, set $\beta_i = q(\alpha_i)$	(2) $\forall i \in I$, set $\beta'_i = q'(\alpha_i)$
(3) –	(3) Choose $r(x) \in_R \mathcal{P}^{s,t}$ s.t. $\forall i \in I$ $r(\alpha_i) = \beta'_i$
(4) Output $q(x)$	(4) Output $r(x)$

Observe that in $Y(s)$, first the polynomial $q'(x)$ is chosen with constant term 0, and then $r(x)$ is chosen with constant term s, subject to it agreeing with q' on $\{\alpha_i\}_{i\in I}$.

Claim 2.4.4 *For every $s \in \mathbb{F}$, it holds that $\{X(s)\} \equiv \{Y(s)\}$.*

Proof: Intuitively, this follows from the fact that the points $\{q(\alpha_i)\}_{i\in i}$ are distributed identically to $\{q'(\alpha_i)\}_{i\in I}$. Formally, define $X(s) = (X_1, X_2)$ and $Y(s) = (Y_1, Y_2)$, where X_1 (resp., Y_1) are the output values from step (2) of the process, and X_2 (resp., Y_2) are the output polynomials from step (4) of the process. From Claim 2.3.2, it immediately follows that $\{X_1\} \equiv \{Y_1\}$. It therefore suffices to show that $\{X_2 \mid X_1\} \equiv \{Y_2 \mid Y_1\}$. Stated equivalently, we wish to show that for every set of field values $\{\beta_i\}_{i\in I}$ and every $h(x) \in \mathcal{P}^{s,t}$,

$$\Pr\Big[X_2(x) = h(x) \mid (\forall i) X_2(\alpha_i) = \beta_i\Big] = \Pr\Big[Y_2(x) = h(x) \mid (\forall i) Y_2(\alpha_i) = \beta_i\Big]$$

where $\{\beta_i\}_{i\in I}$ are the conditioned values in X_1 and Y_1 (we use the same β_i for both since these are identically distributed and we are now conditioning them). First, if there exists an $i \in I$ such that $h(\alpha_i) \neq \beta_i$ then both probabilities above are 0. We now compute these probabilities for the case that $h(\alpha_i) = \beta_i$ for all $i \in I$. We first claim that

$$\Pr\Big[Y_2(x) = h(x) \mid (\forall i) Y_2(\alpha_i) = \beta_i\Big] = \frac{1}{|\mathbb{F}|^{t-|I|}}.$$

This follows immediately from the process for generating the random variable $Y(s)$, because $Y_2(x) = r(x)$ is chosen at random under the constraint that for every $i \in I$, $r(\alpha_i) = \beta_i$. Since $|I|+1$ points are already fixed (the β_i values and the constant term s), there are $|\mathbb{F}|^{t-|I|}$ different polynomials of degree-t that meet these constraints, and Y_2 is chosen uniformly from them.

It remains to show that

$$\Pr\Big[X_2(x) = h(x) \mid (\forall i) X_2(\alpha_i) = \beta_i\Big] = \frac{1}{|\mathbb{F}|^{t-|I|}}.$$

In order to see this, observe that

$$\Pr\left[X_2(x) = h(x) \ \wedge \ (\forall i)X_2(\alpha_i) = \beta_i\right] = \Pr\left[X_2(x) = h(x)\right]$$

because in this case, we consider only polynomials $h(x)$ for which $h(\alpha_i) = \beta_i$ for all $i \in I$, and so the conditioning adds nothing. We conclude that

$$\begin{aligned}\Pr\left[X_2(x) = h(x) \mid (\forall i)X_2(\alpha_i) = \beta_i\right] &= \frac{\Pr\left[X_2(x) = h(x) \ \wedge \ (\forall i)X_2(\alpha_i) = \beta_i\right]}{\Pr\left[(\forall i)X_2(\alpha_i) = \beta_i\right]} \\ &= \frac{\Pr\left[X_2(x) = h(x)\right]}{\Pr\left[(\forall i)X_2(\alpha_i) = \beta_i\right]} = \frac{1}{|\mathbb{F}|^t} \cdot |\mathbb{F}|^{|I|} = \frac{1}{|\mathbb{F}|^{t-|I|}},\end{aligned}$$

where the last equality follows because $q(x) = X_2(x) \in_R \mathcal{P}^{s,t}$ and by Claim 2.3.2 the probability that the points $X_2(\alpha_i) = \beta_i$ for all $i \in I$ equals $|\mathbb{F}|^{-|I|}$. ■

The random variables $X(s)$ and $Y(s)$ can be extended to $X(\vec{s})$ and $Y(\vec{s})$ for any $\vec{s} \in \mathbb{F}^m$ (for some $m \in \mathbb{N}$); the proof of the analogous claim then follows. From this claim, we get:

Claim 2.4.5 *If* $\left\{\widetilde{\text{VIEW}}_I^\pi(\vec{x})\right\} \equiv \left\{\tilde{\mathcal{S}}\left(I, \vec{x}_I, f_I\left(\vec{x}\right)\right)\right\}$, *then* $\left\{\text{VIEW}_I^\pi(\vec{x})\right\} \equiv \left\{\mathcal{S}\left(I, \vec{x}_I, f_I\left(\vec{x}\right)\right)\right\}$.

Proof: In the output reconstruction stage, for every $k \in I$, the corrupted parties receive the points $g_k(\alpha_1), \ldots, g_k(\alpha_n)$ in the real execution, and the points $g'_k(\alpha_1), \ldots, g'_k(\alpha_n)$ in the simulation. Equivalently, we can say that the corrupted parties receive the polynomials $\{g_k(x)\}_{k\in I}$ in a real execution, and the polynomials $\{g'_k(x)\}_{k\in I}$ in the simulation.

In the protocol execution, functionality F_{mult} chooses the polynomial $f_{ab}^{(k)}(x)$ for the output wire of P_k uniformly at random in $\mathcal{P}^{y_k,t}$, and the corrupted parties receive values $\beta_i = f_{ab}^{(k)}(\alpha_i)$ (for every $i \in I$). Finally, as we have just described, in the output stage, the corrupted parties receive the polynomials $f_{ab}^{(k)}(x)$ themselves. Thus, this is the process $X(y_k)$. Extending to all $k \in I$, we have that this is the extended process $X(\vec{s})$ with $\vec{s}$ being the vector containing the corrupted parties' output values $\{y_k\}_{k\in I}$.

In contrast, in the simulation of the multiplication gate leading to the output wire for party P_k, the simulator $\mathcal{S}$ chooses the polynomial $f_{ab}^{(k)}(x)$ uniformly at random in $\mathcal{P}^{0,t}$ (see Step 2c in the specification of $\mathcal{S}$ above), and the corrupted parties receive values $\beta_i = f_{ab}^{(k)}(\alpha_i)$ (for every $i \in I$). Then, in the output stage, $\mathcal{S}$ chose $g'_k(x)$ at random from $\mathcal{P}^{y_k,t}$ under the constraint that $g'_k(\alpha_i) = \beta_i$ for every $i \in I$. Thus, this is the process $Y(y_k)$. Extending to all $k \in I$, we have that this is the extended process $Y(\vec{s})$ with $\vec{s}$ being the vector containing the corrupted parties' output values $\{y_k\}_{k\in I}$. The claim thus follows from Claim 2.4.4. ■

Combining Claims 2.4.3 and 2.4.5 we have that $\{\mathcal{S}\left(I, \vec{x}_I, f_I(\vec{x})\right)\} \equiv \{\text{VIEW}_I^\pi\left(\vec{x}\right)\}$, as required. ■

Privately computing linear functionalities in the real model. Theorem 2.4.2 states that every function can be t-privately computed in the F_{mult}-hybrid model, for *any* $t < n$. However, a look at Protocol 2.4.1 and its proof of security show that F_{mult} is only used for computing multiplication gates in the circuit. Thus, Protocol 2.4.1 can actually be directly used for privately computing any linear functionality f, since such functionalities can be computed by circuits containing only addition and multiplication-by-constant gates. Furthermore, the protocol is secure for any $t < n$; in particular, no honest majority is needed. This yields the following corollary.

Corollary 2.4.6 *Let $t < n$. Then, any linear functionality f can be t-privately computed in the presence of a static semi-honest adversary. In particular, the matrix-multiplication functionality $F^A_{mat}(\vec{x}) = A \cdot \vec{x}$ for matrix $A \in \mathbb{F}^{n \times n}$ can be t-privately computed in the presence of a static semi-honest adversary.*

Corollary 2.4.6 is used below in order to compute the degree-reduction functionality, which is used in order to privately compute F_{mult}.

2.4.3 Privately Computing the F_{mult} Functionality

We have shown how to t-privately compute any functionality in the F_{mult}-hybrid model. In order to achieve private computation in the plain model, it remains to show how to privately compute the F_{mult} functionality. We remark that the threshold needed to privately compute F_{mult} is $t < n/2$, and thus the overall threshold for the generic BGW protocol is $t < n/2$. Recall that the F_{mult} functionality is defined as follows:

$$F_{mult}\Big((f_a(\alpha_1), f_b(\alpha_1)), \ldots, (f_a(\alpha_n), f_b(\alpha_n))\Big) = \Big(f_{ab}(\alpha_1), \ldots, f_{ab}(\alpha_n)\Big)$$

where $f_a(x) \in \mathcal{P}^{a,t}$, $f_b(x) \in \mathcal{P}^{b,t}$, and $f_{ab}(x)$ is a *random* polynomial in $\mathcal{P}^{a \cdot b,t}$.

As we have discussed previously, the simple solution where each party locally multiplies its two shares does not work here, for two reasons. First, the resulting polynomial is of degree $2t$ and not t as required. Second, the resulting polynomial of degree $2t$ is not uniformly distributed amongst all polynomials with the required constant term. Therefore, in order to privately compute the F_{mult} functionality, we first *randomize* the degree-$2t$ polynomial so that it is uniformly distributed, and then reduce its degree to t. That is, F_{mult} is computed according to the following steps:

1. Each party locally multiplies its input shares.
2. The parties run a protocol to generate a random polynomial in $\mathcal{P}^{0,2t}$, and each party receives a share based on this polynomial. Then, each party adds its share of the product (from the previous step) with its share of this polynomial. The resulting shares thus define a polynomial which is uniformly distributed in $\mathcal{P}^{a \cdot b,2t}$.
3. The parties run a protocol to reduce the degree of the polynomial to t, with the result being a polynomial that is uniformly distributed in $\mathcal{P}^{a \cdot b,t}$, as required. This computation uses a t-private protocol for computing *matrix multiplication.* We have already shown how to achieve this in Corollary 2.4.6.

The randomizing (i.e., selecting a random polynomial in $\mathcal{P}^{0,2t}$) and degree-reduction functionalities for carrying out the foregoing steps are formally defined as follows:

- *The randomization functionality:* The randomization functionality is defined as follows:

$$F^{2t}_{rand}(\lambda, \ldots, \lambda) = (r(\alpha_1), \ldots, r(\alpha_n)),$$

where $r(x) \in_R \mathcal{P}^{0,2t}$ is random, and λ denotes the empty string. We will show how to t-privately compute this functionality in Section 2.4.3.

- *The degree-reduction functionality:* Let $h(x) = h_0 + \ldots + h_{2t}x^{2t}$ be a polynomial, and denote by $\mathsf{trunc}_t(h(x))$ the polynomial of degree t with coefficients $h_0, \ldots, h_t$. That is, $\mathsf{trunc}_t(h(x)) = h_0 + h_1 x + \ldots + h_t x^t$ (observe that this is a deterministic functionality). Formally, we define

$$F_{reduce}^{deg}(h(\alpha_1), \ldots, h(\alpha_n)) = (\hat{h}(\alpha_1), \ldots, \hat{h}(\alpha_n))$$

where $\hat{h}(x) = \mathsf{trunc}_t(h(x))$. We will show how to t-privately compute this functionality in Section 2.4.3.

Privately Computing F_{mult} in the $(F_{rand}^{2t}, F_{reduce}^{deg})$-Hybrid Model

We now prove that F_{mult} is reducible to the functionalities F_{rand}^{2t} and F_{reduce}^{deg}; that is, we construct a protocol that t-privately computes F_{mult} given access to ideal functionalities F_{reduce}^{deg} and F_{rand}^{2t}. The full specification appears in Protocol 2.4.7.

Intuitively, this protocol is secure since the randomization step ensures that the polynomial defining the output shares is random. In addition, the parties only see shares of the randomized polynomial and its truncation. Since the randomized polynomial is of degree $2t$, seeing $2t$ shares of this polynomial still preserves privacy. Thus, the t shares of the randomized polynomial together with the t shares of the truncated polynomial (which is of degree t), still gives the adversary no information whatsoever about the secret. (This last point is the crux of the proof.)

PROTOCOL 2.4.7 (t-Privately Computing F_{mult})

- **Input:** Each party P_i holds values β_i, γ_i, such that $\mathsf{reconstruct}_{\vec{\alpha}}(\beta_1, \ldots, \beta_n) \in \mathcal{P}^{a,t}$ and $\mathsf{reconstruct}_{\vec{\alpha}}(\gamma_1, \ldots, \gamma_n) \in \mathcal{P}^{b,t}$ for some $a, b \in \mathbb{F}$.
- **The protocol:**
  1. Each party locally computes $s_i = \beta_i \cdot \gamma_i$.
  2. **Randomize:** Each party P_i sends λ to F_{rand}^{2t} (formally, it writes λ on its oracle tape for F_{rand}^{2t}). Let σ_i be the oracle response for party P_i.
  3. **Reduce the degree:** Each party P_i sends $(s_i + \sigma_i)$ to F_{reduce}^{deg}. Let δ_i be the oracle response for P_i.
- **Output:** Each party P_i outputs δ_i.

We therefore have:

Proposition 2.4.8 *Let $t < n/2$. Then, Protocol 2.4.7 is t-private for F_{mult} in the $(F_{rand}^{2t}, F_{reduce}^{deg})$-hybrid model, in the presence of a static semi-honest adversary.*

Proof: The parties do not receive messages from other parties in the oracle-aided protocol; rather they receive messages from the oracles only. Therefore, our simulator only needs to simulate the oracle-response messages. Since the F_{mult} functionality is probabilistic, we must prove its security using Definition 2.2.2.

In the real execution of the protocol, the corrupted parties' inputs are $\{f_a(\alpha_i)\}_{i \in I}$ and $\{f_b(\alpha_i)\}_{i \in I}$. Then, in the randomize step of the protocol they receive shares σ_i of a random polynomial of degree $2t$ with constant term 0. Denoting this polynomial by $r(x)$, we have that the corrupted parties receive the values $\{r(\alpha_i)\}_{i\in I}$. Next, the parties invoke the functionality F^{deg}_{reduce} and receive back the values δ_i (these are points of the polynomial $\mathsf{trunc}_t(f_a(x)\cdot f_b(x)+r(x))$). These values are actually the parties' outputs, and thus the simulator must make the output of the call to F^{deg}_{reduce} be the shares $\{\delta_i\}_{i\in I}$ of the corrupted parties outputs.

The simulator $\mathcal{S}$:

- **Input:** *The simulator receives as input I, the inputs of the corrupted parties $\{(\beta_i,\gamma_i)\}_{i\in I}$, and their outputs $\{\delta_i\}_{i\in I}$.*
- **Simulation:**
  - *$\mathcal{S}$ chooses $|I|$ values uniformly and independently at random, $\{v_i\}_{i\in I}$.*
  - *For every $i\in I$, the simulator defines the view of the party P_i to be: $(\beta_i,\gamma_i,v_i,\delta_i)$, where (β_i,γ_i) represents P_i's input, v_i represents P_i's oracle response from F^{2t}_{rand}, and δ_i represents P_i's oracle response from F^{deg}_{reduce}.*

We now proceed to prove that the joint distribution of the output of all the parties, together with the view of the corrupted parties is distributed identically to the output of all parties as computed from the functionality F_{mult} and the output of the simulator. We first show that the outputs of all parties are distributed identically in both cases. Then, we show that the view of the corrupted parties is distributed identically, conditioned on the values of the outputs (and inputs) of all parties.

The outputs. Since the inputs and outputs of all the parties lie on the same polynomials, it is enough to show that the polynomials are distributed identically. Let $f_a(x), f_b(x)$ be the input polynomials. Let $r(x)$ be the output of the F^{2t}_{rand} functionality. Finally, denote the truncated result by $\hat{h}(x) \stackrel{\text{def}}{=} \mathsf{trunc}(f_a(x)\cdot f_b(x)+r(x))$.

In the real execution of the protocol, the parties output shares of the polynomial $\hat{h}(x)$. From the way $\hat{h}(x)$ is defined, it is immediate that $\hat{h}(x)$ is a degree-t polynomial that is uniformly distributed in $\mathcal{P}^{a\cdot b,t}$. (In order to see that it is uniformly distributed, observe that with the exception of the constant term, all the coefficients of the degree-$2t$ polynomial $f_a(x)\cdot f_b(x)+r(x)$ are random. Thus the coefficients of $x,\ldots,x^t$ in $\hat{h}(x)$ are random, as required.)

Furthermore, the functionality F_{mult} return shares for a random polynomial of degree t with constant term $f_a(0)\cdot f_b(0)=a\cdot b$. Thus, the outputs of the parties from a real execution and from the functionality are distributed identically.

The view of the corrupted parties. We show that the view of the corrupted parties in the real execution and the simulation are distributed identically, given the inputs and outputs of all parties. Observe that the inputs and outputs define the polynomials $f_a(x)$, $f_b(x)$ and $f_{ab}(x)$. Now, the view that is output by the simulator is

$$\Big\{\{f_a(\alpha_i), f_b(\alpha_i), v_i, f_{ab}(\alpha_i)\}_{i\in I}\Big\}$$

where all the v_i values are uniformly distributed in $\mathbb{F}$, and independent of $f_a(x), f_b(x)$ and $f_{ab}(x)$. It remains to show that in a protocol execution the analogous values – which are the outputs received by the corrupted parties from F_{rand}^{2t} – are also uniformly distributed and independent of $f_a(x), f_b(x)$ and $\hat{h}(x)$ (where $\hat{h}(x)$ is distributed identically to a random $f_{ab}(x)$, as already shown above).

In order to prove this, it suffices to prove that for every vector $\vec{y} \in \mathbb{F}^{|I|}$,

$$\Pr\left[\vec{r} = \vec{y} \mid f_a(x), f_b(x), \hat{h}(x)\right] = \frac{1}{|\mathbb{F}|^{|I|}} \tag{2.4.2}$$

where $\vec{r} = (r(\alpha_{i_1}), \ldots, r(\alpha_{i_{|I|}}))$ for $I = \{i_1, \ldots, i_{|I|}\}$; that is, $\vec{r}$ is the vector of outputs from F_{rand}^{2t}, computed from the polynomial $r(x) \in_R \mathcal{P}^{0,2t}$, that are received by the corrupted parties.

We write $r(x) = r_1(x) + x^t \cdot r_2(x)$, where $r_1(x) \in_R \mathcal{P}^{0,t}$ and $r_2(x) \in_R \mathcal{P}^{0,t}$. In addition, we write $f_a(x) \cdot f_b(x) = h_1(x) + x^t \cdot h_2(x)$, where $h_1(x) \in \mathcal{P}^{ab,t}$ and $h_2(x) \in \mathcal{P}^{0,t}$. Observe that:

$$\hat{h}(x) = \mathsf{trunc}\left(f_a(x) \cdot f_b(x) + r(x)\right) = \mathsf{trunc}\left(h_1(x) + r_1(x) + x^t \cdot (h_2(x) + r_2(x))\right) = h_1(x) + r_1(x)$$

where the last equality holds since the constant term of both $h_2(x)$ and $r_2(x)$ is 0. Rewriting Eq. (2.4.2), we need to prove that for every vector $\vec{y} \in \mathbb{F}^{|I|}$,

$$\Pr\left[\vec{r} = \vec{y} \mid f_a(x), f_b(x), h_1(x) + r_1(x)\right] = \frac{1}{|\mathbb{F}|^{|I|}}$$

where the kth element r_k of $\vec{r}$ is $r_1(\alpha_{i_k}) + (\alpha_{i_k})^t \cdot r_2(\alpha_{i_k})$. The claim follows since $r_2(x)$ is random and independent of $f_a(x), f_b(x), h_1(x)$ and $r_1(x)$. Formally, for any given $y_k \in \mathbb{F}$, the equality $y_k = r_1(\alpha_{i_k}) + (\alpha_{i_k})^t \cdot r_2(\alpha_{i_k})$ holds if and only if $r_2(\alpha_{i_k}) = (\alpha_{i_k})^{-t} \cdot (y_k - r_1(\alpha_{i_k}))$. Since α_{i_k}, y_k and $r_1(\alpha_{i_k})$ are all fixed by the conditioning, the probability follows from Claim 2.3.2.

We conclude that the view of the corrupted parties is identically distributed to the output of the simulator, when conditioning on the inputs and outputs of all parties. ■

Privately Computing F_{rand}^{2t} in the Plain Model

Recall that the randomization functionality is defined as follows:

$$F_{rand}^{2t}(\lambda, \ldots, \lambda) = (r(\alpha_1), \ldots, r(\alpha_n)), \tag{2.4.3}$$

where $r(x) \in_R \mathcal{P}^{0,2t}$, and λ denotes the empty string. The protocol for implementing the functionality works as follows. Each party P_i chooses a random polynomial $q_i(x) \in_R \mathcal{P}^{0,2t}$ and sends the share $q_i(\alpha_j)$ to every party P_j. Then, each party P_i outputs $\delta_i = \sum_{k=1}^{n} q_k(\alpha_i)$. Clearly, the shares $\delta_1, \ldots, \delta_n$ define a polynomial with constant term 0, because all the polynomials in the sum have a zero constant term. Furthermore, the sum of these random $2t$-degree polynomials is a random polynomial in $\mathcal{P}^{0,2t}$, as required. See Protocol 2.4.9 for a formal description.

PROTOCOL 2.4.9 (Privately Computing F^{2t}_{rand})

- **Input:** The parties do not have inputs for this protocol.
- **The protocol:**
  - Each party P_i chooses a random polynomial $q_i(x) \in_R \mathcal{P}^{0,2t}$. Then, for every $j \in \{1, \ldots, n\}$ it sends $s_{i,j} = q_i(\alpha_j)$ to party P_j.
  - Each party P_i receives $s_{1,i}, \ldots, s_{n,i}$ and computes $\delta_i = \sum_{j=1}^{n} s_{j,i}$.
- **Output:** Each party P_i outputs δ_i.

We now prove that Protocol 2.4.9 is t-private for F^{2t}_{rand}.

Claim 2.4.10 *Let $t < n/2$. Then, Protocol 2.4.9 is t-private for the F^{2t}_{rand} functionality, in the presence of a static semi-honest adversary.*

Proof: Intuitively, the protocol is secure because the only messages that the parties receive are random shares of polynomials in $\mathcal{P}^{0,2t}$. The simulator can easily simulate these messages by generating the shares itself. However, in order to make sure that the view of the corrupted parties is consistent with the actual output provided by the functionality, the simulator chooses the shares so that their sum equals δ_i, the output provided by the functionality to each P_i.

The simulator $\mathcal{S}$:

- **Input:** *The simulator receives as input I and the outputs of the corrupted parties $\{\delta_i\}_{i\in I}$.*
- **Simulation:**
  1. *Fix $\ell \notin I$*
  2. *$\mathcal{S}$ chooses $n-1$ random polynomials $q'_j(x) \in \mathcal{P}^{0,2t}$ for every $j \in [n] \setminus \{\ell\}$. Note that for $i \in I$, this involves setting the random tape of P_i so that it results in it choosing $q'_i(x)$.*
  3. *$\mathcal{S}$ sets the values of the remaining polynomial $q'_\ell(x)$ on the points $\{\alpha_i\}_{i\in I}$ by computing $q'_\ell(\alpha_i) = \delta_i - \sum_{j\neq\ell} q'_j(\alpha_i)$ for every $i \in I$.*
  4. *$\mathcal{S}$ sets $(q'_1(\alpha_i), \ldots, q'_n(\alpha_i))$ as the incoming messages of corrupted party P_i in the protocol; observe that all of these points are defined.*
- **Output:** *$\mathcal{S}$ sets the view of each corrupted P_i ($i \in I$) to be the empty input λ, the random tape determined in Step (2) of the simulation, and the incoming messages determined in Step (4).*

We now show that the view of the adversary (containing the views of all corrupted parties) and the output of all parties in a real execution is distributed identically to the output of the simulator and the output of all parties as received from the functionality in an ideal execution.

In order to do this, consider an fictitious simulator $\mathcal{S}'$ who receives the polynomial $r(x)$ instead of the points $\{\delta_i = r(\alpha_i)\}_{i\in I}$. Simulator $\mathcal{S}'$ works in exactly the same way as $\mathcal{S}$ except that it fully defines the remaining polynomial $q'_\ell(x)$ (and not just its values on the points $\{\alpha_i\}_{i\in I}$) by setting $q'_\ell(x) = r(x) - \sum_{j\neq\ell} q'_j(x)$. Then, $\mathcal{S}'$ computes the values $q'_\ell(\alpha_i)$ for every $i \in I$ from $q'_\ell(x)$. The only difference between the simulator $\mathcal{S}$ and the fictitious simulator $\mathcal{S}'$ is with respect to the value of the polynomial $q'_\ell(x)$ on points outside of $\{\alpha_i\}_{i\in I}$. The crucial point to notice is that $\mathcal{S}$ does *not* define these points differently to $\mathcal{S}'$; rather $\mathcal{S}$ does not define them at all. That is, the simulation does not require $\mathcal{S}$ to determine the value of $q'_\ell(x)$ on points outside of $\{\alpha_i\}_{i\in I}$, and so the distributions are identical.

Finally observe that the output distribution generated by $\mathcal{S}'$ is identical to the output of a real protocol. This holds because in a real protocol execution random polynomials $q_1(x),\ldots,q_n(x)$ are chosen and the output points are derived from $\sum_{j=1}^{n} q_j(x)$, whereas in the fictitious simulation with $\mathcal{S}'$ the order is just reversed; i.e., first $r(x)$ is chosen at random and then $q'_1(x),\ldots,q'_n(x)$ are chosen at random under the constraint that their sum equals $r(x)$. Note that this uses the fact that $r(x)$ is randomly chosen. ■

Privately Computing F^{deg}_{reduce} in the Plain Model

Recall that the F^{deg}_{reduce} functionality is defined by

$$F^{deg}_{reduce}(h(\alpha_1),\ldots,h(\alpha_n)) = (\hat{h}(\alpha_1),\ldots,\hat{h}(\alpha_n))$$

where $\hat{h}(x) = \mathsf{trunc}_t(h(x))$ is the polynomial $h(x)$ truncated to degree t (i.e., the polynomial with coefficients $h_0,\ldots,h_t$). We begin by showing that in order to transform a vector of shares of the polynomial $h(x)$ to shares of the polynomial $\mathsf{trunc}_t(h(x))$, it suffices to multiply the input shares by a certain matrix of constants.

Claim 2.4.11 *Let $t < n/2$. Then, there exists a constant matrix $A \in \mathbb{F}^{n\times n}$ such that for every degree-$2t$ polynomial $h(x) = \sum_{j=0}^{2t} h_j \cdot x^j$ and truncated $\hat{h}(x) = \mathsf{trunc}_t(h(x))$, it holds that:*

$$\left(\hat{h}(\alpha_1),\ldots,\hat{h}(\alpha_n)\right)^T = A \cdot \left(h(\alpha_1),\ldots,h(\alpha_n)\right)^T .$$

Proof: Let $\vec{h} = (h_0,\ldots,h_t,\ldots,h_{2t},0,\ldots 0)$ be a vector of length n, and let $V_{\vec{\alpha}}$ be the $n\times n$ Vandermonde matrix for $\vec{\alpha} = (\alpha_1,\ldots,\alpha_n)$. As we have seen in Section 2.3.3, $V_{\vec{\alpha}} \cdot \vec{h}^T = (h(\alpha_1),\ldots,h(\alpha_n))^T$. Since $V_{\vec{\alpha}}$ is invertible, we have that $\vec{h}^T = V_{\vec{\alpha}}^{-1} \cdot (h(\alpha_1),\ldots,h(\alpha_n))^T$. Similarly, letting $\vec{\hat{h}} = (\hat{h}_0,\ldots,\hat{h}_t,0,\ldots 0)$ we have that $\left(\hat{h}(\alpha_1),\ldots,\hat{h}(\alpha_n)\right)^T = V_{\vec{\alpha}} \cdot \vec{\hat{h}}^T$.

Now, let $T = \{1,\ldots,t\}$, and let P_T be the linear projection of T; i.e., P_T is an $n\times n$ matrix such that $P_T(i,j) = 1$ for every $i = j \in T$, and $P_T(i,j) = 0$ for all other values. It thus follows that $P_T \cdot \vec{h}^T = \vec{\hat{h}}^T$. Combining all of the above, we have that

$$\left(\hat{h}(\alpha_1),\ldots,\hat{h}(\alpha_n)\right)^T = V_{\vec{\alpha}} \cdot \vec{\hat{h}}^T = V_{\vec{\alpha}} \cdot P_T \cdot \vec{h}^T = V_{\vec{\alpha}} \cdot P_T \cdot V_{\vec{\alpha}}^{-1} \cdot (h(\alpha_1),\ldots,h(\alpha_n))^T .$$

The claim follows by setting $A = V_{\vec{\alpha}} \cdot P_T \cdot V_{\vec{\alpha}}^{-1}$. ■

By the above claim it follows that the parties can compute F^{deg}_{reduce} by simply multiplying their shares with the constant matrix A from above. That is, the entire protocol for t-privately computing F^{deg}_{reduce} works by the parties t-privately computing the matrix multiplication functionality $F^A_{mat}(\vec{x})$ with the matrix A. By Corollary 2.4.6 (see the end of Section 2.4.2), $F^A_{mat}(\vec{x})$ can be t-privately computed for any $t < n$. Since the entire degree reduction procedure consists of t-privately computing $F^A_{mat}(\vec{x})$, we have the following proposition:

Proposition 2.4.12 *For every $t < n/2$, there exists a protocol that is t-private for F^{deg}_{reduce}, in the presence of a static semi-honest adversary.*

2.4.4 Conclusion

In Section 2.4.3 we proved that there exists a t-private protocol for computing the F_{mult} functionality in the $(F^{2t}_{rand}, F^{deg}_{reduce})$-hybrid model, for any $t < n/2$. Then, in Sections 2.4.3 and 2.4.3 we showed that F^{2t}_{rand} and F^{deg}_{reduce}, respectively, can be t-privately computed (in the plain model) for any $t < n/2$. Finally, in Theorem 2.4.2 we showed that any n-ary functionality can be privately computed in the F_{mult}-hybrid model, for any $t < n$. Combining the above with the modular sequential composition theorem (described in Section 2.2.3), we conclude that:

Theorem 2.4.13 *Let $\mathbb{F}$ be a finite field, let $f : \mathbb{F}^n \to \mathbb{F}^n$ be an n-ary functionality, and let $t < n/2$. Then, there exists a protocol that is t-private for f in the presence of a static semi-honest adversary.*

2.5 Verifiable Secret Sharing (VSS)

2.5.1 Background

Verifiable secret sharing (VSS), defined by Chor et al. [33], is a protocol for sharing a secret in the presence of malicious adversaries. Recall that a secret sharing scheme (with threshold $t+1$) is made up of two stages. In the first stage (called *sharing*), the dealer shares a secret so that any $t+1$ parties can later reconstruct the secret, while any subset of t or fewer parties will learn nothing whatsoever about the secret. In the second stage (called *reconstruction*), a set of $t+1$ or more parties reconstruct the secret. If we consider Shamir's secret-sharing scheme, much can go wrong if the dealer or some of the parties are malicious (e.g., consider the use of secret sharing in Section 2.4). First, in order to share a secret s, the dealer is supposed to choose a random polynomial $q(\cdot)$ of degree t with $q(0) = s$ and then hand each party P_i its share $q(\alpha_i)$. However, nothing prevents the dealer from choosing a polynomial of higher degree. This is a problem because it means that different subsets of $t+1$ parties may reconstruct different values. Thus, the shared value is not well defined. Second, in the reconstruction phase each party P_i provides its share $q(\alpha_i)$. However, a corrupted party can provide a different value, thus effectively changing the value of the reconstructed secret, and the other parties have no way of knowing that the provided value is incorrect. Thus, we must use a method that either prevents the corrupted parties from presenting incorrect shares, or ensures that it is possible to reconstruct the correct secret s given $n-t$ correct shares, even if they are mixed together with t incorrect shares (and no one knows which of the shares are correct or incorrect). Note that in the context of multiparty computation, n parties participate in the reconstruction and not just $t+1$; this is utilized in the following construction.

The BGW protocol for verifiable secret sharing ensures that (for $t < n/3$) the shares received by the honest parties are guaranteed to be $q(\alpha_i)$ for a well-defined degree-t polynomial q, even if the dealer is corrupted. This "secure sharing step" is the challenging part of the protocol. Given such a secure sharing it is possible to use techniques from the field of error-correcting codes in order to reconstruct q (and thus $q(0) = s$) as long as $n-t$ correct shares are provided and $t < n/3$. This is due to the fact that Shamir's secret-sharing scheme when looked at in this context is exactly a Reed-Solomon code, and Reed-Solomon codes can efficiently correct up to t errors, for $t < n/3$.

2.5.2 The Reed-Solomon Code

We briefly describe the Reed-Solomon code, and its use in our context. First, recall that a linear $[n, k, d]$-code over a field $\mathbb{F}$ of size q is a code of length n (meaning that each codeword is a sequence of n field elements), of dimension k (meaning that there are q^k different codewords), and of distance d (meaning that every two codewords are of Hamming distance at least d from each other).

We are interested in constructing a code of length n, dimension $k = t + 1$, and distance $n - t$. The Reed-Solomon code for these parameters is constructed as follows. Let $\mathbb{F}$ be a finite field such that $|\mathbb{F}| > n$, and let $\alpha_1, \ldots, \alpha_n$ be distinct field elements. Let $m = (m_0, \ldots, m_t)$ be a message to be encoded, where each $m_i \in \mathbb{F}$. The encoding of m is as follows:

1. Define a polynomial $p_m(x) = m_0 + m_1 x + \ldots + m_t x^t$ of degree t.

2. Compute the codeword $C(m) = \langle p_m(\alpha_1), \ldots, p_m(\alpha_n)\rangle$.

It is well known that the distance of this code is $n - t$. (In order to see this, recall that for any two different polynomials p_1 and p_2 of degree at most t, there are at most t points α for which $p_1(\alpha) = p_2(\alpha)$. Noting that $m \neq m'$ define different polynomials $p_m \neq p_{m'}$, we have that $C(m)$ and $C(m')$ agree in at most t places.) Let $d(x, y)$ denote the Hamming distance between words $x, y \in \mathbb{F}^n$. The following is a well-known result from the error correcting code literature:

Theorem 2.5.1 *The Reed-Solomon code is a linear $[n, t+1, n-t]$-code over $\mathbb{F}$. In addition, there exists an efficient decoding algorithm that corrects up to $\frac{n-t-1}{2}$ errors. That is, for every $m \in \mathbb{F}^{t+1}$ and every $x \in \mathbb{F}^n$ such that $d(x, C(m)) \leq \frac{n-t-1}{2}$, the decoding algorithm returns m.*

Let $t < n/3$, and so $n \geq 3t + 1$. Plugging this into Theorem 2.5.1, we have that it is possible to efficiently correct up to $\frac{3t+1-t-1}{2} = t$ errors.

Reed-Solomon and Shamir's secret-sharing. Assume that n parties hold shares $\{q(\alpha_i)\}_{i\in[n]}$ of a degree-t polynomial, as in Shamir's secret-sharing scheme. That is, the dealer distributed shares $\{q(\alpha_i)\}_{i\in[n]}$ where $q \in_R \mathcal{P}^{s,t}$ for a secret $s \in \mathbb{F}$. We can view the shares $\langle q(\alpha_1), \ldots, q(\alpha_n)\rangle$ as a Reed-Solomon codeword. Now, in order for the parties to reconstruct the secret from the shares, all parties can just broadcast their shares. Observe that the honest parties provide their correct share $q(\alpha_i)$, whereas the corrupted parties may provide incorrect values. However, since the number of corrupted parties is $t < n/3$, it follows that at most t of the symbols are incorrect. Thus, the Reed-Solomon reconstruction procedure can be run and the honest parties can all obtain the correct polynomial q, and can compute $q(0) = s$.

We conclude that in such a case the corrupted parties cannot effectively cheat in the reconstruction phase. Indeed, even if they provide incorrect values, it is possible for the honest parties to correctly reconstruct the secret (*with probability* 1). Thus, the main challenge in constructing a verifiable secret-sharing protocol is how to force a corrupted dealer to distribute shares that are consistent with some degree-t polynomial.

2.5.3 Bivariate Polynomials

Bivariate polynomials are a central tool used by the BGW verifiable secret sharing protocol (in the sharing stage). We therefore provide a short background to bivariate polynomials in this section.

A bivariate polynomial of degree t is a polynomial over two variables, *each* of which has degree at most t. Such a polynomial can be written as follows:

$$f(x, y) = \sum_{i=0}^{t} \sum_{j=0}^{t} a_{i,j} \cdot x^i \cdot y^j.$$

We denote by $\mathcal{B}^{s,t}$ the set of all bivariate polynomials of degree t and with constant term s. Note that the number of coefficients of a bivariate polynomial in $\mathcal{B}^{s,t}$ is $(t+1)^2 - 1 = t^2 + 2t$ (there are $(t+1)^2$ coefficients, but the constant term is already fixed to be s).

Recall that when considering *univariate* polynomials, $t+1$ points define a unique polynomial of degree t. In this case, each point is a pair (α_k, β_k) and there exists a unique polynomial f such that $f(\alpha_k) = \beta_k$ for all $t+1$ given points $\{(\alpha_k, \beta_k)\}_{k=1}^{t+1}$. The analogous statement for bivariate polynomials is that $t+1$ univariate polynomials of degree t define a unique bivariate polynomial of degree t; see Claim 2.5.2 below. For a degree-t bivariate polynomial $S(x, y)$, fixing the y-value to be some α defines a degree-t univariate polynomial $f(x) = S(x, \alpha)$. Likewise, any $t+1$ fixed values $\alpha_1, \ldots, \alpha_{t+1}$ define $t+1$ degree-t univariate polynomials $f_k(x) = S(x, \alpha_k)$. What we show now is that like in the univariate case, this works in the opposite direction as well. Specifically, given $t+1$ values $\alpha_1, \ldots, \alpha_{t+1}$ and $t+1$ degree-t polynomials $f_1(x), \ldots, f_{t+1}(x)$ there exists a unique bivariate polynomial $S(x, y)$ such that $S(x, \alpha_k) = f_k(x)$, for every $k = 1, \ldots, t+1$. This is formalized in the next claim, which was proven in [46]:

Claim 2.5.2 *Let t be a nonnegative integer, let $\alpha_1, \ldots, \alpha_{t+1}$ be $t+1$ distinct elements in $\mathbb{F}$, and let $f_1(x), \ldots, f_{t+1}(x)$ be $t+1$ polynomials of degree t. Then, there exists a unique bivariate polynomial $S(x, y)$ of degree t such that for every $k = 1, \ldots, t+1$ it holds that*

$$S(x, \alpha_k) = f_k(x). \tag{2.5.1}$$

Proof: Define the bivariate polynomial $S(x, y)$ via the Lagrange interpolation:

$$S(x, y) = \sum_{i=1}^{t+1} f_i(x) \cdot \frac{\prod_{j \neq i}(y - \alpha_j)}{\prod_{j \neq i}(\alpha_i - \alpha_j)}$$

It is easy to see that $S(x, y)$ has degree t. Moreover, for every $k = 1, \ldots, t+1$ it holds that:

$$\begin{aligned} S(x, \alpha_k) &= \sum_{i=1}^{t+1} f_i(x) \cdot \frac{\prod_{j \neq i}(\alpha_k - \alpha_j)}{\prod_{j \neq i}(\alpha_i - \alpha_j)} \\ &= f_k(x) \cdot \frac{\prod_{j \neq k}(\alpha_k - \alpha_j)}{\prod_{j \neq k}(\alpha_k - \alpha_j)} + \sum_{i \in [t+1] \setminus \{k\}} f_i(x) \cdot \frac{\prod_{j \neq i}(\alpha_k - \alpha_j)}{\prod_{j \neq i}(\alpha_i - \alpha_j)} \\ &= f_k(x) + 0 = f_k(x) \end{aligned}$$

and $S(x, y)$ therefore satisfies Eq. (2.5.1). It remains to show that S is unique. Assume that there exist two different t-degree bivariate polynomials $S_1(x, y)$ and $S_2(x, y)$ that satisfy Eq. (2.5.1).

Define the polynomial

$$R(x,y) \stackrel{\text{def}}{=} S_1(x,y) - S_2(x,y) = \sum_{i=0}^{t}\sum_{j=0}^{t} r_{i,j}x^i y^j.$$

We will now show that $R(x,y) = 0$. First, for every $k \in [t+1]$ it holds that:

$$R(x,\alpha_k) = \sum_{i,j=0}^{t} r_{i,j}x^i(\alpha_k)^j = S_1(x,\alpha_k) - S_2(x,\alpha_k) = f_k(x) - f_k(x) = 0,$$

where the last equality follows from Eq. (2.5.1). We can rewrite the univariate polynomial $R(x,\alpha_k)$ as

$$R(x,\alpha_k) = \sum_{i=0}^{t}\left(\left(\sum_{j=0}^{t} r_{i,j}(\alpha_k)^j\right)\cdot x^i\right).$$

As we have seen, $R(x,\alpha_k) = 0$ for every x. Thus, its coefficients are all zeroes,[3] implying that for every fixed $i \in [t+1]$ it holds that $\sum_{j=0}^{t} r_{i,j}(\alpha_k)^j = 0$. This in turn implies that for every fixed $i \in [t+1]$, the polynomial $h_i(x) = \sum_{j=0}^{t} r_{i,j}x^j$ is zero for $t+1$ points (i.e., the points $\alpha_1,\ldots,\alpha_{t+1}$), and so $h_i(x)$ is also the zero polynomial. Thus, its coefficients $r_{i,j}$ equal 0 for every $j \in [t+1]$. This holds for every fixed i, and therefore for every $i,j \in [t+1]$ we have that $r_{i,j} = 0$. We conclude that $R(x,y) = 0$ for every x and y, and hence $S_1(x,y) = S_2(x,y)$. ■

Verifiable secret sharing using bivariate polynomials. The verifiable secret-sharing protocol works by embedding a random univariate degree-t polynomial $q(z)$ with $q(0) = s$ into the bivariate polynomial $S(x,y)$. Specifically, $S(x,y)$ is chosen at random under the constraint that $S(0,z) = q(z)$; the values $q(\alpha_1),\ldots,q(\alpha_n)$ are thus the univariate Shamir-shares embedded into $S(x,y)$. Then, the dealer sends each party P_i two univariate polynomials as intermediate shares; these polynomials are $f_i(x) = S(x,\alpha_i)$ and $g_i(y) = S(\alpha_i,y)$. The "actual" share of each party is $f_i(0)$, which is a share of a univariate polynomial, as in the semi-honest case. However, the two polynomials are given in order perform the verification and consistency of the sharing.

By the definition of these polynomials, it holds that $f_i(\alpha_j) = S(\alpha_j,\alpha_i) = g_j(\alpha_i)$, and $g_i(\alpha_j) = S(\alpha_i,\alpha_j) = f_j(\alpha_i)$. Thus, any two parties P_i and P_j can verify that the univariate polynomials that they received are *pairwise consistent* with each other by checking that $f_i(\alpha_j) = g_j(\alpha_i)$ and $g_i(\alpha_j) = f_j(\alpha_i)$. As we shall see, this prevents the dealer from distributing shares that are not consistent with a single bivariate polynomial. Finally, party P_i defines its output (i.e., "Shamir share") as $f_i(0) = q(\alpha_i)$, as required.

We begin by proving that pairwise consistency checks as described above suffice for uniquely determining the bivariate polynomial S. Specifically:

Claim 2.5.3 *Let $K \subseteq [n]$ be a set of indices such that $|K| \geq t+1$, let $\{f_k(x), g_k(y)\}_{k\in K}$ be a set of pairs of degree-t polynomials, and let $\{\alpha_k\}_{k\in K}$ be distinct non-zero elements in $\mathbb{F}$. If for*

[3]In order to see that all the coefficients of a polynomial which is identical to zero are zeroes, let $p(x) = \sum_{i=0}^{t} a_i x^t$, where $p(x) = 0$ for every x. Let $\vec{a}$ be a vector of the coefficients of p, and let $\vec{\beta}$ be some vector of size $t+1$ of some distinct non-zero elements. Let $V_{\vec{\beta}}$ be the Vandermonde matrix for $\vec{\beta}$. Then, $V_{\vec{\beta}} \cdot \vec{a} = 0$, and therefore $\vec{a} = V_{\vec{\beta}}^{-1} \cdot 0 = 0$.

every $i, j \in K$, it holds that $f_i(\alpha_j) = g_j(\alpha_i)$, then there exists a unique bivariate polynomial S of degree-t in both variables such that $f_k(x) = S(x, \alpha_k)$ and $g_k(y) = S(\alpha_k, y)$ for every $k \in K$.

Proof: Let L be any subset of K of cardinality exactly $t+1$. By Claim 2.5.2, there exists a *unique* bivariate polynomial $S(x, y)$ of degree-t in both variables, for which $S(x, \alpha_\ell) = f_\ell(x)$ for every $\ell \in L$. We now show if $f_i(\alpha_j) = g_j(\alpha_i)$ for all $i, j \in K$, then for every $k \in K$ it holds that $f_k(x) = S(x, \alpha_k)$ and $g_k(y) = S(\alpha_k, y)$.

By the consistency assumption, for every $k \in K$ and $\ell \in L$ we have that $g_k(\alpha_\ell) = f_\ell(\alpha_k)$. Furthermore, by the definition of S from above we have that $f_\ell(\alpha_k) = S(\alpha_k, \alpha_\ell)$. Thus, for all $k \in K$ and $\ell \in L$ it holds that $g_k(\alpha_\ell) = S(\alpha_k, \alpha_\ell)$. Since both $g_k(y)$ and $S(\alpha_k, y)$ are degree-t polynomials, and $g_k(\alpha_\ell) = S(\alpha_k, \alpha_\ell)$ for $t+1$ points α_ℓ, it follows that $g_k(y) = S(\alpha_k, y)$ for every $k \in K$.

It remains to show that $f_k(x) = S(x, \alpha_k)$ for all $k \in K$ (this trivially holds for all $k \in L$ by the definition of S from above, but needs to be proven for $k \in K \setminus L$). By consistency, for every $j, k \in K$, we have that $f_k(\alpha_j) = g_j(\alpha_k)$. Furthermore, we have already proven that $g_j(\alpha_k) = S(\alpha_j, \alpha_k)$ for every $j, k \in K$. Therefore, $f_k(\alpha_j) = S(\alpha_j, \alpha_k)$ for every $j, k \in K$, implying that $f_k(x) = S(x, \alpha_k)$ for every $k \in K$ (because they are degree-t polynomials who have the same value on more than t points). This concludes the proof. ■

We now proceed to prove a "secrecy lemma" for bivariate polynomial secret-sharing. Loosely speaking, we prove that the shares $\{f_i(x), g_i(y)\}_{i \in I}$ (for $|I| \le t$) that the corrupted parties receive do not reveal any information about the secret s. In fact, we show something much stronger: for every two degree-t polynomials q_1 and q_2 such that $q_1(\alpha_i) = q_2(\alpha_i) = f_i(0)$ for every $i \in I$, the distribution over the shares $\{f_i(x), g_i(y)\}_{i \in I}$ received by the corrupted parties when $S(x, y)$ is chosen based on $q_1(z)$ is identical to the distribution when $S(x, y)$ is chosen based on $q_2(z)$. An immediate corollary of this is that no information is revealed about whether the secret equals $s_1 = q_1(0)$ or $s_2 = q_2(0)$.

Claim 2.5.4 *Let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ be n distinct non-zero values, let $I \subset [n]$ with $|I| \le t$, and let q_1 and q_2 be two degree-t polynomials over $\mathbb{F}$ such that $q_1(\alpha_i) = q_2(\alpha_i)$ for every $i \in I$. Then,*

$$\left\{ \{(i, S_1(x, \alpha_i), S_1(\alpha_i, y))\}_{i \in I} \right\} \equiv \left\{ \{(i, S_2(x, \alpha_i), S_2(\alpha_i, y))\}_{i \in I} \right\}$$

where $S_1(x, y)$ and $S_2(x, y)$ are degree-t bivariate polynomial chosen at random under the constraints that $S_1(0, z) = q_1(z)$ and $S_2(0, z) = q_2(z)$, respectively.

Proof: We begin by defining probability ensembles $\mathbb{S}_1$ and $\mathbb{S}_2$, as follows:

$$\begin{aligned} \mathbb{S}_1 &= \left\{ \{(i, S_1(x, \alpha_i), S_1(\alpha_i, y))\}_{i \in I} \mid S_1 \in_R \mathcal{B}^{q_1(0), t} \text{ s.t. } S_1(0, z) = q_1(z) \right\} \\ \mathbb{S}_2 &= \left\{ \{(i, S_2(x, \alpha_i), S_2(\alpha_i, y))\}_{i \in I} \mid S_2 \in_R \mathcal{B}^{q_2(0), t} \text{ s.t. } S_2(0, z) = q_2(z) \right\} \end{aligned}$$

Given this notation, an equivalent formulation of the claim is that $\mathbb{S}_1 \equiv \mathbb{S}_2$.

In order to prove that this holds, we first show that for any set of pairs of degree-t polynomials $Z = \{(i, f_i(x), g_i(y))\}_{i \in I}$, the number of bivariate polynomials in the support of $\mathbb{S}_1$ that are consistent with Z equals the number of bivariate polynomials in the support of $\mathbb{S}_2$ that are consistent with Z, where consistency means that $f_i(x) = S(x, \alpha_i)$ and $g_i(y) = S(\alpha_i, y)$.

First note that if there exist $i, j \in I$ such that $f_i(\alpha_j) \neq g_j(\alpha_i)$ then there does not exist any bivariate polynomial in the support of $\mathbb{S}_1$ or $\mathbb{S}_2$ that is consistent with Z. Also, if there exists an $i \in I$ such that $f_i(0) \neq q_1(\alpha_i)$, then once again there is no polynomial from $\mathbb{S}_1$ or $\mathbb{S}_2$ that is consistent (this holds for $\mathbb{S}_1$ since $f_i(0) = S(0, \alpha_i) = q_1(\alpha_i)$ should hold, and it holds similarly for $\mathbb{S}_2$ because $q_1(\alpha_i) = q_2(\alpha_i)$ for all $i \in I$).

Let $Z = \{(i, f_i(x), g_i(y))\}_{i \in I}$ be a set of degree-t polynomials such that for every $i, j \in I$ it holds that $f_i(\alpha_j) = g_j(\alpha_i)$, and in addition for every $i \in I$ it holds that $f_i(0) = q_1(\alpha_i) = q_2(\alpha_i)$. We begin by counting how many such polynomials exist in the support of $\mathbb{S}_1$. We have that Z contains $|I|$ degree-t polynomials $\{f_i(x)\}_{i \in I}$, and recall that $t+1$ such polynomials $f_i(x)$ fully define a degree-t bivariate polynomial. Thus, we need to choose $t+1-|I|$ more polynomials $f_j(x)$ $(j \neq i)$ that are consistent with $q_1(z)$ and with $\{g_i(y)\}_{i \in I}$. In order for a polynomial $f_j(x)$ to be consistent in this sense, it must hold that $f_j(\alpha_i) = g_i(\alpha_j)$ for every $i \in I$, and in addition that $f_j(0) = q_1(\alpha_j)$. Thus, for each such $f_j(x)$ that we add, $|I|+1$ values of f_j are already determined. Since the values of f_j at $t+1$ points determine a degree-t univariate polynomial, it follows that an additional $t-|I|$ points can be chosen in all possible ways and the result will be consistent with Z. We conclude that there exist $\left(|\mathbb{F}|^{t-|I|}\right)^{(t+1-|I|)}$ ways to choose S_1 according to $\mathbb{S}_1$ that will be consistent. (Note that if $|I| = t$ then there is just one way.) The important point here is that the exact same calculation holds for S_2 chosen according to $\mathbb{S}_2$, and thus exactly the same number of polynomials from $\mathbb{S}_1$ are consistent with Z as from $\mathbb{S}_2$.

Now, let $Z = \{(i, f_i(x), g_i(y))\}_{i \in I}$ be a set of $|I|$ pairs of univariate degree-t polynomials. We have already shown that the number of polynomials in the support of $\mathbb{S}_1$ that are consistent with Z equals the number of polynomials in the support of $\mathbb{S}_2$ that are consistent with Z. Since the polynomials S_1 and S_2 (in $\mathbb{S}_1$ and $\mathbb{S}_2$, respectively) are chosen randomly among those consistent with Z, it follows that the probability that Z is obtained is exactly the same in both cases, as required. ■

2.5.4 The Verifiable Secret Sharing Protocol

In the VSS functionality, the dealer inputs a polynomial $q(x)$ of degree t, and each party P_i receives its Shamir share $q(\alpha_i)$ based on that polynomial. The "verifiable" part is that if q is of degree greater than t, then the parties reject the dealer's shares and output $\perp$. The functionality is formally defined as follows:

FUNCTIONALITY 2.5.5 (The F_{VSS} functionality)

$$F_{VSS}(q(x), \lambda, \ldots, \lambda) = \begin{cases} (q(\alpha_1), \ldots, q(\alpha_n)) & \text{if } \deg(q) \leq t \\ (\perp, \ldots, \perp) & \text{otherwise} \end{cases}$$

Observe that the secret $s = q(0)$ is only implicitly defined in the functionality; it is however well defined. Thus, in order to share a secret s, the functionality is used by having the dealer first choose a random polynomial $q \in_R \mathcal{P}^{s,t}$ (where $\mathcal{P}^{s,t}$ is the set of all degree-t univariate polynomials with constant term s) and then run F_{VSS} with input $q(x)$.

We implement the sharing of univariate polynomial using a different verifiable sharing functionality for distribution a *bivariate* polynomial. This gives a modular exposition of the protocol. In addition, sharing bivariate polynomial is an important building block in our construction of the simpler multiplication protocol, that we will describe in Chapter 3. Thus, this sub-functionality is important in its own right.

In functionality for distributing bivariate polynomial (denoted[4] by $\widetilde{F}_{VSS}$), the dealer holds a bivariate polynomial $S(x, y)$ of degree-t in both variables, and each party P_i receives its shares on this polynomial – $f_i(x), g_i(y)$ based on S. The trusted party verifies that if the S is of degree greater than t, then the parties reject and output $\perp$. We define the functionality $\widetilde{F}_{VSS}$ as follows:

FUNCTIONALITY 2.5.6 (The $\widetilde{F}_{VSS}$ functionality)

$$\widetilde{F}_{VSS}(S(x,y), \lambda, \ldots, \lambda) = \begin{cases} ((f_1(x), g_1(y)), \ldots, (f_n(x), g_n(y))) & \text{if } \deg(S) \leq t \\ (\perp, \ldots, \perp) & \text{otherwise} \end{cases},$$

where $f_i(x) = S(x, \alpha_i)$, $g_i(y) = S(\alpha_i, y)$.

We describe the protocol for sharing a bivariate polynomial in the next subsection.

Implementing F_{VSS} in the $\widetilde{F}_{VSS}$-hybrid model. We now proceed to the protocol that implements sharing of a univariate polynomial using a functionality for sharing a bivariate polynomial. The protocol is very simple, and there is no interaction between the parties rather than the invocation of $\widetilde{F}_{VSS}$-functionality. Specifically, the dealer holds as input a polynomial $q(z)$ and chooses a bivariate polynomial $S(x, y)$ of degree-t uniformly at random under the constraint that $S(0, z) = q(z)$. The parties then invoke the $\widetilde{F}_{VSS}$ functionality, where the dealer inputs $S(x, y)$. Then, upon receiving outputs $f_i(x), g_i(y)$, the parties simply outputs $f_i(0) = S(0, \alpha_i) = q(\alpha_i)$. By the security of $\widetilde{F}_{VSS}$, $S(0, z)$ is a univariate polynomial of degree-t, and thus $q(z)$ is well-defined. See Protocol 2.5.7.

PROTOCOL 2.5.7 (Securely Computing F_{VSS} in the $\widetilde{F}_{VSS}$-hybrid model)

- **Input:** The dealer $D = P_1$ holds a polynomial $q(x)$ of degree at most t (if not, then the honest dealer just aborts at the onset). The other parties $P_2, \ldots, P_n$ have no input.
- **Common input:** The description of a field $\mathbb{F}$ and n non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **The protocol:**
  1. The dealer selects a uniformly distributed bivariate polynomial $S(x, y) \in \mathcal{B}^{q(0),t}$, under the constraint that $S(0, z) = q(z)$.
  2. The parties invoke the $\widetilde{F}_{VSS}$ functionality where P_1 is dealer and inputs $S(x, y)$, each other party has no input.
- **Output** If the output of $\widetilde{F}_{VSS}$ is $(f_i(x), g_i(y))$, output $f_i(0)$. Otherwise, output $\perp$.

Theorem 2.5.8 *Let $t < n/3$. Then, Protocol 2.5.7 is t-secure for the F_{VSS} functionality in the $\widetilde{F}_{VSS}$-hybrid model, in the presence of a static malicious adversary.*

Proof: We separately deal with the case that the dealer is honest and the case where the dealer is corrupted. When the dealer is corrupted, security is a simple reduction to the security of $\widetilde{F}_{VSS}$, and the guarantee that the outputs of $\widetilde{F}_{VSS}$ are shares on bivariate polynomial of degree-t. When the dealer is honest, a rather more involved argument is needed. In particular,

[4]Note that $\widetilde{F}_{VSS}$ denotes sharing a bivariate polynomial, whereas F_{VSS} denotes sharing a univariate polynomial.

giving the output shares (which are shares of a univariate polynomial), the simulator has to give shares of a bivariate polynomial to the corrupted parties. However, using Claim 2.5.4, the simulator can just choose this bivariate polynomial uniformly at random.

Case 1 – the Dealer is Honest

In the ideal execution, the dealer sends $q(x)$ to the trusted party computing F_{VSS} and each honest party P_j receives $q(\alpha_j)$, outputs it, and never outputs $\perp$. The corrupted parties have no input and so the adversary has no influence on the output of the honest parties.

The simulator $\mathcal{SIM}$.

- *$\mathcal{SIM}$ internally invokes $\mathcal{A}$ on the auxiliary input z.*
- Interacting with the trusted party: *$\mathcal{SIM}$ receives the output of the corrupted parties $\{q(\alpha_i)\}_{i\in I}$.*
- *$\mathcal{SIM}$ chooses any polynomial $q'(x)$ of degree-t under the constraint that: $q'(\alpha_i) = q(\alpha_i)$ for every $i \in I$.*
- *$\mathcal{SIM}$ chooses a bivariate polynomial $S'(x,y)$ of degree-t in both variables under the constraint that $S'(0,z) = q'(z)$.*
- *$\mathcal{SIM}$ internally gives the adversary $\mathcal{A}$ the bivariate shares $\{S'(x,\alpha_i), S'(\alpha_i,y)\}_{i\in I}$ as the output of $\widetilde{F}_{VSS}$ functionality, outputs whatever $\mathcal{A}$ outputs and halts.*

We need to show that the joint distribution of the outputs of the honest parties and the view of the adversary is identical in both executions. It is clear that the output of the honest parties is the same in both executions, since the dealer is honest. In the real execution, the input of the dealer is always a degree-t polynomial, and by the security of $\widetilde{F}_{VSS}$ the output of each honest party is $f_i(0) = S(0,\alpha_i) = q(\alpha_i)$. In the ideal execution, the output of the honest parties is clearly $q(\alpha_i)$, as required. Next, the view of the parties is simply the output of the $\widetilde{F}_{VSS}$-functionality. Using Cleaim 2.5.4, the distribution of the shares on the random bivariate polynomial $S(x,y)$ (chosen in the real execution), is identical to the distribution of the shares of the random bivariate polynomial $S'(x,y)$ (chosen in the ideal execution), since the underlying univariate polynomials $q(x), q'(x)$, respectively, satisfy $q(\alpha_i) = q'(\alpha_i)$ for every $i \in I$. This concludes the proof for honest dealer.

Case 2 – the Dealer is Corrupted

This case is straightforward. The simulator receives from the corrupted dealer the polynomial $S(x,y)$ – its input to the (internally simulated) $\widetilde{F}_{VSS}$. In case $S(x,y)$ is valid, it sends the univariate polynomial $q(y) \overset{\text{def}}{=} S(0,y)$ to the trusted parties, and each honest party P_j receives its shares on $S(x,y)$ and outputs $q(\alpha_j)$. In case $S(x,y)$ is invalid, the simulator sends the trusted party an invalid polynomial x^{2t} (i.e., univariate polynomial with degree greater than t), and each honest party outputs $\perp$. This is exactly as in the real execution – where here $\widetilde{F}_{VSS}$ checks the polynomial and gives the parties their (bivariate) shares on this polynomial, and then each party outputs its share $f_i(0)$. For completeness, we formally describe the simulator $\mathcal{SIM}$.

The simulator $\mathcal{SIM}$.

- *$\mathcal{SIM}$ internally invokes the adversary $\mathcal{A}$ on the auxiliary input z.*
- *$\mathcal{SIM}$ internally receives from $\mathcal{A}$ the input $S(x,y)$ of the dealer P_1 to the functionality $\widetilde{F}_{VSS}$.*
- *If $S(x,y)$ is of degree-t in both variables, $\mathcal{SIM}$ sends the univariate polynomial $S(0,z)$ to the trusted party computing F_{VSS}, and gives the adversary $\mathcal{A}$ the bivariate shares $\{S(\alpha_i,y),S(x,\alpha_i)\}_{i\in I}$ as the outputs of the simulated $\widetilde{F}_{VSS}$ functionality.*
- *If $S(x,y)$ is of degree higher than t, $\mathcal{SIM}$ sends x^{2t} (i.e., an invalid polynomial) to the trusted party computing F_{VSS}, and gives the adversary the values $\{\perp\}_{i\in I}$ as outputs from $\widetilde{F}_{VSS}$.*
- *$\mathcal{SIM}$ outputs whatever $\mathcal{A}$ outputs and halts.*

■

2.5.5 Sharing a Bivariate Polynomial

We now proceed to the implementation of $\widetilde{F}_{VSS}$ functionality. We recall that the functionality is defined as follows (Functionality 2.5.6):

$$\widetilde{F}_{VSS}(S(x,y),\lambda,\ldots,\lambda)=\begin{cases}((f_1(x),g_1(y)),\ldots,(f_n(x),g_n(y))) & \text{if } \deg(S)\leq t\\ (\perp,\ldots,\perp) & \text{otherwise}\end{cases},$$

where $f_i(x)=S(x,\alpha_i)$, $g_i(y)=S(\alpha_i,y)$.

The protocol idea. We present the VSS protocol of BGW with the simplification of the complaint phase suggested by [45]. The protocol uses private point-to-point channels between each pair of parties and an *authenticated* broadcast channel (meaning that the identity of the broadcaster is given).

The input of the dealer is the polynomial $S(x,y)$ of degree-t in both variables. The dealer then sends each party P_i two polynomials that are derived from $S(x,y)$: the polynomial $f_i(x)=S(x,\alpha_i)$ and the polynomial $g_i(y)=S(\alpha_i,y)$. As we have shown in Claim 2.5.4, t pairs of polynomials $f_i(x),g_i(y)$ received by the corrupted parties reveal nothing about the constant term of S (i.e., the secret being shared). In addition, given these polynomials, the parties can verify that they have consistent inputs. Specifically, since $g_i(\alpha_j)=S(\alpha_i,\alpha_j)=f_j(\alpha_i)$, it follows that each pair of parties P_i and P_j can check that their polynomials fulfill $f_i(\alpha_j)=g_j(\alpha_i)$ and $g_i(\alpha_j)=f_j(\alpha_i)$ by sending each other these points. If all of these checks pass, then by Claim 2.5.3 it follows that all the polynomials are derived from a single bivariate polynomial $S(x,y)$, and thus the sharing is valid and the secret is fully determined.

The problem that arises is what happens if the polynomials are not all consistent; i.e., if P_j receives from P_i values $f_i(\alpha_j),g_i(\alpha_j)$ such that $f_j(\alpha_i)\neq g_i(\alpha_j)$ or $g_j(\alpha_i)\neq f_i(\alpha_j)$. This can happen if the dealer is corrupted, or if P_i is corrupted. In such a case, P_j issues a "complaint" by broadcasting its inconsistent values $(j,i,f_j(\alpha_i),g_j(\alpha_i))$ defined by the shares $f_j(x),g_j(y)$ it received from the dealer. Then, the dealer checks if these values are correct, and if they are not then it is required to broadcast the correct polynomials for that complaining party. We stress that in this case the dealer broadcasts the *entire polynomials* $f_j(x)$ and $g_j(y)$ defining P_j's share, and this enables all other parties P_k to verify that these polynomials are consistent with their own shares, thus verifying their validity. Note that if the values broadcast *are* correct (e.g., in the case that the dealer is honest and P_i sent P_j incorrect values) then the dealer does not broadcast P_j's polynomials. This ensures that an honest dealer does not reveal the shares of honest parties.

This strategy is sound since if the dealer is honest, then all honest parties will have consistent values. Thus, the only complaints will be due to corrupted parties complaining falsely (in which case the dealer will broadcast the *corrupted parties* polynomials, which gives them no more information), or due to corrupted parties sending incorrect values to honest parties (in which case the dealer does not broadcast anything, as mentioned). In contrast, if the dealer is not honest, then all honest parties will reject and output $\perp$ unless it re-sends consistent polynomials to all, thereby guaranteeing that $S(x, y)$ is fully defined again, as required. This complaint resolution must be carried out carefully in order to ensure that security is maintained. We defer more explanation about how this works until after the full specification, given in Protocol 2.5.7.

PROTOCOL 2.5.9 (Securely Computing $\widetilde{F}_{VSS}$)

- **Input:** The dealer $D = P_1$ holds a bivariate polynomial $S(x,y)$ of degree at most t in both variables (if not, then the honest dealer just aborts at the onset). The other parties $P_2, \ldots, P_n$ have no input.
- **Common input:** The description of a field $\mathbb{F}$ and n non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **The protocol:**
  1. **Round 1 (send shares) – the dealer:**
     (a) For every $i \in \{1, \ldots, n\}$, the dealer defines the polynomials $f_i(x) \stackrel{\text{def}}{=} S(x, \alpha_i)$ and $g_i(y) \stackrel{\text{def}}{=} S(\alpha_i, y)$. It then sends to each party P_i the polynomials $f_i(x)$ and $g_i(y)$.
  2. **Round 2 (exchange subshares) – each party P_i:**
     (a) Store the polynomials $f_i(x)$ and $g_i(y)$ that were received from the dealer. (If $f_i(x)$ or $g_i(y)$ is of degree greater than t then truncate it to be of degree t.)
     (b) For every $j \in \{1, \ldots, n\}$, send $f_i(\alpha_j)$ and $g_i(\alpha_j)$ to party P_j.
  3. **Round 3 (broadcast complaints) – each party P_i:**
     (a) For every $j \in \{1, \ldots, n\}$, let (u_j, v_j) denote the values received from player P_j in Round 2 (these are supposed to be $u_j = f_j(\alpha_i)$ and $v_j = g_j(\alpha_i)$).
     If $u_j \neq g_i(\alpha_j)$ or $v_j \neq f_i(\alpha_j)$, then broadcast $\mathsf{complaint}(i, j, f_i(\alpha_j), g_i(\alpha_j))$.
     (b) If no parties broadcast a complaint, then every party P_i outputs $f_i(0)$ and halts.
  4. **Round 4 (resolve complaints) – the dealer:** For every complaint message received, do the following:
     (a) Upon viewing a message $\mathsf{complaint}(i, j, u, v)$ broadcast by P_i, check that $u = S(\alpha_j, \alpha_i)$ and $v = S(\alpha_i, \alpha_j)$. (Note that if the dealer and P_i are honest, then it holds that $u = f_i(\alpha_j)$ and $v = g_i(\alpha_j)$.) If the above condition holds, then do nothing. Otherwise, broadcast $\mathsf{reveal}(i, f_i(x), g_i(y))$.
  5. **Round 5 (evaluate complaint resolutions) – each party P_i:**
     (a) For every $j \neq k$, party P_i marks (j,k) as a joint complaint if it viewed two messages $\mathsf{complaint}(k, j, u_1, v_1)$ and $\mathsf{complaint}(j, k, u_2, v_2)$ broadcast by P_k and P_j, respectively, such that $u_1 \neq v_2$ or $v_1 \neq u_2$. If there exists a joint complaint (j,k) for which the dealer did not broadcast $\mathsf{reveal}(k, f_k(x), g_k(y))$ nor $\mathsf{reveal}(j, f_j(x), g_j(y))$, then go to Step 6 (and do not broadcast consistent). Otherwise, proceed to the next step.
     (b) Consider the set of $\mathsf{reveal}(j, f_j(x), g_j(y))$ messages sent by the dealer (truncating the polynomials to degree t if necessary as in Step 2a):
        i. If there exists a message in the set with $j = i$ then reset the stored polynomials $f_i(x)$ and $g_i(y)$ to the new polynomials that were received, and go to Step 6 (without broadcasting consistent).
        ii. If there exists a message in the set with $j \neq i$ and for which $f_i(\alpha_j) \neq g_j(\alpha_i)$ or $g_i(\alpha_j) \neq f_j(\alpha_i)$, then go to Step 6 (without broadcasting consistent).
     If the set of reveal messages does not contain a message that fulfills either one of the above conditions, then proceed to the next step.
     (c) Broadcast the message consistent.
  6. **Output decision (if there were complaints) – each party P_i:** If at least $n - t$ parties broadcast consistent, output $(f_i(x), g_i(y))$. Otherwise, output $\perp$.

The security of Protocol 2.5.9. Before we prove that Protocol 2.5.9 is t-secure for the $\widetilde{F}_{VSS}$ functionality, we present an intuitive argument as to why this holds. First, consider the case that the dealer is honest. In this case, all of the polynomials received by the parties are consistent (i.e., for every pair P_i, P_j it holds that $f_i(\alpha_j) = g_j(\alpha_i)$ and $f_j(\alpha_i) = g_i(\alpha_j)$). Thus, an honest party P_j only broadcasts a complaint if a corrupted party sends it incorrect values and the values included in that complaint are known already to the adversary. However, if this occurs then the dealer will *not* send a reveal of the honest party's polynomials (because its values are correct). Furthermore, if any corrupted party P_i broadcasts a complaint with incorrect values (u, v), the dealer can send the correct reveal message (this provides no additional information to the adversary since the reveal message just contains the complainant's shares). In such a case, the check carried out by each honest party P_j in Step 5(b)ii will pass and so every honest party will broadcast consistent. Thus, at least $n-t$ parties broadcast consistent (since there are at least $n-t$ honest parties) and so every honest party P_j outputs $f_j(x) = S(x, \alpha_j), g_j(y) = S(\alpha_j, y)$.

Next, consider the case that the dealer is corrupted. In this case, the honest parties may receive polynomials that are not consistent with each other; that is, honest P_j and P_k may receive polynomials $f_j(x), g_j(y)$ and $f_k(x), g_k(y)$ such that either $f_j(\alpha_k) \neq g_k(\alpha_j)$ or $f_k(\alpha_j) \neq g_j(\alpha_k)$. However, in such a case both honest parties complain, and the dealer must send a valid reveal message (in the sense described below) or no honest party will broadcast consistent. In order for $n-t$ parties to broadcast consistent, there must be at least $(n-t)-t = t+1$ honest parties that broadcast consistent. This implies that these $t+1$ or more honest parties all received polynomials $f_j(x)$ and $g_j(y)$ in the first round that are *pairwise consistent* with each other and with all of the "fixed" values in the reveal messages. Thus, by Claim 2.5.3 the polynomials $f_j(x)$ and $g_j(y)$ of these $t+1$ (or more) parties are all derived from a unique degree-t bivariate polynomial $S(x, y)$, meaning that $f_j(x) = S(x, \alpha_j)$ and $g_j(y) = S(\alpha_j, y)$. (The parties who broadcasted consistent are those that make up the set K in Claim 2.5.3.)

The above suffices to argue that the polynomials of all the honest parties that broadcast consistent are derived from a unique $S(x, y)$. It remains to show that if at least $t+1$ honest parties broadcast consistent, then the polynomials of all the other honest parties that do not broadcast consistent are also derived from the same $S(x, y)$. Assume that this is not the case. That is, there exists an honest party P_j such that $f_j(x) \neq S(x, \alpha_j)$ (an analogous argument can be made with respect to $g_j(x)$ and $S(\alpha_j, y)$). Since $f_j(x)$ is of degree-t this implies that $f_j(\alpha_k) = S(\alpha_k, \alpha_j)$ for at most t points α_k. Thus, P_j's points are pairwise consistent with at most t honest parties that broadcast consistent (since for all of these parties $g_k(y) = S(\alpha_k, y)$). This implies that there must have been a joint complaint between P_j and an honest party P_k who broadcasted consistent, and so this complaint must have been resolved by the dealer broadcasting polynomials $f_j(x)$ and $g_j(y)$ such that $f_j(\alpha_k) = g_k(\alpha_j)$ for all P_k who broadcasted consistent (otherwise, they would not have broadcasted consistent). We now proceed to the formal proof.

Theorem 2.5.10 *Let $t < n/3$. Then, Protocol 2.5.9 is t-secure for the $\widetilde{F}_{VSS}$ functionality in the presence of a static malicious adversary.*

Proof: Let $\mathcal{A}$ be an adversary in the real world. We show the existence of a simulator $\mathcal{SIM}$ such that for any set of corrupted parties I and for all inputs, the output of all parties and the adversary $\mathcal{A}$ in an execution of the real protocol with $\mathcal{A}$ is identical to the outputs in an execution with $\mathcal{SIM}$ in the ideal model. We separately deal with the case that the dealer is honest and the case that the dealer is corrupted. Loosely speaking, when the dealer is honest we show that the honest parties always accept the dealt shares, and in particular that the adversary cannot falsely generate complaints that will interfere with the result. In the case that the dealer is corrupted the proof is more involved and consists of showing that if the dealer resolves complaints so that at least $n - t$ parties broadcast consistent, then this implies that at the end of the protocol all honest parties hold consistent shares, as required.

Case 1 – the Dealer is Honest

In this case in an *ideal execution*, the dealer sends $S(x, y)$ to the trusted party and each honest party P_j receives $S(x, \alpha_j), S(\alpha_j, y)$ from the trusted party, outputs it, and never outputs $\perp$. Observe that none of the corrupted parties have input and so the adversary has no influence on the output of the honest parties. We begin by showing that this always holds in a *real execution* as well; i.e., in a real execution each honest party P_j always outputs $S(x, \alpha_j), S(\alpha_j, y)$ and never outputs $\perp$.

Since the dealer is honest, its input is always a valid bivariate polynomial and sends each party the prescribed values. In this case, an honest party P_j always outputs either $S(x, \alpha_j), S(\alpha_j, y)$ or $\perp$. This is due to the fact that its polynomial $f_j(x)$ will never be changed, because it can only be changed if a $\mathsf{reveal}(j, f'_j(x), g_j(y))$ message is sent with $f'_j(x) \neq f_j(x)$. However, an honest dealer never does this. Thus, it remains to show that P_j never outputs $\perp$. In order to see this, recall that an honest party outputs bivariate shares and not $\perp$ if and only if at least $n - t$ parties broadcast consistent. Thus, it suffices to show that all honest parties broadcast consistent. An honest party P_j broadcasts consistent if and only if the following conditions hold:

1. The dealer resolves all conflicts: Whenever a pair of complaint messages $\mathsf{complaint}(k, \ell, u_1, v_1)$ and $\mathsf{complaint}(\ell, k, u_2, v_2)$ were broadcast such that $u_1 \neq v_2$ and $v_1 \neq u_2$ for some k and ℓ, the dealer broadcasts a reveal message for ℓ or k or both in Step 4a (or else P_j would not broadcast consistent as specified in Step 5a).

2. The dealer did not broadcast $\mathsf{reveal}(j, f_j(x), g_j(y))$. (See Step 5(b)i.)

3. Every revealed polynomial fits P_j's polynomials: Whenever the dealer broadcasts a message $\mathsf{reveal}(k, f_k(x), g_k(y))$, it holds that $g_k(\alpha_j) = f_j(\alpha_k)$ and $f_k(\alpha_j) = g_j(\alpha_k)$. (See Step 5(b)ii.)

Since the dealer is honest, whenever there is a conflict between two parties, the dealer will broadcast a reveal message. This is due to the fact that if $u_1 \neq v_2$ or $u_2 \neq v_1$, it cannot hold that both (u_1, v_1) and (u_2, v_2) are consistent with $S(x, y)$ (i.e., it cannot be that $u_1 = S(\alpha_\ell, \alpha_k)$ and $v_1 = S(\alpha_k, \alpha_\ell)$ as well as $u_2 = S(\alpha_k, \alpha_\ell)$ and $v_2 = S(\alpha_\ell, \alpha_k)$). Thus, by its instructions, the dealer will broadcast at least one reveal message, and so condition (1) holds. In addition, it is immediate that since the dealer is honest, condition (3) also holds. Finally, the dealer broadcasts a $\mathsf{reveal}(j, f_j(x), g_j(y))$ message if and only if P_j sends a complaint with an *incorrect* pair (u, v); i.e., P_j broadcast (j, k, u, v) where either $u \neq f_j(\alpha_k)$ or $v \neq g_j(\alpha_k)$. However, since both the dealer and P_j are honest, any complaint sent by P_j will be with the correct (u, v) values. Thus, the dealer will not broadcast a reveal of P_j's polynomials and condition (2) also holds. We conclude that every honest party broadcasts consistent and so all honest parties P_j output $f_j(x), g_j(y)$, as required.

Since the outputs of the honest parties are fully determined by the honest dealer's input, it remains to show the existence of an ideal-model adversary/simulator $\mathcal{SIM}$ that can generate the *view of the adversary* $\mathcal{A}$ in an execution of the real protocol, given only the outputs $f_i(x), g_i(y)$ of the corrupted parties P_i for every $i \in I$.

The simulator $\mathcal{SIM}$:

- *$\mathcal{SIM}$ invokes $\mathcal{A}$ on the auxiliary input z.*
- Interaction with the trusted party: *$\mathcal{SIM}$ receives the output values $\{f_i(x), g_i(y)\}_{i \in I}$.*
- Generating the view of the corrupted parties: *$\mathcal{SIM}$ chooses any polynomial $S'(x,y)$ under the constraint that $S'(x, \alpha_i) = f_i(x)$ and $S'(\alpha_i, y) = g_i(y)$ for every $i \in I$ (Such a polynomial always exists. In particular, $f_i(x), g_i(y)$ are derived from such a bivariate polynomial). Then, $\mathcal{SIM}$ runs all honest parties (including the honest dealer) in an interaction with $\mathcal{A}$, with the dealer input polynomial as $S'(x,y)$.*
- *$\mathcal{SIM}$ outputs whatever $\mathcal{A}$ outputs, and halts.*

We now prove that the distribution generated by $\mathcal{SIM}$ is as required. First, observe that all that the corrupted parties see in the simulation by $\mathcal{SIM}$ is determined by the adversary and the *sequence* of polynomial pairs $\{(f_i(x), g_i(y))\}_{i \in I}$, where $f_i(x)$ and $g_i(y)$ are selected based on $S'(x,y)$, as described in the protocol. In order to see this, note that the only information sent after Round 1 are parties' complaints, complaint resolutions, and consistent messages. However, when the dealer is honest any complaint sent by an honest party P_j can only be due it receiving incorrect (u_i, v_i) from a corrupted party P_i (i.e., where either $u_i \neq f_j(\alpha_i)$ or $v_i \neq g_j(\alpha_i)$ or both). Such a complaint is of the form $(j, i, f_j(\alpha_i), g_j(\alpha_i))$, which equals $(j, i, g_i(\alpha_j), f_i(\alpha_j))$ since the dealer is honest, and so this complaint is determined by $(f_i(x), g_i(x))$ where $i \in I$. In addition, since the honest parties' complaints always contain correct values, the dealer can only send reveal messages reveal$(i, f_i(x), g_i(x))$ where $i \in I$; once again this information is already determined by the polynomial pairs of Round 1. Thus, all of the messages sent by $\mathcal{SIM}$ in the simulation can be computed from the sequence $\{(f_i(x), g_i(y))\}_{i \in I}$ only. Next, observe that the above is also true for a real protocol execution as well. Thus, the only difference between the real and ideal executions is whether the sequence $\{(f_i(x), g_i(y)\}_{i \in I}$ is based on the real polynomial $S(x,y)$ or the simulator-chosen polynomial $S'(x,y)$. However, by the way $S'(x,y)$ is chosen in the simulation, these polynomials are exactly the same. This completes the proof of the case that the dealer is honest.

Case 2 – the Dealer is Corrupted

In this case, the adversary $\mathcal{A}$ controls the dealer. Briefly speaking, the simulator $\mathcal{SIM}$ just plays the role of all honest parties. Recall that all actions of the parties, apart from the dealer, are deterministic and that these parties have no inputs. If the simulated execution is such that the parties output $\perp$, the simulator sends an invalid polynomial (say $S(x,y) = x^{2t}$) to the trusted party. Otherwise, the simulator uses the fact that it sees all "shares" sent by $\mathcal{A}$ to honest parties in order to interpolate and find the polynomial $S(x,y)$, which it then sends to the trusted party computing the functionality. That is, here the simulator invokes the trusted party after simulating an execution of the protocol. We now formally describe the simulator:

The simulator $\mathcal{SIM}$:

1. *$\mathcal{SIM}$ invokes $\mathcal{A}$ on its auxiliary input z.*

2. *$\mathcal{SIM}$ plays the role of all the $n - |I|$ honest parties interacting with $\mathcal{A}$, as specified by the protocol, running until the end.*

3. *Let* num *be the number of (honest and corrupted) parties P_j that broadcast* $\mathsf{consistent}$ *in the simulation:*

   (a) *If* $\mathsf{num} < n - t$, *then $\mathcal{SIM}$ sends the trusted party the polynomial $S'(x,y) = x^{2t}$ as the dealer input (this causes the trusted party to send $\perp$ as output to all parties in the ideal model).*

   (b) *If* $\mathsf{num} \geq n-t$, *then $\mathcal{SIM}$ defines a polynomial $S'(x,y)$ as follows. Let $K \subset [n] \setminus I$ be the set of all honest parties that broadcast* $\mathsf{consistent}$ *in the simulation. $\mathcal{SIM}$ finds the unique degree-t bivariate polynomial S' that is guaranteed to exist by Claim 2.5.3 for this set K (later we will show why Claim 2.5.3 can be used). $\mathcal{SIM}$ sends $S'(x,y)$ to the trusted party (we stress that $S'(x,y)$ is not necessarily equal to the polynomial $S(x,y)$ that the dealer – equivalently P_1 – receives as input).*

   $\mathcal{SIM}$ receives the output $\{q'(\alpha_i)\}_{i \in I}$ of the corrupted parties from the trusted party. (Since these values are already known to $\mathcal{SIM}$, they are not used. Nevertheless, $\mathcal{SIM}$ must send $q'(x)$ to the trusted party since this results in the honest parties receiving their output from F_{VSS}.)

4. *$\mathcal{SIM}$ halts and outputs whatever $\mathcal{A}$ outputs.*

Observe that all parties, as well as the simulator, are *deterministic*. Thus, the outputs of all parties are fully determined both in the real execution of the protocol with $\mathcal{A}$ and in the ideal execution with $\mathcal{SIM}$. We therefore show that the outputs of the adversary and the parties in a real execution with $\mathcal{A}$ are equal to the outputs in an ideal execution with $\mathcal{SIM}$.

First, observe that the simulator plays the role of all the honest parties in an ideal execution, following the exact protocol specification. Since the honest parties have no input, the messages sent by the simulator in the ideal execution are exactly the same as those sent by the honest parties in a real execution of the protocol. Thus, the value that is output by $\mathcal{A}$ in a real execution *equals* the value that is output by $\mathcal{A}$ in the ideal execution with $\mathcal{SIM}$. It remains to show that the outputs of the honest parties are also the same in the real and ideal executions. Let OUTPUT_J denote the outputs of the parties P_j for all $j \in J$. We prove:

Claim 2.5.11 *Let $J = [n] \setminus I$ be the set of indices of the honest parties. For every adversary $\mathcal{A}$ controlling I including the dealer, every polynomial $S(x,y)$ and every auxiliary input $z \in \{0,1\}^*$ for $\mathcal{A}$, it holds that:*

$$\text{OUTPUT}_J\left(\text{REAL}_{\pi,\mathcal{A}(z),I}\left(S(x,y),\lambda,\ldots,\lambda\right)\right) = \text{OUTPUT}_J\left(\text{IDEAL}_{F_{VSS},\mathcal{S}(z),I}\left(S(x,y),\lambda,\ldots,\lambda\right)\right).$$

Proof: Let $\vec{x} = (S(x,y),\lambda,\ldots,\lambda)$ be the vector of inputs. We separately analyze the case that in the *real* execution some honest party outputs $\perp$ and the case where no honest party outputs $\perp$.

Case 1: There exists a $j \in J$ such that $\text{OUTPUT}_j(\vec{x}) = \perp$. We show that in this case all the honest parties output $\perp$ in both the real and ideal executions. Let j be such that $\text{OUTPUT}_j(\text{REAL}_{\pi,\mathcal{A}(z),I}(\vec{x})) = \perp$. By the protocol specification, an honest party P_j outputs $\perp$ (in the real world) if and only if it receives less than $n-t$ "consistent" messages over the broadcast channel. Since these messages are broadcast, it holds that all the parties receive the same messages. Thus, if an honest P_j output $\perp$ in the real execution, then each honest party received less than $n-t$ such "consistent" messages, and so every honest party outputs $\perp$ (in the real execution).

We now claim that in the ideal execution, all honest parties also output $\perp$. The output of the honest parties in the ideal execution are determined by the trusted third party, based on the input sent by $\mathcal{SIM}$. It follows by the specification of $\mathcal{SIM}$ that all honest parties output $\perp$ if and only if $\mathcal{SIM}$ sends x^{2t} to the trusted third party. As we have mentioned, the simulator $\mathcal{SIM}$ follows the instructions of the honest parties exactly in the simulation. Thus, if in a real execution with $\mathcal{A}$ less than $n-t$ parties broadcast consistent, then the same is also true in the simulation with $\mathcal{SIM}$. (We stress that *exactly the same messages* are sent by $\mathcal{A}$ and the honest parties in a real protocol execution and in the simulation with $\mathcal{SIM}$.) Now, by the instructions of $\mathcal{SIM}$, if less than $n-t$ parties broadcast consistent, then $\mathsf{num} < n-t$, and $\mathcal{SIM}$ sends $S(x,y) = x^{2t}$ to the trusted party. We conclude that all honest parties output $\perp$ in the ideal execution as well.

Case 2: For every $j \in J$ it holds that $\text{OUTPUT}_j(\text{REAL}_{\pi,\mathcal{A}(z),I}(\vec{x})) \neq \perp$. By what we have discussed above, this implies that in the simulation with $\mathcal{SIM}$, at least $n-t$ parties broadcast consistent. Since $n \geq 3t+1$ this implies that at least $3t+1-t \geq 2t+1$ parties broadcast consistent. Furthermore, since there are at most t corrupted parties, we have that at least $t+1$ *honest* parties broadcast consistent. Recall that an honest party P_j broadcasts consistent if and only if the following conditions hold (cf. the case of honest dealer):

1. The dealer resolves all conflicts (Step 5a of the protocol).

2. The dealer did not broadcast $\mathsf{reveal}(j, f_j(x), g_j(y))$ (Step 5(b)i of the protocol).

3. Every revealed polynomial fits P_j's polynomials (Step 5(b)ii of the protocol).

Let $K \subset [n]$ be the set of honest parties that broadcast consistent as in Step 3b of $\mathcal{SIM}$. For each of these parties the above conditions hold. Thus, for every $i, j \in K$ it holds that $f_i(\alpha_j) = g_j(\alpha_i)$ and so Claim 2.5.3 can be applied. This implies that there exists a *unique* bivariate polynomial $S'(x,y)$ such that $S'(x,\alpha_k) = f_k(x)$ and $S'(\alpha_k, y) = g_k(y)$ for every $k \in K$. Now, since $\mathcal{SIM}$ sends $S'(x,y)$ to the trusted party in an ideal execution, we have that all honest parties P_j output $S'(x,\alpha_j), S'(\alpha_j, y)$ in an ideal execution. We now prove that the same also holds in a real protocol execution.

We stress that the polynomial $S'(x,y)$ is defined as a deterministic function of the transcript of messages sent by $\mathcal{A}$ in a real or ideal execution. Furthermore, since the execution is deterministic, the exact same polynomial $S'(x,y)$ is defined in both the real and ideal executions. It therefore remains to show that each honest party P_j outputs $S'(x,\alpha_j), S'(\alpha_j,y)$ in a real execution. We first observe that any honest party P_k for $k \in K$ clearly outputs $f_k(x) = S'(x,\alpha_k), g_k(y) = S'(\alpha_k, y)$. This follows from the fact that by the protocol description, each party P_i that does not output $\perp$ outputs these polynomials. Thus, each such P_k outputs $(f_k(x), g_k(y))$.

It remains to show that every honest party P_j for $j \notin K$ also outputs $S'(x, \alpha_j), S'(\alpha_j, y)$; i.e., it remains to show that every honest party P_j who did *not* broadcast consistent also outputs univariate polynomials that are consistent with $S'(x, y)$ defined by the parties that did broadcast consistent. Let $f'_j(x)$ and $g'_j(x)$ be the polynomials that P_j holds after the possible replacement in Step 5(b)i of the protocol (note that these polynomials may be different from the original polynomials that P_j received from the dealer at the first stage). We stress that this party P_j did not broadcast consistent, and therefore we cannot rely on the conditions above. However, for every party P_k ($k \in K$) who broadcast consistent, we are guaranteed that the polynomials $f_k(x)$ and $g_k(y)$ are consistent with the values of the polynomials of P_j; that is, it holds that $f_k(\alpha_j) = g'_j(\alpha_k)$ and $g_k(\alpha_j) = f'_j(\alpha_k)$. This follows from the fact that all conflicts are properly resolved (and so if they were inconsistent then a reveal message must have been sent to make them consistent). This implies that for $t + 1$ points $k \in K$, it holds that $f'_j(\alpha_k) = S'(\alpha_k, \alpha_j)$, and so since $f'_j(x)$ is a polynomial of degree t (by the truncation instruction; see the protocol specification) it follows that $f'_j(x) = S'(x, \alpha_j)$ (because both are degree-t polynomials in x), and $g'_j(y) = S'(\alpha_j, y)$, and thus the honest parties $j \notin K$ also output consistent polynomials on $S'(x, y)$. This completes the proof. ∎

This completes the proof of Theorem 2.5.10. ∎

Efficiency. We remark that in the case that no parties behave maliciously in Protocol 2.5.7, the protocol merely involves the dealer sending two polynomials to each party, and each party sending two field elements to every other party. Specifically, if no party broadcasts a complaint, then the protocol can conclude immediately after Round 3.

2.6 Multiplication in the Presence of Malicious Adversaries

2.6.1 High-Level Overview

In this section, we show how to securely compute shares of the product of shared values, in the presence of a malicious adversary controlling any $t < n/3$ parties. We use the simplification of the original multiplication protocol of [22] that appears in [51]. We start with a short overview of the simplification of [51] in the semi-honest model, and then we show how to move to the malicious case.

Assume that the values on the input wires are a and b, respectively, and that each party holds degree-t shares a_i and b_i. Recall that the values $a_i \cdot b_i$ define a (non random) degree-$2t$ polynomial that hides $a \cdot b$. The semi-honest multiplication protocol of [22] works by first rerandomizing this degree-$2t$ polynomial, and then reducing its degree to degree-t while preserving the constant term which equals $a \cdot b$ (see Section 2.4.3). Recall also that the degree-reduction works by running the BGW protocol for a linear function, where the first step involves each party sharing its input by a degree-t polynomial. In our case, the parties' inputs are themselves shares of a degree-$2t$ polynomial, and thus each party "subshares" its share.

The method of [51] simplifies this protocol by replacing the two different stages of rerandomization and degree-reduction with a single step. The simplification is based on an observation that a specific linear combination of all the subshares of all $a_i \cdot b_i$ defines a *random* degree-t

polynomial that hides $a \cdot b$ (where the randomness of the polynomial is derived from the randomness of the polynomials used to define the subshares). Thus, the protocol involves first subsharing the share-product values $a_i \cdot b_i$, and then carrying out a local linear combination of the obtained subshares.

The main problem and difficulty that arises in the case of malicious adversaries is that corrupted parties may not subshare the correct values $a_i \cdot b_i$. We therefore need a mechanism that forces the corrupted parties to distribute the correct values, without revealing any information. Unfortunately, it is not possible to simply have the parties VSS-subshare their share products $a_i \cdot b_i$ and then use error correction to correct any corrupt values. This is due to the fact that the shares $a_i \cdot b_i$ lie on a degree-$2t$ polynomial, which in turn defines a Reed-Solomon code of parameters $[n, 2t+1, n-2t]$. For such a code, it is possible to correct up to $\frac{n-2t-1}{2}$ errors (see Section 2.5.2); plugging in $n = 3t+1$ we have that it is possible to correct up to $\frac{t}{2}$ errors. However, there are t corrupted parties and so incorrect values supplied by more than half of them cannot be corrected.[5] The BGW protocol therefore forces the parties to distribute correct values, using the following steps:

1. The parties first distribute subshares of their input shares on each wire (rather than the product of their input shares) to all other parties in a verifiable way. That is, each party P_i distributes subshares of a_i and subshares of b_i. Observe that the input shares are points on degree-t polynomials. Thus, these shares constitute a Reed-Solomon code with parameters $[n, t+1, n-t]$ for which it is possible to correct up to t errors. There is therefore enough redundancy to correct errors, and so any incorrect values provided by corrupted parties can be corrected. This operation is carried out using the $F_{VSS}^{subshare}$ functionality, described in Section 2.6.4.

2. Next, each party distributes subshares of the product $a_i \cdot b_i$. The protocol for subsharing the product uses the separate subshares of a_i and b_i obtained in the previous step, in order to verify that the correct product $a_i \cdot b_i$ is shared. Stated differently, this step involves a protocol for verifying that a party distributes shares of $a_i \cdot b_i$ (via a degree-t polynomial), given shares of a_i and shares of b_i (via degree-t polynomials). This step is carried out using the F_{VSS}^{mult} functionality, described in Section 2.6.6. In order to implement this step, we introduce a new functionality called F_{eval} in Section 2.6.5.

3. Finally, after the previous step, all parties verifiably hold (degree-t) subshares of all the products $a_i \cdot b_i$ of every party. As described above, shares of the product $a \cdot b$ can be obtained by computing a linear function of the subshares obtained in the previous step. Thus, each party just needs to carry out a local computation on the subshares obtained. This is described in Section 2.6.7.

Before we show how to securely compute the $F_{VSS}^{subshare}$ functionality, we present relevant preliminaries in Sections 2.6.2 and 2.6.3. Specifically, in Section 2.6.2 we introduce the notion of *corruption-aware functionalities*. These are functionalities whose behavior may depend on

[5]We remark that in the case of $t < n/4$ (i.e., $n \geq 4t+1$), the parties can correct errors directly on degree-$2t$ polynomials. Therefore, the parties can distribute subshares of the products $a_i \cdot b_i$, and correct errors on these shares using (a variant of) the $F_{VSS}^{subshare}$ functionality directly. Thus, overall, the case of $t < n/4$ is significantly simpler, since there is no need for the F_{VSS}^{mult} subprotocol that was mentioned in the second step described above. A full specification of this simplification is described in Section 2.9; the description assumes familiarity with the material appearing in Sections 2.6.2, 2.6.3, 2.6.4 and 2.6.7, and therefore should be read after these sections.

which parties are corrupted. We use this extension of standard functionalities in order to prove the BGW protocol in a modular fashion. Next, in Section 2.6.3 we present a subprotocol for securely computing matrix multiplication over a shared vector. This will be used in the protocol for securely computing $F_{VSS}^{subshare}$, which appears in Section 2.6.4.

2.6.2 Corruption-Aware Functionalities and Their Use

In the standard definition of secure computation (see Section 2.2.2 and [27, 53]) the functionality defines the desired input/output behavior of the computation. As such, it merely receives inputs from the parties and provides outputs. However, in some cases, we wish to provide the corrupted parties, equivalently the adversary, with some additional power over the honest parties.

In order to see why we wish to do this, consider the input sharing phase of the BGW protocol, where each party distributes its input using secret sharing. This is achieved by running n executions of VSS where in the ith copy party P_i plays the dealer with a polynomial $q_i(x)$ defining its input. The question that arises now is what security is obtained when running these VSS invocations in *parallel*, and in particular we need to define *the ideal functionality that such parallel VSS executions fulfills.* Intuitively, the security of the VSS protocol guarantees that all shared values are independent. Thus, one could attempt to define the "parallel VSS" functionality as follows:

FUNCTIONALITY 2.6.1 (Parallel VSS (naive attempt) – F_{VSS}^n)

1. The parallel F_{VSS}^n functionality receives inputs $q_1(x), \ldots, q_n(x)$ from parties $P_1, \ldots, P_n$, respectively. If P_i did not send a polynomial $q_i(x)$, or $\deg(q_i) > t$, then F_{VSS}^n defines $q_i(x) = \bot$ for every x.
2. For every $i = 1, \ldots, n$, the functionality F_{VSS}^n sends $(q_1(\alpha_i), \ldots, q_n(\alpha_i))$ to party P_i.

This is the naive extension of the single F_{VSS} functionality (Functionality 2.5.5), and at first sight seems to be the appropriate ideal functionality for a protocol consisting of n *parallel* executions of Protocol 2.5.7 for computing F_{VSS}. However, we now show that this protocol does not securely compute the parallel VSS functionality as defined.

Recall that the adversary is *rushing*, which means that it can receive the honest parties' messages in a given round before sending its own. In this specific setting, the adversary can see the corrupted parties' shares of the honest parties' polynomials before it chooses the corrupted parties' input polynomials (since these shares of the honest parties' polynomials are all sent to the corrupted parties in the first round of Protocol 2.5.7). Thus, the adversary can choose the corrupted parties' polynomials in a way that is related to the honest parties' polynomials. To be specific, let P_j be an honest party with input $q_j(x)$, and let P_i be a corrupted party. Then, the adversary can first see P_i's share $q_j(\alpha_i)$, and then choose $q_i(x)$ so that $q_i(\alpha_i) = q_j(\alpha_i)$, for example. In contrast, the adversary in the ideal model with F_{VSS}^n *cannot* achieve this effect since it receives no information about the honest parties' polynomials before all input polynomials, including those of the corrupted parties, are sent to the trusted party. Thus, n parallel executions of Protocol 2.5.7 does *not* securely compute F_{VSS}^n as defined in Functionality 2.6.1.

Despite the above, we stress that in many cases (and, in particular, in the application of parallel VSS in the BGW protocol) this adversarial capability is of no real concern. Intuitively, this is due to the fact that $q_j(\alpha_i)$ is actually independent of the constant term $q_j(0)$ and so making $q_i(\alpha_i)$ depend on $q_j(\alpha_i)$ is of no consequence in this application. Nevertheless, the adversary *can* set $q_i(x)$ in this way in the real protocol (due to rushing), but *cannot do so* in the ideal model with functionality F_{VSS}^n (as in Functionality 2.6.1). Therefore, the protocol consisting of n parallel calls to F_{VSS} does *not* securely compute the F_{VSS}^n functionality. Thus, one has to either modify the protocol or change the functionality definition, or both. Observe that the fact that in some applications we don't care about this adversarial capability is immaterial: The problem is that the protocol does not securely compute Functionality 2.6.1 and thus something has to be changed.

One possible modification to both the protocol and functionality is to run the F_{VSS} executions sequentially in the real protocol and define an ideal (reactive) functionality where each party P_i first receives its shares $q_1(\alpha_i), \ldots, q_{i-1}(\alpha_i)$ from the previous VSS invocations before sending its own input polynomial $q_i(x)$. This solves the aforementioned problem since the ideal (reactive) functionality allows each party to make its polynomial depend on shares previously received. However, this results in a protocol that is not constant round, which is a significant disadvantage.

Another possible modification is to leave the protocol unmodified (with n parallel calls to F_{VSS}), and change the ideal functionality as follows. First, the *honest* parties send their input polynomials $q_j(x)$ (for every $j \notin I$). Next, the corrupted parties receive their shares on these polynomials (i.e., $q_j(\alpha_i)$ for every $j \notin I$ and $i \in I$), and finally the corrupted parties send their polynomials $q_i(x)$ (for every $i \in I$) to the trusted party. This reactive functionality captures the capability of the adversary to choose the corrupted parties' polynomials based on the shares $q_j(\alpha_i)$ that it views on the honest parties' polynomials, but nothing more. Formally, we define:

FUNCTIONALITY 2.6.2 (Corruption-aware parallel VSS – F_{VSS}^n)

F_{VSS}^n receives a set of indices $I \subseteq [n]$ and works as follows:

1. F_{VSS}^n receives an input polynomial $q_j(x)$ from every *honest* P_j ($j \notin I$).
2. F_{VSS}^n sends the (ideal model) adversary the corrupted parties' shares $\{q_j(\alpha_i)\}_{j \notin I}$ for every $i \in I$, based on the honest parties' polynomials.
3. F_{VSS}^n receives from the (ideal model) adversary an input polynomial $q_i(x)$ for every $i \in I$.
4. F_{VSS}^n sends the shares $(q_1(\alpha_j), \ldots, q_n(\alpha_j))$ to every party P_j ($j = 1, \ldots, n$).[6]

This modification to the definition of F_{VSS}^n solves our problem. However, the standard definition of security, as referred in Section 2.2.2, does not allow us to define a functionality in this way. This is due to the fact that the standard formalism does not distinguish between honest and malicious parties. Rather, the functionality is supposed to receive inputs from each honest and corrupt party in the same way, and in particular does not "know" which parties are corrupted. We therefore augment the standard formalism to allow corruption-aware functionalities (CA functionalities) that *receive the set I of the identities of the corrupted parties as additional auxiliary input* when invoked. We proceed by describing the changes required to the standard (stand-alone) definition of security of Section 2.2.2 in order to incorporate corruption awareness.

[6]It actually suffices to send the shares $(q_1(\alpha_j), \ldots, q_n(\alpha_j))$ only to parties P_j for $j \notin I$ since all other parties have already received these values. Nevertheless, we present it in this way for the sake of clarity.

Definition. The formal definition of security for a corruption-aware functionality is the same as Definition 2.2.3 with the sole change being that f is a function of the subset of corrupted parties and the inputs; formally, $f : 2^{[n]} \times (\{0,1\}^*)^n \to (\{0,1\}^*)^n$. We denote by $f_I(\vec{x}) = f(I, \vec{x})$ the function f with the set of corrupted parties fixed to $I \subset [n]$. Then, we require that for every subset I (of cardinality at most t), the distribution $\text{IDEAL}_{f_I,\mathcal{S}(z),I}(\vec{x})$ is distributed identically to $\text{REAL}_{\pi,\mathcal{A}(z),I}(\vec{x})$. We stress that in the ideal model, the subset I that is given to a corruption-aware functionality as auxiliary input (upon initialization) is the same subset I of corrupted parties that the adversary controls. Moreover, the functionality receives this subset I at the very start of the ideal process, in the exact same way as the (ideal model) adversary receives the auxiliary input z, the honest parties receive their inputs, and so on. We also stress that the honest parties (both in the ideal and real models) do *not* receive the set I, since this is something that is of course not known in reality (and so the security notion would be nonsensical). Formally,

Definition 2.6.3 *Let $f : 2^{[n]} \times (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a* corruption-aware *n-ary functionality and let π be a protocol. We say that π is t-*secure for *f if for every probabilistic adversary $\mathcal{A}$ in the real model, there exists a probabilistic adversary $\mathcal{S}$ of comparable complexity in the ideal model, such that for every $I \subset [n]$ of cardinality at most t, every $\vec{x} \in (\{0,1\}^*)^n$ where $|x_1| = \ldots = |x_n|$, and every $z \in \{0,1\}^*$, it holds that: $\left\{\text{IDEAL}_{f_I,\mathcal{S}(z),I}(\vec{x})\right\} \equiv \left\{\text{REAL}_{\pi,\mathcal{A}(z),I}(\vec{x})\right\}$.*

We stress that since we only consider static adversaries here, the set I is fully determined before the execution begins, and thus this is well defined.

This idea of having the behavior of the functionality depend on the adversary and/or the identities of the corrupted parties was introduced by [28] in order to provide more flexibility in defining functionalities, and is heavily used in the universal composability framework.[7]

The hybrid model and modular composition. In the hybrid model, where the parties have oracle tapes for some ideal functionality (trusted party), in addition to regular communication tapes, the same convention for corruption awareness is followed as in the ideal model. Specifically, an execution in the $\mathcal{G}_I$-hybrid model, denoted $\text{HYBRID}^{\mathcal{G}_I}_{f,\mathcal{A}(z),I}(\vec{x})$, is parameterized by the set I of corrupted parties, and this set I is given to functionality $\mathcal{G}$ upon initialization of the system just like the auxiliary input is given to the adversary. As mentioned above, I is fixed ahead of time and so this is well-defined. We stress again that the honest parties do not know the set of indices I, and real messages sent by honest parties and their input to the ideal functionality are independent of I.

In more detail, in an ideal execution the behavior of the trusted party depends heavily on the set of corrupted parties I, and in some sense, its exact code is fixed *only after* we determine the set of corrupted parties I. In contrast, in a real execution the specification of the protocol is independent of the set I, and the code that the honest parties execute is fixed ahead of time and is the same one *for any* set of corrupted parties I. An execution in the hybrid model is something in between: the code of the honest parties is independent of I and is fixed ahead of

[7]In the UC framework, the adversary can communicate directly with the ideal functionality and it is mandated that the adversary notifies the ideal functionality (i.e., trusted party) of the identities of all corrupted parties. Furthermore, ideal functionalities often utilize this information (i.e., they are corruption aware) since the way that the universal composability framework is defined typically requires functionalities to treat the inputs of honest and corrupted parties differently. See Section 6 of the full version of [28] for details.

time (like in the real model); however, the code of the aiding functionality is fixed only after we set I (as in the ideal model).

In our proof below, some of the functionalities are corruption aware and some are not; in particular, as we will describe, our final functionality for secure computation with the BGW protocol is *not* corruption aware. In order to be consistent with respect to the definition, we work with corruption-aware functionalities only and remark that *any* ordinary functionality f (that is *not* corruption aware) can be rewritten as a fictitiously corruption-aware functionality f_I where the functionality just ignores the auxiliary input I. An important observation is that a protocol that securely computes this fictitiously corruption-aware functionality, securely computes the original functionality in the standard model (i.e., when the functionality does not receive the set I as an auxiliary input). This holds also for protocols that use corruption-aware functionalities as subprotocols (as we will see, this is the case with the final BGW protocol). This observation relies on the fact that a protocol is always corruption unaware, and that the simulator knows the set I in *both* the corruption aware and the standard models. Thus, the simulator is able to simulate the corruption-aware subprotocol, even in the standard model. Indeed, since the corruption-aware functionality f_I ignores the set I, and since the simulator knows I in both models, the two ensembles $\text{IDEAL}_{f_I,\mathcal{S}(z),I}(\vec{x})$ (in the corruption-aware model) and $\text{IDEAL}_{f,\mathcal{S}(z),I}(\vec{x})$ (in the standard model) are identical. Due to this observation, we are able to conclude that the resulting BGW protocol securely computes any standard (not corruption aware) functionality in the *standard model*, even though it uses corruption-aware subprotocols.

Regarding composition, the sequential modular composition theorems of [27, 53] do not consider corruption-aware functionalities. Nevertheless, it is straightforward to see that the proofs hold also for this case, with no change whatsoever. Thus, the method described in Section 2.2.3 for proving security in a modular way can be used with corruption-aware functionalities as well.

Discussion. The augmentation of the standard definition with corruption-aware functionalities enables more flexibility in protocol design. Specifically, it is possible to model the situation where corrupted parties can learn more than just the specified output, or can obtain some other "preferential treatment" (like in the case of parallel VSS where they are able to set their input polynomials as a partial function of the honest parties' input). In some sense, this implies a weaker security guarantee than in the case where all parties (honest and corrupted) receive the same treatment. However, since the ideal functionality is specified so that the "weakness" is explicitly stated, the adversary's advantage is well defined.

This approach is not foreign to modern cryptography and has been used before. For example, secure encryption is defined while allowing the adversary a negligible probability of learning information about the plaintext. A more significant example is the case of two-party secure computation. In this case, the ideal model is defined so that the corrupted party explicitly receives the output first and can then decide whether or not the honest party also receives output. This is weaker than an ideal model in which both parties receive output and so "complete fairness" is guaranteed. However, since complete fairness cannot be achieved (in general) without an honest majority, this weaker ideal model is used, and the security weakness is explicitly modeled.

In the context of this paper, we use corruption awareness in order to enable a modular analysis of the BGW protocol. In particular, for some of the subprotocols used in the BGW protocol, it seems hard to define an appropriate ideal functionality that is not corruption aware. Nevertheless, our final result regarding the BGW protocol is for standard functionalities. That is, when we state that *every functionality* can be securely computed by BGW (with the appropriate corruption threshold), we refer to regular functionalities and not to corruption-aware ones.

The reason why the final BGW protocol works for corruption unaware functionalities *only* is due to the fact that the protocol emulates the computation of a *circuit* that computes the desired functionality. However, not every corruption-aware functionality can be computed by a circuit that receives inputs from the parties only, without also having the identities of the set of corrupted parties as auxiliary input. Since the real protocol is never allowed to be "corruption aware", this means that such functionalities cannot be realized by the BGW protocol. We remark that this is in fact *inherent*, and there exist corruption-aware functionalities that cannot be securely computed by *any* protocol. In particular, consider the functionality that just announces to all parties who is corrupted. Since a corrupted party may behave like an honest one, it is impossible to securely compute such a functionality.

Finally, we note that since we already use corruption awareness anyhow in our definitions of functionalities (for the sake of feasibility and/or efficiency), we sometimes also use it in order to simplify the definition of a functionality. For example, consider a secret sharing reconstruction functionality. As we have described in Section 2.5.2, when $t < n/3$, it is possible to use Reed-Solomon error correction to reconstruct the secret, even when up to t incorrect shares are received. Thus, an *ideal* functionality for reconstruction can be formally defined by having the trusted party run the Reed-Solomon error correction procedure. Alternatively, we can define the ideal functionality so that it receive shares from the *honest parties* only, and reconstructs the secret based on these shares only (which are guaranteed to be correct). This latter formulation is corruption aware, and has the advantage of making it clear that the adversary cannot influence the outcome of the reconstruction in any way.

Convention. For the sake of clarity, we will describe (corruption-aware) functionalities as having direct communication with the (ideal) adversary. In particular, the corrupted parties will not send input or receive output, and all such communication will be between the adversary and functionality. This is equivalent to having the corrupted parties send input as specified by the adversary.

Moreover, we usually omit the set of corrupted parties I in the notation of a corruption-aware functionality (i.e., we write $\mathcal{G}$ instead of $\mathcal{G}_I$). However, in the definition of any corruption-aware functionality we add an explicit note that the functionality receives as auxiliary input the set of corrupted parties I. In addition, for any protocol in the corruption-aware hybrid model, we add an "aiding ideal-functionality initialization" step, to explicitly emphasize that the aiding ideal functionalities receive the set I upon initialization.

2.6.3 Matrix Multiplication in the Presence of Malicious Adversaries

We begin by showing how to securely compute the matrix-multiplication functionality, that maps the input vector $\vec{x}$ to $\vec{x} \cdot A$ for a fixed matrix A, where the ith party holds x_i and all parties receive the entire vector $\vec{x} \cdot A$ for output. Beyond being of interest in its own right, this serves as a good warm-up to secure computation in the malicious setting. In addition, we will explicitly use this as a subprotocol in the computation of $F_{VSS}^{subshare}$ in Section 2.6.4.

The basic matrix-multiplication functionality is defined by a matrix $A \in \mathbb{F}^{n \times m}$, and the aim of the parties is to securely compute the length-m vector $(y_1, \ldots, y_m) = (x_1, \ldots, x_n) \cdot A$, where $x_1, \ldots, x_n \in \mathbb{F}$ are their respective inputs. We will actually need to define something more involved, but we begin by explaining how one can securely compute the basic functionality. Note first that matrix multiplication is a linear functionality (i.e., it can be computed by circuits containing only addition and multiplication-by-constant gates). Thus, we can use the same methodology as was described at the end of Section 2.4.2 for privately computing any linear functionality, in the semi-honest model. Specifically, the inputs are first shared. Next, each party locally computes the linear functionality on the shares it received. Finally, the parties send their resulting shares in order to reconstruct the output. The difference here in the malicious setting is simply that the *verifiable* secret sharing functionality is used for sharing the inputs, and Reed-Solomon decoding (as described in Section 2.5.2) is used for reconstructing the output. Thus, the basic matrix multiplication functionality can be securely computed as follows:

1. *Input sharing phase:* Each party P_i chooses a random polynomial $g_i(x)$ under the constraint that $g_i(0) = x_i$. Then, P_i shares its polynomial $g_i(x)$ using the ideal F_{VSS} functionality. After all polynomials are shared, party P_i has the shares $g_1(\alpha_i), \ldots, g_n(\alpha_i)$.

2. *Matrix multiplication emulation phase:* Given the shares from the previous step, each party computes its Shamir-share of the output vector of the matrix multiplication by computing $\vec{y}^i = (g_1(\alpha_i), \ldots, g_n(\alpha_i)) \cdot A$. Note that:

$$\vec{y}^i = (g_1(\alpha_i), \ldots, g_n(\alpha_i)) \cdot A = [g_1(\alpha_i), g_2(\alpha_i), \ldots, g_n(\alpha_i)] \cdot \begin{bmatrix} a_{1,1} & \ldots & a_{1,m} \\ a_{2,1} & \ldots & a_{2,m} \\ \vdots & & \vdots \\ a_{n,1} & \ldots & a_{n,m} \end{bmatrix}$$

   and so the jth element in $\vec{y}^i$ equals $\sum_{\ell=1}^{n} g_\ell(\alpha_i) \cdot a_{\ell,j}$. Denoting the jth element in $\vec{y}^i$ by y_j^i, we have that $y_j^1, \ldots, y_j^n$ are Shamir-shares of the jth element of $\vec{y} = (g_1(0), \ldots, g_n(0)) \cdot A$.

3. *Output reconstruction phase:*

   (a) Each party P_i sends its vector $\vec{y}^i$ to all other parties.

   (b) Each party P_i reconstructs the secrets from all the shares received, thereby obtaining $\vec{y} = (g_1(0), \ldots, g_n(0)) \cdot A$. This step involves running (local) error correction on the shares, in order to neutralize any incorrect shares sent by the malicious parties. Observe that the vectors sent in the protocol constitute the rows in the matrix

$$\begin{bmatrix} \leftarrow & \vec{y}^1 & \rightarrow \\ \leftarrow & \vec{y}^2 & \rightarrow \\ & \vdots & \\ \leftarrow & \vec{y}^n & \rightarrow \end{bmatrix} = \begin{bmatrix} \sum_{\ell=1}^{n} g_\ell(\alpha_1) \cdot a_{\ell,1} & \cdots & \sum_{\ell=1}^{n} g_\ell(\alpha_1) \cdot a_{\ell,m} \\ \sum_{\ell=1}^{n} g_\ell(\alpha_2) \cdot a_{\ell,1} & \cdots & \sum_{\ell=1}^{n} g_\ell(\alpha_2) \cdot a_{\ell,m} \\ \vdots & & \vdots \\ \sum_{\ell=1}^{n} g_\ell(\alpha_n) \cdot a_{\ell,1} & \cdots & \sum_{\ell=1}^{n} g_\ell(\alpha_n) \cdot a_{\ell,m} \end{bmatrix}$$

   and the jth *column* of the matrix constitutes Shamir-shares on the polynomial with constant term $\sum_{\ell=1}^{n} g_\ell(0) \cdot a_{j,\ell}$, which is the jth element in the output. Thus, Reed-Solomon error correction can be applied to the columns in order to correct any incorrect shares and obtain the correct output.

The above protocol computes the correct output: The use of F_{VSS} in the first step prevents any malicious corrupted party from sharing an invalid polynomial, while the use of error correction in the last step ensures that the corrupted parties cannot adversely influence the output.

However, as we have mentioned, we need matrix multiplication in order to secure compute the $F_{VSS}^{subshare}$ functionality in Section 2.6.4. In this case, the functionality that is needed is a little more involved than basic matrix multiplication. First, instead of each party P_i inputting a value x_i, we need its input to be a degree-t polynomial $g_i(x)$ and the constant term $g_i(0)$ takes the place of x_i.[8] Next, In addition to obtaining the result $\vec{y} = (g_1(0), \ldots, g_n(0)) \cdot A$ of the matrix multiplication, each party P_i also outputs the shares $g_1(\alpha_i), \ldots, g_n(\alpha_i)$ that it received on the input polynomials of the parties. Based on the above, one could define the functionality as

$$F_{mat}^{A}(g_1, \ldots, g_n) = \bigg((\vec{y}, \{g_\ell(\alpha_1)\}_{\ell=1}^{n}), (\vec{y}, \{g_\ell(\alpha_2)\}_{\ell=1}^{n}) \ldots, (\vec{y}, \{g_\ell(\alpha_n)\}_{\ell=1}^{n}) \bigg),$$

where $\vec{y} = (g_1(0), \ldots, g_n(0)) \cdot A$. Although this looks like a very minor difference, as we shall see below, it significantly complicates things. In particular, we will need to define a corruption aware variant of this functionality.

We now explain why inputting polynomials $g_1(x), \ldots, g_n(x)$ rather than values $x_1, \ldots, x_n$ (and likewise outputting the shares) makes a difference. In the protocol that we described above for matrix multiplication, each party P_i sends its shares $\vec{y}^i$ of the output. Now, the vectors $\vec{y}^1, \ldots, \vec{y}^n$ are *fully determined* by the input polynomials $g_1(x), \ldots, g_n(x)$. However, in the ideal execution, the simulator only receives a subset of the shares and cannot simulate all of them. (Note that the simulator cannot generate random shares since the $\vec{y}^i$ vectors are fully and deterministically determined by the input.) To be concrete, consider the case that only party P_1 is corrupted. In this case, the ideal adversary receives as output $\vec{y} = (g_1(0), \ldots, g_n(0)) \cdot A$ and the shares $g_1(\alpha_1), \ldots, g_n(\alpha_1)$. In contrast, the real adversary sees all of the vectors $\vec{y}^2, \ldots, \vec{y}^n$ sent by the honest parties in the protocol. Now, these vectors are a *deterministic* function of the input polynomials $g_1(x), \ldots, g_n(x)$ and of the fixed matrix A. Thus, the simulator in the ideal model must be able to generate the *exact* messages sent by the honest parties (recall that the distinguisher knows all of the inputs and outputs and so can verify that the output transcript is truly consistent with the inputs). However, it is impossible for a simulator who is given only $\vec{y}$ and the shares $g_1(\alpha_1), \ldots, g_n(\alpha_1)$ to generate these exact messages, since it doesn't have enough information. In an extreme example, consider the case that $m = n$, the matrix A is the identity matrix, and the honest parties' polynomials are random. In this case, $\vec{y}^i = (g_1(\alpha_i), \ldots, g_n(\alpha_i))$. By the properties of random polynomials, the simulator cannot generate $\vec{y}^i$ for $i \neq 1$ given only $\vec{y} = (g_1(0), \ldots, g_n(0))$, the shares $(g_1(\alpha_1), \ldots, g_n(\alpha_1))$ and the polynomial $g_1(x)$.

One solution to the above is to modify the protocol by somehow adding randomness, thereby making the $\vec{y}^i$ vectors not a deterministic function of the inputs. However, this would add complexity to the protocol and turns out to be unnecessary. Specifically, we only construct this protocol for its use in securely computing $F_{VSS}^{subshare}$, and the security of the protocol for computing $F_{VSS}^{subshare}$ is maintained even if the adversary explicitly learns the vector of m polynomials $\vec{Y}(x) = (Y_1(x), \ldots, Y_m(x)) = (g_1(x), \ldots, g_n(x)) \cdot A$. (Denoting the jth column of A by $(a_{1,j}, \ldots, a_{n,j})^T$, we have that $Y_j(x) = \sum_{\ell=1}^{n} g_\ell(x) \cdot a_{\ell,j}$.) We therefore modify the functionality definition so that the adversary receives $\vec{Y}(x)$, thereby making it corruption aware (observe that the basic output $(g_1(0), \ldots, g_n(0)) \cdot A$ is given by $\vec{Y}(0)$). Importantly, given this additional information, it is possible to simulate the protocol based on the methodology described above

[8]This is needed because in $F_{VSS}^{subshare}$ the parties need to output $g_i(x)$ and so need to know it. It would be possible to have the functionality choose $g_i(x)$ and provide it in the output, but then exactly the same issue would arise.

(VSS sharing, local computation, and Reed-Solomon reconstruction), and prove its security.

Before formally defining the F^A_{mat} functionality, we remark that we also use corruption awareness in order to deal with the fact that the first step of the protocol for computing F^A_{mat} involves running parallel VSS invocations, one for each party to distribute shares of its input polynomial. As we described in Section 2.6.2 this enables the adversary to choose the corrupted parties' polynomials $g_i(x)$ (for $i \in I$) after seeing the corrupted parties' shares on the honest parties' polynomials (i.e., $g_j(\alpha_i)$ for every $j \notin I$ and $i \in I$). We therefore model this capability in the functionality definition.

FUNCTIONALITY 2.6.4 (Functionality F^A_{mat} for matrix multiplication, with $A \in \mathbb{F}^{n\times m}$)

The F^A_{mat}-functionality receives as input a set of indices $I \subseteq [n]$ and works as follows:

1. F^A_{mat} receives the inputs of the honest parties $\{g_j(x)\}_{j\notin I}$; if a polynomial $g_j(x)$ is not received or its degree is greater than t, then F^A_{mat} resets $g_j(x) = 0$.
2. F^A_{mat} sends shares $\{g_j(\alpha_i)\}_{j\notin I; i\in I}$ to the (ideal) adversary.
3. F^A_{mat} receives the corrupted parties' polynomials $\{g_i(x)\}_{i\in I}$ from the (ideal) adversary; if a polynomial $g_i(x)$ is not received or its degree is greater than t, then F^A_{mat} resets $g_i(x) = 0$.
4. F^A_{mat} computes $\vec{Y}(x) = (Y_1(x), \ldots, Y_m(x)) = (g_1(x), \ldots, g_n(x)) \cdot A$.
5. (a) For every $j \notin I$, functionality F^A_{mat} sends party P_j the entire length-m vector $\vec{y} = \vec{Y}(0)$, together with P_j's shares $(g_1(\alpha_j), \ldots, g_n(\alpha_j))$ on the input polynomials.
   (b) In addition, functionality F^A_{mat} sends the (ideal) adversary its output: the vector of polynomials $\vec{Y}(x)$, and the corrupted parties' outputs ($\vec{y}$ together with $(g_1(\alpha_i), \ldots, g_n(\alpha_i))$, for every $i \in I$).

We have already described the protocol intended to securely compute Functionality 2.6.4 and motivated its security. We therefore proceed directly to the formal description of the protocol (see Protocol 2.6.5) and its proof of security. We recall that since all our analysis is performed in the corruption-aware model, we describe the functionality in the corruption-aware hybrid model. Thus, although the F_{VSS} functionality (Functionality 2.5.5) is a standard functionality, we refer to it as a "fictitiously corruption-aware" functionality, as described in Section 2.6.2.

The figure below illustrates Step 5 of Protocol 2.6.5. Each party receives a vector from every other party. These vectors (placed as rows) all form a matrix, whose columns are at most distance t from codewords who define the output.

$$\begin{bmatrix} \leftarrow & \vec{Y}(\alpha_1) & \rightarrow \\ \leftarrow & \vec{Y}(\alpha_2) & \rightarrow \\ & \vdots & \\ \leftarrow & \vec{Y}(\alpha_n) & \rightarrow \end{bmatrix} = \begin{bmatrix} \hat{Y}_1(\alpha_1) & \cdots & \boldsymbol{\hat{Y}_k(\alpha_1)} & \cdots & \hat{Y}_m(\alpha_1) \\ \hat{Y}_1(\alpha_2) & \cdots & \boldsymbol{\hat{Y}_k(\alpha_2)} & \cdots & \hat{Y}_m(\alpha_2) \\ \vdots & & \vdots & & \vdots \\ \hat{Y}_1(\alpha_n) & \cdots & \boldsymbol{\hat{Y}_k(\alpha_n)} & \cdots & \hat{Y}_m(\alpha_n) \end{bmatrix}$$

Figure 2.1: The vectors received by P_i form a matrix; error correction is run on the *columns*.

PROTOCOL 2.6.5 (Securely computing F_{mat}^A in the F_{VSS}-hybrid model)

- **Inputs:** Each party P_i holds a polynomial $g_i(x)$.
- **Common input:** A field description $\mathbb{F}$, n distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$, and a matrix $A \in \mathbb{F}^{n \times m}$.
- **Aiding ideal functionality initialization:** Upon invocation, the trusted party computing the (fictitiously corruption-aware) functionality F_{VSS} is given the set of corrupted parties I.
- **The protocol:**
  1. Each party P_i checks that its input polynomial is of degree-t; if not, it resets $g_i(x) = 0$. It then invokes the F_{VSS} functionality as dealer with $g_i(x)$ as its private input.
  2. At the end of Step 1, each party P_i holds the values $g_1(\alpha_i), \ldots, g_n(\alpha_i)$. If any value equals $\perp$, then P_i replaces it with 0.
  3. Denote $\vec{x}^i = (g_1(\alpha_i), \ldots, g_n(\alpha_i))$. Then, each party P_i locally computes $\vec{y}^i = \vec{x}^i \cdot A$ (equivalently, for every $k = 1, \ldots, m$, each P_i computes $Y_k(\alpha_i) = \sum_{\ell=1}^n g_\ell(\alpha_i) \cdot a_{\ell,k}$ where $(a_{1,k}, \ldots, a_{n,k})^T$ is the kth column of A, and stores $\vec{y}^i = (Y_1(\alpha_i), \ldots, Y_m(\alpha_i))$).
  4. Each party P_i sends $\vec{y}^i$ to every P_j ($1 \leq j \leq n$).
  5. For every $j = 1, \ldots, n$, denote the vector received by P_i from P_j by $\hat{\vec{Y}}(\alpha_j) = (\hat{Y}_1(\alpha_j), \ldots, \hat{Y}_m(\alpha_j))$. (If any value is missing, it replaces it with 0. We stress that different parties may hold different vectors if a party is corrupted.) Each P_i works as follows:
     - For every $k = 1, \ldots, m$, party P_i locally runs the Reed-Solomon decoding procedure (with $d = 2t + 1$) on the possibly corrupted codeword $(\hat{Y}_k(\alpha_1), \ldots, \hat{Y}_k(\alpha_n))$ to get the codeword $(Y_k(\alpha_1), \ldots, Y_k(\alpha_n))$; see Figure 2.1. It then reconstructs the polynomial $Y_k(x)$ and computes $y_k = Y_k(0)$.
- **Output:** P_i outputs $(y_1, \ldots, y_m)$ as well as the shares $g_1(\alpha_i), \ldots, g_n(\alpha_i)$.

Theorem 2.6.6 *Let $t < n/3$. Then, Protocol 2.6.5 is t-secure for the F_{mat}^A functionality in the F_{VSS}-hybrid model, in the presence of a static malicious adversary.*

Proof: We begin by describing the simulator $\mathcal{S}$. The simulator $\mathcal{S}$ interacts externally with the trusted party computing F_{mat}^A, and internally invokes the (hybrid model) adversary $\mathcal{A}$, hence simulating an execution of Protocol 2.6.5 for $\mathcal{A}$. As such, $\mathcal{S}$ has external communication with the trusted party computing F_{mat}^A, and internal communication with the real adversary $\mathcal{A}$. As part of the internal communication with $\mathcal{A}$, the simulator hands $\mathcal{A}$ messages that $\mathcal{A}$ expects to see from the honest parties in the protocol execution. In addition, $\mathcal{S}$ simulates the interaction of $\mathcal{A}$ with the ideal functionality F_{VSS} and hands it the messages it expects to receives from F_{VSS} in Protocol 2.6.5. $\mathcal{S}$ works as follows:

1. *$\mathcal{S}$ internally invokes $\mathcal{A}$ with the auxiliary input z.*

2. *After the honest parties send their inputs to the trusted party computing F_{mat}^A, $\mathcal{S}$ receives shares $\{g_j(\alpha_i)\}_{j \notin I, i \in I}$ on its (external) incoming communication tape from F_{mat}^A (see Step 2 in Functionality 2.6.4).*

3. *$\mathcal{S}$ internally simulates the ideal F_{VSS} invocations for the honest dealers (Step 1 in Protocol 2.6.5): For every $j \notin I$, $\mathcal{S}$ simulates the invocation of F_{VSS} where P_j plays as the dealer, and sends to the adversary $\mathcal{A}$ the shares $\{g_j(\alpha_i)\}_{i \in I}$ (where $g_j(\alpha_i)$ is the appropriate value received from F^A_{mat} in the previous step).*

4. *$\mathcal{S}$ internally simulates the F_{VSS} invocations for the corrupted dealers (Step 1 in Protocol 2.6.5): For every $i \in I$, the simulator $\mathcal{S}$ receives from $\mathcal{A}$ the polynomial $g_i(x)$ that $\mathcal{A}$ sends as input to the F_{VSS} functionality with P_i as dealer.*

   (a) *If $\deg(g_i(x)) \leq t$, then $\mathcal{S}$ stores $g_i(x)$ and internally hands $\mathcal{A}$ the output $\{g_i(\alpha_k)\}_{k \in I}$ that $\mathcal{A}$ expects to receive as the corrupted parties' outputs from this F_{VSS} invocation.*

   (b) *If $\deg(g_i(x)) > t$ or was not sent by $\mathcal{A}$, then $\mathcal{S}$ replaces it with the constant polynomial $g_i(x) = 0$, and internally hands $\mathcal{A}$ the output $\perp$ that $\mathcal{A}$ expects to receive as the corrupted parties' outputs from this F_{VSS} invocation.*

5. *$\mathcal{S}$ externally sends the trusted party computing F^A_{mat} the polynomials $\{g_i(x)\}_{i \in I}$ as the inputs of the corrupted parties (see Step 3 in Functionality 2.6.4).*

6. *At this point, the functionality F^A_{mat} has all the parties' inputs, and so it computes the vector of polynomials $\vec{Y}(x) = (g_1(x), \ldots, g_n(x)) \cdot A$, and $\mathcal{S}$ receives back the following output from F^A_{mat} (see Step 5 in Functionality 2.6.4):*

   (a) *The vector of polynomials $\vec{Y}(x) = (g_1(x), \ldots, g_n(x)) \cdot A$,*

   (b) *The output vector $\vec{y} = (y_1, \ldots, y_m)$, and*

   (c) *The shares $(g_1(\alpha_i), \ldots, g_n(\alpha_i))$ for every $i \in I$.*

7. *For every $j \notin I$ and $i \in I$, $\mathcal{S}$ internally hands the adversary $\mathcal{A}$ the vector $\vec{y^j} = (Y_1(\alpha_j), \ldots, Y_m(\alpha_j))$ as the vector that honest party P_j sends to all other parties (in Step 4 of Protocol 2.6.5).*

8. *$\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.*

We now prove that for every $I \subset [n]$ with $|I| \leq t$:

$$\left\{\text{IDEAL}_{F^A_{mat}, \mathcal{S}(z), I}(\vec{x})\right\}_{z \in \{0,1\}^*; \vec{x} \in \mathbb{F}^n} \equiv \left\{\text{HYBRID}^{F_{VSS}}_{\pi, \mathcal{A}(z), I}(\vec{x})\right\}_{z \in \{0,1\}^*; \vec{x} \in \mathbb{F}^n}. \tag{2.6.1}$$

In order to see why this holds, observe first that in the F_{VSS}-hybrid model, the honest parties actions in the protocol are *deterministic* (the randomness in the real protocol is "hidden" inside the protocol for securely computing F_{VSS}), as is the simulator $\mathcal{S}$ and the ideal functionality F^A_{mat}. Thus, it suffices to separately show that the view of the adversary is identical in both cases, and the outputs of the honest parties are identical in both cases.

By inspection of the protocol and simulation, it follows that the shares $\{(g_1(\alpha_i), \ldots, g_n(\alpha_i))\}_{i \in I}$ of the corrupted parties on the honest parties inputs and the vector of polynomials $\vec{Y}(x)$ as received by $\mathcal{S}$, provide it all the information necessary to generate the *exact* messages that the corrupted parties would receive in a real execution of Protocol 2.6.5. Thus, the view of the adversary is identical in the ideal execution and in the protocol execution.

Next, we show that the honest party's outputs are identical in both distributions. In order to see this, it suffices to show that the vector of polynomials $\vec{Y}(x) = (Y_1(x), \ldots, Y_m(x))$ computed by F^A_{mat} in Step 4 of the functionality specification is identical to the vector of polynomials

$(Y_1(x), \ldots, Y_m(x))$ computed by each party in Step 5 of Protocol 2.6.5 (since this defines the outputs). First, the polynomials of the honest parties are clearly the same in both cases. Furthermore, since the adversary's view is the same it holds that the polynomials $g_i(x)$ sent by $\mathcal{S}$ to the trusted party computing F^A_{mat} are exactly the same as the polynomials used by $\mathcal{A}$ in Step 1 of Protocol 2.6.5. This follows from the fact that the F_{VSS} functionality is used in this step and so the polynomials of the corrupted parties obtained by $\mathcal{S}$ from $\mathcal{A}$ are exactly the same as used in the protocol. Now, observe that each polynomial $Y_k(x)$ computed by the honest parties is obtained by applying Reed-Solomon decoding to the word $(\hat{Y}_k(\alpha_1), \ldots, \hat{Y}_k(\alpha_n))$. The crucial point is that the honest parties compute the values $\hat{Y}_k(\alpha_i)$ correctly, and so for every $j \notin I$ it holds that $\hat{Y}_k(\alpha_j) = Y_k(\alpha_j)$. Thus, at least $n - t$ elements of the word $(\hat{Y}_k(\alpha_1), \ldots, \hat{Y}_k(\alpha_n))$ are "correct" and so the polynomial $Y_k(x)$ reconstructed by all the honest parties in the error correction is the same $Y_k(x)$ as computed by F^A_{mat} (irrespective of what the corrupted parties send). This completes the proof. ■

2.6.4 The $F^{subshare}_{VSS}$ Functionality for Sharing Shares

Defining the functionality. We begin by defining the $F^{subshare}_{VSS}$ functionality. Informally speaking, this functionality is a way for a set of parties to verifiably give out shares of values that are themselves shares. Specifically, assume that the parties $P_1, \ldots, P_n$ hold values $f(\alpha_1), \ldots, f(\alpha_n)$, respectively, where f is a degree-t polynomial either chosen by one of the parties or generated jointly in the computation. The aim is for each party to *share its share* $f(\alpha_i)$ – and not any other value – with all other parties (see Figure 2.2). In the semi-honest setting, this can be achieved simply by having each party P_i choose a random polynomial $g_i(x)$ with constant term $f(\alpha_i)$ and then send each P_j the share $g_i(\alpha_j)$. However, in the malicious setting, it is necessary to force the corrupted parties to share the *correct* value and nothing else; this is the main challenge. We stress that since there are more than t honest parties, their shares fully determine $f(x)$, and so the "correct" share of a corrupted party is well defined. Specifically, letting $f(x)$ be the polynomial defined by the honest parties' shares, the aim here is to ensure that a corrupted P_i provides shares using a degree-t polynomial with constant term $f(\alpha_i)$.

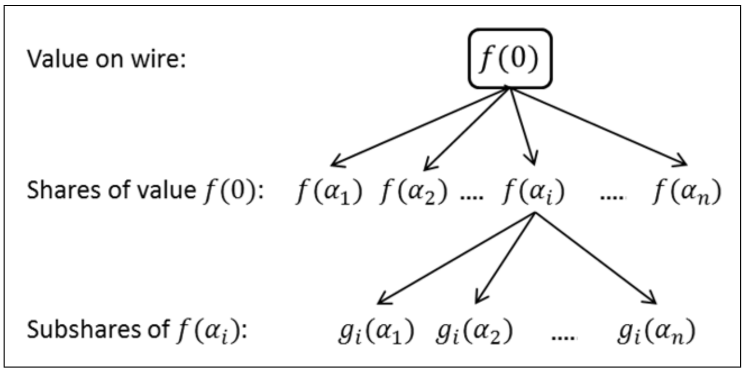

Value on wire:
$f(0)$
Shares of value $f(0)$:
$f(\alpha_1)$ $f(\alpha_2)$ .... $f(\alpha_i)$ .... $f(\alpha_n)$
Subshares of $f(\alpha_i)$:
$g_i(\alpha_1)$ $g_i(\alpha_2)$ .... $g_i(\alpha_n)$

Figure 2.2: The subsharing process: P_i distributes shares of its share $f(\alpha_i)$

The functionality definition is such that if a corrupted party P_i does not provide a valid input (i.e., it does not input a degree-t polynomial $g_i(x)$ such that $g_i(0) = f(\alpha_i)$), then $F^{subshare}_{VSS}$

defines a *new polynomial* $g'_i(x)$ which is the constant polynomial $g'_i(x) = f(\alpha_i)$ for all x, and uses $g'_i(x)$ in place of $g_i(x)$ in the outputs. This ensures that the constant term of the polynomial is always $f(\alpha_i)$, as required.

We define $F_{VSS}^{subshare}$ as a corruption-aware functionality (see Section 2.6.2). Among other reasons, this is due to the fact that the parties distributes subshares of their shares. As we described in Section 2.6.2 this enables the adversary to choose the corrupted parties' polynomials $g_i(x)$ (for $i \in I$) after seeing the corrupted parties' shares of the honest parties' polynomials (i.e., $g_j(\alpha_i)$ for every $j \notin I$ and $i \in I$).

In addition, in the protocol the parties invoke the F_{mat}^{A} functionality (Functionality 2.6.4) with (the transpose of) the parity-check matrix H of the appropriate Reed-Solomon code (this matrix is specified below where we explain its usage in the protocol). This adds complexity to the definition of $F_{VSS}^{subshare}$ because additional information revealed by F_{mat}^{A} to the adversary needs to be revealed by $F_{VSS}^{subshare}$ as well. We denote the matrix multiplication functionality with (the transpose of) the parity-check matrix H by F_{mat}^{H} from here on. Recall that the adversary's output from F_{mat}^{H} includes $\vec{Y}(x) = (g_1(x), \ldots, g_n(x)) \cdot H^T$; see Step 5 in Functionality 2.6.4. Thus, in order to simulate the call to F_{mat}^{H}, the ideal adversary needs this information. We deal with this in the same way as in F_{mat}^{H}, by having the functionality $F_{VSS}^{subshare}$ provide the ideal adversary with the additional vector of polynomials $(g_1(x), \ldots, g_n(x)) \cdot H^T$. As we will see later, this does not interfere with our use of $F_{VSS}^{subshare}$ in order to achieve secure multiplication (which is our ultimate goal). Although it is too early to really see why this is the case, we nevertheless remark that when H is the parity-check matrix of the Reed-Solomon code, the vector $(g_1(0), \ldots, g_n(0)) \cdot H^T$ can be determined based on the corrupted parties' inputs (because we know that the honest parties' values are always "correct"), and the vector $(g_1(x), \ldots, g_n(x)) \cdot H^T$ is random under this constraint. Thus, these outputs can be simulated.

FUNCTIONALITY 2.6.7 (Functionality $F_{VSS}^{subshare}$ for subsharing shares)

$F_{VSS}^{subshare}$ receives a set of indices $I \subseteq [n]$ and works as follows:

1. $F_{VSS}^{subshare}$ receives the inputs of the honest parties $\{\beta_j\}_{j \notin I}$. Let $f(x)$ be the unique degree-t polynomial determined by the points $\{(\alpha_j, \beta_j)\}_{j \notin I}$.[9]
2. For every $j \notin I$, $F_{VSS}^{subshare}$ chooses a random degree-t polynomial $g_j(x)$ under the constraint that $g_j(0) = \beta_j = f(\alpha_j)$.
3. $F_{VSS}^{subshare}$ sends the shares $\{g_j(\alpha_i)\}_{j \notin I; i \in I}$ to the (ideal) adversary.
4. $F_{VSS}^{subshare}$ receives polynomials $\{g_i(x)\}_{i \in I}$ from the (ideal) adversary; if a polynomial $g_i(x)$ is not received or if $g_i(x)$ is of degree higher than t, then $F_{VSS}^{subshare}$ sets $g_i(x) = 0$.
5. $F_{VSS}^{subshare}$ *determines the output polynomials* $g'_1(x), \ldots, g'_n(x)$:
   (a) For every $j \notin I$, $F_{VSS}^{subshare}$ sets $g'_j(x) = g_j(x)$.
   (b) For every $i \in I$, if $g_i(0) = f(\alpha_i)$ then $F_{VSS}^{subshare}$ sets $g'_i(x) = g_i(x)$. Otherwise it sets $g'_i(x) = f(\alpha_i)$, (i.e., $g'_i(x)$ is the constant polynomial equalling $f(\alpha_i)$ everywhere).
6. (a) For every $j \notin I$, $F_{VSS}^{subshare}$ sends the polynomial $g'_j(x)$ and the shares $(g'_1(\alpha_j), \ldots, g'_n(\alpha_j))$ to party P_j.
   (b) Functionality $F_{VSS}^{subshare}$ sends the (ideal) adversary the vector of polynomials $\vec{Y}(x) = (g_1(x), \ldots, g_n(x)) \cdot H^T$, where H is the parity-check matrix of the appropriate Reed-Solomon code (see below). In addition, it sends the corrupted parties' outputs $g'_i(x)$ and $(g'_1(\alpha_i), \ldots, g'_n(\alpha_i))$ for every $i \in I$.

Background to implementing $F_{VSS}^{subshare}$. Let $G \in \mathbb{F}^{(t+1)\times n}$ be the generator matrix for a (generalized) Reed-Solomon code of length $n = 3t+1$, dimension $k = t+1$ and distance $d = 2t+1$. In matrix notation, the encoding of a vector $\vec{a} = (a_0, \ldots, a_t) \in \mathbb{F}^{t+1}$ is given by $\vec{a} \cdot G$, where:

$$G \overset{\text{def}}{=} \begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \vdots & \vdots & & \vdots \\ \alpha_1^t & \alpha_2^t & \ldots & \alpha_n^t \end{pmatrix}.$$

Letting $f(x) = \sum_{\ell=0}^{t} a_\ell \cdot x^\ell$ be a degree-t polynomial, the Reed-Solomon encoding of $\vec{a} = (a_0, \ldots, a_t)$ is the vector $\langle f(\alpha_1), \ldots, f(\alpha_n)\rangle$. Let $H \in \mathbb{F}^{2t\times n}$ be the parity-check matrix of G; that is, H is a rank $2t$ matrix such that $G \cdot H^T = 0^{(t+1)\times 2t}$. W.l.o.g, the matrix H is of the form:

$$H = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \vdots & \vdots & & \vdots \\ \alpha_1^{2t-1} & \alpha_2^{2t-1} & \ldots & \alpha_n^{2t-1} \end{pmatrix} \cdot \begin{pmatrix} v_1 & & 0 \\ & \ddots & \\ 0 & & v_n \end{pmatrix} \qquad (2.6.2)$$

for non-zero values $v_1, \ldots, v_n$. The syndrome of a word $\vec{\beta} \in \mathbb{F}^n$ is given by $S(\vec{\beta}) = \vec{\beta} \cdot H^T \in \mathbb{F}^{2t}$. A basic fact from error-correcting codes is that, for any codeword $\vec{\beta} = \vec{a} \cdot G$, it holds that $S(\vec{\beta}) = 0^{2t}$. Moreover, for every error vector $\vec{e} \in \{0,1\}^n$, it holds that $S(\vec{\beta} + \vec{e}) = S(\vec{e})$. If $\vec{e}$ is of distance at most t from $\vec{0}$ (i.e., $\sum e_i \leq t$), then it is possible to correct the vector $\vec{\beta} + \vec{e}$ and to obtain the original vector $\vec{\beta}$. An important fact is that a sub-procedure of the Reed-Solomon decoding algorithm can extract the error vector $\vec{e}$ from the syndrome vector $S(\vec{e})$ alone. That is, given a possibly corrupted codeword $\vec{\gamma} = \vec{\beta} + \vec{e}$, the syndrome vector is computed as $S(\vec{\gamma}) = S(\vec{e})$ and is given to this sub-procedure, which returns $\vec{e}$. From $\vec{e}$ and $\vec{\gamma}$, the codeword $\vec{\beta}$ can be extracted easily.

The protocol. In the protocol, each party P_i chooses a random polynomial $g_i(x)$ whose constant term equals its input share β_i; let $\vec{\beta} = (\beta_1, \ldots, \beta_n)$. Recall that the input shares are the shares of some polynomial $f(x)$. Thus, for all honest parties P_j it is guaranteed that $g_j(0) = \beta_j = f(\alpha_j)$. In contrast, there is no guarantee regarding the values $g_i(0)$ for corrupted P_i. Let $\vec{\gamma} = (g_1(0), \ldots, g_n(0))$. It follows that $\vec{\gamma}$ is a word that is at most distance t from the vector $\vec{\beta} = (f(\alpha_1), \ldots, f(\alpha_n))$, which is a Reed-Solomon codeword of length $n = 3t+1$. Thus, it is possible to correct the word $\vec{\gamma}$ using Reed-Solomon error correction. The parties send the chosen polynomials $(g_1(x), \ldots, g_n(x))$ to F_{mat}^H (i.e., Functionality 2.6.4 for matrix multiplication with the transpose of the parity-check matrix H described above). F_{mat}^H hands each party P_i the output $(g_1(\alpha_i), \ldots, g_n(\alpha_i))$ and $(s_1, \ldots, s_{2t}) = \vec{\gamma} \cdot H^T$, where the latter equals the syndrome $S(\vec{\gamma})$ of the input vector $\vec{\gamma}$. The parties use the syndrome in order to each locally carry out error

[9]If all of the points sent by the honest parties lie on a single degree-t polynomial, then this guarantees that $f(x)$ is the unique degree-t polynomial for which $f(\alpha_j) = \beta_j$ for all $j \notin I$. If not all the points lie on a single degree-t polynomial, then no security guarantees are obtained. However, since the honest parties all send their prescribed input, $f(x)$ will always be as desired. This can be formalized using the notion of a partial functionality [53, Sec. 7.2]. Alternatively, it can be formalized by having the ideal functionality give all of the honest parties' inputs to the adversary and letting the adversary singlehandedly determine all of the outputs of the honest parties, in the case that the condition does not hold. This makes any protocol secure vacuously (since anything can be simulated).

correction and obtain the error vector $\vec{e} = (e_1, \ldots, e_n) = \vec{\gamma} - \vec{\beta}$. Note that $\vec{e}$ has the property that for every i, $g_i(0) - e_i = f(\alpha_i)$, and it can be computed from the syndrome alone, using the sub-procedure mentioned above. This error vector now provides the honest parties with all the information that they need to compute the output. Specifically, if $e_i = 0$, then this implies that P_i used a "correct" polynomial $g_i(x)$ for which $g_i(0) = f(\alpha_i)$, and so the parties can just output the shares $g_i(\alpha_j)$ that they received as output from F_{mat}^H. In contrast, if $e_i \neq 0$ then the parties know that P_i is corrupted, and can all send each other the shares $g_i(\alpha_j)$ that they received from F_{mat}^H. This enables them to reconstruct the polynomial $g_i(x)$, again using Reed-Solomon error correction, and compute $g_i(0) - e_i = f(\alpha_i)$. Thus, they obtain the actual share of the corrupted party and can set $g_i'(x) = f(\alpha_i)$, as required in the functionality definition. See Protocol 2.6.8 for the full specification.

One issue that must be dealt with in the proof of security is due to the fact that the syndrome $\vec{\gamma} \cdot H^T$ is revealed in the protocol, and is seemingly not part of the output. However, recall that the adversary receives the vector of polynomials $\vec{Y}(x) = (g_1(x), \ldots, g_n(x)) \cdot H^T$ from $F_{VSS}^{subshare}$ and the syndrome is just $\vec{Y}(0)$. This is therefore easily simulated.

PROTOCOL 2.6.8 (Securely computing $F_{VSS}^{subshare}$ in the F_{mat}^H-hybrid model)

- **Inputs:** Each party P_i holds a value β_i; we assume that the points (α_j, β_j) of the honest parties all lie on a single degree-t polynomial (see the definition of $F_{VSS}^{subshare}$ above and Footnote 9 therein).
- **Common input:** A field description $\mathbb{F}$ and n distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$, which determine the matrix $H \in \mathbb{F}^{2t \times n}$ which is the parity-check matrix of the Reed-Solomon code (with parameters as described above).
- **Aiding ideal functionality initialization:** Upon invocation, the trusted party computing the corruption-aware functionality F_{mat}^H receives the set of corrupted parties I.
- **The protocol:**
  1. Each party P_i chooses a random degree-t polynomial $g_i(x)$ under the constraint that $g_i(0) = \beta_i$
  2. The parties invoke the F_{mat}^H functionality (i.e., Functionality 2.6.4 for matrix multiplication with the transpose of the parity-check matrix H). Each party P_i inputs the polynomial $g_i(x)$ from the previous step, and receives from F_{mat}^H as output the shares $g_1(\alpha_i), \ldots, g_n(\alpha_i)$, the degree-$t$ polynomial $g_i(x)$ and the length $2t$ vector $\vec{s} = (s_1, \ldots, s_{2t}) = (g_1(0), \ldots, g_n(0)) \cdot H^T$. Recall that $\vec{s}$ is the syndrome vector of the possible corrupted codeword $\vec{\gamma} = (g_1(0), \ldots, g_n(0))$.
  3. Each party locally runs the Reed-Solomon decoding procedure using $\vec{s}$ only, and receives back an error vector $\vec{e} = (e_1, \ldots, e_n)$.
  4. For every k such that $e_k = 0$: each party P_i sets $g_k'(\alpha_i) = g_k(\alpha_i)$.
  5. For every k such that $e_k \neq 0$:
     (a) Each party P_i sends $g_k(\alpha_i)$ to every P_j.
     (b) Each party P_i receives $g_k(\alpha_1), \ldots, g_k(\alpha_n)$; if any value is missing, it sets it to 0. P_i runs the Reed-Solomon decoding procedure on the values to reconstruct $g_k(x)$.
     (c) Each party P_i computes $g_k(0)$, and sets $g_k'(\alpha_i) = g_k(0) - e_k$ (which equals $f(\alpha_k)$).
- **Output:** P_i outputs $g_i(x)$ and $g_1'(\alpha_i), \ldots, g_n'(\alpha_i)$.

Theorem 2.6.9 *Let $t < n/3$. Then, Protocol 2.6.8 is t-secure for the $F_{VSS}^{subshare}$ functionality in the F_{mat}^{H}-hybrid model, in the presence of a static malicious adversary.*

Proof: We begin by describing the simulator $\mathcal{S}$. The simulator interacts externally with the ideal functionality $F_{VSS}^{subshare}$, while internally simulating the interaction of $\mathcal{A}$ with the honest parties and F_{mat}^{H}.

1. *$\mathcal{S}$ internally invokes $\mathcal{A}$ with the auxiliary input z.*

2. *After the honest parties send their polynomials $\{g_j(x)\}_{j\notin I}$ to the trusted party computing $F_{VSS}^{subshare}$, $\mathcal{S}$ receives the shares $\{g_j(\alpha_i)\}_{j\notin I, i\in I}$ from $F_{VSS}^{subshare}$ (see Step 3 in Functionality 2.6.7).*

3. *$\mathcal{S}$ begins to internally simulate the invocation of F_{mat}^{H} by sending $\mathcal{A}$ the shares $\{g_j(\alpha_i)\}_{j\notin I, i\in I}$ as its first output from the call to F_{mat}^{H} in the protocol (that is, it simulates Step 2 in Functionality 2.6.4).*

4. *$\mathcal{S}$ internally receives from $\mathcal{A}$ the polynomials $\{g_i(x)\}_{i\in I}$ that $\mathcal{A}$ sends to F_{mat}^{H} in the protocol (Step 3 of Functionality 2.6.4).*

5. *$\mathcal{S}$ externally sends the $F_{VSS}^{subshare}$ functionality the polynomials $\{g_i(x)\}_{i\in I}$ that were received in the previous step (as expected by Step 4 in Functionality 2.6.7). For the rest of the execution, if $\deg(g_i) > t$ for some $i \in I$, $\mathcal{S}$ resets $g_i(x) = 0$.*

6. *$\mathcal{S}$ externally receives its output from $F_{VSS}^{subshare}$ (Step 6b of Functionality 2.6.7), which is comprised of the vector of polynomials $\vec{Y}(x) = (g_1(x), \ldots, g_n(x)) \cdot H^T$, and the corrupted parties' outputs: polynomials $\{g'_i(x)\}_{i\in I}$ and the shares $\{g'_1(\alpha_i), \ldots, g'_n(\alpha_i)\}_{i\in I}$. Recall that $g'_j(x) = g_j(x)$ for every $j \notin I$. Moreover, for every $i \in I$, if $g_i(0) = f(\alpha_i)$ then $g'_i(x) = g_i(x)$, and $g'_i(x) = f(\alpha_i)$ otherwise.*

7. *$\mathcal{S}$ concludes the internal simulation of F_{mat}^{H} by preparing the output that the internal $\mathcal{A}$ expects to receive from F_{mat}^{H} (Step 5 of Functionality 2.6.4) in the protocol, as follows:*

   (a) *$\mathcal{A}$ expects to receive the vector of polynomials $\vec{Y}(x) = (g_1(x), \ldots, g_n(x)) \cdot H^T$ from F_{mat}^{H}; however, $\mathcal{S}$ received this exact vector of polynomials from $F_{VSS}^{subshare}$ and so just hands it internally to $\mathcal{A}$.*

   (b) *In addition, $\mathcal{A}$ expects to receive the corrupted parties' outputs $\vec{y} = \vec{Y}(0)$ and the shares $\{(g_1(\alpha_i), \ldots, g_n(\alpha_i))\}_{i\in I}$. Simulator $\mathcal{S}$ can easily compute $\vec{y} = \vec{Y}(0)$ since it has the actual polynomials $\vec{Y}(x)$. In addition, $\mathcal{S}$ already received the shares $\{g_j(\alpha_i)\}_{j\notin I; i\in I}$ from $F_{VSS}^{subshare}$ and can compute the missing shares using the polynomials $\{g_i(x)\}_{i\in I}$. Thus, $\mathcal{S}$ internally hands $\mathcal{A}$ the values $\vec{y} = \vec{Y}(0)$ and $\{(g_1(\alpha_i), \ldots, g_n(\alpha_i))\}_{i\in I}$, as expected by $\mathcal{A}$.*

8. *$\mathcal{S}$ proceeds with the simulation of the protocol as follows. $\mathcal{S}$ computes the error vector $\vec{e} = (e_1, \ldots, e_n)$ by running the Reed-Solomon decoding procedure on the syndrome vector $\vec{s}$, that it computes as $\vec{s} = \vec{Y}(0)$ (using $\vec{Y}(x)$ that it received from $F_{VSS}^{subshare}$). Then, for every $i \in I$ for which $e_i \neq 0$ and for every $j \notin I$, $\mathcal{S}$ internally simulates P_j sending $g_i(\alpha_j)$ to all parties (Step 5a of Protocol 2.6.8).*

9. $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.

We now prove that for every $I \subset [n]$ with $|I| \leq t$:

$$\left\{\text{IDEAL}_{F_{VSS}^{subshare},\mathcal{S}(z),I}(\vec{x})\right\}_{z\in\{0,1\}^*;\vec{x}\in\mathbb{F}^n} \equiv \left\{\text{HYBRID}^{F_{mat}^{H}}_{\pi,\mathcal{A}(z),I}(\vec{x})\right\}_{z\in\{0,1\}^*;\vec{x}\in\mathbb{F}^n}.$$

The main point to notice is that the simulator has enough information to perfectly emulate the honest parties' instructions. The only difference is that in a real protocol execution, the honest parties P_j choose the polynomials $g_j(x)$, whereas in an ideal execution the functionality $F_{VSS}^{subshare}$ chooses the polynomials $g_j(x)$ for every $j \notin I$. However, in both cases they are chosen at random under the constraint that $g_j(0) = \beta_j$. Thus, the distributions are identical. Apart from that, $\mathcal{S}$ has enough information to generate the exact messages that the honest parties would send. Finally, since all honest parties receive the same output from F_{mat}^{A} in the protocol execution, and this fully determines $\vec{e}$, we have that all honest parties obtain the exact same view in the protocol execution and thus all output the exact same value. Furthermore, by the error correction procedure, for every k such that $e_k \neq 0$, they reconstruct the same $g_k(x)$ sent by $\mathcal{A}$ to F_{mat}^{A} and so all define $g'_k(\alpha_j) = g_k(0) - e_k$.

A fictitious simulator $\mathcal{S}'$. We construct a fictitious simulator $\mathcal{S}'$ who generates the entire output distribution of both the honest parties and adversary as follows. For every $j \notin I$, simulator $\mathcal{S}'$ receives for input a *random* polynomial $g_j(x)$ under the constraint that $g_j(0) = \beta_j$. Then, $\mathcal{S}'$ invokes the adversary $\mathcal{A}$ and emulates the honest parties and the aiding functionality F_{mat}^{H} in a protocol execution with $\mathcal{A}$, using the polynomials $g_j(x)$. Finally, $\mathcal{S}'$ outputs whatever $\mathcal{A}$ outputs, together with the output of each honest party. (Note that $\mathcal{S}'$ does not interact with a trusted party and is a stand-alone machine.)

The output distributions. It is clear that the output distribution generated by $\mathcal{S}'$ is *identical* to the output distribution of the adversary and honest parties in a real execution, since the polynomials $g_j(x)$ are chosen randomly exactly like in a real execution and the rest of the protocol is emulated by $\mathcal{S}'$ exactly according to the honest parties' instructions.

It remains to show that the output distribution generated by $\mathcal{S}'$ is *identical* to the output distribution of an ideal execution with $\mathcal{S}$ and a trusted party computing $F_{VSS}^{subshare}$. First, observe that both $\mathcal{S}'$ and $\mathcal{S}$ are *deterministic* machines. Thus, it suffices to separately show that the adversary's view is identical in both cases (given the polynomials $\{g_j(x)\}_{j\notin I}$), and the outputs of the honest parties are identical in both case (again, given the polynomials $\{g_j(x)\}_{j\notin I}$). Now, the messages generated by $\mathcal{S}$ and $\mathcal{S}'$ for $\mathcal{A}$ are identical throughout. This holds because the shares $\{g_j(\alpha_i)\}_{j\notin I;i\in I}$ of the honest parties that $\mathcal{A}$ receives from F_{mat}^{H} are the same ($\mathcal{S}$ receives them from $F_{VSS}^{subshare}$ and $\mathcal{S}'$ generates them itself from the input), as is the vector $\vec{Y}(x) = (g_1(x), \ldots, g_n(x)) \cdot H^T$ and the rest of the output from F_{mat}^{H} for $\mathcal{A}$. Finally, in Step 8 of the specification of $\mathcal{S}$ above, the remainder of the simulation after F_{mat}^{H} is carried out by running the honest parties' instructions. Thus, the messages are clearly identical and $\mathcal{A}$'s view is identical in both executions by $\mathcal{S}$ and $\mathcal{S}'$.

We now show that the output of the honest parties' as generated by $\mathcal{S}'$ is identical to their output in the ideal execution with $\mathcal{S}$ and the trusted party, given the polynomials $\{g_j(x)\}_{j\notin I}$. In the ideal execution with $\mathcal{S}$, the output of each honest party P_j is determined by the trusted party computing $F_{VSS}^{subshare}$ to be $g'_j(x)$ and $(g'_1(\alpha_j), \ldots, g'_n(\alpha_j))$. For every $j \notin I$, $F_{VSS}^{subshare}$ sets

$g'_j(x) = g_j(x)$. Likewise, since the inputs of all the honest parties lie on the same degree-t polynomial, denoted f (and so $f(\alpha_j) = \beta_j$ for every $j \notin I$), we have that the error correction procedure of Reed-Solomon decoding returns an error vector $\vec{e} = (e_1, \dots, e_n)$ such that for every k for which $g_k(0) = f(\alpha_k)$ it holds that $e_k = 0$. In particular, this holds for every $j \notin I$. Furthermore, F^H_{mat} guarantees that all honest parties receive the same vector $\vec{s}$ and so the error correction yields the same error vector $\vec{e}$ for every honest party. Thus, for every $j, \ell \notin I$ we have that each honest party P_ℓ sets $g'_j(\alpha_\ell) = g_j(\alpha_\ell)$, as required.

Regarding the corrupted parties' polynomials $g_i(x)$ for $i \in I$, the trusted party computing $F^{subshare}_{VSS}$ sets $g'_i(x) = g_i(x)$ if $g_i(0) = f(\alpha_i)$, and sets $g'_i(x)$ to be a constant polynomial equalling $f(\alpha_i)$ everywhere otherwise. This exact output is obtained by the honest parties for the same reasons as above: all honest parties receive the same $\vec{s}$ and thus the same $\vec{e}$. If $e_i = 0$ then all honest parties P_j set $g'_i(\alpha_j) = g_i(\alpha_j)$, whereas if $e_i \neq 0$ then the error correction enables them to reconstruct the polynomial $g_i(x)$ exactly and compute $f(\alpha_i) = g_i(0)$. Then, by the protocol every honest P_j sets its share $g'_i(\alpha_j) = f(\alpha_i) - e_i$, exactly like the trusted party. This completes the proof. ■

2.6.5 The F_{eval} Functionality for Evaluating a Shared Polynomial

In the protocol for verifying the multiplication of shares presented in Section 2.6.6 (The F^{mult}_{VSS} functionality), the parties need to process "complaints" (which are claims by some of the parties that others supplied incorrect values). These complaints are processed by evaluating some shared polynomials at the point of the complaining party. Specifically, given shares $f(\alpha_1), \dots, f(\alpha_n)$, of a polynomial f, the parties need to compute $f(\alpha_k)$ for a predetermined k, without revealing anything else. (To be more exact, the shares of the honest parties define a unique polynomial f, and the parties should obtain $f(\alpha_k)$ as output.)

We begin by formally defining this functionality. The functionality is parameterized by an index k that determines at which point the polynomial is to be evaluated. In addition, we define the functionality to be corruption-aware in the sense that the polynomial is reconstructed from the honest party's inputs alone (and the corrupted parties' shares are ignored). We stress that it is possible to define the functionality so that it runs the Reed-Solomon error correction procedure on the input shares. However, defining it as we do makes it clear that the corrupted parties can have no influence whatsoever on the output. See Functionality 2.6.10 for a full specification.

FUNCTIONALITY 2.6.10 (Functionality F^k_{eval} for evaluating a polynomial on α_k)

F^k_{eval} receives a set of indices $I \subseteq [n]$ and works as follows:

1. The F^k_{eval} functionality receives the inputs of the honest parties $\{\beta_j\}_{j \notin I}$. Let $f(x)$ be the unique degree-t polynomial determined by the points $\{(\alpha_j, \beta_j)\}_{j \notin I}$. (If not all the points lie on a single degree-t polynomial, then no security guarantees are obtained; see Footnote 9.)
2. (a) For every $j \notin I$, F^k_{eval} sends the output pair $(f(\alpha_j), f(\alpha_k))$ to party P_j.
   (b) For every $i \in I$, F^k_{eval} sends the output pair $(f(\alpha_i), f(\alpha_k))$ to the (ideal) adversary, as the output of P_i.

Equivalently, in function notation, we have:

$$F_{eval}^k\Big(\beta_1,\ldots,\beta_n\Big) = \Big(((f(\alpha_1),f(\alpha_k)),\ldots,(f(\alpha_n),f(\alpha_k))\Big)$$

where f is the result of Reed-Solomon decoding on $(\beta_1,\ldots,\beta_n)$. We remark that although each party P_i already holds $f(\alpha_i)$ as part of its input, we need the output to include this value in order to simulate. This will not make a difference in its use, since $f(\alpha_i)$ is anyway supposed to be known to P_i.

Background. We show that the share $f(\alpha_k)$ can be obtained by a linear combination of all the input shares $(\beta_1,\ldots,\beta_n)$. The parties' inputs are a vector $\vec{\beta} \stackrel{\text{def}}{=} (\beta_1,\ldots,\beta_n)$ where for every $j \notin I$ it holds that $\beta_j = f(\alpha_j)$. Thus, the parties' inputs are computed by

$$\vec{\beta} = V_{\vec{\alpha}} \cdot \vec{f}^T,$$

where $V_{\vec{\alpha}}$ is the Vandermonde matrix (see Eq. (2.3.2)), and $\vec{f}$ is the vector of coefficients for the polynomial $f(x)$. We remark that $\vec{f}$ is of length n, and is padded with zeroes beyond the $(t+1)$th entry. Let $\vec{\alpha}_k = (1,\alpha_k,(\alpha_k)^2,\ldots,(\alpha_k)^{n-1})$ be the kth row of $V_{\vec{\alpha}}$. Then the output of the functionality is

$$f(\alpha_k) = \vec{\alpha}_k \cdot \vec{f}^T.$$

We have:

$$\vec{\alpha}_k \cdot \vec{f}^T = \vec{\alpha}_k \cdot \left(V_{\vec{\alpha}}^{-1} \cdot V_{\vec{\alpha}}\right) \cdot \vec{f}^T = \left(\vec{\alpha}_k \cdot V_{\vec{\alpha}}^{-1}\right) \cdot \Big(V_{\vec{\alpha}} \cdot \vec{f}^T\Big) = \left(\vec{\alpha}_k \cdot V_{\vec{\alpha}}^{-1}\right) \cdot \vec{\beta}^T \tag{2.6.3}$$

and so there exists a vector of *fixed constants* $(\vec{\alpha}_k \cdot V_{\vec{\alpha}}^{-1})$ such that the inner product of this vector and the inputs yields the desired result. In other words, F_{eval}^k is simply a linear function of the parties' inputs.

The protocol. Since F_{eval}^k is a linear function of the parties' inputs (which are themselves shares), it would seem that it is possible to use the same methodology for securely computing F_{mat}^A (or even directly use F_{mat}^A). However, this would allow the parties to input any value they wish in the computation. In contrast, the linear function that computes F_{eval}^k (i.e., the linear combination of Eq. (2.6.3)) must be computed on the *correct* shares, where "correct" means that they *all* lie on the same degree-t polynomial. This problem is solved by having the parties subshare their input shares using a more robust input sharing stage that guarantees that all the parties input their "correct share". Fortunately, we already have a functionality that fulfills this exact purpose: the $F_{VSS}^{subshare}$ functionality of Section 2.6.4. Therefore, the protocol consists of a *robust* input sharing phase (i.e., an invocation of $F_{VSS}^{subshare}$), a computation phase (which is non-interactive), and an output reconstruction phase. See Protocol 2.6.11 for the full description.

Informally speaking, the security of the protocol follows from the fact that the parties only see subshares that reveal nothing about the original shares. Then, they see n shares of a random polynomial $Q(x)$ whose secret is the value being evaluated, enabling them to reconstruct that secret. Since the secret is obtained by the simulator/adversary as the legitimate output in the ideal model, this can be simulated perfectly.

The main subtlety that needs to be dealt with in the proof of security is due to the fact that the $F_{VSS}^{subshare}$ functionality actually "leaks" some additional information to the adversary, beyond the vectors $(g_1'(\alpha_i),\ldots,g_n'(\alpha_i))$ for all $i \in I$. Namely, the adversary also receives the vector of polynomials $\vec{Y}(x) = (g_1(x), \ldots, g_n(x)) \cdot H^T$, where H is the parity-check matrix for the Reed-Solomon code, and $g_i(x)$ is the polynomial sent by the adversary to $F_{VSS}^{subshare}$ for the corrupted P_i and may differ from $g'_i(x)$ if the constant term of $g_i(x)$ is incorrect (for honest parties $g'_j(x) = g_j(x)$ always).

PROTOCOL 2.6.11 (Securely computing F_{eval}^{k} in the $F_{VSS}^{subshare}$-hybrid model)

- **Inputs:** Each party P_i holds a value β_i; we assume that the points (α_j, β_j) for every honest P_j all lie on a single degree-t polynomial f (see the definition of F_{eval}^{k} above and Footnote 9).
- **Common input:** The description of a field $\mathbb{F}$ and n distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **Aiding ideal functionality initialization:** Upon invocation, the trusted party computing the corruption-aware functionality $F_{VSS}^{subshare}$ receives the set of corrupted parties I.
- **The protocol:**
  1. The parties invoke the $F_{VSS}^{subshare}$ functionality with each party P_i using β_i as its private input. At the end of this stage, each party P_i holds $g'_1(\alpha_i), \ldots, g'_n(\alpha_i)$, where all the $g'_i(x)$ are of degree t, and for every i, $g'_i(0) = f(\alpha_i)$.
  2. Each party P_i locally computes: $Q(\alpha_i) = \sum_{\ell=1}^{n} \lambda_\ell \cdot g'_\ell(\alpha_i)$, where $(\lambda_1, \ldots, \lambda_n) = \vec{\alpha}_k \cdot V_{\vec{\alpha}}^{-1}$. Each party P_i sends $Q(\alpha_i)$ to all P_j.
  3. Each party P_i receives all the shares $\hat{Q}(\alpha_j)$ from each other party $1 \le j \le n$ (if any value is missing, replace it with 0). Note that some of the parties may hold different values if a party is corrupted. Then, given the possibly corrupted codeword $(\hat{Q}(\alpha_1), \ldots, \hat{Q}(\alpha_n))$, each party runs the Reed-Solomon decoding procedure and receives the codeword $(Q(\alpha_1), \ldots, Q(\alpha_n))$. It then reconstructs $Q(x)$ and computes $Q(0)$.
- **Output:** Each party P_i outputs $(\beta_i, Q(0))$.

The intuition as to why this vector of polynomials $\vec{Y}(x)$ can be simulated is due to the following special property of the syndrome function. Let $\vec{\gamma} = (\gamma_1, \ldots, \gamma_n)$ be the inputs of the parties (where for $i \notin I$ it may be the case that $\gamma_i \neq f(\alpha_i)$). (We denote the "correct" input vector by $\vec{\beta}$ – meaning $\vec{\beta} = (f(\alpha_1), \ldots, f(\alpha_n))$ – and the actual inputs used by the parties by $\vec{\gamma}$.) The vector $\vec{\gamma}$ defines a word that is of distance at most t from the valid codeword $(f(\alpha_1), \ldots, f(\alpha_n))$. Thus, there exists an *error vector* $\vec{e}$ of weight at most t such that $\vec{\gamma} - \vec{e} = (f(\alpha_1), \ldots, f(\alpha_n)) = \vec{\beta}$. The syndrome function $S(\vec{x}) = \vec{x} \cdot H^T$ has the property that $S(\vec{\gamma}) = S(\vec{\beta} + \vec{e}) = S(\vec{e})$; stated differently, $(\beta_1, \ldots, \beta_n) \cdot H^T = \vec{e} \cdot H^T$. Now, $\vec{e}$ is actually fully known to the simulator. This is because for every $i \in I$ it receives $f(\alpha_i)$ from F_{eval}^{k}, and so when $\mathcal{A}$ sends $g_i(x)$ to $F_{VSS}^{subshare}$ in the protocol simulation, the simulator can simply compute $e_i = g_i(0) - f(\alpha_i)$. Furthermore, for all $j \notin I$, it is always the case that $e_j = 0$. Thus, the simulator can compute $\vec{e} \cdot H^T = \vec{\beta} \cdot H^T = (g_1(0), \ldots, g_n(0)) \cdot H^T = \vec{Y}(0)$ from the corrupted parties' input and output only (and the adversary's messages). In addition, the simulator has the values $g_1(\alpha_i), \ldots, g_n(\alpha_i)$ for every $i \in I$ and so can compute $\vec{Y}(\alpha_i) = (g_1(\alpha_i), \ldots, g_n(\alpha_i)) \cdot H^T$. As we will show, the vector of polynomials $\vec{Y}(x)$ is a series of random degree-t polynomials under the constraints $\vec{Y}(0)$ and $\{\vec{Y}(\alpha_i)\}_{i \in I}$ that $\mathcal{S}$ can compute. (Actually, when $|I| = t$ there are $t + 1$ constraints and so this vector is *fully determined*. In this case, its actually values are known to the simulator; otherwise, the simulator can just choose random polynomials that fulfill the constraints.) Finally, the same is true regarding the polynomial $Q(x)$: the simulator knows $|I|+1$ constraints (namely $Q(0)=f(\alpha_k)$ and $Q(\alpha_i)=\sum_{\ell=1}^{n}\lambda_\ell\cdot g'_\ell(\alpha_i)$), and can choose Q to be random under these constraints in order to simulate the honest parties sending $Q(\alpha_j)$ for every $j\notin I$. We now formally prove this.

Theorem 2.6.12 *Let $t<n/3$. Then, Protocol 2.6.11 is t-secure for the F^k_{eval} functionality in the $F^{subshare}_{VSS}$-hybrid model, in the presence of a static malicious adversary.*

Proof: The simulator interacts externally with a trusted party computing F^k_{eval}, while internally simulating the interaction of $\mathcal{A}$ with the trusted party computing $F^{subshare}_{VSS}$ and the honest parties. We have already provided the intuition behind how the simulator works, and thus proceed directly to its specification.

The simulator $\mathcal{S}$:

1. *$\mathcal{S}$ receives the ideal adversary's output from F^k_{eval}: the pair $(f(\alpha_i),f(\alpha_k))$ for every $i\in I$ (recall that the corrupted parties have no input in F^k_{eval} and so it just receives output).*
2. *$\mathcal{S}$ internally invokes $\mathcal{A}$ with the auxiliary input z, and begins to simulate the protocol execution.*
3. *$\mathcal{S}$ internally simulates the $F^{subshare}_{VSS}$ invocations:*
   (a) *$\mathcal{S}$ simulates $\mathcal{A}$ receiving the shares $\{g_j(\alpha_i)\}_{j\notin I;i\in I}$ (Step 3 in the $F^{subshare}_{VSS}$ functionality): For every $j\notin I$, $\mathcal{S}$ chooses uniformly at random a polynomial $g_j(x)$ from $\mathcal{P}^{0,t}$, and sends $\mathcal{A}$ the values $\{g_j(\alpha_i)\}_{j\notin I;i\in I}$.*
   (b) *$\mathcal{S}$ internally receives from $\mathcal{A}$ the inputs $\{g_i(x)\}_{i\in I}$ of the corrupted parties to $F^{subshare}_{VSS}$ (Step 4 in the $F^{subshare}_{VSS}$ specification). If for any $i\in I$, $\mathcal{A}$ did not send some polynomial $g_i(x)$, then $\mathcal{S}$ sets $g_i(x)=0$.*
   (c) *For every $i\in I$, $\mathcal{S}$ checks that $\deg(g_i)\le t$ and that $g_i(0)=f(\alpha_i)$. If this check passes, $\mathcal{S}$ sets $g'_i(x)=g_i(x)$. Otherwise, $\mathcal{S}$ sets $g'_i(x)=f(\alpha_i)$. (Recall that $\mathcal{S}$ has $f(\alpha_i)$ from its output from F^k_{eval}.)*
   (d) *For every $j\notin I$, $\mathcal{S}$ sets $g'_j(x)=g_j(x)$.*
   (e) *$\mathcal{S}$ internally gives the adversary $\mathcal{A}$ the outputs (as in Step 6b of the $F^{subshare}_{VSS}$ functionality):*
      i. *The vector of polynomials $\vec{Y}(x)$, which is chosen as follows:*
         - *$\mathcal{S}$ sets $(e_1,\dots,e_n)$ such that $e_j=0$ for every $j\notin I$, and $e_i=g_i(0)-f(\alpha_i)$ for every $i\in I$.*
         - *$\mathcal{S}$ chooses $\vec{Y}(x)$ to be a random vector of degree-t polynomials under the constraints that $\vec{Y}(0)=(e_1,\dots,e_n)\cdot H^T$, and for every $i\in I$ it holds that $\vec{Y}(\alpha_i)=(g_1(\alpha_i),\dots,g_n(\alpha_i))\cdot H^T$.*

         Observe that if $|I|=t$, then all of the polynomials in $\vec{Y}(x)$ are fully determined by the above constraints.
      ii. *The polynomials and values $g'_i(x)$ and $\{g'_1(\alpha_i),\dots,g'_n(\alpha_i)\}$ for every $i\in I$*
4. *$\mathcal{S}$ simulates the sending of the shares $Q(\alpha_j)$:*

(a) *$\mathcal{S}$ chooses a random polynomial $Q(x)$ of degree t under the constraints that:*

- *$Q(0) = f(\alpha_k)$.*
- *For every $i \in I$, $Q(\alpha_i) = \sum_{\ell=1}^{n} \gamma_\ell \cdot g'_\ell(\alpha_i)$.*

(b) *For every $j \notin I$, $\mathcal{S}$ internally simulates honest party P_j sending the value $Q(\alpha_j)$ (Step 2 in Protocol 2.6.11).*

5. *$\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.*

We now prove that for every $I \subseteq [n]$, such that $|I| \leq t$,

$$\left\{\text{IDEAL}_{F^k_{eval},\mathcal{S}(z),I}(\vec{\beta})\right\}_{\vec{\beta}\in\mathbb{F}^n,z\in\{0,1\}^*} \equiv \left\{\text{HYBRID}^{F^{subshare}_{VSS}}_{\pi,\mathcal{A}(z),I}(\vec{\beta})\right\}_{\vec{\beta}\in\mathbb{F}^n,z\in\{0,1\}^*}.$$

There are three differences between the simulation with $\mathcal{S}$ and $\mathcal{A}$, and an execution of Protocol 2.6.11 with $\mathcal{A}$. First, $\mathcal{S}$ chooses the polynomials $g_j(x)$ to have constant terms of 0 instead of constant terms $f(\alpha_j)$ for every $j \notin I$. Second, $\mathcal{S}$ computes the vector of polynomials $\vec{Y}(x)$ based on the given constraints, rather that it being computed by $F^{subshare}_{VSS}$ based on the polynomials $(g_1(x), \ldots, g_n(x))$. Third, $\mathcal{S}$ chooses a random polynomial $Q(x)$ under the described constraints, rather than it being computed as a function of all the polynomials $g'_1(x), \ldots, g'_n(x)$.

We construct three fictitious hybrid simulators, and modify $\mathcal{S}$ one step at a time.

The fictitious simulator $\mathcal{S}_1$: Simulator $\mathcal{S}_1$ is exactly the same as $\mathcal{S}$, except that it receives for input the values $\beta_j = f(\alpha_j)$, for every $j = 1, \ldots, n$. Then, for every $j \notin I$, instead of choosing $g_j(x) \in_R \mathcal{P}^{0,t}$, the fictitious simulator $\mathcal{S}_1$ chooses $g_j(x) \in_R \mathcal{P}^{f(\alpha_j),t}$. We stress that $\mathcal{S}_1$ runs in the ideal model with the same trusted party running F^k_{eval} as $\mathcal{S}$, and the honest parties receive output as specified by F^k_{eval} when running with the ideal adversary $\mathcal{S}$ or $\mathcal{S}_1$.

We claim that for every $I \subseteq [n]$, such that $|I| \leq t$,

$$\left\{\text{IDEAL}_{F^k_{eval},\mathcal{S}_1(z,\vec{\beta}),I}(\vec{\beta})\right\}_{\vec{\beta}\in\mathbb{F}^n,z\in\{0,1\}^*} \equiv \left\{\text{IDEAL}_{F^k_{eval},\mathcal{S}(z),I}(\vec{\beta})\right\}_{\vec{\beta}\in\mathbb{F}^n,z\in\{0,1\}^*}$$

In order to see that the above holds, observe that both $\mathcal{S}$ and $\mathcal{S}_1$ can work when given the points of the inputs shares $\{g_j(\alpha_i)\}_{i\in I,j\notin I}$ and they don't actually need the polynomials themselves. Furthermore, the only difference between $\mathcal{S}$ and $\mathcal{S}_1$ is whether these points are derived from polynomials with zero constant terms, or with the "correct" ones. That is, there exists a machine $\mathcal{M}$ that receives points $\{g_j(\alpha_i)\}_{i\in I;j\notin I}$ and runs the simulation strategy with $\mathcal{A}$ while interacting with F^k_{eval} in an ideal execution, such that:

- If $g_j(0) = 0$ then the joint output of $\mathcal{M}$ and the honest parties in the ideal execution is exactly that of $\text{IDEAL}_{F^k_{eval},\mathcal{S}(z),I}(\vec{\beta})$; i.e., an ideal execution with the original simulator.
- If $g_j(0) = f(\alpha_j)$ then the joint output of $\mathcal{M}$ and the honest parties in the ideal execution is exactly that of $\text{IDEAL}_{F^k_{eval},\mathcal{S}_1(z,\vec{\beta}),I}(\vec{\beta})$; i.e., an ideal execution with the fictitious simulator.

By Claim 2.3.4, the points $\{g_j(\alpha_i)\}_{i\in I;j\notin I}$ when $g_j(0) = 0$ are identically distributed to the points $\{g_j(\alpha_i)\}_{i\in I;j\notin I}$ when $g_j(0) = f(\alpha_j)$. Thus, the joint outputs of the adversary and honest parties in both simulations are identical.

The fictitious simulator $\mathcal{S}_2$: Simulator $\mathcal{S}_2$ is exactly the same as $\mathcal{S}_1$, except that it computes the vector of polynomials $\vec{Y}(x)$ in the same way that $F_{VSS}^{subshare}$ computes it in the real execution. Specifically, for every $j \notin I$, $\mathcal{S}_2$ chooses random polynomials $g_j(x)$ under the constraint that $g_j(0) = f(\alpha_j)$ just like honest parties. In addition, for every $i \in I$, it uses the polynomials $g_i(x)$ sent by $\mathcal{A}$. We claim that for every $I \subseteq [n]$, such that $|I| \leq t$,

$$\left\{\text{IDEAL}_{F^k_{eval},\mathcal{S}_2(z,\vec{\beta}),I}(\vec{\beta})\right\}_{\vec{\beta}\in\mathbb{F}^n,z\in\{0,1\}^*} \equiv \left\{\text{IDEAL}_{F^k_{eval},\mathcal{S}_1(z,\vec{\beta}),I}(\vec{\beta})\right\}_{\vec{\beta}\in\mathbb{F}^n,z\in\{0,1\}^*}$$

This follows from the aforementioned property of the syndrome function $S(\vec{x}) = \vec{x} \cdot H^T$. Specifically, let $\vec{\gamma}$ be the parties' actually inputs (for $j \notin I$ we are given that $\gamma_j = f(\alpha_j)$, but nothing is guaranteed about the value of γ_i for $i \in I$), and let $\vec{e} = (e_1, \ldots, e_n)$ be the error vector (for which $\gamma_i = f(\alpha_i) + e_i$). Then, $S(\vec{\gamma}) = S(\vec{e})$. If $|I| = t$, then the constraints fully define the vector of polynomials $\vec{Y}(x)$, and by the property of the syndrome these constraints are identical in both simulations by $\mathcal{S}_1$ and $\mathcal{S}_2$. Otherwise, if $|I| < t$, then $\mathcal{S}_1$ chooses $\vec{Y}(x)$ at random under $t+1$ constraints, whereas $\mathcal{S}_2$ computes $\vec{Y}(x)$ from the actual values. Consider each polynomial $Y_\ell(x)$ separately (for $\ell = 1, \ldots, 2t-1$). Then, for each polynomial there is a set of $t+1$ constraints and each is chosen at random under those constraints. Consider the random processes $X(s)$ and $Y(s)$ before Claim 2.4.4 in Section 2.4.2 (where the value "s" here for $Y_\ell(x)$ is the ℓth value in the vector $\vec{e} \cdot H^T$). Then, by Claim 2.4.4, the distributions are identical.

The fictitious simulator $\mathcal{S}_3$: Simulator $\mathcal{S}_3$ is the same as $\mathcal{S}_2$, except that it computes the polynomial $Q(x)$ using the polynomials $g'_1(x), \ldots, g'_n(x)$ instead of under the constraints. The fact that this is identical follows the exact same argument regarding $\vec{Y}_\ell(x)$ using Claim 2.4.4 in Section 2.4.2. Thus,

$$\left\{\text{IDEAL}_{F^k_{eval},\mathcal{S}_3(z,\vec{\beta}),I}(\vec{\beta})\right\}_{\vec{\beta}\in\mathbb{F}^n,z\in\{0,1\}^*} \equiv \left\{\text{IDEAL}_{F^k_{eval},\mathcal{S}_2(z,\vec{\beta}),I}(\vec{\beta})\right\}_{\vec{\beta}\in\mathbb{F}^n,z\in\{0,1\}^*}$$

Completing the proof: Observe that the view of $\mathcal{A}$ in $\text{IDEAL}_{F^k_{eval},\mathcal{S}_3(z,\vec{\beta}),I}(\vec{\beta})$ is exactly the same as in a real execution. It remains to show that the honest parties output the same in both this execution and in the $F_{VSS}^{subshare}$-hybrid execution of Protocol 2.6.11. Observe that $\mathcal{S}_3$ (and $\mathcal{S}_1/\mathcal{S}_2$) send no input to the trusted party in the ideal model. Thus, we just need to show that the honest parties always output $f(\alpha_k)$ in a real execution, when f is the polynomial defined by the input points $\{\beta_j\}_{j\notin I}$ of the honest parties. However, this follows immediately from the guarantees provided the $F_{VSS}^{subshare}$ functionality and by the Reed-Solomon error correction procedure. In particular, the only values received by the honest parties in a real execution are as follows:

1. Each honest P_j receives $g'_1(\alpha_j), \ldots, g'_n(\alpha_j)$, where it is guaranteed by $F_{VSS}^{subshare}$ that for every $i = 1, \ldots, n$ we have $g'_i(0) = f(\alpha_i)$. Thus, these values are *always* correct.

2. Each honest P_j receives values $(\hat{Q}(\alpha_1), \ldots, \hat{Q}(\alpha_n))$. Now, since $n - t$ of these values are sent by honest parties, it follows that this is a vector that is of distance at most t from the codeword $(Q(\alpha_1), \ldots, Q(\alpha_n))$. Thus, the Reed-Solomon correction procedure returns this codeword to every honest party, implying that the correct polynomial $Q(x)$ is reconstructed, and the honest party outputs $Q(0) = f(\alpha_k)$, as required.

This completes the proof. ■

2.6.6 The F_{VSS}^{mult} Functionality for Sharing a Product of Shares

Recall that in the semi-honest protocol for multiplication, each party locally computes the product of its *shares on the input wires* and distributes shares of this product to all other parties (i.e., it defines a polynomial with constant term that equals the product of its shares). In the protocol for malicious adversaries, the same procedure needs to be followed. However, in contrast to the semi-honest case, a mechanism is needed to enforce the malicious parties to indeed use the product of their shares. As in F_{eval}^{k}, we can force the malicious parties to distribute subshares of their "correct" shares using the $F_{VSS}^{subshare}$ functionality. Given a correct subsharing of shares, it is possible to obtain a subsharing of the product of the shares. This second step is the aim of the F_{VSS}^{mult} functionality.

In more detail, the F_{VSS}^{mult} functionality is used *after* the parties have all obtained subshares of each other's shares. Concretely, the functionality has a dealer with two degree-t polynomials $A(x)$ and $B(x)$ for input, and all parties already have shares of these polynomials. (In the use of the functionality in F_{mult} for actual multiplication, $A(x)$ and $B(x)$ are such that $A(0)$ is the dealer's share on one value and $B(0)$ is the dealer's share on another value.) Then, the output of the functionality is shares of a random polynomial with constant term $A(0) \cdot B(0)$ for all parties. Given that $A(0)$ and $B(0)$ are the dealer's shares on the original polynomials, the output is a subsharing of the product of shares, exactly as in the semi-honest case. See formal description of the functionality in Functionality 2.6.13.

FUNCTIONALITY 2.6.13 (Functionality F_{VSS}^{mult} for sharing a product of shares)

F_{VSS}^{mult} receives a set of indices $I \subseteq [n]$ and works as follows:

1. The F_{VSS}^{mult} functionality receives an input pair (a_j, b_j) from every honest party P_j ($j \notin I$). (The dealer P_1 also has polynomials $A(x), B(x)$ such that $A(\alpha_j) = a_j$ and $B(\alpha_j) = b_j$, for every $j \notin I$.)
2. F_{VSS}^{mult} computes the unique degree-t polynomials A and B such that $A(\alpha_j) = a_j$ and $B(\alpha_j) = b_j$ for every $j \notin I$ (if no such A or B exist of degree-t, then F_{VSS}^{mult} behaves differently as in Footnote 9).
3. If the dealer P_1 is honest ($1 \notin I$), then:
   (a) F_{VSS}^{mult} chooses a random degree-t polynomial C under the constraint that $C(0) = A(0) \cdot B(0)$.
   (b) *Outputs for honest:* F_{VSS}^{mult} sends the dealer P_1 the polynomial $C(x)$, and for every $j \notin I$ it sends $C(\alpha_j)$ to P_j.
   (c) *Outputs for adversary:* F_{VSS}^{mult} sends the shares $(A(\alpha_i), B(\alpha_i), C(\alpha_i))$ to the (ideal) adversary, for every $i \in I$.
4. If the dealer P_1 is corrupted ($1 \in I$), then:
   (a) F_{VSS}^{mult} sends $(A(x), B(x))$ to the (ideal) adversary.
   (b) F_{VSS}^{mult} receives a polynomial C as input from the (ideal) adversary.
   (c) If either $\deg(C) > t$ or $C(0) \neq A(0) \cdot B(0)$, then F_{VSS}^{mult} resets $C(x) = A(0) \cdot B(0)$; that is, the constant polynomial equalling $A(0) \cdot B(0)$ everywhere.
   (d) *Outputs for honest:* F_{VSS}^{mult} sends $C(\alpha_j)$ to P_j, for every $j \notin I$.
   (There is no more output for the adversary in this case.)

We remark that although the dealing party P_1 is supposed to already have $A(x), B(x)$ as part of its input and each party P_i is also supposed to already have $A(\alpha_i)$ and $B(\alpha_i)$ as part of its input, this information is provided as output in order to enable simulation. Specifically, the simulator needs to know the corrupted parties "correct points" in order to properly simulate the protocol execution. In order to ensure that the simulator has this information (since the adversary is not guaranteed to have its correct points as input), it is provided by the functionality. In our use of F_{VSS}^{mult} in the multiplication protocol, this information is always known to the adversary anyway, and so there is nothing leaked by having it provided again by the functionality.

As we have mentioned, this functionality is used once the parties already hold shares of a and b (where a and b are the original shares of the dealer). The aim of the functionality is for them to now obtain shares of $a \cdot b$ via a degree-t polynomial C such that $C(0) = A(0) \cdot B(0) = a \cdot b$. We stress that a and b are not values on the wires, but rather are the *shares* of the dealing party of the original values on the wires.

The protocol idea. Let $A(x)$ and $B(x)$ be polynomials such that $A(0) = a$ and $B(0) = b$; i.e., $A(x)$ and $B(x)$ are the polynomials used to share a and b. The idea behind the protocol is for the dealer to first define a sequence of t polynomials $D_1(x), \ldots, D_t(x)$, all of degree-t, such that $C(x) \stackrel{\text{def}}{=} A(x) \cdot B(x) - \sum_{\ell=1}^{t} x^\ell \cdot D_\ell(x)$ is a random degree-t polynomial with constant term equalling $a \cdot b$; recall that since each of $A(x)$ and $B(x)$ are of degree t, the polynomial $A(x) \cdot B(x)$ is of degree $2t$. We will show below how the dealer can choose $D_1(x), \ldots, D_t(x)$ such that all the coefficients from $t+1$ to $2t$ in $A(x) \cdot B(x)$ are canceled out, and the resulting polynomial $C(x)$ is of degree-t (and random). The dealer then shares the polynomials $D_1(x), \ldots, D_t(x)$ and each party locally computes its share of $C(x)$. An important property is that the constant term of $C(x)$ equals $A(0) \cdot B(0) = a \cdot b$ for *every* possible choice of polynomials $D_1(x), \ldots, D_t(x)$. This is due to the fact that each $D_\ell(x)$ is multiplied by x^ℓ (with $\ell \geq 1$) and so these do not affect $C(0)$. This guarantees that even if the dealer is malicious and does not choose the polynomials $D_1(x), \ldots, D_t(x)$ correctly, the polynomial $C(x)$ must have the correct constant term (but it will not necessarily be of degree t, as we explain below).

In more detail, after defining $D_1(x), \ldots, D_t(x)$, the dealer shares them all using F_{VSS}; this ensures that all polynomials are of degree-t and all parties have correct shares. Since each party already holds a valid share of $A(x)$ and $B(x)$, this implies that each party can *locally compute* its share of $C(x)$. Specifically, given $A(\alpha_j)$, $B(\alpha_j)$ and $D_1(\alpha_j), \ldots, D_t(\alpha_j)$, party P_j can simply compute $C(\alpha_j) = A(\alpha_j) \cdot B(\alpha_j) - \sum_{\ell=1}^{t} (\alpha_j)^\ell \cdot D_\ell(\alpha_j)$. The crucial properties are that **(a)** if the dealer is honest, then all the honest parties hold valid shares of a random degree-t polynomial with constant term $a \cdot b$, as required, and **(b)** if the dealer is malicious, all honest parties are guaranteed to hold valid shares of a polynomial with constant term $a \cdot b$ (but with no guarantee regarding the degree). Thus, all that remains is for the parties to verify that the shares that they hold for $C(x)$ define a degree-t polynomial.

It may be tempting to try to solve this problem by having the dealer share $C(x)$ using F_{VSS}, and then having each party check that the share that it received from this F_{VSS} equals the value $C(\alpha_j)$ that it computed from its shares $A(\alpha_j), B(\alpha_j), D_1(\alpha_j), \ldots, D_t(\alpha_j)$. If not, then like in Protocol 2.5.7 for VSS, the parties broadcast complaints. If more than t complaints are broadcast then the honest parties know that the dealer is corrupted (more than t complaints are needed since the corrupted parties can falsely complain when the dealer is honest). They can then broadcast their input shares to reconstruct $A(x), B(x)$ and all define their output shares to be $a \cdot b = A(0) \cdot B(0)$. Since F_{VSS} guarantees that the polynomial shared is of degree-t and we already know that the computed polynomial has the correct constant term, this seems to provide the guarantee that the parties hold shares of a degree-t polynomial with constant term $A(0) \cdot B(0)$. However, the assumption that $t+1$ correct shares (as is guaranteed by viewing at most t complaints) determines that the polynomial computed is of degree-t, or that the polynomial shared with VSS has constant term $A(0) \cdot B(0)$ is *false*. This is due to the fact that it is possible for the dealer to define the polynomials $D_1(x), \ldots, D_t(x)$ so that $C(x)$ is a degree $2t$ polynomial that agrees with some other degree-t polynomial $C'(x)$ on up to $2t$ of the honest parties' points α_j, but for which $C'(0) \neq a \cdot b$. A malicious dealer can then share $C'(x)$ using F_{VSS} and no honest parties would detect any cheating.[10] Observe that at least one honest party would detect cheating and would complain (because $C(x)$ can only agree with $C'(x)$ on $2t$ of the points, and there are at least $2t+1$ honest parties). However, this is not enough to act upon because, as described, when the dealer is honest up to t of the parties could present fake complaints because they are malicious.

We solve this problem by having the parties *unequivocally verify every complaint* to check if it is legitimate. If the complaint is legitimate, then they just reconstruct the initial shares a and b and all output the constant share $a \cdot b$. In contrast, if the complaint is not legitimate, the parties just ignore it. This guarantees that if no honest parties complain (legitimately), then the degree-t polynomial $C'(x)$ shared using F_{VSS} agrees with the computed polynomial $C(x)$ on at least $2t+1$ points. Since $C(x)$ if of degree at most $2t$, this implies that $C(x) = C'(x)$ and so it is actually of degree-t, as required. In order to unequivocally verify complaints, we use the new functionality defined in Section 2.6.5 called F_{eval} that reconstructs the input share of the complainant (given that all honest parties hold valid shares of a degree-t polynomial). Now, if a party P_k complains legitimately, then this implies that $C'(\alpha_k) \neq A(\alpha_k) \cdot B(\alpha_k) - \sum_{\ell=1}^{t} (\alpha_k)^\ell \cdot D_\ell(\alpha_k)$. Observe that the parties are guaranteed to have valid shares of all the polynomials $C'(x), D_1(x), \ldots, D_t(x)$ since they are shared using F_{VSS}, and also shares of $A(x)$ and $B(x)$ by the assumption on the inputs. Thus, they can use F_{eval}^k to obtain all of the values $A(\alpha_k)$, $B(\alpha_k)$, $D_1(\alpha_k), \ldots, D_t(\alpha_k)$, and $C'(\alpha_k)$ and then each party can just check if $C'(\alpha_k)$ equals $C(\alpha_k) \stackrel{\text{def}}{=} A(\alpha_k) \cdot B(\alpha_k) - \sum_{\ell=1}^{t} (\alpha_k)^\ell D_\ell(\alpha_k)$. If yes, then the complaint is false, and is ignored. If no, then the complaint is valid, and thus they reconstruct $a \cdot b$.

Observe that if the dealer is honest, then no party can complain legitimately. In addition, when the dealer is honest and an illegitimate complaint is sent by a corrupted party, then this complaint is verified using F_{eval} which reveals nothing more than the complainants shares. Since the complainant in this case is corrupted, and so its share is already known to the adversary, this reveals no additional information.

Constructing the polynomial $C(x)$. As we have mentioned above, the protocol works by having the dealer choose t polynomials $D_1(x), \ldots, D_t(x)$ that are specially designed so that $C(x) = A(x) \cdot B(x) - \sum_{\ell=1}^{t} x^\ell \cdot D_\ell(x)$ is a *uniformly distributed* polynomial in $\mathcal{P}^{a \cdot b, t}$, where

[10]An alternative strategy could be to run the verification strategy of Protocol 2.5.7 for VSS on the shares $C(\alpha_j)$ that the parties computed in order to verify that it is a degree-t polynomial. The problem with this strategy is that if $C(x)$ is not a degree-t polynomial, then the protocol for F_{VSS} *changes* the points that the parties receive so that it is a degree-t polynomial. However, in this process, the constant term of the resulting polynomial may also change. Thus, there will no longer be any guarantee that the honest parties hold shares of a polynomial with the correct constant term.

$A(0) = a$ and $B(0) = b$. We now show how the dealer chooses these polynomials. The dealer first defines the polynomial $D(x)$:

$$D(x) \overset{\text{def}}{=} A(x) \cdot B(x) = a \cdot b + d_1 x + \ldots + d_{2t} x^{2t}$$

($D(x)$ is of degree $2t$ since both $A(x)$ and $B(x)$ are of degree-t). Next it defines the polynomials:

$$\begin{aligned}
D_t(x) &= r_{t,0} + r_{t,1}x + \ldots + r_{t,t-1}x^{t-1} + d_{2t}x^t \\
D_{t-1}(x) &= r_{t-1,0} + r_{t-1,1}x + \ldots + r_{t-1,t-1}x^{t-1} + (d_{2t-1} - r_{t,t-1}) \cdot x^t \\
D_{t-2}(x) &= r_{t-2,0} + r_{t-2,1}x + \ldots + r_{t-2,t-1}x^{t-1} + (d_{2t-2} - r_{t-1,t-1} - r_{t,t-2}) \cdot x^t \\
&\vdots \\
D_1(x) &= r_{1,0} + r_{1,1}x + \ldots r_{1,t-1}x^{t-1} + (d_{t+1} - r_{t,1} - r_{t-1,2} - \ldots - r_{2,t-1})\, x^t
\end{aligned}$$

where all $r_{i,j} \in_R \mathbb{F}$ are random values, and the d_i values are the coefficients from $D(x) = A(x) \cdot B(x)$.[11] That is, in each polynomial $D_\ell(x)$ all coefficients are random expect for the t^{th} coefficient, which is included in the $(t + \ell)$th coefficient of $D(x)$. More exactly, for $1 \leq \ell \leq t$ polynomial $D_\ell(x)$ is defined by:

$$D_\ell(x) = r_{\ell,0} + r_{\ell,1} \cdot x + \cdots + r_{\ell,t-1} \cdot x^{t-1} + \left(d_{t+\ell} - \sum_{m=\ell+1}^{t} r_{m,t+\ell-m} \right) \cdot x^t \qquad (2.6.4)$$

and the polynomial $C(x)$ is computed by:

$$C(x) = D(x) - \sum_{\ell=1}^{t} x^\ell \cdot D_\ell(x).$$

Before proceeding, we show that when the polynomials $D_1(x), \ldots, D_t(x)$ are chosen in this way, it holds that $C(x)$ is a degree-t polynomial with constant term $A(0) \cdot B(0) = a \cdot b$. Specifically, the coefficients in $D(x)$ for powers greater than t cancel out. For every polynomial $D_\ell(x)$, we have that: $D_\ell(x) = r_{\ell,0} + r_{\ell,1} \cdot x + \cdots + r_{\ell,t-1} \cdot x^{t-1} + R_{\ell,t} \cdot x^t$, where

$$R_{\ell,t} = d_{t+\ell} - \sum_{m=\ell+1}^{t} r_{m,t+\ell-m}. \qquad (2.6.5)$$

(Observe that the sum of the *indices* (i, j) of the $r_{i,j}$ values inside the sum is *always* $t + \ell$ exactly.) We now analyze the structure of the polynomial $\sum_{\ell=1}^{t} x^\ell \cdot D_\ell(x)$. First, observe that it is a polynomial of degree $2t$ with constant term 0 (the constant term is 0 since $\ell \geq 1$). Next, the coefficient of the monomial x^ℓ is the *sum* of the coefficients of the ℓth column in Table 2.1; in the table, the coefficients of the polynomial $D_\ell(x)$ are written in the ℓth row and are shifted ℓ places to the right since $D_\ell(x)$ is multiplied by x^ℓ.

We will now show that for every $k = 1, \ldots, t$ the coefficient of the monomial x^{t+k} in the polynomial $\sum_{\ell=1}^{t} x^\ell \cdot D_\ell(x)$ equals d_{t+k}. Now, the sum of the $(t + k)$th column of the above table (for $1 \leq k \leq t$) is

$$R_{k,t} + r_{k+1,t-1} + r_{k+2,t-2} + \cdots + r_{t,k} = R_{k,t} + \sum_{m=k+1}^{t} r_{m,t+k-m}.$$

[11]The **naming convention** for the $r_{i,j}$ values is as follows. In the first $t - 1$ coefficients, the first index in every $r_{i,j}$ value is the index of the polynomial and the second is the place of the coefficient. That is, $r_{i,j}$ is the jth coefficient of polynomial $D_i(x)$. The values for the t^{th} coefficient are used in the other polynomials as well, and are chosen to cancel out; see below.

	x	x^2	x^3	$\dots$	x^t	x^{t+1}	x^{t+2}	$\dots$	x^{2t-2}	x^{2t-1}	x^{2t}
D_t					$r_{t,0}$	$r_{t,1}$	$r_{t,2}$	$\dots$	$r_{t,t-2}$	$r_{t,t-1}$	$R_{t,t}$
D_{t-1}				$\dots$	$r_{t-1,1}$	$r_{t-1,2}$	$r_{t-1,3}$	$\dots$	$r_{t-1,t-1}$	$R_{t-1,t}$	
D_{t-2}				$\dots$	$r_{t-2,2}$	$r_{t-2,3}$	$r_{t-2,4}$	$\dots$	$R_{t-2,t}$		
$\vdots$				$\iddots$	$\vdots$	$\vdots$	$\vdots$	$\iddots$			
D_3			$r_{3,0}$	$\dots$	$r_{3,t-3}$	$r_{3,t-2}$	$r_{3,t-1}$	$\dots$			
D_2		$r_{2,0}$	$r_{2,1}$	$\dots$	$r_{2,t-2}$	$r_{2,t-1}$	$R_{2,t}$				
D_1	$r_{1,0}$	$r_{1,1}$	$r_{1,2}$	$\dots$	$r_{1,t-1}$	$R_{1,t}$					

Table 2.1: Coefficients of the polynomial $\sum_{\ell=1}^{t} x^\ell \cdot D_\ell(x)$.

Combining this with the definition of $R_{k,t}$ in Eq. (2.6.5), we have that all of the $r_{i,j}$ values cancel out, and the sum of the $(t+k)$th column is just d_{t+k}. We conclude that the $(t+k)$th coefficient of $C(x) = D(x) - \sum_{\ell=1}^{t} x^\ell \cdot D_\ell(x)$ equals $d_{t+k} - d_{t+k} = 0$, and thus $C(x)$ is of degree t, as required. The fact that $C(0) = a \cdot b$ follows immediately from the fact that each $D_\ell(x)$ is multiplied by x^ℓ and so this does not affect the constant term of $D(x)$. Finally, observe that the coefficients of $x, x^2, \dots, x^t$ are all random (since for every $i = 1, \dots, t$ the value $r_{i,0}$ appears only in the coefficient of x^i). Thus, the polynomial $C(x)$ also has random coefficients everywhere except for the constant term.

The protocol. See Protocol 2.6.15 for a full specification in the $(F_{VSS}, F^1_{eval}, \dots, F^n_{eval})$-hybrid model. From here on, we write the F_{eval}-hybrid model to refer to all n functionalities $F^1_{eval}, \dots, F^n_{eval}$.

We have the following theorem:

Theorem 2.6.14 *Let $t < n/3$. Then, Protocol 2.6.15 is t-secure for the F^{mult}_{VSS} functionality in the (F_{VSS}, F_{eval})-hybrid model, in the presence of a static malicious adversary.*

Proof: We separately prove the security of the protocol when the dealer is honest and when the dealer is corrupted.

Case 1 – the dealer P_1 is honest: The simulator interacts externally with F^{mult}_{VSS}, while internally simulating the interaction of $\mathcal{A}$ with the honest parties and F_{VSS}, F_{eval} in Protocol 2.6.15. Since the dealer is honest, in all invocations of F_{VSS} the adversary has no inputs to these invocations and just receives shares. Moreover, as specified in the F^{mult}_{VSS} functionality, the ideal adversary/simulator $\mathcal{S}$ has no input to F^{mult}_{VSS} and it just receives the correct input shares $(A(\alpha_i), B(\alpha_i))$ and the output shares $C(\alpha_i)$ for every $i \in I$. The simulator $\mathcal{S}$ simulates the view of the adversary by choosing random degree-t polynomials $D_2(x), \dots, D_t(x)$, and then choosing $D_1(x)$ randomly under the constraint that for every $i \in I$ it holds that

$$\alpha_i \cdot D_1(\alpha_i) = A(\alpha_i) \cdot B(\alpha_i) - C(\alpha_i) - \sum_{\ell=2}^{t} (\alpha_i)^\ell \cdot D_\ell(\alpha_i).$$

This computation makes sense because

$$C(x) = D(x) - \sum_{\ell=1}^{t} x^\ell \cdot D_\ell(x) = A(x) \cdot B(x) - x \cdot D_1(x) - \sum_{\ell=2}^{t} x^\ell \cdot D_\ell(x)$$

implying that

$$x \cdot D_1(x) = A(x) \cdot B(x) - C(x) - \sum_{\ell=2}^{t} x^\ell \cdot D_\ell(x).$$

PROTOCOL 2.6.15 (Securely computing F_{VSS}^{mult} in the F_{VSS}-F_{eval}-hybrid model)

- **Input:**
  1. The dealer P_1 holds two degree-t polynomials A and B.
  2. Each party P_i holds a pair of shares a_i and b_i such that $a_i = A(\alpha_i)$ and $b_i = B(\alpha_i)$.
- **Common input:** A field description $\mathbb{F}$ and n distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **Aiding ideal functionality initialization:** Upon invocation, the trusted party computing the (fictitiously corruption-aware) functionality F_{VSS} and the corruption-aware functionality F_{eval} receives the set of corrupted parties I.
- **The protocol:**
  1. *Dealing phase:*
     (a) The dealer P_1 defines the degree-$2t$ polynomial $D(x) = A(x) \cdot B(x)$; denote $D(x) = a \cdot b + \sum_{\ell=1}^{2t} d_\ell \cdot x^\ell$.

     (b) P_1 chooses t^2 values $\{r_{k,j}\}$ uniformly and independently at random from $\mathbb{F}$, where $k = 1, \ldots, t$, and $j = 0, \ldots, t-1$.

     (c) For every $\ell = 1, \ldots, t$, the dealer P_1 defines the polynomial $D_\ell(x)$:
     $$D_\ell(x) = \left(\sum_{m=0}^{t-1} r_{\ell,m} \cdot x^m\right) + \left(d_{\ell+t} - \sum_{m=\ell+1}^{t} r_{m,t+\ell-m}\right) \cdot x^t.$$

     (d) P_1 computes the polynomial:
     $$C(x) = D(x) - \sum_{\ell=1}^{t} x^\ell \cdot D_\ell(x).$$

     (e) P_1 invokes the F_{VSS} functionality as dealer with input $C(x)$; each party P_i receives $C(\alpha_i)$.

     (f) P_1 invokes the F_{VSS} functionality as dealer with input $D_\ell(x)$ for every $\ell = 1, \ldots, t$; each party P_i receives $D_\ell(\alpha_i)$.
  2. *Verify phase:* Each party P_i works as follows:
     (a) If any of the $C(\alpha_i), D_\ell(\alpha_i)$ values equals $\perp$ then P_i proceeds to the *reject phase* (note that if one honest party received $\perp$ then all did).

     (b) Otherwise, P_i computes $c'_i = a_i \cdot b_i - \sum_{\ell=1}^{t} (\alpha_i)^\ell \cdot D_\ell(\alpha_i)$. If $c'_i \neq C(\alpha_i)$ then P_i broadcasts $(\mathsf{complaint}, i)$.

     (c) If any party P_k broadcast $(\mathsf{complaint}, k)$ then go to the *complaint resolution phase.*
  3. *Complaint resolution phase:* Run the following for every $(\mathsf{complaint}, k)$ message:
     (a) Run $t+3$ invocations of F_{eval}^k: in each of the invocations each party P_i inputs the corresponding value $a_i, b_i, C(\alpha_i), D_1(\alpha_i), \ldots, D_t(\alpha_i)$.

     (b) Let $A(\alpha_k), B(\alpha_k), \tilde{C}(\alpha_k), \tilde{D}_1(\alpha_k), \ldots, \tilde{D}_t(\alpha_k)$ be the respective outputs that all parties receive from the invocations. Compute $\tilde{C}'(\alpha_k) = A(\alpha_k) \cdot B(\alpha_k) - \sum_{\ell=1}^{t} (\alpha_k)^\ell \cdot \tilde{D}_\ell(\alpha_k)$. (We denote these polynomials by $\tilde{C}, \tilde{D}_\ell, \ldots$ since if the dealer is not honest they may differ from the specified polynomials above.)

     (c) If $\tilde{C}(\alpha_k) \neq \tilde{C}'(\alpha_k)$, then proceed to the *reject phase.*
  4. *Reject phase* (skip to the output if not explicitly instructed to run the reject phase):
     (a) Every party P_i broadcasts the pair (a_i, b_i). Let $\vec{a} = (a_1, \ldots, a_n)$ and $\vec{b} = (b_1, \ldots, b_n)$ be the broadcast values (where zero is used for any value not broadcast). Then, P_i computes $A'(x)$ and $B'(x)$ to be the outputs of Reed-Solomon decoding on $\vec{a}$ and $\vec{b}$, respectively.

     (b) Every party P_i sets $C(\alpha_i) = A'(0) \cdot B'(0)$.
- **Output:** Every party P_i outputs $C(\alpha_i)$.

As we will see, the polynomials $D_\ell(x)$ chosen by an honest dealer have the same distribution as those chosen by $\mathcal{S}$ (they are random under the constraint that

$$C(\alpha_i) = A(\alpha_i) \cdot B(\alpha_i) - \sum_{\ell=1}^{t} (\alpha_i)^\ell \cdot D_\ell(\alpha_i)$$

for all $i \in I$). In order to simulate the complaints, observe that no honest party broadcasts a $\mathsf{complaint}$. Furthermore, for every $(\mathsf{complaint}, i)$ value broadcast by a corrupted P_i ($i \in I$), the complaint resolution phase can easily be simulated since $\mathcal{S}$ knows the correct values $\tilde{A}(\alpha_i) = A(\alpha_i)$, $\tilde{B}(\alpha_i) = B(\alpha_i)$, $\tilde{C}(\alpha_i) = C(\alpha_i)$. Furthermore, for every $\ell = 1, \ldots, t$, $\mathcal{S}$ uses $\tilde{D}_\ell(\alpha_i) = D_\ell(\alpha_i)$ as chosen initially in the simulation as the output from F^i_{eval}. We now formally describe the simulator.

The simulator $\mathcal{S}$:

1. *$\mathcal{S}$ internally invokes the adversary $\mathcal{A}$ with the auxiliary input z.*
2. *$\mathcal{S}$ externally receives from F^{mult}_{VSS} the values $(A(\alpha_i), B(\alpha_i), C(\alpha_i))$ for every $i \in I$ (Step 3c in Functionality 2.6.13). (Recall that the adversary has no input to F^{mult}_{VSS} in the case that the dealer is honest.)*
3. *$\mathcal{S}$ chooses $t-1$ random degree-t polynomials $D_2(x), \ldots, D_t(x)$.*
4. *For every $i \in I$, $\mathcal{S}$ computes:*
$$D_1(\alpha_i) = (\alpha_i)^{-1} \cdot \left(A(\alpha_i) \cdot B(\alpha_i) - C(\alpha_i) - \sum_{\ell=2}^{t} (\alpha_i)^\ell \cdot D_\ell(\alpha_i) \right)$$
5. *$\mathcal{S}$ simulates the F_{VSS} invocations, and simulates every corrupted party P_i (for every $i \in I$) internally receiving outputs $C(\alpha_i)$, $D_1(\alpha_i), \ldots, D_t(\alpha_i)$ from F_{VSS} in the respective invocations (Steps 1e and 1f of Protocol 2.6.15).*
6. *For every $k \in I$ for which $\mathcal{A}$ instructs the corrupted party P_k to broadcast a $(\mathsf{complaint}, k)$ message, $\mathcal{S}$ simulates the* complaint resolution phase *(Step 3 of Protocol 2.6.15) by internally simulating the $t+3$ invocations of F^k_{eval}: For every $i \in I$, the simulator internally hands the adversary $(A(\alpha_i), A(\alpha_k))$, $(B(\alpha_i), B(\alpha_k))$, $(C(\alpha_i), C(\alpha_k))$ and $\{(D_\ell(\alpha_i), D_\ell(\alpha_k))\}_{\ell=1}^t$ as P_i's outputs from the respective invocation of F^k_{eval}.*
7. *$\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs, and halts.*

We prove that for every for every $I \subseteq [n]$, every $z \in \{0,1\}^*$ and all vectors of inputs $\vec{x}$,

$$\left\{ \text{IDEAL}_{F^{mult}_{VSS}, \mathcal{S}(z), I}(\vec{x}) \right\} \equiv \left\{ \text{HYBRID}^{F_{VSS}, F_{eval}}_{\pi, \mathcal{A}(z), I}(\vec{x}) \right\}.$$

We begin by showing that the outputs of the honest parties are distributed identically in an ideal execution with $\mathcal{S}$ and in a real execution of the protocol with $\mathcal{A}$ (the protocol is actually run in the (F_{VSS}, F_{eval})-hybrid model, but we say "real" execution to make for a less cumbersome description). Then, we show that the view of the adversary is distributed identically, when the output of the honest parties is given.

The honest parties' outputs. We analyze the distribution of the output of honest parties. Let the inputs of the honest parties be shares of the degree-t polynomials $A(x)$ and $B(x)$. Then, in the ideal model the trusted party chooses a polynomial $C(x)$ that is distributed uniformly at random in $\mathcal{P}^{A(0)\cdot B(0),t}$, and sends each party P_j the output $(A(\alpha_j), B(\alpha_j), C(\alpha_j))$.

In contrast, in a protocol execution, the honest dealer chooses $D_1(x), \ldots, D_t(x)$ and then derives $C(x)$ from $D(x) = A(x)\cdot B(x)$ and the polynomial $D_1(x), \ldots, D_t(x)$; see Steps 1a to 1d in Protocol 2.6.15. It is immediate that the polynomial C computed by the dealer in the protocol is such that $C(0) = A(0) \cdot B(0)$ and that each honest party P_j outputs $C(\alpha_j)$. This is due to the fact that, since the dealer is honest, all the complaints that are broadcasted are resolved with the result that $\tilde{C}(\alpha_k) \neq \tilde{C}'(\alpha_k)$, and so the *reject phase* is never reached. Thus, the honest parties output shares of a polynomial $C(x)$ with the correct constant term. It remains to show that $C(x)$ is of degree-t and is *uniformly distributed* in $\mathcal{P}^{A(0)\cdot B(0),t}$. In the discussion above, we have already shown that $\deg(C) \leq t$, and that every coefficient of $C(x)$ is random, except for the constant term.

We conclude that $C(x)$ as computed by the honest parties is uniformly distributed in $\mathcal{P}^{A(0)\cdot B(0),t}$ and so the distribution over the outputs of the honest parties in a real protocol execution is identical to their output in an ideal execution.

The adversary's view. We now show that the view of the adversary is identical in the real protocol and ideal executions, given the honest parties' inputs and outputs. Fix the honest parties' input shares $(A(\alpha_j), B(\alpha_j))$ and output shares $C(\alpha_j)$ for every $j \notin I$. Observe that these values fully determine the degree-t polynomials $A(x), B(x), C(x)$ since there are more than t points. Now, the view of the adversary in a real protocol execution is comprised of the shares

$$\left\{D_1(\alpha_i)\right\}_{i\in I}, \ldots, \left\{D_t(\alpha_i)\right\}_{i\in I}, \left\{C(\alpha_i)\right\}_{i\in I} \tag{2.6.6}$$

received from the F_{VSS} invocations, and of the messages from the complaint resolution phase. In the complaint resolution phase, the adversary merely sees some subset of the shares in Eq. (2.6.6). This is due to the fact that in this corruption case where the dealer is honest, only corrupted parties complain. Since $C(x)$ is fixed (since we are conditioning over the input and output of the honest parties), we have that it suffices for us to show that the $D_1(\alpha_i), \ldots, D_t(\alpha_i)$ values are identically distributed in an ideal execution and in a real protocol execution.

Formally, denote by $D_1^S(x), \ldots, D_t^S(x)$ the polynomials chosen by $\mathcal{S}$ in the simulation, and by $D_1(x), \ldots, D_t(x)$ the polynomials chosen by the honest dealer in a protocol execution. Then, it suffices to prove that

$$\left\{D_1^S(\alpha_i), \ldots, D_t^S(\alpha_i) \mid A(x), B(x), C(x)\right\}_{i\in I} \equiv \left\{D_1(\alpha_i), \ldots, D_t(\alpha_i) \mid A(x), B(x), C(x)\right\}_{i\in I} \tag{2.6.7}$$

In order to prove this, we show that for every $\ell = 1, \ldots, t$,

$$\begin{aligned}&\left\{D_\ell^S(\alpha_i) \mid A(x), B(x), C(x), D_{\ell+1}^S(\alpha_i), \ldots, D_t^S(\alpha_i)\right\}_{i\in I} \\ &\qquad \equiv \left\{D_\ell(\alpha_i) \mid A(x), B(x), C(x), D_{\ell+1}(\alpha_i), \ldots, D_t(\alpha_i)\right\}_{i\in I} .\end{aligned} \tag{2.6.8}$$

Combining all of the above (from $\ell = t$ downto $\ell = 1$), we derive Eq. (2.6.7).

We begin by proving Eq. (2.6.8) for $\ell > 1$, and leave the case of $\ell = 1$ for last. Let

$\ell \in \{2, \ldots, t\}$. It is clear that the points $\{D_\ell^S(\alpha_i)\}_{i \in I}$ are uniformly distributed, because the simulator $\mathcal{S}$ chose $D_\ell^S(x)$ uniformly at random, and independently of $A(x), B(x), C(x)$ and $D_{\ell+1}^S(x), \ldots, D_t^S(x)$. In contrast, in the protocol, there seems to be dependence between $D_\ell(x)$ and the polynomials $A(x), B(x), C(x)$ and $D_{\ell+1}(x), \ldots, D_t(x)$. In order to see that this is not a problem, note that

$$D_\ell(x) = r_{\ell,0} + r_{\ell,1} \cdot x + \cdots + r_{\ell,t-1} \cdot x^{t-1} + \left(d_{\ell+t} - \sum_{m=\ell+1}^{t} r_{m,t+\ell-m} \right) \cdot x^t$$

where the values $r_{\ell,0}, \ldots, r_{\ell,t-1}$ are all random and do *not* appear in any of the polynomials $D_{\ell+1}(x), \ldots, D_t(x)$, nor of course in $A(x)$ or $B(x)$; see Table 2.1. Thus, the only dependency is in the t^{th} coefficient (since the values $r_{m,t+\ell-m}$ appear in the polynomials $D_{\ell+1}(x), \ldots, D_t(x)$). However, by Claim 2.3.5 it holds that if $D_\ell(x)$ is a degree-t polynomial in which its *first* t coefficients are uniformly distributed, then any t points $\{D_\ell(\alpha_i)\}_{i \in I}$ are uniformly distributed. Finally, regarding the polynomial $C(x)$ observe that the m^{th} coefficient of $C(x)$, for $1 \leq m \leq t$ in the real protocol includes the random value $r_{1,m-1}$ (that appears in no other polynomials; see Table 2.1), and the constant term is always $A(0) \cdot B(0)$. Since $r_{1,m-1}$ are random and appear only in $D_1(x)$, this implies that $D_\ell(x)$ is independent of $C(x)$. This completes the proof of Eq. (2.6.8) for $\ell > 1$.

It remains now to prove Eq. (2.6.8) for the case $\ell = 1$; i.e., to show that the points $\{D_1^S(\alpha_i)\}_{i \in I}$ and $\{D_1(\alpha_i)\}_{i \in I}$ are identically distributed, conditioned on $A(x), B(x), C(x)$ and all the points $\{D_2(\alpha_i), \ldots, D_t(\alpha_i)\}_{i \in I}$. Observe that the polynomial $D_1(x)$ chosen by the dealer in the real protocol is fully determined by $C(x)$ and $D_2(x), \ldots, D_t(x)$. Indeed, an equivalent way of describing the dealer is for it to choose all $D_2(x), \ldots, D_t(x)$ as before, to choose $C(x)$ uniformly at random in $\mathcal{P}^{a \cdot b, t}$ and then to choose $D_1(x)$ as follows:

$$D_1(x) = x^{-1} \cdot \left(A(x) \cdot B(x) - C(x) - \sum_{k=2}^{t} x^k \cdot D_k(x) \right). \tag{2.6.9}$$

Thus, once $D_2(x), \ldots, D_t(x), A(x), B(x), C(x)$ are fixed, the polynomial $D_1(x)$ is fully determined. Likewise, in the simulation, the points $\{D_1(\alpha_i)\}_{i \in I}$ are fully determined by $\{D_2(\alpha_i), \ldots, D_t(\alpha_i), A(\alpha_i), B(\alpha_i), C(\alpha_i)\}_{i \in I}$. Thus, the actual values $\{D_1(\alpha_i)\}_{i \in I}$ are the same in the ideal execution and real protocol execution, when conditioning as in Eq. (2.6.8). (Intuitively, the above proof shows that the distribution over the polynomials in a real execution is identical to choosing a random polynomial $C(x) \in \mathcal{P}^{A(0) \cdot B(0), t}$ and random points $D_2(\alpha_i), \ldots, D_t(\alpha_i)$, and then choosing random polynomials $D_2(x), \ldots, D_t(x)$ that pass through these points, and determining $D_1(x)$ so that Eq. (2.6.9) holds.)

We conclude that the view of the corrupted parties in the protocol is identically distributed to the adversary's view in the ideal simulation, given the outputs of the honest parties. Combining this with the fact that the outputs of the honest parties are identically distributed in the protocol and ideal executions, we conclude that the joint distributions of the adversary's output and the honest parties' outputs in the ideal and real executions are identical.

Case 2 – the dealer is corrupted: In the case that the dealer P_1 is corrupted, the ideal adversary sends a polynomial $C(x)$ to the trusted party computing F_{VSS}^{mult}. If the polynomial is of degree at most t and has the constant term $A(0) \cdot B(0)$, then this polynomial determines the output of the honest parties. Otherwise, the polynomial $C(x)$ determining the output shares of the honest parties is the constant polynomial equalling $A(0) \cdot B(0)$ everywhere.

Intuitively, the protocol is secure in this corruption case because any deviation by a corrupted dealer from the prescribed instructions is unequivocally detected in the verify phase via the F_{eval} invocations. Observe also that in the (F_{VSS}, F_{eval})-hybrid model, the adversary receives no messages from the honest parties except for those sent in the complaint phase. However, the adversary already knows the results of these complaints in any case. In particular, since the adversary (in the ideal model) knows $A(x)$ and $B(x)$, and it dealt the polynomials $C(x), D_1(x), \ldots, D_t(x)$, it knows exactly where a complaint will be sent and it knows the values revealed by the F_{eval}^k calls.

We now formally describe the simulator (recall that the ideal adversary receives the polynomials $A(x), B(x)$ from F_{VSS}^{mult}; this is used to enable the simulation).

The simulator $\mathcal{S}$:

1. *$\mathcal{S}$ internally invokes $\mathcal{A}$ with the auxiliary input z.*

2. *$\mathcal{S}$ externally receives the polynomials $A(x), B(x)$ from F_{VSS}^{mult} (Step 4a in Functionality 2.6.13).*

3. *$\mathcal{S}$ internally receives the polynomials $C(x)$, $D_1(x), \ldots, D_t(x)$ that $\mathcal{A}$ instructs the corrupted dealer to use in the F_{VSS} invocations (Steps 1e and 1f of Protocol 2.6.15).*

4. *If $\deg(C) > t$ or if $\deg(D_\ell) > t$ for some $1 \leq \ell \leq t$, then $\mathcal{S}$ proceeds to Step 8 below (simulating reject).*

5. *For every $k \notin I$ such that $C(\alpha_k) \neq A(\alpha_k) \cdot B(\alpha_k) - \sum_{\ell=1}^{t} (\alpha_k)^\ell \cdot D_\ell(\alpha_k)$, the simulator $\mathcal{S}$ simulates the honest party P_k broadcasting the message $(\mathsf{complaint}, k)$. Then, $\mathcal{S}$ internally simulates the "complaint resolution phase" (Step 3 in Protocol 2.6.15). In this phase, $\mathcal{S}$ uses the polynomials $A(x), B(x), C(x)$ and $D_1(x), \ldots, D_t(x)$ in order to compute the values output in the F_{eval}^k invocations. If there exists such a $k \notin I$ as above, then $\mathcal{S}$ proceeds to Step 8 below.*

6. *For every $(\mathsf{complaint}, k)$ message (with $k \in I$) that was internally broadcast by the adversary $\mathcal{A}$ in the name of a corrupted party P_k, the simulator $\mathcal{S}$ uses the polynomials $A(x), B(x), C(x)$ and $D_1(x), \ldots, D_t(x)$ in order to compute the values output in the F_{eval}^k invocations, as above. Then, if there exists an $i \in I$ such that $C(\alpha_k) \neq A(\alpha_k) \cdot B(\alpha_k) - \sum_{\ell=1}^{t} (\alpha_k)^\ell \cdot D_\ell(\alpha_k)$, simulator $\mathcal{S}$ proceeds to Step 8 below.*

7. *If $\mathcal{S}$ reaches this point, then it externally sends the polynomial $C(x)$ obtained from $\mathcal{A}$ above to F_{VSS}^{mult} (Step 4b in Functionality 2.6.13). It then skips to Step 9 below.*

8. Simulating reject: *(Step 4 in Protocol 2.6.15)*

   (a) *$\mathcal{S}$ externally sends $\hat{C}(x) = x^{t+1}$ to the trusted party computing F_{VSS}^{mult} (i.e., $\mathcal{S}$ sends a polynomial $\hat{C}$ such that $\deg(\hat{C}) > t$).*

   (b) *$\mathcal{S}$ internally simulates every honest party P_j broadcasting $a_j = A(\alpha_j)$ and $b_j = B(\alpha_j)$ as in the reject phase.*

9. *$\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs, and halts.*

The simulator obtains $A(x), B(x)$ from F_{VSS}^{mult} and can therefore compute the actual inputs $a_j = A(\alpha_j)$ and $b_j = B(\alpha_j)$ held by all honest parties P_j ($j \notin I$). Therefore, the view of the adversary in the simulation is clearly identical to its view in a real execution. We now show that the output of the honest parties in the ideal model and in a real protocol execution are identical, *given* the view of the corrupted parties/adversary. We have two cases in the ideal model/simulation:

1. *Case 1 – $\mathcal{S}$ does not simulate reject ($\mathcal{S}$ does not run Step 8):* This case occurs if

   (a) All the polynomials $C(x)$, $D_1(x), \ldots, D_t(x)$ are of degree-t, *and*

   (b) For every $j \notin I$, it holds that $C(\alpha_j) = A(\alpha_j) \cdot B(\alpha_j) - \sum_{\ell=1}^{t} (\alpha_j)^\ell \cdot D_\ell(\alpha_j)$, *and*

   (c) If any corrupt P_i broadcast $(\mathsf{complaint}, i)$ then $C(\alpha_i) = A(\alpha_i) \cdot B(\alpha_i) - \sum_{\ell=1}^{t} (\alpha_i)^\ell \cdot D_\ell(\alpha_i)$.

   The polynomials obtained by $\mathcal{S}$ from $\mathcal{A}$ in the simulation are the same polynomials used by $\mathcal{A}$ in the F_{VSS} calls in the real protocol. Thus, in this case, in the protocol execution it is clear that each honest party P_j will output $C(\alpha_j)$.

   In contrast, in the ideal model, each honest P_j will outputs $C(\alpha_j)$ as long as $\deg(C) \leq t$ and $C(0) = A(0) \cdot B(0)$. Now, let $C'(x) = A(x) \cdot B(x) - \sum_{\ell=1}^{t} x^\ell \cdot D_\ell(x)$. By the definition of C' and the fact that each $D_\ell(x)$ is guaranteed to be of degree-t, we have that $C'(x)$ is of degree at most $2t$. Furthermore, in this case, we know that for every $j \notin I$, it holds that $C(\alpha_j) = A(\alpha_j) \cdot B(\alpha_j) - \sum_{\ell=1}^{t} (\alpha_j)^\ell \cdot D_\ell(\alpha_j) = C'(\alpha_j)$. Thus, $C(x) = C'(x)$ on at least $2t + 1$ points $\{\alpha_j\}_{j \notin I}$. This implies that $C(x) = C'(x)$, and in particular $C(0) = C'(0)$. Since $C'(0) = A(0) \cdot B(0)$ *irrespective* of the choice of the polynomials $D_1(x), \ldots, D_t(x)$, we conclude that $C(0) = A(0) \cdot B(0)$. The fact that $C(x)$ is of degree-t follows from the conditions of this case. Thus, we conclude that in the ideal model, every honest party P_j also outputs $C(\alpha_j)$, exactly as in a protocol execution.

2. *Case 2 – $\mathcal{S}$ simulates reject ($\mathcal{S}$ runs Step 8):* This case occurs if any of (a), (b) or (c) above do not hold. When this occurs in a protocol execution, all honest parties run the reject phase in the real execution and output the value $A(0) \cdot B(0)$. Furthermore, in the ideal model, in any of these cases the simulator $\mathcal{S}$ sends the polynomial $\hat{C}(x) = x^{t+1}$ to F_{VSS}^{mult}. Now, upon input of $C(x)$ with $\deg(C) > t$, functionality F_{VSS}^{mult} sets $C(x) = A(0) \cdot B(0)$ and so all honest parties output the value $A(0) \cdot B(0)$, exactly as in a protocol execution.

This concludes the proof. ■

2.6.7 The F_{mult} Functionality and its Implementation

We are finally ready to show how to securely compute the product of shared values, in the presence of malicious adversaries. As we described in the high-level overview in Section 2.6.1, the multiplication protocol works by first having each party share subshares of its two input shares (using $F_{VSS}^{subshare}$), and then share the product of the shares (using F_{VSS}^{mult}). Finally, given shares of the product of each party's two input shares, a sharing of the product of the *input values* is obtained via a local computation of a linear function by each party.

The functionality. We begin by defining the multiplication functionality for the case of malicious adversaries. In the semi-honest setting, the F_{mult} functionality was defined as follows:

$$F_{mult}\Big((f_a(\alpha_1), f_b(\alpha_1)), \ldots, (f_a(\alpha_n), f_b(\alpha_n))\Big) = \Big(f_{ab}(\alpha_1), \ldots, f_{ab}(\alpha_n)\Big)$$

where f_{ab} is a random degree-t polynomial with constant term $f_a(0) \cdot f_b(0) = a \cdot b$. We stress that unlike in F_{VSS}^{mult}, here the values a and b are the actual values on the incoming wires to the multiplication gate (and not shares).

In the malicious setting, we need to define the functionality with more care. First, the corrupted parties are able to influence the output and determine the shares of the corrupted parties in the output polynomial. In order to see why this is the case, recall that the multiplication works by the parties running F_{VSS}^{mult} multiple times (in each invocation a different party plays the dealer) and then computing a linear function of the subshares obtained. Since each corrupted party can choose which polynomial $C(x)$ is used in F_{VSS}^{mult} when it is the dealer, the adversary can singlehandedly determine the shares of the corrupted parties in the final polynomial that hides the product of the values. This is similar to the problem that arises when running F_{VSS} in parallel, as described in Section 2.6.2. We therefore define the F_{mult} multiplication functionality as a reactive corruption-aware functionality. See Functionality 2.6.16 for a full specification.

FUNCTIONALITY 2.6.16 (Functionality F_{mult} for emulating a multiplication gate)

F_{mult} receives a set of indices $I \subseteq [n]$ and works as follows:

1. The F_{mult} functionality receives the inputs of the honest parties $\{(\beta_j, \gamma_j)\}_{j \notin I}$. Let $f_a(x), f_b(x)$ be the unique degree-t polynomials determined by the points $\{(\alpha_j, \beta_j)\}_{j \notin I}$, $\{(\alpha_j, \gamma_j)\}_{j \notin I}$, respectively. (If such polynomials do not exist then no security is guaranteed; see Footnote 9.)
2. F_{mult} sends $\{(f_a(\alpha_i), f_b(\alpha_i))\}_{i \in I}$ to the (ideal) adversary.[12]
3. F_{mult} receives points $\{\delta_i\}_{i \in I}$ from the (ideal) adversary (if some δ_i is not received, then it is set to equal 0).
4. F_{mult} chooses a random degree-t polynomial $f_{ab}(x)$ under the constraints that:
   (a) $f_{ab}(0) = f_a(0) \cdot f_b(0)$, and
   (b) For every $i \in I$, $f_{ab}(\alpha_i) = \delta_i$.

   (such a degree-t polynomial always exists since $|I| \le t$).
5. The functionality F_{mult} sends the value $f_{ab}(\alpha_j)$ to every honest party P_j $(j \notin I)$.

Before proceeding, we remark that the F_{mult} functionality is sufficient for use in circuit emulation. Specifically, the only difference between it and the definition of multiplication in the semi-honest case is the ability of the adversary to determine its own points. However, since f_{ab} is of degree-t, the ability of $\mathcal{A}$ to determine t points of f_{ab} reveals nothing about $f_{ab}(0) = a \cdot b$. A formal proof of this is given in Section 2.7.

The protocol idea. We are now ready to show how to multiply in the $F_{VSS}^{subshare}$ and F_{VSS}^{mult} hybrid model. Intuitively, the parties first distribute subshares of their shares and subshares of

[12]As with F_{eval} and F_{VSS}^{mult}, the simulator needs to receive the correct shares of the corrupted parties in order to simulate, and so this is also received as output. Since this information is anyway given to the corrupted parties, this makes no difference to the use of the functionality for secure computation.

the product of their shares, using $F_{VSS}^{subshare}$ and F_{VSS}^{mult}, respectively. Note that F_{VSS}^{mult} assumes that the parties already hold correct subshares,; this is achieved by first running $F_{VSS}^{subshare}$ on the input shares. Next, we use the method from [51] to have the parties directly compute shares of the *product of the values* on the input wires, from the subshares of the *product of their shares.* This method is based on the following observation. Let $f_a(x)$ and $f_b(x)$ be two degree-t polynomials such that $f_a(0) = a$ and $f_b(0) = b$, and let $h(x) = f_a(x) \cdot f_b(x) = a \cdot b + h_1 \cdot x + h_2 \cdot x^2 + \ldots + h_{2t} \cdot x^{2t}$. Letting $V_{\vec{\alpha}}$ be the Vandermonde matrix for $\vec{\alpha}$, and recalling that $V_{\vec{\alpha}}$ is invertible, we have that

$$V_{\vec{\alpha}} \cdot \begin{pmatrix} ab \\ h_1 \\ \vdots \\ h_{2t} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} h(\alpha_1) \\ h(\alpha_2) \\ \vdots \\ h(\alpha_n) \end{pmatrix} \quad \text{and so} \quad \begin{pmatrix} ab \\ h_1 \\ \vdots \\ h_{2t} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = V_{\vec{\alpha}}^{-1} \cdot \begin{pmatrix} h(\alpha_1) \\ h(\alpha_2) \\ \vdots \\ h(\alpha_n) \end{pmatrix}.$$

Let $\lambda_1, \ldots, \lambda_n$ be the first row of $V_{\vec{\alpha}}^{-1}$. It follows that

$$a \cdot b = \lambda_1 \cdot h(\alpha_1) + \ldots + \lambda_n \cdot h(\alpha_n) = \lambda_1 \cdot f_a(\alpha_1) \cdot f_b(\alpha_1) + \ldots + \lambda_n \cdot f_a(\alpha_n) \cdot f_b(\alpha_n).$$

Thus the parties simply need to compute a linear combination of the products $f_a(\alpha_\ell) \cdot f_b(\alpha_\ell)$ for $\ell = 1, \ldots, n$. Using $F_{VSS}^{subshare}$ and F_{VSS}^{mult}, as described above, the parties first distribute random shares of the values $f_a(\alpha_\ell) \cdot f_b(\alpha_\ell)$, for every $\ell = 1, \ldots, n$. That is, let $C_1(x), \ldots, C_n(x)$ be random degree-t polynomials such that for every ℓ it holds that $C_\ell(0) = f_a(\alpha_\ell) \cdot f_b(\alpha_\ell)$; the polynomial $C_\ell(x)$ is shared using F_{VSS}^{mult} where P_ℓ is the dealer (since P_ℓ's input shares are $f_a(\alpha_\ell)$ and $f_b(\alpha_\ell)$). Then, the result of the sharing via F_{VSS}^{mult} is that each party P_i holds $C_1(\alpha_i), \ldots, C_n(\alpha_i)$. Thus, each P_i can locally compute $Q(\alpha_i) = \sum_{\ell=1}^{n} \lambda_\ell \cdot C_\ell(\alpha_i)$ and we have that the parties hold shares of the polynomial $Q(x) = \sum_{\ell=1}^{n} \lambda_\ell \cdot C_\ell(x)$. By the fact that $C_\ell(0) = f_a(\alpha_\ell) \cdot f_b(\alpha_\ell)$ for every ℓ, it follows that

$$Q(0) = \sum_{\ell=1}^{n} \lambda_\ell \cdot C_\ell(0) = \sum_{\ell=1}^{n} \lambda_\ell \cdot f_a(\alpha_\ell) \cdot f_b(\alpha_\ell) = a \cdot b. \tag{2.6.10}$$

Furthermore, since all the $C_\ell(x)$ polynomials are of degree-t, the polynomial $Q(x)$ is also of degree-t, implying that the parties hold a valid sharing of $a \cdot b$, as required. Full details of the protocol are given in Protocol 2.6.17.

The correctness of the protocol is based on the above discussion. Intuitively, the protocol is secure since the invocations of $F_{VSS}^{subshare}$ and F_{VSS}^{mult} provide shares to the parties that reveal nothing. However, recall that the adversary's output from $F_{VSS}^{subshare}$ includes the vector of polynomials $\vec{Y}(x) = (g_1(x), \ldots, g_n(x)) \cdot H^T$, where $g_1, \ldots, g_n$ are the polynomials defining the parties' input shares, and H is the parity-check matrix of the appropriate Reed-Solomon code; see Section 2.6.4. In the context of Protocol 2.6.17, this means that the adversary also obtains the vectors of polynomials $\vec{Y}_A(x) = (A_1(x), \ldots, A_n(x)) \cdot H^T$ and $\vec{Y}_B(x) = (B_1(x), \ldots, B_n(x)) \cdot H^T$. Thus, we must also show that these vectors can be generated by the simulator for the adversary. The strategy for doing so is exactly as in the simulation of F_{eval} in Section 2.6.5.

PROTOCOL 2.6.17 (Computing F_{mult} in the ($F_{VSS}^{subshare}$, F_{VSS}^{mult})-hybrid model)

- **Input:** Each party P_i holds a_i, b_i, where $a_i = f_a(\alpha_i)$, $b_i = f_b(\alpha_i)$ for some polynomials $f_a(x), f_b(x)$ of degree t, which hide a, b, respectively. (If not all the points lie on a single degree-t polynomial, then no security guarantees are obtained. See Footnote 9.)
- **Common input:** A field description $\mathbb{F}$ and n distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **Aiding ideal functionality initialization:** Upon invocation, the trusted party computing the corruption-aware functionalities $F_{VSS}^{subshare}$ and F_{VSS}^{mult} receives the set of corrupted parties I.
- **The protocol:**
  1. The parties invoke the $F_{VSS}^{subshare}$ functionality with each party P_i using a_i as its private input. Each party P_i receives back shares $A_1(\alpha_i), \ldots, A_n(\alpha_i)$, and a polynomial $A_i(x)$. (Recall that for every i, the polynomial $A_i(x)$ is of degree-t and $A_i(0) = f_a(\alpha_i) = a_i$.)
  2. The parties invoke the $F_{VSS}^{subshare}$ functionality with each party P_i using b_i as its private input. Each party P_i receives back shares $B_1(\alpha_i), \ldots, B_n(\alpha_i)$, and a polynomial $B_i(x)$.
  3. For every $i = 1, \ldots, n$, the parties invoke the F_{VSS}^{mult} functionality as follows:
     (a) *Inputs:* In the ith invocation, party P_i plays the dealer. All parties P_j ($1 \leq j \leq n$) send F_{VSS}^{mult} their shares $A_i(\alpha_j), B_i(\alpha_j)$.
     (b) *Outputs:* The dealer P_i receives $C_i(x)$ where $C_i(x) \in_R \mathcal{P}^{A_i(0) \cdot B_i(0), t}$, and every party P_j ($1 \leq j \leq n$) receives the value $C_i(\alpha_j)$.
  4. At this stage, each party P_i holds values $C_1(\alpha_i), \ldots, C_n(\alpha_i)$, and locally computes $Q(\alpha_i) = \sum_{\ell=1}^{n} \lambda_\ell \cdot C_\ell(\alpha_i)$, where $(\lambda_1, \ldots, \lambda_n)$ is the first row of the matrix $V_{\vec{\alpha}}^{-1}$.
- **Output:** Each party P_i outputs $Q(\alpha_i)$.

We prove the following:

Theorem 2.6.18 *Let $t < n/3$. Then, Protocol 2.6.17 is t-secure for the F_{mult} functionality in the ($F_{VSS}^{subshare}$, F_{VSS}^{mult})-hybrid model, in the presence of a static malicious adversary.*

Proof: As we have mentioned, in our analysis here we assume that the inputs of the honest parties all lie on two polynomials of degree t; otherwise (vacuous) security is immediate as described in Footnote 9. We have already discussed the motivation behind the protocol and therefore proceed directly to the simulator. The simulator externally interacts with the trusted party computing F_{mult}, internally invokes the adversary $\mathcal{A}$, and simulates the honest parties in Protocol 2.6.17 and the interaction with the $F_{VSS}^{subshare}$ and F_{VSS}^{mult} functionalities.

The simulator $\mathcal{S}$:

1. *$\mathcal{S}$ internally invokes $\mathcal{A}$ with the auxiliary input z.*
2. *$\mathcal{S}$ externally receives from the trusted party computing F_{mult} the values $(f_a(\alpha_i), f_b(\alpha_i))$, for every $i \in I$ (Step 2 of Functionality 2.6.16).*
3. *$\mathcal{S}$* simulates the first invocation of *$F_{VSS}^{subshare}$ (Step 1 of Protocol 2.6.17):*
   (a) *For every $j \notin I$, $\mathcal{S}$ chooses a polynomial $A_j(x) \in_R \mathcal{P}^{0,t}$ uniformly at random.*

(b) $\mathcal{S}$ internally hands $\mathcal{A}$ the values $\{A_j(\alpha_i)\}_{j\notin I; i\in I}$ as if coming from $F_{VSS}^{subshare}$. (Step 3 of Functionality 2.6.7)

(c) $\mathcal{S}$ internally receives from $\mathcal{A}$ a set of polynomials $\{A_i(x)\}_{i\in I}$ (i.e., the inputs of the corrupted parties to $F_{VSS}^{subshare}$). If any polynomial is missing, then $\mathcal{S}$ sets it to be the constant polynomial 0. (Step 4 of Functionality 2.6.7)

(d) For every $i \in I$, $\mathcal{S}$ performs the following checks (exactly as Step 5b in Functionality 2.6.7):

i. $\mathcal{S}$ checks that $A_i(0) = f_a(\alpha_i)$, and

ii. $\mathcal{S}$ checks that the degree of $A_i(x)$ is t.

If both checks pass, then it sets $A'_i(x) = A_i(x)$. Otherwise, $\mathcal{S}$ sets $A'_i(x)$ to be the constant polynomial that equals $f_a(\alpha_i)$ everywhere (recall that $\mathcal{S}$ received $f_a(\alpha_i)$ from F_{mult} in Step 2 and so can carry out this check and set the output to be these values if necessary).

For every $j \notin I$, $\mathcal{S}$ sets $A'_j(x) = A_j(x)$.

(e) $\mathcal{S}$ computes the vector of polynomials $\vec{Y}_A(x)$ that $\mathcal{A}$ expects to receive from $F_{VSS}^{subshare}$ (in a real execution, $\vec{Y}_A(x) = (A_1(x), \ldots, A_n(x)) \cdot H^T$). In order to do this, $\mathcal{S}$ first computes the error vector $\vec{e}^A = (e_1^A, \ldots, e_n^A)$ as follows: for every $j \notin I$ it sets $e_j^A = 0$, and for every $i \in I$ it sets $e_i^A = A_i(0) - f(\alpha_i)$. Then, $\mathcal{S}$ chooses a vector of random polynomials $\vec{Y}_A(x) = (Y_1(x), \ldots, Y_n(x))$ under the constraints that **(a)** *$\vec{Y}_A(0) = (e_1^A, \ldots, e_n^A) \cdot H^T$, and* **(b)** *$\vec{Y}_A(\alpha_i) = (A_1(\alpha_i), \ldots, A_n(\alpha_i)) \cdot H^T$ for every $i \in I$.*

(f) $\mathcal{S}$ internally hands $\mathcal{A}$ its output from $F_{VSS}^{subshare}$. Namely, it hands $\mathcal{A}$ the polynomials $\{A'_i(x)\}_{i\in I}$, the shares $\{A'_1(\alpha_i), \ldots, A'_n(\alpha_i)\}_{i\in I}$, and the vector of polynomials $\vec{Y}_A(x)$ computed above. (Step 6b of Functionality 2.6.7)

4. $\mathcal{S}$ simulates the second invocation of $F_{VSS}^{subshare}$: *This simulation is carried out in an identical way using the points $\{f_b(\alpha_i)\}_{i\in I}$. Let $B_1(x), \ldots, B_n(x)$ and $B'_1(x), \ldots, B'_n(x)$ be the polynomials used by $\mathcal{S}$ in the simulation of this step (and so $\mathcal{A}$ receives from $\mathcal{S}$ as output from $F_{VSS}^{subshare}$ the values $\{B'_i(x)\}_{i\in I}$, $\{B'_1(\alpha_i), \ldots, B'_n(\alpha_i)\}_{i\in I}$ and $\vec{Y}_B(x)$ computed analogously to above).*

   At this point $\mathcal{S}$ holds a set of degree-t polynomials $\{A'_\ell(x), B'_\ell(x)\}_{\ell\in[n]}$, where for every $j \notin I$ it holds that $A'_j(0) = B'_j(0) = 0$, and for every $i \in I$ it holds that $A'_i(0) = f_a(\alpha_i)$ and $B'_i(0) = f_b(\alpha_i)$.

5. For every $j \notin I$, $\mathcal{S}$ simulates the F_{VSS}^{mult} invocation where the honest party P_j is dealer *(Step 3 in Protocol 2.6.17):*

   (a) $\mathcal{S}$ chooses a uniformly distributed polynomial $C'_j(x) \in_R \mathcal{P}^{0,t}$.

   (b) $\mathcal{S}$ internally hands the adversary $\mathcal{A}$ the shares $\{(A'_j(\alpha_i), B'_j(\alpha_i), C'_j(\alpha_i))\}_{i\in I}$, as if coming from F_{VSS}^{mult} (Step 3c in Functionality 2.6.13). .

6. For every $i \in I$, $\mathcal{S}$ simulates the F_{VSS}^{mult} invocation where the corrupted party P_i is dealer:

   (a) $\mathcal{S}$ internally hands the adversary $\mathcal{A}$ the polynomials $(A'_i(x), B'_i(x))$ as if coming from F_{VSS}^{mult} (Step 4a of Functionality 2.6.13).

(b) $\mathcal{S}$ internally receives from $\mathcal{A}$ the input polynomial $C_i(x)$ of the corrupted dealer that $\mathcal{A}$ sends to F_{VSS}^{mult} (Step 4b of Functionality 2.6.13).

i. If the input is a polynomial C_i such that $\deg(C_i) \leq t$ *and* $C_i(0) = A'_i(0) \cdot B'_i(0) = f_a(\alpha_i) \cdot f_b(\alpha_i)$*, then $\mathcal{S}$ sets $C'_i(x) = C_i(x)$.*

ii. Otherwise, $\mathcal{S}$ sets $C'_i(x)$ to be the constant polynomial equalling $f_a(\alpha_i) \cdot f_b(\alpha_i)$ everywhere.

At this point, $\mathcal{S}$ holds polynomials $C'_1(x), \ldots, C'_n(x)$, where for every $j \notin I$ it holds that $C'_j(0) = 0$ and for every $i \in I$ it holds that $C'_j(0) = f_a(\alpha_i) \cdot f_b(\alpha_i)$.

7. *For every $i \in I$, the simulator $\mathcal{S}$ computes $Q(\alpha_i) = \sum_{\ell=1}^{n} \lambda_\ell \cdot C'_\ell(\alpha_i)$, where $C'_1(x), \ldots, C'_n(x)$ are as determined by $\mathcal{S}$ above, and sends the set $\{Q(\alpha_i)\}_{i \in I}$ to the F_{mult} functionality (this is the set $\{\delta_i\}_{i \in I}$ in Step 3 of Functionality 2.6.16).*

8. *$\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.*

The differences between the simulation with $\mathcal{S}$ and $\mathcal{A}$, and a real execution of Protocol 2.6.17 with $\mathcal{A}$ are as follows. First, for every $j \notin I$, $\mathcal{S}$ chooses the polynomials $A'_j(x), B'_j(x)$, and $C'_j(x)$ to have constant terms of 0 instead of constant terms $f_a(\alpha_j), f_b(\alpha_j)$, and $f_a(\alpha_j) \cdot f_b(\alpha_j)$, respectively. Second, the vectors $\vec{Y}_A(x)$ and $\vec{Y}_B(x)$ are computed by $\mathcal{S}$ using the error vector, and not using the actual polynomials $A_1(x), \ldots, A_n(x)$ and $B_1(x), \ldots, B_n(x)$, as computed by $F_{VSS}^{subshare}$ in the protocol execution. Third, in an ideal execution the output shares are generated by F_{mult} choosing a random degree-t polynomial $f_{ab}(x)$ under the constraints that $f_{ab}(0) = f_a(0) \cdot f_b(0)$, and $f_{ab}(\alpha_i) = \delta_i$ for every $i \in I$. In contrast, in a real execution, the output shares are derived from the polynomial $Q(x) = \sum_{\ell=1}^{n} \lambda_\ell \cdot C'_\ell(x)$. Apart from these differences, the executions are identical since $\mathcal{S}$ is able to run the checks of the $F_{VSS}^{subshare}$ and F_{VSS}^{mult} functionalities exactly as they are specified.

Our proof proceeds by constructing intermediate fictitious simulators to bridge between the real and ideal executions.

The fictitious simulator $\mathcal{S}_1$. Let $\mathcal{S}_1$ be exactly the same as $\mathcal{S}$, except that it receives for input the values $f_a(\alpha_j), f_b(\alpha_j)$, for every $j \notin I$. Then, instead of choosing $A'_j(x) \in_R \mathcal{P}^{0,t}$, $B'_j(x) \in_R \mathcal{P}^{0,t}$, and $C'_j(x) \in_R \mathcal{P}^{0,t}$, the fictitious simulator $\mathcal{S}_1$ chooses $A'_j(x) \in_R \mathcal{P}^{f_a(\alpha_j),t}$, $B'_j(x) \in_R \mathcal{P}^{f_b(\alpha_j),t}$, and $C'_j(x) \in_R \mathcal{P}^{f_a(\alpha_j) \cdot f_b(\alpha_j),t}$. We stress that $\mathcal{S}_1$ runs in the ideal model with the same trusted party running F_{mult} as $\mathcal{S}$, and the honest parties receive output as specified by F_{mult} when running with the ideal adversary $\mathcal{S}$ or $\mathcal{S}_1$.

The ideal executions with $\mathcal{S}$ and $\mathcal{S}_1$. We begin by showing that the joint output of the adversary and honest parties is identical in the original simulation by $\mathcal{S}$ and the fictitious simulation by $\mathcal{S}_1$. That is,

$$\left\{ \text{IDEAL}_{F_{mult}, \mathcal{S}(z), I}(\vec{x}) \right\}_{\vec{x} \in (\{0,1\}^*)^n, z \in \{0,1\}^*} \equiv \left\{ \text{IDEAL}_{F_{mult}, \mathcal{S}_1(z'), I}(\vec{x}) \right\}_{\vec{x} \in (\{0,1\}^*)^n, z \in \{0,1\}^*}$$

where z' contains the same z as $\mathcal{A}$ receives, together with the $f_a(\alpha_j), f_b(\alpha_j)$ values for every $j \notin I$. In order to see that the above holds, observe that both $\mathcal{S}$ and $\mathcal{S}_1$ can work when given the points of the inputs shares $\{(A'_j(\alpha_i), B'_j(\alpha_i))\}_{i \in I, j \notin I}$ and the outputs shares $\{C'_j(\alpha_i)\}_{i \in I; j \notin I}$ and they don't actually need the polynomials themselves. Furthermore, the only difference between $\mathcal{S}$ and $\mathcal{S}_1$ is whether these polynomials are chosen with zero constant terms, or with the "correct" ones. That is, there exists a machine $\mathcal{M}$ that receives points $\{A'_j(\alpha_i), B'_j(\alpha_i)_{i\in I;j\notin I}, \{C'_j(\alpha_i)\}_{i\in I;j\notin I}$ and runs the simulation strategy with $\mathcal{A}$ while interacting with F_{mult} in an ideal execution, such that:

- If $A'_j(0) = B'_j(0) = C'_j(0) = 0$ then the joint output of $\mathcal{M}$ and the honest parties in the ideal execution is exactly that of $\text{IDEAL}_{F_{mult},\mathcal{S}(z),I}(\vec{x})$; i.e., an ideal execution with the original simulator.
- If $A'_j(0) = f_a(\alpha_j)$, $B'_j(0) = f_b(\alpha_j)$ and $C'_j(0) = f_a(\alpha_j)\cdot f_b(\alpha_j)$ then the joint output of $\mathcal{M}$ and the honest parties in the ideal execution is exactly that of $\text{IDEAL}_{F_{mult},\mathcal{S}_1(z'),I}(\vec{x})$; i.e., an ideal execution with the fictitious simulator $\mathcal{S}_1$.

By Claim 2.3.4, the points $\{A'_j(\alpha_i), B'_j(\alpha_i), C'_j(\alpha_i)\}_{i\in I;j\notin I}$ when $A'_j(0) = B'_j(0) = C'_j(0) = 0$ are identically distributed to the points $\{A'_j(\alpha_i), B'_j(\alpha_i), C'_j(\alpha_i)\}_{i\in I;j\notin I}$ when $A'_j(0) = f_a(\alpha_j)$, $B'_j(0) = f_b(\alpha_j)$ and $C'_j(0) = f_a(\alpha_j)\cdot f_b(\alpha_j)$. Thus, the joint outputs of the adversary and honest parties in both simulations are identical.

The fictitious simulator $\mathcal{S}_2$. Let $\mathcal{S}_2$ be exactly the same as $\mathcal{S}_1$, except that instead of computing $\vec{Y}_A(x)$ and $\vec{Y}_B(x)$ via the error vectors $(e_1^A,\ldots,e_n^A)$ and $(e_1^B,\ldots,e_n^B)$, it computes them like in a real execution. Specifically, it uses the actual polynomials $A_1(x),\ldots,A_n(x)$; observe that $\mathcal{S}_2$ has these polynomials since it chose them.[13] The fact that

$$\left\{\text{IDEAL}_{F_{mult},\mathcal{S}_2(z'),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n,z\in\{0,1\}^*} \equiv \left\{\text{IDEAL}_{F_{mult},\mathcal{S}_1(z'),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n,z\in\{0,1\}^*}$$

follows from exactly the same argument as in F_{eval} regarding the construction of the vector of polynomials $\vec{Y}(x)$, using the special property of the Syndrome function.

An ideal execution with $\mathcal{S}_2$ and a real protocol execution. It remains to show that the joint outputs of the adversary and honest parties are identical in a real protocol execution and in an ideal execution with $\mathcal{S}_2$:

$$\left\{\text{HYBRID}^{F_{VSS}^{subshare},F_{VSS}^{mult}}_{\pi,\mathcal{A}(z),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n,z\in\{0,1\}^*} \equiv \left\{\text{IDEAL}_{F_{mult},\mathcal{S}_2(z'),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n,z\in\{0,1\}^*}.$$

The only difference between these two executions is the way the polynomial defining the output is chosen. Recall that in an ideal execution the output shares are generated by F_{mult} choosing a random degree-t polynomial $f_{ab}(x)$ under the constraints that $f_{ab}(0) = f_a(0)\cdot f_b(0)$, and $f_{ab}(\alpha_i) = \delta_i$ for every $i\in I$. In contrast, in a real execution, the output shares are derived from the polynomial $Q(x) = \sum_{\ell=1}^n \lambda_\ell \cdot C'_\ell(x)$. However, by the way that $\mathcal{S}_2$ is defined, we have that each $\delta_i = Q(\alpha_i) = \sum_{\ell=1}^n \lambda_\ell\cdot C'_\ell(\alpha_i)$ where all polynomials $C'_1(x),\ldots,C'_n(x)$ are chosen with the correct constant terms. Thus, it remains to show that the following distributions are identical:

[13]We remark that the original $\mathcal{S}$ could not work in this way since our proof that the simulations by $\mathcal{S}$ and $\mathcal{S}_1$ are identical uses the fact that the points $\{A'_j(\alpha_i), B'_j(\alpha_i)_{i\in I;j\notin I}, \{C'_j(\alpha_i)\}_{i\in I;j\notin I}$ alone suffice for simulation. This is true when computing $\vec{Y}_A(x)$ and $\vec{Y}_B(x)$ via the error vectors, but not when computing them from the actual polynomials as $\mathcal{S}_2$ does.

- *Ideal with $\mathcal{S}_2$:* Choose a degree-t polynomial $f_{ab}(x)$ at random under the constraints that $f_{ab}(0) = f_a(0) \cdot f_b(0)$, and $f_{ab}(\alpha_i) = Q(\alpha_i) = \sum_{\ell=1}^{n} \lambda_\ell \cdot C'_\ell(\alpha_i)$ for every $i \in I$.

- *Real execution:* Compute $f_{ab}(x) = Q(x) = \sum_{\ell=1}^{n} \lambda_\ell \cdot C'_\ell(x)$.

We stress that in both cases, the polynomials $C'_1(x), \ldots, C'_n(x)$ have exactly the same distribution.

Observe that if $|I| = t$, then the constraints in the ideal execution with $\mathcal{S}_2$ fully define $f_{ab}(x)$ to be exactly the same polynomial as in the real execution (this is due to the fact that the constraints define $t+1$ points on a degree-t polynomial).

If $|I| < t$, then the polynomial $f_{ab}(x)$ in the ideal execution with $\mathcal{S}_2$ can be chosen by choosing $t - |I|$ random values $\beta_\ell \in_R \mathbb{F}$ (for $\ell \notin I$) and letting $f_{ab}(x)$ be the unique polynomial fulfilling the given constraints and passing through the points $(\alpha_\ell, \beta_\ell)$. Consider now the polynomial $f_{ab}(x)$ generated in a real execution. Fix any $j \notin I$. By the way that Protocol 2.6.17 works, $C'_j(x)$ is a random polynomial under the constraint that $C'_j(0) = f_a(\alpha_j) \cdot f_b(\alpha_j)$. By Corollary 2.3.3, given points $\{(\alpha_i, C'_j(\alpha_i))\}_{i \in I}$ and a "secret" $s = C'_j(0)$, it holds that any subset of $t - |I|$ points of $\{C'_j(\alpha_\ell)\}_{\ell \notin I}$ are *uniformly distributed* (note that none of the points in $\{C'_j(\alpha_\ell)\}_{\ell \notin I}$ are seen by the adversary). This implies that for any $t - |I|$ points α_ℓ (with $\ell \notin I$) the points $f_{ab}(\alpha_\ell)$ in the polynomial $f_{ab}(x)$ computed in a real execution are uniformly distributed. This is therefore exactly the same as choosing $t - |I|$ values $\beta_\ell \in_R \mathbb{F}$ at random (with $\ell \notin I$), and setting f_{ab} to be the unique polynomial such that $f_{ab}(\alpha_\ell) = \beta_\ell$ in addition to the above constraints. Thus, the polynomials $f_{ab}(x)$ computed in an ideal execution with $\mathcal{S}_2$ and in a real execution are identically distributed. This implies that the $\text{HYBRID}_{\pi,\mathcal{A}(z),I}^{F_{VSS}^{subshare}, F_{VSS}^{mult}}(\vec{x})$ and $\text{IDEAL}_{F_{mult},\mathcal{S}_2(z'),I}(\vec{x})$ distributions are identical, as required. ■

Securely computing F_{mult} in the plain model. The following corollary is obtained by combining the following:

- Theorem 2.5.8 (securely compute F_{VSS} in the plain model),
- Theorem 2.6.6 (securely compute F_{mat}^{A} in the F_{VSS}-hybrid model),
- Theorem 2.6.9 (securely compute $F_{VSS}^{subshare}$ in the F_{mat}^{A}-hybrid model),
- Theorem 2.6.12 (securely compute F_{eval} in the $F_{VSS}^{subshare}$-hybrid model),
- Theorem 2.6.14 (securely compute F_{VSS}^{mult} in the F_{VSS}, F_{eval}-hybrid model), and
- Theorem 2.6.18 (securely compute F_{mult} in the $F_{VSS}^{subshare}, F_{VSS}^{mult}$-hybrid model)

and using the modular sequential composition theorem of [27]. We have:

Corollary 2.6.19 *Let $t < n/3$. Then, there exists a protocol that is t-secure for F_{mult} functionality in the plain model with private channels, in the presence of a static malicious adversary.*

More efficient constant-round multiplication [8]. The protocol that we have presented is very close to that described by BGW. However, it is possible to use these techniques to achieve a more efficient multiplication protocol. For example, observe that if the parties already hold shares of all other parties' shares, then these can be used directly in F_{VSS}^{mult} without running $F_{VSS}^{subshare}$ at all. Now, the verifiable secret sharing protocol of [22] presented in Section 2.5 is based on bivariate polynomials, and so all parties do indeed receive shares of all other parties' shares. This means that it is possible to modify Protocol 2.6.17 so that the parties proceed directly to F_{VSS}^{mult} without using $F_{VSS}^{subshare}$ at all. Furthermore, the output of each party P_i in F_{VSS}^{mult} is the share $C(\alpha_i)$ received via the F_{VSS} functionality; see Protocol 2.6.15. Once again, using VSS based on bivariate polynomials, this means that the parties can actually output the shares of all other parties' shares as well. Applying the linear computation of $Q(x)$ to these bivariate shares, we conclude that it is possible to include the shares of all other parties as additional output from Protocol 2.6.17. Thus, the next time that F_{mult} is called, the parties will again already have the shares of all other parties' shares and $F_{VSS}^{subshare}$ need not be called. This is a significant efficiency improvement. (Note that unless some of the parties behave maliciously, F_{VSS}^{mult} itself requires $t+1$ invocations of F_{VSS} and nothing else. With this efficiency improvement, we have that the entire cost of F_{mult} is $n \cdot (t+1)$ invocations of F_{VSS}.) See [8] for more details on this and other ways to further utilize the properties of bivariate secret sharing in order to obtain simpler and much more efficient multiplication protocols.

We remark that there exist protocols that are *not* constant round and have far more efficient communication complexity; see [18] for such a protocol. In addition, in the case of $t < n/4$, there is a much more efficient solution for constant-round multiplication presented in BGW itself; see Section 2.9 for a brief description.

2.7 Secure Computation in the (F_{VSS}, F_{mult})-Hybrid Model

2.7.1 Securely Computing any Functionality

In this section we show how to t-securely compute any functionality f in the (F_{VSS}, F_{mult})-hybrid model, in the presence of a malicious adversary controlling any $t < n/3$ parties. We also assume that all inputs are in a known field $\mathbb{F}$ (with $|\mathbb{F}| > n$), and that the parties all have an arithmetic circuit C over $\mathbb{F}$ that computes f. As in the semi-honest case, we assume that $f : \mathbb{F}^n \to \mathbb{F}^n$ and so the input and output of each party is a single field element.

The protocol here is almost identical to Protocol 2.4.1 for the semi-honest case; the only difference is that the verifiable secret-sharing functionality F_{VSS} is used in the input stage, and the F_{mult} functionality used for multiplication gates in the computation stage is the corruption-aware one defined for the case of malicious adversaries (see Section 2.6.7). See Section 2.5.4 for the definition of F_{VSS} (Functionality 2.5.5), and see Functionality 2.6.16 for the definition of F_{mult}. Observe that the definition of F_{VSS} is such that the effect is identical to that of Shamir secret sharing in the presence of semi-honest adversaries. Furthermore, the correctness of F_{mult} ensures that at every intermediate stage the (honest) parties hold correct shares on the wires of the circuit. In addition, observe that F_{mult} reveals nothing to the adversary except for its points on the input wires, which it already knows. Thus, the adversary learns nothing in the computation stage, and after this stage the parties all hold correct shares on the circuit-output wires. The protocol is therefore concluded by having the parties send their shares on the output wires to the appropriate recipients (i.e., if party P_j is supposed to receive the output on a certain wire, then all parties send their shares on that wire to P_j). This step introduces a difficulty that does not arise in the semi-honest setting; some of the parties may send *incorrect* values on these wires. Nevertheless, as we have seen, this can be easily solved since it is guaranteed that more than two-thirds of the shares are correct and so each party can apply Reed-Solomon decoding to ensure that the final output obtained is correct. See Protocol 2.7.1 for full details.

PROTOCOL 2.7.1 (t-Secure Computation of f in the (F_{mult}, F_{VSS})-Hybrid Model)

- **Inputs:** Each party P_i has an input $x_i \in \mathbb{F}$.
- **Common input:** Each party P_i holds an arithmetic circuit C over a field $\mathbb{F}$ of size greater than n, such that for every $\vec{x} \in \mathbb{F}^n$ it holds that $C(\vec{x}) = f(\vec{x})$, where $f : \mathbb{F}^n \to \mathbb{F}^n$. The parties also hold a description of $\mathbb{F}$ and distinct non-zero values $\alpha_1, \ldots, \alpha_n$ in $\mathbb{F}$.
- **Aiding ideal functionality initialization:** Upon invocation, the trusted parties computing the (fictitiously corruption-aware) functionality F_{VSS} and the corruption-aware functionality F_{mult} receive the set of corrupted parties I.
- **The protocol:**
  1. **The input sharing stage:**
     (a) Each party P_i chooses a polynomial $q_i(x)$ uniformly at random from the set $\mathcal{P}^{x_i,t}$ of degree-t polynomials with constant-term x_i. Then, P_i invokes the F_{VSS} functionality as dealer, using $q_i(x)$ as its input.
     (b) Each party P_i records the values $q_1(\alpha_i), \ldots, q_n(\alpha_i)$ that it received from the F_{VSS} functionality invocations. If the output from F_{VSS} is $\perp$ for any of these values, P_i replaces the value with 0.
  2. **The circuit emulation stage:** Let $G_1, \ldots, G_\ell$ be a predetermined topological ordering of the gates of the circuit. For $k = 1, \ldots, \ell$ the parties work as follows:
     - *Case 1 – G_k is an addition gate:* Let β_i^k and γ_i^k be the shares of input wires held by party P_i. Then, P_i defines its share of the output wire to be $\delta_i^k = \beta_i^k + \gamma_i^k$.
     - *Case 2 – G_k is a multiplication-by-a-constant gate with constant c:* Let β_i^k be the share of the input wire held by party P_i. Then, P_i defines its share of the output wire to be $\delta_i^k = c \cdot \beta_i^k$.
     - *Case 3 – G_k is a multiplication gate:* Let β_i^k and γ_i^k be the shares of input wires held by party P_i. Then, P_i sends (β_i^k, γ_i^k) to the ideal functionality F_{mult} and receives back a value δ_i^k. Party P_i defines its share of the output wire to be δ_i^k.
  3. **The output reconstruction stage:**
     (a) Let $o_1, \ldots, o_n$ be the output wires, where party P_i's output is the value on wire o_i. For every $i = 1, \ldots, n$, denote by $\beta_1^i, \ldots, \beta_n^i$ the shares that the parties hold for wire o_i. Then, each P_j sends P_i the share β_j^i.
     (b) Upon receiving all shares, P_i runs the Reed-Solomon decoding procedure on the possible corrupted codeword $(\beta_1^i, \ldots, \beta_n^i)$ to obtain a codeword $(\tilde{\beta}_1^i, \ldots, \tilde{\beta}_n^i)$. Then, P_i computes $\mathsf{reconstruct}_{\vec{\alpha}}(\tilde{\beta}_1^i, \ldots, \tilde{\beta}_n^i)$ and obtains a polynomial $g_i(x)$. Finally, P_i then defines its output to be $g_i(0)$.

We now prove that Protocol 2.7.1 can be used to securely compute any functionality. We stress that the theorem holds for regular functionalities only, and not for corruption-aware functionalities (see Section 2.6.2). This is because not every corruption-aware functionality can be computed by a circuit that receives inputs from the parties only, without having the set of identities of the corrupted parties as auxiliary input (such a circuit is what is needed for Protocol 2.7.1).

Theorem 2.7.2 *Let $f : \mathbb{F}^n \to \mathbb{F}^n$ be any n-ary functionality, and let $t < n/3$. Then, Protocol 2.7.1 (with auxiliary-input C to all parties) is t-secure for f in the (F_{VSS}, F_{mult})-hybrid model, in the presence of a static malicious adversary.*

Proof: Intuitively, security here follows from the fact that a corrupted party in Protocol 2.7.1 cannot do anything but choose its input as it wishes. In order to see this, observe that the entire protocol is comprised of F_{VSS} and F_{mult} calls, and in the latter the adversary receives no information in its output and has no influence whatsoever on the outputs of the honest parties. Finally, the adversary cannot affect the outputs of the honest parties due to the Reed-Solomon decoding carried out in the output stage. The simulator internally invokes $\mathcal{A}$ and simulates the honest parties in the protocol executions and the invocations of F_{VSS} and F_{mult} functionalities and externally interacts with the trusted party computing f. We now formally describe the simulator.

The Simulator $\mathcal{S}$:

- *$\mathcal{S}$ internally invokes $\mathcal{A}$ with its auxiliary input z.*
- **The input sharing stage:**
  1. *For every $j \notin I$, $\mathcal{S}$ chooses a uniformly distributed polynomial $q_j(x) \in_R \mathcal{P}^{0,t}$ (i.e., degree-t polynomial with constant term 0), and for every $i \in I$, it internally sends the adversary $\mathcal{A}$ the shares $q_j(\alpha_i)$ as it expects from the F_{VSS} invocations.*
  2. *For every $i \in I$, $\mathcal{S}$ internally obtains from $\mathcal{A}$ the polynomial $q_i(x)$ that it instructs P_i to send to the F_{VSS} functionality when P_i is the dealer. If* $\deg(q_i(x)) \leq t$, *$\mathcal{S}$ simulates F_{VSS} sending $q_i(\alpha_\ell)$ to P_ℓ for every $\ell \in I$. Otherwise, $\mathcal{S}$ simulates F_{VSS} sending $\perp$ to P_ℓ for every $\ell \in I$, and resets $q_i(x)$ to be a constant polynomial equalling zero everywhere.*
  3. *For every $j \in \{1, \ldots, n\}$, denote the circuit-input wire that receives P_j's input by w_j. Then, for every $i \in I$, simulator $\mathcal{S}$ stores the value $q_j(\alpha_i)$ as the share of P_i on the wire w_j.*
- **Interaction with the trusted party:**
  1. *$\mathcal{S}$ externally sends the trusted party computing f the values $\{x_i = q_i(0)\}_{i \in I}$ as the inputs of the corrupted parties.*
  2. *$\mathcal{S}$ receives from the trusted party the outputs $\{y_i\}_{i \in I}$ of the corrupted parties.*
- **The circuit emulation stage:** *Let $G_1, \ldots, G_\ell$ be the gates of the circuit according to their topological ordering. For $k = 1, \ldots, \ell$:*
  1. Case 1 – G_k is an addition gate: *Let β_i^k and γ_i^k be the shares that $\mathcal{S}$ has stored for the input wires to G_k for the party P_i. Then, for every $i \in I$, $\mathcal{S}$ computes the value $\delta_i^k = \beta_i^k + \gamma_i^k$ as the share of P_i for the output wire of G_k and stores this values.*

2. Case 2 – G_k is a multiplication-by-a-constant gate with constant c: *Let β_i^k be the share that $\mathcal{S}$ has stored for the input wire to G_k for P_i. Then, for every $i \in I$, $\mathcal{S}$ computes the value $\delta_i^k = c \cdot \beta_i^k$ as the share of P_i for the output wire of G_k and stores this value.*
3. Case 3 – G_k is a multiplication gate: *$\mathcal{S}$ internally simulates the trusted party computing F_{mult} for $\mathcal{A}$, as follows. Let β_i^k and γ_i^k be the shares that $\mathcal{S}$ has stored for the input wires to G_k for the party P_i. Then, $\mathcal{S}$ first hands $\{(\beta_i^k, \gamma_i^k)\}_{i\in I}$ to $\mathcal{A}$ as if coming from F_{mult} (see Step 2 of Functionality 2.6.16) Next, it obtains from $\mathcal{A}$ values $\{\delta_i^k\}_{i\in I}$ as the input of the corrupted parties for the functionality F_{mult} (See step 3 of Functionality 2.6.16). If any δ_i^k is not sent, then $\mathcal{S}$ sets $\delta_i^k = 0$. Finally, $\mathcal{S}$ stores δ_i^k as the share of P_i for the output wire of G_k. (Note that the adversary has no output from F_{mult} beyond receiving its own (β_i^k, γ_i^k) values.)*

- **The output reconstruction stage:** *For every $i \in I$, simulator $\mathcal{S}$ works as follows. Denote by o_i the circuit-output wire that contains the output of party P_i, and let $\{\beta_\ell^i\}_{\ell\in I}$ be the shares that $\mathcal{S}$ has stored for wire o_i for all corrupted parties P_ℓ ($\ell \in I$). Then, $\mathcal{S}$ chooses a random polynomial $q_i'(x)$ under the constraint that $q_i'(\alpha_\ell) = \beta_\ell^i$ for all $\ell \in I$, and $q_i'(0) = y_i$, where y_i is the output of P_i received by $\mathcal{S}$ from the trusted party computing f. Finally, for every $j \notin I$, $\mathcal{S}$ simulates the honest party P_j sending $q_i'(\alpha_j)$ to P_i.*

A fictitious simulator $\mathcal{S}'$: We begin by constructing a fictitious simulator $\mathcal{S}'$ that works exactly like $\mathcal{S}$ except that it receives as input all of the input values $\vec{x} = (x_1, \ldots, x_n)$, and chooses the polynomials $q_j(x) \in_R \mathcal{P}^{x_j,t}$ of the honest parties with the correct constant term instead of with constant term 0. Apart from this, $\mathcal{S}'$ works exactly like $\mathcal{S}$ and interacts with a trusted party computing f in the ideal model.

The original and fictitious simulations. We now show that the joint output of the adversary and honest parties is identical in the original and fictitious simulations. That is,

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n, z\in\{0,1\}^*} \equiv \left\{\text{IDEAL}_{f,\mathcal{S}'(\vec{x},z),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n, z\in\{0,1\}^*}. \tag{2.7.1}$$

This follows immediately from the fact that both $\mathcal{S}$ and $\mathcal{S}'$ can work identically when receiving the points $\{q_j(\alpha_i)\}_{i\in I; j\notin I}$ externally. Furthermore, the only difference between them is if $q_j(\alpha_i) \in_R \mathcal{P}^{0,t}$ or $q_j(\alpha_i) \in_R \mathcal{P}^{x_j,t}$, for every $j \notin I$. Thus, there exists a single machine $\mathcal{M}$ that runs in the ideal model with a trusted party computing f, and that receives points $\{q_j(\alpha_i)\}_{i\in I;j\notin I}$ and runs the simulation using these points. Observe that if $q_j(\alpha_i) \in_R \mathcal{P}^{0,t}$ for every $j \notin I$, then the joint output of $\mathcal{M}$ and the honest parties in the ideal execution is exactly the same as in the ideal execution with $\mathcal{S}$. In contrast, if $q_j(\alpha_i) \in_R \mathcal{P}^{x_j,t}$ for every $j \notin I$, then the joint output of $\mathcal{M}$ and the honest parties in the ideal execution is exactly the same as in the ideal execution with the fictitious simulator $\mathcal{S}'$. By Claim 2.3.4, these points are identically distributed in both cases, and thus the joint output of $\mathcal{M}$ and the honest parties are identically distributed in both cases; Eq. (2.7.1) follows.

The fictitious simulation and a protocol execution. We now proceed to show that:

$$\left\{\text{IDEAL}_{f,\mathcal{S}'(\vec{x},z),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n, z\in\{0,1\}^*} \equiv \left\{\text{HYBRID}^{F_{VSS},F_{mult}}_{\pi,\mathcal{A}(z),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n, z\in\{0,1\}^*}.$$

We first claim that the output of the honest parties are identically distributed in the real execution and the alternative simulation. This follows immediately from the fact that the inputs to F_{VSS} fully determine the inputs $\vec{x}$, which in turn fully determine the output of the circuit. In order to see this, observe that F_{mult} always sends shares of the product of the input shares (this holds as long as the honest parties send "correct" inputs which they always do), and the local computation in the case of multiplication-by-a-constant and addition gates is trivially correct. Thus, the honest parties all hold correct shares of the outputs on the circuit-output wires. Finally, by the Reed-Solomon decoding procedure (with code length n and dimension $t+1$), it is possible to correct up to $\frac{n-t}{2} > \frac{3t-t}{2} = t$ errors. Thus, the values sent by the corrupted parties in the output stage have no influence whatsoever on the honest parties' outputs.

Next, we show that the view of the adversary $\mathcal{A}$ in the fictitious simulation with $\mathcal{S}'$ is identical to its view in real protocol execution, conditioned on the honest parties' outputs $\{y_j\}_{j \notin I}$. It is immediate that these views are identical up to the output stage. This is because $\mathcal{S}'$ uses the same polynomials as the honest parties in the input stage, and in the computation stage $\mathcal{A}$ receives no output at all (except for its values on the input wires for multiplication gates which are already known). It thus remains to show that the values $\{q'_i(\alpha_j)\}_{i \in I; j \notin I}$ received by $\mathcal{A}$ from $\mathcal{S}'$ in the output stage are identically distributed to the values received by $\mathcal{A}$ from the honest parties P_j.

Assume for simplicity that the output wire comes directly from a multiplication gate. Then, F_{mult} chooses the polynomial that determines the shares on the wire at random, under the constraint that it has the correct constant term (which in this case we know is y_i, since we have already shown that the honest parties' outputs are correct). Since this is *exactly* how $\mathcal{S}'$ chooses the value, we have that the distributions are identical. This concludes the proof. ■

Putting it all together. We conclude with a corollary that considers the plain model with private channels. The corollary is obtained by combining Theorem 2.5.8 (securely computing F_{VSS} in the plain model), Corollary 2.6.19 (securely computing F_{mult} in the plain model) and Theorem 2.7.2 (securely computing f in the F_{VSS}, F_{mult}-hybrid model), and using the modular sequential composition theorem of [27]:

Corollary 2.7.3 *For every functionality $f : \mathbb{F}^n \to \mathbb{F}^n$ and $t < n/3$, there exists a protocol that is t-secure for f in the plain model with private channels, in the presence of a static malicious adversary.*

2.7.2 Communication and Round Complexity

We begin by summarizing the *communication complexity* of the BGW protocol (as presented here) in the case of malicious adversaries. We consider both the cost in the "optimistic case" where no party deviates from the protocol specification, and in the "pessimistic case" where some party does deviate. We remark that since the protocol achieves perfect security, nothing can be gained by deviating, except possible to make the parties run longer. Thus, in general, one would expect that the typical cost of running the protocol is the "optimistic cost". In addition, we separately count the number of field elements sent over the point-to-point private channels, and the number of elements sent over a broadcast channel. (The "BGW" row in the table counts the overall cost of computing a circuit C with $|C|$ multiplication gates.)

Protocol	**Optimistic Cost**	**Pessimistic Cost**
$\boldsymbol{F_{VSS}}$:	$O(n^2)$ over pt-2-pt No broadcast	$O(n^2)$ over pt-2-pt $O(n^2)$ broadcast
$\boldsymbol{F_{VSS}^{subshare}}$:	$O(n^3)$ over pt-2-pt No broadcast	$O(n^3)$ over pt-2-pt $O(n^3)$ broadcast
$\boldsymbol{F_{eval}}$:	$O(n^3)$ over pt-2-pt No broadcast	$O(n^3)$ over pt-2-pt $O(n^3)$ broadcast
$\boldsymbol{F_{VSS}^{mult}}$:	$O(n^3)$ over pt-2-pt No broadcast	$O(n^5)$ over pt-2-pt $O(n^5)$ broadcast
$\boldsymbol{F_{mult}}$:	$O(n^4)$ over pt-2-pt No broadcast	$O(n^6)$ over pt-2-pt $O(n^6)$ broadcast
BGW:	$O(\lvert C\rvert \cdot n^4)$ over pt-2-pt No broadcast	$O(\lvert C\rvert \cdot n^6)$ over pt-2-pt $O(\lvert C\rvert \cdot n^6)$ broadcast

Regarding *round complexity*, since we use the sequential composition theorem, all calls to functionalities must be sequential. However, in Section 2.8 we will see that all subprotocols can actually be run concurrently, and thus in parallel. In this case, we have that all the protocols for computing F_{VSS}, $F_{VSS}^{subshare}$, F_{eval}, F_{VSS}^{mult} and F_{mult} have a *constant number of rounds.* Thus, each level of the circuit C can be computed in $O(1)$ rounds, and the overall round complexity is linear in the depth of the circuit C. This establishes the complexity bounds stated in Theorem 1.

2.8 Adaptive Security, Composition and the Computational Setting

Our proof of the security of the BGW protocol in the semi-honest and malicious cases relates to the *stand-alone model* and to the case of *static corruptions*. In addition, in the information-theoretic setting, we consider perfectly-secure private channels. In this section, we show that our proof of security for the limited stand-alone model with static corruptions suffices for obtaining security in the much more complex settings of composition and adaptive corruptions (where the latter is for a weaker variant; see below). This is made possible due to the fact that the BGW protocol is *perfectly secure*, and not just statistically secure.

Security under composition. In [75, Theorem 3] it was proven that any protocol that computes a functionality f with perfect security and has a straight-line black-box simulator (as is the case with all of our simulators), securely computes f under the definition of (static) universal composability [28] (or equivalently, concurrent general composition [78]). Using the terminology UC-secure to mean secure under the definition of universal composability, we have the following corollary:

Corollary 2.8.1 *For every functionality f, there exists a protocol for UC-securely computing f in the presence of* static *semi-honest adversaries that corrupt up to $t < n/2$ parties, in the private channels model. Furthermore, there exists a protocol for UC-securely computing f in the presence of* static *malicious adversaries that corrupt up to $t < n/3$ parties, in the private channels model.*

Composition in the computational setting. There are two differences between the computational and information-theoretic settings. First, in the information-theoretic setting there are ideally private channels, whereas in the computational setting it is typically only assumed that there are authenticated channels. Second, in the information-theoretic setting, the adversary does not necessarily run in polynomial time. Nevertheless, as advocated by [53, Sec. 7.6.1] and adopted in Definition 2.2.3, we consider simulators that run in time that is polynomial in the running-time of the adversary. Thus, if the real adversary runs in polynomial-time, then so does the simulator, as required for the computational setting. This is also means that it is possible to replace the ideally private channels with public-key encryption. We state our corollary here for computational security for the most general setting of UC-security (although an analogous corollary can of course be obtained for the more restricted stand-alone model as well). The corollary is obtained by replacing the private channels in Corollary 2.8.1 with UC-secure channels that can be constructed using semantically-secure public-key encryption [28, 30]. We state the corollary only for the case of malicious adversaries since the case of semi-honest adversaries has already been proven in [31] for any $t < n$.

Corollary 2.8.2 *Assuming the existence of semantically-secure public-key encryption, for every functionality f, there exists a protocol for UC-securely computing f in the presence of* static *malicious adversaries that corrupt up to $t < n/3$ parties, in the authenticated channels model.*

We stress that the above protocol requires no common reference string or other setup (beyond that required for obtaining authenticated channels). This is the first full proof of the existence of such a UC-secure protocol.

Adaptive security with inefficient simulation. In general, security in the presence of a static adversary does not imply security in the presence of an adaptive adversary, even for perfectly-secure protocols [29]. This is true, for example, for the definition of security of adaptive adversaries that appears in [27]. However, there is an alternative definition of security (for static and adaptive adversaries) due to [41] that requires a straight-line black-box simulator, and also the existence of a committal round at which point the transcript of the protocol fully defines all of the parties' inputs. Furthermore, it was shown in [29] that security in the presence of static adversaries in the strong sense of [41] *does* imply security in the presence of adaptive adversaries (also in the strong sense of [41]), as long as the simulator is allowed to be inefficient (i.e., the simulator is not required to be of *comparable complexity* to the adversary; see Definition 2.2.3). It turns out that all of the protocols in this paper meet this definition. Thus, applying the result of [29] we can conclude that all of the protocols in this paper are secure in the presence of adaptive adversaries with inefficient simulation, under the definition of [41]. Finally, we observe that any protocol that is secure in the presence of adaptive adversaries under the definition of [41] is also secure in the presence of adaptive adversaries under the definition of [27]. We therefore obtain security in the presence of adaptive adversaries with *inefficient simulation* "for free". This is summarized as follows.

Corollary 2.8.3 *For every functionality f, there exists a protocol for securely computing f in the presence of adaptive semi-honest adversaries that corrupt up to $t < n/2$ parties with, in the private channels model (with inefficient simulation). Furthermore, there exists a protocol for securely computing f in the presence of adaptive malicious adversaries that corrupt up to $t < n/3$ parties, in the private channels model (with inefficient simulation).*

2.9 Multiplication in the Case of $t < n/4$

In this section, we describe how to securely compute shares of the product of shared values, in the presence of a malicious adversary controlling only $t < n/4$ parties. This is much simpler than the case of $t < n/3$, since in this case there is enough redundancy to correct errors in polynomials with degree-$2t$. Due to this, it is similar in spirit to the semi-honest multiplication protocol, using the simplification of [51]. In this section, we provide a full-description of this simpler and more efficient protocol, without a proof of security. In our presentation here, we assume familiarity with the material appearing in Sections 2.6.2, 2.6.3, 2.6.4 and 2.6.7.

High-level description of the protocol. Recall that the multiplication protocol works by having the parties compute a linear function of the product of their shares. That is, each party locally multiplies its two shares, and then subshares the result using a degree-t polynomial. The final result is then a specific linear combination of these subshares. Similarly to the case of $t < n/3$ we need a mechanism that verifies that the corrupted parties have shared the correct products. In this case where $t < n/4$, this can be achieved by directly using the error correction property of the Reed-Solomon code, since we can correct degree-$2t$ polynomials. The high-level protocol is as follows:

- Each party holds inputs a_i and b_i, which are shares of two degree-t polynomials that hide values a and b, respectively.
- Each party locally computes the product $a_i \cdot b_i$. The parties then distribute subshares of $a_i \cdot b_i$ to all other parties in a verifiable way using a variant of the $F_{VSS}^{subshare}$. Observe that the products are points on degree-$2t$ polynomials. Thus, these shares constitute a Reed-Solomon code with parameters $[4t+1, 2t+1, 2t+1]$ for which it is possible to correct up to t errors. There is therefore enough redundancy to correct errors, unlike the case where $t < n/3$ where t errors can not necessarily be corrected on a $2t$-degree polynomial. This enables us to design a variant of the $F_{VSS}^{subshare}$ functionality (Section 2.6.4) that works directly on the products $a_i \cdot b_i$.
- At this point, all parties verifiably hold (degree-t) subshares of the product of the input shares of every party. As shown in [51], shares of the product of the values on the wires can be obtained by computing a linear function of the subshares obtained in the previous step.

In the following, we show how to slightly modify the $F_{VSS}^{subshare}$ *functionality* (Section 2.6.4) to work with the case of $t < n/4$ (as we will explain, the protocol actually remains the same). In addition, we provide a full specification for the protocol that implements the multiplication functionality, F_{mult}; i.e., the modifications to Protocol 2.6.17.

We stress that in the case that $t < n/3$ it is not possible to run $F_{VSS}^{subshare}$ directly on the products $a_i \cdot b_i$ of the input shares since they define a degree-$2t$ polynomial and so at most $\frac{n-2t-1}{2} = t/2$ errors can be corrected. Thus, it is necessary to run $F_{VSS}^{subshare}$ separately on a_i and b_i, and then use the F_{mult} functionality to achieve a sharing of $a_i \cdot b_i$. It follows that in this case of $t < n/4$, there is no need for the involved F_{VSS}^{mult} functionality, making the protocol simpler and more efficient.

The $F_{VSS}^{subshare}$ functionality and protocol. We reconsider the definition of the $F_{VSS}^{subshare}$ functionality, and present the necessary modifications for the functionality. Here, we assume that the inputs of the $3t+1$ honest parties $\{(\alpha_j, \beta_j)\}_{j \notin I}$ define a degree-$2t$ polynomial instead of a degree-t polynomial. The definition of the functionality remains unchanged except for this modification.

We now proceed to show that Protocol 2.6.8 that implements the $F_{VSS}^{subshare}$ functionality works as is also for this case, where the inputs are shares of a degree-$2t$ polynomial. In order to see this, recall that there are two steps in the protocol that may be affected by the change of the inputs and should be reconsidered: (1) the parity check matrix H, which is the parameter for the F_{mat}^{H}-functionality, and (2) Step 3, where each party locally computed the error vector using the syndrome vector (the output of the F_{mat}^{H}), and the error correction procedure of the Reed-Solomon code. These steps could conceivably be different since in this case the parameters of the Reed-Solomon codes are different. Regarding the parity-check matrix, the same matrix is used for both cases. Recall that the case of $t < n/3$ defines a Reed-Solomon code with parameters $[3t+1, t+1, 2t+1]$, and the case of $t < n/4$ defines a code with parameters $[4t+1, 2t+1, 2t+1]$. Moreover, recall that a Reed-Solomon code with parameters $[n, k, n-k+1]$ has a parity-check matrix $H \in \mathbb{F}^{(n-k)\times n}$. In the case of $n = 3t+1$ we have that $k = t+1$ and so $n-k = 2t$. Likewise, in the case of $n = 4t+1$, we have that $k = 2t+1$ and so $n-k = 2t$. It follows that in both case, the parity-check matrix H is of dimension $2t \times n$, and so is the same (of course, for different values of t a different matrix is used, but what we mean is that the protocol description is exactly the same). Next, in Step 3 of the protocol, each party locally executes the Reed-Solomon error correction procedure given the syndrome vector that is obtained using F_{mat}^{H}. This procedure depends on the distance of the code. However, this is $2t+1$ in both cases and so the protocol description remains exactly the same.

The protocol for F_{mult}. We now proceed to the specification of the functionality F_{mult}. As we have mentioned, this protocol is much simpler than Protocol 2.6.17 since the parties can run the $F_{VSS}^{subshare}$ functionality directly on the product of their inputs, instead of first running it on a_i, then on b_i, and then using F_{mult} to obtain a sharing of $a_i \cdot b_i$. The protocol is as follows:

PROTOCOL 2.9.1 (Computing F_{mult} in the $F_{VSS}^{subshare}$-hybrid model (with $t < n/4$))

- **Input:** Each party P_i holds a_i, b_i, where $a_i = f_a(\alpha_i)$, $b_i = f_b(\alpha_i)$ for some polynomials $f_a(x), f_b(x)$ of degree t, which hide a, b, respectively. (If not all the points lie on a single degree-t polynomial, then no security guarantees are obtained. See Footnote 9.)
- **Common input:** A field description $\mathbb{F}$ and n distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **The protocol:**
  1. Each party locally computes $c_i = a_i \cdot b_i$.
  2. The parties invoke the $F_{VSS}^{subshare}$ functionality with each party P_i using c_i as its private input. Each party P_i receives back shares $C_1(\alpha_i), \ldots, C_n(\alpha_i)$, and a polynomial $C_i(x)$. (Recall that for every i, the polynomial $C_i(x)$ is of degree-t and $C_i(0) = c_i = a_i \cdot b_i = f_a(\alpha_i) \cdot f_b(\alpha_i)$)
  3. Each party locally computes $Q(\alpha_i) = \sum_{j=1}^{n} \lambda_j \cdot C_j(\alpha_i)$, where $(\lambda_1, \ldots, \lambda_n)$ is the first row of the matrix $V_{\vec{\alpha}}^{-1}$ (see Section 2.6.7).
- **Output:** Each party P_i outputs $Q(\alpha_i)$.

Chapter 3

Efficient Perfectly-Secure Multiplication Protocol

In the previous chapter, we provided a full description and proof of the BGW protocol. In this chapter we focus on the multiplication protocol of BGW. We observe that with some natural adaptations, one of subprotocols of BGW is redundant and can be saved. As a result, we come up with a new multiplication protocol that is more efficient and simpler. We present a full specification of our multiplication protocol together with a full proof of its security.

3.1 Introduction

3.1.1 An Overview of the Multiplication Protocol

We give a brief overview of the BGW multiplication protocol. More details can be found in Section 2.6.7. Recall that the original BGW multiplication protocol follows the invariant that each wire in the circuit is hidden by a random univariate polynomial $f(x)$ of degree-t, and the share of each party is a point $(\alpha_i, f(\alpha_i))$. The protocol for emulating a multiplication gate works as follows:

1. *Subsharing:* Given shares $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ of values a and b, each party *shares its share* with all other parties. Using the fact that each set of shares $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$ lies on a degree-t polynomial, the honest parties can verify (and correct) the subshares distributed by the corrupted parties. This step is carried out using the functionality $F_{VSS}^{subshare}$ (Section 2.6.4), which guarantees that all parties distribute subshares of the correct shares.

2. *Multiplication of subshares – F_{VSS}^{mult}:* Each party P_i plays the role of dealer in a protocol for which the result is that all parties hold shares (with threshold t) of the product $a_i \cdot b_i$ of its initial shares a_i and b_i. This step uses the fact that all parties hold subshares of a_i, b_i as carried out in the previous item. The subsharing is necessary in order to enable verification that the product $a_i \cdot b_i$ has been shared (see Section 2.6.6).

3. *Linear combination:* As described in [51], once the parties all hold shares of $a_1 b_1, \ldots, a_n b_n$,

they can each carry out a local linear combination of their shares, with the result being that they hold shares $c_1, \ldots, c_n$ of $a \cdot b$ (see Section 2.6.7).

We present a new BGW-based protocol that is more efficient than the original BGW protocol. In a nutshell, this protocol uses the bivariate structure introduced by BGW for the purpose of VSS throughout the entire multiplication protocol. Hirt et al. [65] also observed that the use of the bivariate polynomial can offer efficiency improvements; however they do not utilize this to the fullest extent possible. We will show how this approach enables us to completely avoid the use of $F_{VSS}^{subshare}$ and compute the other subprotocols for the multiplication procedure more efficiently.

In more detail, recall that in the BGW protocol, each wire in the circuit is hidden by a random univariate polynomial of degree-t. In our proposed protocol, each wire in the circuit is hidden by a (random) *bivariate polynomial* $F(x, y)$ of degree-t in both variables (see Section 2.5.3 for a definition of bivariate polynomial). As a result, the share of each party is the pair of degree-t polynomials $\langle F(x, \alpha_i), F(\alpha_i, y)\rangle$. We note that in the BGW protocol the VSS sharing is carried out using a bivariate polynomial; however after the initial sharing the parties resort back to the shares of a univariate polynomial, by setting their shares for further computations to $F(0, \alpha_i)$ (see Section 2.5.4). In contrast, we will preserve the shares of the bivariate but at times will also use univariate polynomials.

3.1.2 $F_{VSS}^{subshare}$ for Free

As described above, in order to carry out the "multiplication of subshares" step, the parties need to each have shares of all the other parties' univariate shares. In the original BGW construction, this is done by the $F_{VSS}^{subshare}$ functionality. Informally speaking, the $F_{VSS}^{subshare}$ functionality is a way for a set of parties to verifiably give out shares of values that are themselves shares. Specifically, assume that the parties $P_1, \ldots, P_n$ hold values $f(\alpha_1), \ldots, f(\alpha_n)$, respectively, where f is a degree-t polynomial. The goal is for each party to *share its share* $f(\alpha_i)$ with all other parties while ensuring that a malicious P_i shares its correct $f(\alpha_i)$ and not something else. The protocol for achieving this sub-sharing is highly non trivial, relies heavily on properties of the error correction and the Reed-Solomon decoding algorithm. Moreover, it involves n invocations of VSS plus the transmission of $O(n^3)$ field elements over private channels (see Section 2.6.4).

Our first important observation is that in the bivariate case the subshares of each share *are already distributed* among the parties. In order to see this, assume that the value on the wire is some element a, and the is hidden using some bivariate polynomial $F(x, y)$. Recall that each party P_i holds shares $\langle F(x, \alpha_i), F(\alpha_i, y)\rangle$. Based on this, we can define the univariate "Shamir" sharing of a via the polynomial $g_0(y) = F(0, y)$; due to the properties of the bivariate sharing, $g_0(y)$ is a univariate polynomial of degree-t that hides a. Furthermore, since each party P_i holds the polynomial $f_i(x) = F(x, \alpha_i)$, it can locally compute its share $a_i = f_i(0) = F(0, \alpha_i) = g_0(\alpha_i)$ on the univariate polynomial $g_0(y)$.

We now claim that for every i, it holds that all the other parties P_j actually already have univariate shares of a_i. These shares of a_i are defined via the polynomial $f_i(x) = F(x, \alpha_i)$. This is due to the fact that each P_j holds the polynomial $g_j(y) = F(\alpha_j, y)$ and so can compute $g_j(\alpha_i) = F(\alpha_j, y) = f_i(\alpha_j)$. In conclusion, $f_i(x)$ is a degree-t polynomial, where its constant term is the actual "Shamir" share of P_i, and each party can locally compute its share on this polynomial. Thus, all of the subshares that are computed via the $F_{VSS}^{subshare}$ functionality in the original BGW protocol can actually be locally computed by each party using the bivariate shares.

An additional important point is that the parties already hold bivariate shares after the input sharing phase (after each party distributes its share using VSS). However, these bivariate shares need to be maintained throughout the circuit emulation phase. This is in fact the technically more involved part of our construction, and requires some modifications from the original protocol.

3.1.3 Our Results

We present a full specification and a full proof of security for the proposed protocol. Our protocol preserves a constant round complexity for a single multiplication, as the original multiplication protocol of BGW. The communication complexity of our protocol in the worst case (i.e., when some parties behave maliciously) is $O(n^5)$ field elements over point-to-point channels and $O(n^4)$ field elements over a broadcast channel. This is in contrast to the original BGW protocol which has worst-case communication complexity of $O(n^6)$ field elements over point-to-point channels and $O(n^6)$ field elements over a broadcast channel. We remark that in the case that no parties actually cheat, both of the protocols have communication complexity of only $O(n^4)$ field elements over point-to-point channels, and require no message broadcasts at all.

We also consider our work as addressing the question whether or not it is possible to construct protocols with round and communication complexity that are both low. Our protocol takes the first step by reducing the communication complexity of BGW and [37] while maintaining constant round complexity per multiplication.

Concurrent composition and adaptive security. Our protocol achieves *perfect security*, as in the original work of BGW. We stress that perfect security is not just a question of aesthetics, but rather provides a substantive advantage over protocols that are only proven statistically secure. First, in [75] it is shown that if a protocol is perfectly secure in the stand-alone setting and has a black-box straight-line simulator, then it is also secure under concurrent general composition, or equivalently, universal composition [28]. Since all our simulations are straight-line, we derive security for universal security for free. Second, in [29] it was shown that any protocol that is proven perfectly secure under the security definition of [41] is also secure in the presence of adaptive adversaries with inefficient simulation. The additional requirements of the definition of [41] clearly hold for all BGW protocols and subprotocols. Thus, we obtain adaptive security, albeit with the weaker guarantee provided by inefficient simulation (in particular, this does not imply adaptive security in the computational setting).

Related work. We compare our protocol to those in the existing literature. The only other protocol for perfectly-secure multiplication for any $t < n/3$ that is constant round (and in particular does not depend on the number of participating parties) is that of Cramer et al. [37]. This protocol works in a different way to the BGW protocol, and has worst-case communication complexity of $O(n^5)$ field elements over point-to-point channels and $O(n^5)$ field elements over a broadcast channel, in contrast to $O(n^4)$ broadcasts in our protocol. Furthermore, in the case that no parties actually cheat, the cost of [37] is $O(n^4)$ field elements over point-to-point channels and $O(n^3)$ field elements over a broadcast channel, in contrast to $O(n^4)$ field elements over point-to-point channels (and no broadcast at all) in our protocol.

There has been a considerable amount of work focused on improving the communication complexity of information-theoretic protocols using the player elimination technique [65, 64, 17, 66, 39, 18]. This work culminated in (amortized) linear communication complexity in [18], providing highly efficient protocols for achieving perfect secure computation. However, all of these works have round complexity that depends on the number of participating parties, and not just on the depth of the circuit being computed. This is inherent in the player elimination technique since every time cheating is detected, two players are eliminated and some computations are repeated by the remaining parties. Thus, this technique yields protocols that have round complexity of at least $\Omega(t)$. We remark that the round complexity of these protocols are actually higher; e.g., the round complexity of [65] is $O(d + n^2)$ where d is the depth of the circuit. Although in many cases player elimination would give a more efficient protocol, there are some cases where it would not; for example, when a low-depth circuit is computed by many parties. In addition, from a theoretical perspective the question of low round and communication complexity is an important one. These protocols are therefore incomparable.

Appendix B. The specification of the protocol is full of detailed and contain full proofs. As a result, the specification of the protocol is not consecutive. A reader who may find it beneficial and more convenient to read the full specification continuously may refer to Appendix B.

3.2 Preliminaries and Definitions

We follow the definitions of perfect secure computation, as presented in Section 2.2. In addition, we work in the corruption-aware model (Section 2.6.2), and our functionalities receive as input the set of corrupted parties $I \subset [n]$.

In the following, we briefly recall the definitions and claims regarding bivariate polynomials (Section 2.5.3), and present some additional claims that are needed for the proofs of our construction. Afterward, we recall the definition of the bivariate secret-sharing functionality, the $\widetilde{F}_{VSS}$ functionality (Section 2.5.5), which plays an essential role in our construction.

3.2.1 Properties of Bivariate Polynomials

We state some claims regarding distributions of univariate and bivariate polynomials. Some of the claims are the bivariate analogues to the univariate claims given in Section 2.3. This section is based on Section 2.5.3.

Recall that a bivariate polynomial of degree-t in both variables is defined as:

$$S(x,y) = \sum_{i=0}^{t}\sum_{j=0}^{t} a_{i,j}x^i y^j \ .$$

We denote by $\mathcal{B}^{s,t}$ the set of all bivariate polynomials of degree-t in both variables with constant term s (and recall that $\mathcal{P}^{s,t}$ denotes the set of all univariate polynomials of degree-t with constant term s). We stated several claims in the context of bivariate polynomials in Section 2.5.3. The first was simply the "interpolation claim" of bivariate polynomials (Claim 2.5.2). This Claim is the analogue to the interpolation of univariate polynomial, where here instead of "$t+1$ points $\{(\alpha_i, \beta_i)\}$ uniquely define a univariate polynomial", we have that "$t+1$ univariate polynomials $\{(\alpha_i, f_i(x))\}$ uniquely define a bivariate polynomial".

An alternative claim for interpolation of bivariate polynomial is the following. Instead of having $t+1$ univariate polynomials ("shares") $\{(\alpha_i, f_i(x))\}$ in order to define the bivariate polynomial $S(x,y)$, it is enough to have a set of cardinality t of pairs of polynomials $(f_i(x), g_i(y))$ that are consistent (i.e., for every pair (i,j) we have that $f_i(\alpha_j) = g_j(\alpha_i)$), and the secret (constant-term) s. There exists a unique bivariate polynomial that satisfies all the constraints above. Formally:

Claim 3.2.1 *Let t be a nonnegative integer, let $\alpha_1, \ldots, \alpha_t$ be t distinct elements in $\mathbb{F}$, let $s \in \mathbb{F}$ be some constant, and let $\{f_1(x), \ldots, f_t(x), g_1(y), \ldots, g_t(y)\}$ be a set of $2t$ polynomials of degree t, such that for every pair $(i,j) \in \{1, \ldots, t\}^2$ it holds that: $f_i(\alpha_j) = g_j(\alpha_i)$. Then, there exists a unique bivariate polynomial $S(x,y)$ of degree t such that $S(0,0) = s$ and for every $i = 1, \ldots, t$ it holds that $S(x, \alpha_i) = f_i(x)$, $S(\alpha_i, y) = g_i(y)$.*

Proof: We define the bivariate polynomial that satisfies all the conditions. We already have a set of t univariate polynomials, and therefore we need only one additional polynomial in order to apply Claim 2.5.2, the "interpolation claim" for bivariate polynomials. After applying this Claim, we will show that this resulting polynomial is unique and satisfies all the necessary conditions.

Define the unique degree-t polynomial $f'(x)$ that passes through the following $t+1$ points: The point $(0, s)$ and the points $\{(\alpha_i, g_i(0))\}$ for every $i = 1, \ldots, t$. Thus, it holds that $f'(0) = s$ and for every $i = 1, \ldots, t$: $f'(\alpha_i) = g_i(0)$.

Now, giving the polynomial $(0, f'(x))$, and the set of polynomials $((\alpha_1, f_1(x)), \ldots, (\alpha_t, f_t(x)))$, we have $t+1$ polynomials of degree-t. Thus, using Claim 2.5.2, there exists a unique polynomial $S(x,y)$ such that $S(x, \alpha_i) = f_i(x)$ for every $i = 1, \ldots, t$ and $S(x, 0) = f'(x)$. We claim that S is the unique polynomial that satisfies all the conditions in the statement of the claim.

For every $i = 1, \ldots, t$, we have that $S(\alpha_i, y)$ is a univariate polynomial with degree-t. It holds that for every $j = 1, \ldots, t$, $S(\alpha_i, \alpha_j) = f_j(\alpha_i) = g_i(\alpha_j)$, and $S(\alpha_i, 0) = f'(\alpha_i) = g_i(0)$. We therefore have that the polynomials $S(\alpha_i, y)$ and $g_i(y)$ are both degree-t polynomials that agree on $t+1$ points, and therefore they are identical, and $S(\alpha_i, y) = g_i(y)$. Finally, $S(0,0) = f'(0) = s$, and so S satisfies all the requirements above. The uniqueness of S follows from the uniqueness of S from Claim 2.5.2, which concludes the proof. ■

We recall that in Section 2.5.3, we proved Claim 2.5.4. This Claim shows that for any two degree-t univariate polynomials $q_1(x), q_2(x)$ such that $q_1(\alpha_i) = q_2(\alpha_i)$ for every $i \in I$, the shares of the corrupted parties on two random bivariate polynomials $S_1(x,y), S_2(x,y)$ that satisfy $S_1(x,0) = q_1(x)$ and $S_2(x,0) = q_2(x)$ are distributed identically. The following is a simple Corollary of this Claim:

Corollary 3.2.2 *For any set of distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$, any pair of values $s, s' \in \mathbb{F}$, any subset $I \subset [n]$ where $|I| = \ell \leq t$, it holds that: Then,*

$$\left\{ \{(i, S_1(x, \alpha_i), S_1(\alpha_i, y))\}_{i \in I} \right\} \equiv \left\{ \{(i, S_2(x, \alpha_i), S_2(\alpha_i, y))\}_{i \in I} \right\}$$

where $S_1(x,y)$ and $S_2(x,y)$ are degree-t bivariate polynomial chosen at random from $\mathcal{B}^{s,t}$, $\mathcal{B}^{s',t}$, respectively.

Proof: The choice of $S_1(x,y)$ from $\mathcal{B}^{s,t}$, and S_2 from $\mathcal{B}^{s',t}$ is equivalent to the following. We simply choose two polynomials $q_1(x), q_2(x)$ uniformly at random from $\mathcal{P}^{s,t}$, $\mathcal{P}^{s',t}$, respectively. Then, we choose S_1, S_2 uniformly at random under the constraints that $S_1(x,0) = q_1(x)$ and $S_2(x,0) = q_2(x)$. Using this representation, we can apply Claim 2.3.2 to get that $\{q_1(\alpha_i)\}_{i\in I} \equiv \{q_2(\alpha_i)\}_{i\in I}$. Thus, for every set of elements $\{\beta_i\}_{i\in I}$ in $\mathbb{F}$, we have that:

$$\left\{ \{(i, S_1(x,\alpha_i), S_1(\alpha_i,y))\}_{i\in I} \mid \bigwedge_{i\in I} q_1(\alpha_i) = \beta_i \right\}$$
$$\equiv \left\{ \{(i, S_2(x,\alpha_i), S_2(\alpha_i,y))\}_{i\in I} \mid \bigwedge_{i\in I} q_2(\alpha_i) = \beta_i \right\}$$

where the latter follows from Claim 2.5.4. Since the conditionals ensembles are identical, the corollary follows. ■

3.2.2 Verifiable Secret Sharing of a Bivariate Polynomial

In Section 2.5.5 we introduced the $\widetilde{F}_{VSS}$ functionality – a functionality for sharing a degree-t bivariate polynomial. This functionality plays an important role in our construction, and we redefine it here for completeness. The functionality is defined as follows:

$$\widetilde{F}_{VSS}(S(x,y), \lambda, \ldots, \lambda) = \begin{cases} ((f_1(x), g_1(y)), \ldots, (f_n(x), g_n(y))) & \text{if } \deg(S) \le t \\ (\bot, \ldots, \bot) & \text{otherwise} \end{cases},$$

where $f_i(x) = S(x,\alpha_i)$, $g_i(y) = S(\alpha_i, y)$. For further information, and for the implementation of this functionality, see Section 2.5.5. Moreover, we recall that the protocols for implementing F_{VSS} and $\widetilde{F}_{VSS}$ have the same communication complexity ($O(n^2)$ field elements in point-to-point channel and no broadcast in the optimistic case, and additional $O(n^2)$ field elements in the pessimistic case). In fact, the implementation of F_{VSS} is just a single invocation of $\widetilde{F}_{VSS}$, with some additional local computation before and after the invocation.

3.3 The Multiplication Protocol

3.3.1 A High-Level Overview

We start with a high-level overview of the (modified) BGW protocol. Recall that the BGW protocol works by having the parties compute the desired function f by securely emulating the computation of an arithmetic circuit computing f. In this computation, the parties compute shares of the output of a circuit gate given shares of the input wires of that gate. By our invariant, the secret sharing scheme is based on bivariate polynomial; that is, we assume that the secret $s \in \mathbb{F}$ is hidden by some bivariate polynomial $S(x,y)$ of degree-t in both variables, where the share of each party P_i is the two univariate degree-t polynomials $\langle S(x,\alpha_i), S(\alpha_i,y)\rangle$.

The computation of addition gates and multiplication-with-a-constant gates can be emulated locally. That is, assume that the values on the inputs wires are a and b, and assume that these are hidden by $A(x,y)$ and $B(x,y)$, respectively. Then, each party computes shares of the output wire using its shares on the inputs wires locally. Specifically, P_i holds the shares $\langle A(x,\alpha_i), A(\alpha_i,y)\rangle$ and $\langle B(x,\alpha_i), B(\alpha_i,y)\rangle$. Consider the polynomial $C(x,y) = A(x,y) + B(x,y)$. This is a polynomial of degree-t in both variables and its constant term is $C(0,0) = A(0,0) + B(0,0) = a+b$. Moreover, each party P_i can locally compute its share $\langle C(x,\alpha_i), C(\alpha_i,y)\rangle$ by simply adding its two shares: $\langle A(x,\alpha_i) + B(x,\alpha_i), A(\alpha_i,y) + B(\alpha_i,y)\rangle$. Multiplication by a constant gate is computed similarly.

In this section, we show how to securely compute bivariate shares of the product of shared values, in the presence of a malicious adversary controlling any $t < n/3$ parties. The protocol for emulating the computation of a multiplication gate is very similar to the original construction of BGW (see Section 2.6.1). In particular, giving the shares $\langle A(x,\alpha_i), A(\alpha_i,y)\rangle$ and $\langle B(x,\alpha_i), B(\alpha_i,y)\rangle$, we refer to the values $A(0,\alpha_i), B(0,\alpha_i)$ as the univariate shares of party P_i. These two values correspond to the univariate polynomials $A(0,y)$ and $B(0,y)$, that hide the values a and b, respectively. Like the univariate case, we have two univariate polynomials that hide the values on the wires, and each party holds a share on each polynomial. Thus, the univariate polynomial $A(0,y)\cdot B(0,y)$ is a degree-$2t$ polynomial with the desired constant term ab, and each party can compute its share on this polynomial by multiplying $A(0,\alpha_i)\cdot B(0,\alpha_i)$ (simply by multiplying its two "univariate shares"). As was shown in [51] (see also Section 2.6.7), there exists constants $\gamma_1,\ldots,\gamma_n$ such that:

$$ab = \gamma_1 \cdot A(0,\alpha_i)\cdot B(0,\alpha_i) + \ldots + \gamma_n \cdot A(0,\alpha_n)\cdot B(0,\alpha_n) \qquad (3.3.1)$$

Our goal is to provide the parties bivariate sharing (of degree-t) of the product ab. The protocol is similar to the original construction of BGW, where we save the invocation of $F_{VSS}^{subshare}$, and have some modification of F_{VSS}^{mult}. In particular, the multiplication protocol proceeds as follows:

1. Each party distributes bivariate sharing (of degree-t) of the product $A(0,\alpha_i)\cdot B(0,\alpha_i)$ in a verifiable way. The protocol for sharing the product uses the (univariate) subshares of $A(0,\alpha_i)$ and $B(0,\alpha_i)$ where the parties *already hold* by their bivariate shares of $A(x,y)$ and $B(x,y)$. This step is carried out using the $\widetilde{F}_{VSS}^{mult}$ functionality[1] in Section 3.3.4, which is the bivariate analogue to the F_{VSS}^{mult} functionality (Section 2.6.6). In order to implement this step, we introduce the functionality called $\widetilde{F}_{eval}$, which is the univariate analogue to F_{eval}. In addition, we present yet another functionality $\widetilde{F}_{extend}$, that enables a dealer to extends univariate shares to bivariate shares. This functionality is formalized in Section 3.3.2.

2. Let $C_i(x,y)$ be the bivariate polynomial that hides $A(0,\alpha_i)\cdot B(0,\alpha_i)$. After the previous step, each party P_j hold bivariate shares $\langle C_i(x,\alpha_j), C_i(\alpha_j,y)\rangle$ for every i. Using the linear combination as in Eq. (3.3.1), each party obtains shares of $C(x,y) = \sum_{i=1}^{n}\gamma_i \cdot C_i(x,y)$, and it holds that:

$$C(0,0) = \sum_{i=1}^{n}\gamma_i \cdot C_i(0,0) = \sum_{i=1}^{n}\gamma_i \cdot A(0,\alpha_i)\cdot B(0,\alpha_i) = ab\ .$$

   Thus, after each party distributes bivariate sharing of the product of its univariate shares $(A(0,\alpha_i)\cdot B(0,\alpha_i))$, the parties just perform a local linear combination in order to obtain

[1]By convention, we denote bivariate-sharing based functionalities with a tilde.

shares on the output polynomial.

The multiplication functionality is denoted by $\widetilde{F}_{mult}$, which is the bivariate analogue to F_{mult} functionality. See Section 3.3.5.

Organization. We now proceed to formally describe and prove each one of the functionalities. In Section 3.3.2 we define the $\widetilde{F}_{extend}$ functionality that enables a dealer to transform a univariate sharing to bivariate sharing. In Section 3.3.3 we present the $\widetilde{F}_{eval}$ functionality which is the bivariate analogue of F_{eval}. In Section 3.3.4 we present the $\widetilde{F}_{VSS}^{mult}$ functionality, and in Section 3.3.5 we present the $\widetilde{F}_{mult}$ functionality.

3.3.2 The $\widetilde{F}_{extend}$ Functionality for Transforming Univariate to Bivariate

As we will show below (in Section 3.3.3) and as we have seen regarding $F_{VSS}^{subshare}$, it is possible to utilize the additional information provided by a bivariate secret sharing in order to obtain higher efficiency. However, some of the intermediate sharings used in the multiplication protocol are inherently univariate. Thus, we introduce a new functionality called $\widetilde{F}_{extend}$ that enables a dealer to efficiently extend shares of a univariate polynomial $q(x)$ of degree-t to a sharing based on a bivariate polynomial $S(x, y)$ of degree-t in both variables, with the guarantee that $q(x) = S(x, 0)$. In the functionality definition, the dealer receives the polynomial $q(x)$ that is distributed via the inputs of the honest parties. Although in any use of the functionality this is already known to the dealer, we need it for technical reasons in the simulation when the dealer is corrupted. See Functionality 3.3.1 for a full definition (observe that the dealer has as input the univariate polynomial $q(x)$ and a bivariate polynomial $S(x, y)$ such that $S(x, 0) = q(x)$).

FUNCTIONALITY 3.3.1 (The $\widetilde{F}_{extend}$ Functionality for Extending Shares)

$\widetilde{F}_{extend}$ receives a set of indices $I \subseteq [n]$ and works as follows:

1. The $\widetilde{F}_{extend}$ functionality receives the shares of the honest parties $\{\beta_j\}_{j \notin I}$. Let $q(x)$ be the unique degree-t polynomial determined by the points $\{(\alpha_j, \beta_j)\}_{j \notin I}$. (If no such polynomial exists then no security is guaranteed[2].)
2. In case that the dealer is corrupted, $\widetilde{F}_{extend}$ sends $q(x)$ to the (ideal) adversary.
3. $\widetilde{F}_{extend}$ receives $S(x, y)$ from the dealer. Then, it checks that $S(x, y)$ is of degree-t in both variables *and* $S(x, 0) = q(x)$.
4. If both condition holds, define the output of P_i to be the pair of univariate polynomials $\langle S(x, \alpha_i), S(\alpha_i, y)\rangle$. Otherwise, define the output of P_i to be $\perp$.
5. (a) $\widetilde{F}_{extend}$ sends the outputs to each honest party P_j ($j \notin I$).

   (b) $\widetilde{F}_{extend}$ sends the output of each corrupted party P_i ($i \in I$) to the (ideal) adversary.

The protocol that implements this functionality is simple and efficient, but the argument for its security is delicate. The dealer distributes shares of $S(x, y)$, using the bivariate VSS protocol ($\widetilde{F}_{VSS}$). Each party receives shares $S(x, \alpha_i), S(\alpha_i, y)$, and checks that $S(\alpha_i, 0) = q(\alpha_i)$. If not,

[2]If all of the points sent by the honest parties lie on a single degree-t polynomial, then this guarantees that $f(x)$ is the unique degree-t polynomial for which $f(\alpha_j) = \beta_j$ for all $j \notin I$. If not all the points lie on a single degree-t polynomial, then no security guarantees are obtained. However, since the honest parties all send their prescribed input, $f(x)$ will always be as desired. This can be formalized using the notion of a partial functionality [53, Sec. 7.2].

it broadcasts a complaint. The parties accept the shares of $S(x,y)$ if and only if there are at most t complaints. See Protocol 3.3.2.

PROTOCOL 3.3.2 (Securely Computing $\widetilde{F}_{extend}$ in the $\widetilde{F}_{VSS}$-Hybrid Model)

- **Input:** The dealer P_1 holds a univariate polynomial of degree-t, $q(x)$, and a degree-t bivariate polynomial $S(x,y)$ that satisfies $S(x,0) = q(x)$. Each party P_i holds $q(\alpha_i)$.
- **Common input:** A field description $\mathbb{F}$ and n distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **Aiding ideal functionality initialization:** Upon invocation, the trusted party computing the (fictitiously corruption-aware) functionality $\widetilde{F}_{VSS}$ receives the set of corrupted parties I.
- **The protocol:**
  1. The parties invoke the $\widetilde{F}_{VSS}$ functionality, where P_1 (the dealer) uses the bivariate polynomial $S(x,y)$, and any other party inputs λ (the empty string).
  2. If $\widetilde{F}_{VSS}$ returns $\perp$, each outputs $\perp$ and halts.
  3. Otherwise, let $\langle S(x,\alpha_i), S(\alpha_i,y)\rangle$ be the output shares of $\widetilde{F}_{VSS}$. If $S(\alpha_i,0) \neq q(\alpha_i)$, then broadcast $\mathsf{complaint}(i)$.
- **Output:** If more than t parties broadcast $\mathsf{complaint}$, output $\perp$. Otherwise, output $\langle f_i(x), g_i(y)\rangle = \langle S(x,\alpha_i), S(\alpha_i,y)\rangle$.

We now give an intuitive argument as to why the protocol securely computes the functionality. First, assume that the dealer is honest. In this case, the dealer inputs a degree-t bivariate polynomial that satisfies $S(x,0) = q(x)$, as required. The bivariate VSS functionality $\widetilde{F}_{VSS}$ ensures that the honest parties receive the correct shares. Now, since the polynomial satisfies the requirement, none of the honest parties complain. As a result, at most t parties complain, and all the honest parties accept the new bivariate shares.

The case where the dealer is corrupted is more subtle. At first, it may seem possible that t honest parties receive inconsistent shares and broadcast a $\mathsf{complaint}$, while the remaining $t+1$ honest parties receive consistent shares and remain silent (together with all the corrupted parties). In such a case, only t complaints would be broadcast and so the parties would accept the bivariate polynomial even though it is not consistent with the inputs of all honest parties. Fortunately, as we show, such a situation can actually never occur. This is due to the fact that the $\widetilde{F}_{VSS}$ functionality ensures that the bivariate polynomial that is distributed is of degree-t in both variables, and due to the fact that the inputs of the honest parties lie on a polynomial with degree-t. As we show in the proof, this implies that if there exists a set of $t+1$ honest parties for which the bivariate polynomial agrees with their inputs, then this bivariate polynomial *must* satisfy $S(x,0) = q(x)$. In other words, we prove that once $t+1$ of the bivariate shares are consistent with the points of $t+1$ of the honest parties, then *all* of the bivariate shares must be consistent with *all* of the honest parties' points.

Theorem 3.3.3 *Let $t < n/3$. Then, Protocol 3.3.2 is t-secure for the $\widetilde{F}_{extend}$ functionality in the $\widetilde{F}_{VSS}$-hybrid model, in the presence of a static malicious adversary.*

Proof: We separate to two case, depending whether the dealer is honest or not.

Case 1 - an Honest Dealer. An honest dealer always distributes a bivariate polynomial that satisfies the condition, and so no honest party broadcasts a complaint. The view of the adversary is the output of the $\widetilde{F}_{VSS}$ functionality, which is exactly the output of the trusted party computing $\widetilde{F}_{extend}$ in the ideal execution. Therefore, the simulation is straightforward.

The simulator $\mathcal{S}$.

1. *$\mathcal{S}$ invokes the adversary $\mathcal{A}$ with the auxiliary information z.*
2. *The honest parties send their shares to the trusted party computing $\widetilde{F}_{extend}$, and the dealer sends a bivariate polynomial $S(x,y)$ that satisfies the condition. Then, $\mathcal{S}$ receives from $\widetilde{F}_{extend}$ the univariate values $\{q(\alpha_i)\}_{i\in I}$ and the bivariate shares $\{S(x,\alpha_i), S(\alpha_i,y)\}_{i\in I}$.*
3. *$\mathcal{S}$ sends the set of polynomials $\{\langle S(x,\alpha_i), S(\alpha_i,y)\rangle\}_{i\in I}$ to the adversary $\mathcal{A}$ as the output of $\widetilde{F}_{VSS}$ (Step 1 in Protocol 3.3.2).*
4. *$\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.*

Since the adversary has no input to the $\widetilde{F}_{extend}$ functionality, the output of the honest parties in the ideal execution are always the bivariate shares on $S(x,y)$. Since in the real execution no honest party complains, there are at most t complaints and therefore the honest parties always outputs their output from $\widetilde{F}_{extend}$, that is, their bivariate shares on $S(x,y)$. The view of the adversary in the ideal and real executions consists of the output of the $\widetilde{F}_{VSS}$ functionality. Therefore, overall, for every real adversary $\mathcal{A}$ controlling parties I where $|I| \leq t$ and I does not include the dealer P_1, every univariate polynomial $q(x)$, every bivariate polynomial $S(x,y)$ with degree-t that satisfies $S(x,0) = q(x)$, and all auxiliary inputs $z \in \{0,1\}^*$, it holds that:

$$\left\{\text{HYBRID}^{\widetilde{F}_{VSS}}_{\pi,\mathcal{A}(z),I}(\vec{x})\right\} = \left\{\text{IDEAL}_{\widetilde{F}_{extend},\mathcal{S}(z),I}(\vec{x})\right\}$$

where $\vec{x} = ((q(x), S(x,y), q(\alpha_1)), q(\alpha_2), \ldots, q(\alpha_n))$.

We proceed to the second case - where the dealer is corrupted.

Case 2 - a Corrupted Dealer. The simulator interacts externally with a trusted party computing $\widetilde{F}_{extend}$, while internally simulating the interaction of $\mathcal{A}$ with the trusted party computing $\widetilde{F}_{VSS}$ and the honest parties. Here, in order to simulate the complaints of the honest parties, the simulator has to know the polynomial $q(x)$ that is defined implicitly by the inputs of the honest parties. Therefore, it received this polynomial from the $\widetilde{F}_{extend}$ functionality (see Step 2 in Functionality 3.3.1).

The simulator $\mathcal{S}$.

1. *$\mathcal{S}$ internally invokes $\mathcal{A}$ with the auxiliary input z.*
2. *Recall that the honest parties send their univariate shares to the trusted party computing $\widetilde{F}_{extend}$. Then, the functionality reconstructs the underlying polynomial $q(x)$, and sends it to the simulator $\mathcal{S}$ (Step 2 in Functionality 3.3.1).*
3. *$\mathcal{S}$ internally receives from $\mathcal{A}$ the bivariate polynomial $S'(x,y)$ that the dealer sends to the (simulated) trusted party computing $\widetilde{F}_{VSS}$ (Step 1 in Protocol 3.3.2).*

$\mathcal{S}$ checks the polynomial $S'(x,y)$. If this polynomial is of degree-t in both variables, then it accepts this polynomial and internally sends $\mathcal{A}$ as the output of $\widetilde{F}_{VSS}$ the set of bivariate shares $\{\langle S'(x,\alpha_i), S'(\alpha_i,y)\rangle\}_{i\in I}$. Otherwise it internally sends $\mathcal{A}$ the value $\perp$ for every $i \in I$, and externally sends x^{2t} to $\widetilde{F}_{extend}$ (i.e., an invalid polynomial), outputs whatever $\mathcal{A}$ outputs, and halts.

4. *$\mathcal{S}$ externally sends to the $\widetilde{F}_{extend}$ functionality the bivariate polynomial $S'(x,y)$ as the input of the dealer.*
5. *For every $j \notin I$, $\mathcal{S}$ checks that $q(\alpha_j) = S'(\alpha_j, 0)$. If this check fails, it internally sends the adversary $\mathcal{A}$ the message* complaint(j) *as a broadcast message of honest P_j.*
6. *$\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.*

We now prove that for every $I \subseteq [n]$ such that $1 \in I$, $|I| \leq t$, every $z \in \{0,1\}^*$ it holds that:

$$\left\{\text{HYBRID}^{\widetilde{F}_{VSS}}_{\pi,\mathcal{A}(z),I}(\vec{x})\right\}_{\vec{x}\in\{0,1\}^*} = \left\{\text{IDEAL}_{\widetilde{F}_{extend},\mathcal{S}(z),I}(\vec{x})\right\}_{\vec{x}\in\{0,1\}^*}$$

where $\vec{x} = ((q(x), S(x,y), q(\alpha_1)), q(\alpha_2), \ldots, q(\alpha_n))$.

It is clear from the description of the simulator that the view of the corrupted party is the same in both executions, since the simulator plays the role of the honest parties in the ideal execution. It is left to show that the output of the honest parties is distributed identically in both worlds, conditioning on the view of the corrupted parties.

If the dealer sends an invalid bivariate polynomial to $\widetilde{F}_{VSS}$, then it is easy to see that in both executions the output of all honest parties is $\perp$. Thus, we left with the case where the polynomial $S'(x,y)$ is of degree-t in both variables. If the polynomial satisfies the condition $S'(x,0) = q(x)$, then in the ideal execution all the honest parties output their shares on this bivariate polynomial, since the simulator sends S' to $\widetilde{F}_{extend}$ functionality. Likewise, in the real execution no honest party broadcasts complaint, and the parties will accept this bivariate polynomial (exactly like the case of an honest dealer).

The only case that is left is where the dealer sends a polynomial of degree-t such that $S'(x,0) \neq q(x)$. In the ideal world, the functionality $\widetilde{F}_{extend}$ rejects this polynomial and all honest parties output $\perp$. We therefore need to show that the same happens in the real execution, or, equivalently, that strictly more than t parties broadcast complaint.

Assume by contradiction that $S'(x,0) \neq q(x)$ and that less than $t+1$ parties broadcast complaint. Therefore, there must be at least $t+1$ honest parties that are satisfied with their shares. The $\widetilde{F}_{VSS}$ functionality ensures that $S'(x,y)$ is of degree-t in both variables, and thus $S'(x,0)$ is a univariate polynomial of degree t. For every honest party that does not broadcast complaint it holds that $S'(\alpha_j,0) = q(\alpha_j)$. Since both polynomials $S'(x,0)$, $q(x)$ are polynomials with degree-t, and since they agree in at least $t+1$ points, these two polynomials are identical. We conclude that $S'(x,0) = q(x)$ in contradiction. Therefore, it must be that at least $t+1$ parties broadcast complaint, and so all the parties in the real execution output $\perp$, like in the ideal execution. This completes the proof. ■

3.3.3 The $\widetilde{F}^k_{eval}$ Functionality for Evaluating a Shared Bivariate Polynomial

Another tool that we need is the $\widetilde{F}^k_{eval}$ functionality. Given a sharing of a bivariate polynomial $S(x,y)$ of degree-t in both variables, the functionality enables the parties to evaluate the bivariate polynomial on some point α_k, or equivalently to learn the bivariate sharing $\langle S(x,\alpha_k), S(\alpha_k,y)\rangle$. The $\widetilde{F}^k_{eval}$ functionality is formally defined in Functionality 3.3.4.

FUNCTIONALITY 3.3.4 (The Functionality $\widetilde{F}^k_{eval}$)

$\widetilde{F}^k_{eval}$ receives a set of indices $I \subseteq [n]$ and works as follows:

1. The $\widetilde{F}^k_{eval}$ functionality receives from each honest party P_j the pair of degree-t polynomials $(f_j(x), g_j(y))$, for every $j \notin I$. Let $S(x,y)$ be the single bivariate polynomial with degree-t in both variables that satisfies $S(x,\alpha_j) = f_j(x)$, $S(\alpha_j,y) = g_j(y)$ for every $j \notin I$. (If no such $S(x,y)$ exists, then no security is guaranteed; see Footnote 2).
2. (a) For every $j \notin I$, F^k_{eval} sends the output pair $\langle S(x,\alpha_k), S(\alpha_k,y)\rangle$.
   (b) In addition, for every $i \in I$, F^k_{eval} sends the output pair $\langle S(x,\alpha_k), S(\alpha_k,y)\rangle$ to the (ideal) adversary.

The protocol computing $\widetilde{F}^k_{eval}$ is straightforward and very efficient. Given input $\langle f_i(x), g_i(y)\rangle$ (which under a similar assumption on the inputs as in Footnote 2 equals $S(x,\alpha_i)$, $S(\alpha_i,y)$), each party P_i evaluates its polynomials on the point α_k and sends $f_i(\alpha_k), g_i(\alpha_k)$ (equivalently, $S(\alpha_k,\alpha_i), S(\alpha_i,\alpha_k)$)) to every other party; broadcast is not needed for this. Assume for now that all parties send the correct values. Thus, at the end of this step, each party holds the values $(f_1(\alpha_k),\dots,f_n(\alpha_k))$ and $(g_1(\alpha_k),\dots,g_n(\alpha_k))$. Now, recall that for every $i,j \in [n]$, $f_i(\alpha_j) = g_j(\alpha_i) = S(\alpha_j,\alpha_i)$. Therefore, each party actually holds the sequences $(g_k(\alpha_1),\dots,g_k(\alpha_n))$ and $(f_k(\alpha_1),\dots,f_k(\alpha_n))$, respectively, and can reconstruct the polynomials $g_k(y)$ and $f_k(x)$. See Protocol 3.3.5.

In case the corrupted parties send incorrect values (or do not send any values), since $f_k(x)$ and $g_k(y)$ are both degree-t polynomials, each party can reconstruct the polynomials by using Reed-Solomon decoding.

The simplicity and efficiency of this protocol demonstrates the benefits of the approach of utilizing the bivariate shares throughout the entire multiplication protocol.

Theorem 3.3.6 *Let $t < n/3$. Then, Protocol 3.3.5 is t-secure for the $\widetilde{F}^k_{eval}$ functionality in the presence of a static malicious adversary.*

Proof: The functionality $\widetilde{F}^k_{eval}$ does not receive any values from the ideal adversary, and so the simulator does not send any value to the trusted party. The simulator only needs to generate the view of the corrupted parties. That is, it needs to generate the points:

$$\{S(\alpha_k,\alpha_j), (S(\alpha_j,\alpha_k))\}_{j\notin I}$$

This set of points can be computed easily from the polynomials $S(x,\alpha_k), S(\alpha_k,y)$ that the simulator receives from the $\widetilde{F}_{eval}$–functionality.

The simulator $\mathcal{S}$.

1. $\mathcal{S}$ invokes the adversary $\mathcal{A}$ on the auxiliary input z.

2. $\mathcal{S}$ receives from the $\widetilde{F}_{eval}$ functionality the polynomials $(S(x,\alpha_k), S(\alpha_k,y))$ as the output of all parties. Let $f_k(x) \stackrel{\text{def}}{=} S(x,\alpha_k)$, and $g_k(y) \stackrel{\text{def}}{=} S(\alpha_k,y)$.

PROTOCOL 3.3.5 (Securely Computing $\widetilde{F}^{k}_{eval}$)

- **Input:** Each party holds two degree-t polynomials $f_i(x), g_i(y)$.

  (It is assumed that there there exists a single bivariate polynomial $S(x,y)$ such that for every $i \in [n]$: $S(x,\alpha_i) = f_i(x)$ and $S(\alpha_i, y) = g_i(y)$. If such a polynomial does not exists, then no security is guaranteed; see the definition of the functionality).
- **Common input:** A field description $\mathbb{F}$ and n distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **The protocol:**
  1. Each party P_i sends to each party the values $(f_i(\alpha_k), g_i(\alpha_k))$ (which are the values $(g_k(\alpha_i), f_k(\alpha_i)) = (S(\alpha_k, \alpha_i), S(\alpha_i, \alpha_k))$, respectively).
  2. At the end of this stage each party holds the sequences $(\hat{S}(\alpha_1, \alpha_k), \ldots, \hat{S}(\alpha_n, \alpha_k))$ and $(\hat{S}(\alpha_k, \alpha_1), \ldots, \hat{S}(\alpha_k, \alpha_n))$, where each is a possibly corrupted codeword of distance at most-t from each one of the following codewords, respectively:
  $$\begin{aligned} S(\alpha_1,\alpha_k),\ldots,S(\alpha_n,\alpha_k)) &= (g_k(\alpha_1),\ldots,g_k(\alpha_n)), \\ S(\alpha_k,\alpha_1),\ldots,S(\alpha_k,\alpha_n)) &= (f_k(\alpha_1),\ldots,f_k(\alpha_n)) \ . \end{aligned}$$
  3. Using the Reed-Solomon decoding procedure, each party decodes the above codewords and reconstructs the polynomials $S(x,\alpha_k) = f_k(x)$ and $S(\alpha_k, y) = g_k(y)$.
- **Output:** Each party outputs $\langle S(x,\alpha_k), S(\alpha_k,y)\rangle = \langle f_k(x), g_k(y)\rangle$.

3. *For every $j \notin I$, and for every $i \in I$, $\mathcal{S}$ simulates party P_j sending to P_i the values $(f_j(\alpha_k), g_j(\alpha_k)) = (g_k(\alpha_j), f_k(\alpha_j)) = (S(\alpha_k,\alpha_j), S(\alpha_j,\alpha_k))$ as in the first step of the protocol.*

4. *$\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.*

We now show that for every real adversary $\mathcal{A}$ controlling parties I where $|I| \le t$, for every bivariate polynomial $S(x,y)$ with degree-t in both variables x,y and all auxiliary input $z \in \{0,1\}^*$ it holds that:
$$\left\{\text{IDEAL}_{\widetilde{F}_{eval},\mathcal{S}(z),I}(\vec{x})\right\} \equiv \left\{\text{REAL}_{\pi,\mathcal{A}(z),I}(\vec{x})\right\}$$
where $\vec{x} = ((f_1(x), g_1(y)), \ldots, (f_n(x), g_n(y))$, and for every $j \in [n]$, $f_j(x) = S(x,\alpha_j)$, $g_j(y) = S(\alpha_j, y)$.

The view of the adversary is exactly the same in both executions, since it holds that:
$$f_j(\alpha_k) = S(\alpha_k,\alpha_j) = g_k(\alpha_j) \qquad \text{and} \qquad g_j(\alpha_k) = S(\alpha_j,\alpha_k) = f_k(\alpha_j).$$

Since the corrupted parties have no inputs to the functionality, it is enough to show that the output of the honest parties is the same in both executions. In the ideal, the parties output $\langle S(x,\alpha_k), S(\alpha_k,y)\rangle$. In the real, since $S(x,y)$ is a bivariate polynomial with degree-t in both variables, the polynomials $S(x,\alpha_k), S(\alpha_k,y)$ are both of degree-t. As a result, the values $(S(\alpha_1,\alpha_k),\ldots,S(\alpha_n,\alpha_k))$ are a Reed-Solomon codeword of length n and dimension $t+1$, and therefore it is possible to correct up to $\frac{n-t}{2} > \frac{3t-t}{2} = t$ errors. Thus, the values sent by the corrupted parties in the output stage have no influence whatsoever on the honest parties' outputs, and thus the honest parties reconstruct the correct polynomials $\langle S(x,\alpha_k), S(\alpha_k,y)\rangle$.

■

3.3.4 The $\widetilde{F}_{VSS}^{mult}$ Functionality for Sharing a Product of Shares

The main step for achieving secure multiplication is a method for a party P_i to share the product of its shares[3] $a \cdot b$, while preventing a corrupted P_i from sharing an incorrect value. In the univariate case, the parties use $F_{VSS}^{subshare}$ to first share their shares, and then use F_{VSS}^{mult} to distribute shares of the product of their shares. In this section, we revisit the multiplication for the bivariate case. In this case, the parties hold shares of univariate polynomials that hide a party P_i's shares a, b, exactly as in the univariate solution with functionality F_{VSS}^{mult}. We stress that in our case these shares are univariate (i.e. points on a polynomial) and not bivariate shares (i.e. univariate polynomials) since we are referring to the *subshares*. Nevertheless, as we have shown, these can be separately extended to bivariate sharings of a and b using $\widetilde{F}_{extend}$. Our goal with $\widetilde{F}_{VSS}^{mult}$ is for the parties to obtain a sharing of $a \cdot b$, by holding shares of a bivariate polynomial $C(x, y)$ whose constant term is the desired product. See Functionality 3.3.7 for a specification.

FUNCTIONALITY 3.3.7 (The $\widetilde{F}_{VSS}^{mult}$ functionality for sharing a product of shares)

$\widetilde{F}_{VSS}^{mult}$ receives a set of indices $I \subseteq [n]$ and works as follows:

1. The $\widetilde{F}_{VSS}^{mult}$ functionality receives an input pair (a_j, b_j) from every honest party P_j $(j \notin I)$. The dealer P_1 has polynomials $A(x), B(x)$ such that $A(\alpha_j) = a_j$ and $B(\alpha_j) = b_j$, for every j.
2. $\widetilde{F}_{VSS}^{mult}$ computes the unique degree-t univariate polynomials A and B such that $A(\alpha_j) = a_j$ and $B(\alpha_j) = b_j$ for every $j \notin I$ (if no such A or B exist of degree-t, then $\widetilde{F}_{VSS}^{mult}$ behaves differently as in Footnote 2).
3. If the dealer P_1 is honest $(1 \notin I)$, then:
   (a) $\widetilde{F}_{VSS}^{mult}$ chooses a random degree-t bivariate polynomial $C(x, y)$ under the constraint that $C(0, 0) = A(0) \cdot B(0)$.
   (b) *Outputs for honest:* $\widetilde{F}_{VSS}^{mult}$ sends the dealer P_1 the polynomial $C(x, y)$, and for every $j \notin I$ it sends the bivariate shares $\langle C(x, \alpha_j), C(\alpha_j, y) \rangle$.
   (c) *Output for adversary:* For every $i \in I$, the functionality $\widetilde{F}_{VSS}^{mult}$ sends the univariate shares $A(\alpha_i), B(\alpha_i)$, and the bivariate shares $\langle C(x, \alpha_i), C(\alpha_i, y) \rangle$.
4. If the dealer P_1 is corrupted $(1 \in I)$, then:
   (a) $\widetilde{F}_{VSS}^{mult}$ sends $(A(x), B(x))$ to the (ideal) adversary.
   (b) $\widetilde{F}_{VSS}^{mult}$ receives a polynomial C as input from the (ideal) adversary.
   (c) If either $\deg(C) > t$ or $C(0, 0) \neq A(0) \cdot B(0)$, then $\widetilde{F}_{VSS}^{mult}$ resets $C(x, y) = A(0) \cdot B(0)$. That is, the constant polynomial equalling $A(0) \cdot B(0)$ everywhere.
   (d) *Output for honest:* $\widetilde{F}_{VSS}^{mult}$ sends the bivariate shares $\langle C(x, \alpha_j), C(\alpha_j, y) \rangle$ to P_j for every $j \notin I$.
   (There is no more output for the adversary in this case.)

The protocol. Recall the idea behind the univariate protocol as appear in BGW. The dealer chooses the univariate polynomials $D_1(x), \ldots, D_t(x)$ so that $C(x) = A(x) \cdot B(x) - \sum_{\ell=1}^{t} x^{\ell} \cdot D_{\ell}(x)$ is a degree-t polynomial with constant term $a \cdot b$. Note that since each polynomial D_{ℓ} is multiplied

[3]For the sake of clarity and to reduce the number of indices, in this section we refer to a and b as the shares of P_i (and not the secret), and to a_j and b_j the univariate subshares that P_j holds of P_i's shares a and b.

by x^ℓ, we have that the constant-term of $C(x)$ is always $a \cdot b$ (i.e., $A(0) \cdot B(0)$). The protocol shows how P_1 can choose the polynomials such that all the coefficient of degree higher than t are canceled, which ensure that $C(x)$ is of degree-t exactly. In case P_1 uses "incorrect" polynomials, then $C(x)$ may be of degree hight than t. The parties then verify that $C(x)$ is of degree-t (using a special verification process), and in case the verification succeeds, they output their shares on $C(x)$. See Section 2.6.6.

In the bivariate case, we wish to get shares of a bivariate polynomial $C(x, y)$. The parties hold as input shares of the univariate polynomial $A(x), B(x)$ as in the BGW protocol. Then, the dealer chooses $D_1, \ldots, D_t$ is a similar way as the original protocol (i.e., see Eq. (2.6.4) in Section 2.6.6). However, here the dealer distributes bivariate sharing instead of a univariate, and therefore for every polynomial $D_\ell(x)$ it chooses a bivariate polynomial $D_\ell(x, y)$ uniformly at random under the constraint that $D_\ell(x, 0) = D_\ell(x)$. Then, it distributes each one of these bivariate polynomial using the bivariate sharing functionality $\widetilde{F}_{VSS}$. This ensures that all the polynomials $D_1(x, 0), \ldots, D_t(x, 0)$ are of degree-t. In addition, this comes at no additional cost over the univariate protocol since the BGW VSS protocol anyway uses bivariate polynomials. At this point, each party holds shares of the univariate polynomials $A(x), B(x)$, and shares of the t bivariate polynomials $D_1(x, y), \ldots, D_t(x, y)$. From the construction, the univariate polynomial defined by:

$$C'(x) = A(x) \cdot B(x) - \sum_{k=1}^{t} x^k \cdot D_k(x, 0)$$

is a random polynomial with constant-term $a \cdot b$, and each party P_i can locally compute its share $C'(\alpha_i)$ on this polynomial. However, as in the univariate case, if the dealer did not choose the polynomials $D_\ell(x, y)$ as instructed, the polynomial $C'(x)$ may not be of degree-t, and in fact can be any polynomial of degree $2t$ (but no more since all the polynomials were shared using VSS and so are of degree at most t). We must therefore check the degree of $C'(x)$.

The verification of the polynomial $C'(x)$. At this point, the dealer chooses a random bivariate polynomial $C(x, y)$ of degree-t in both variables under the constraint that $C(x, 0) = C'(x)$, and shares it using the bivariate VSS functionality $\widetilde{F}_{VSS}$. This guarantees that the parties hold shares of a degree-t bivariate polynomial $C(x, y)$. If this polynomial satisfies

$$C(x, 0) = C'(x) = A(x) \cdot B(x) - \sum_{\ell=1}^{t} x^\ell \cdot D_\ell(x, 0)$$

then $C(0, 0) = A(0) \cdot B(0) = a \cdot b$, and we are done.

We therefore want to check that indeed $C(x, 0) = C'(x)$. Each party P_i holds shares of the polynomial $C(x, y)$, and so it holds the univariate polynomials $C(x, \alpha_i)$, $C(\alpha_i, y)$. Moreover, it has already computed its share $C'(\alpha_i)$. Thus, it can check that $C(\alpha_i, 0) = C'(\alpha_i)$. Since $C'(x)$ is of degree at most $2t$, and since $C(x, y)$ is of degree-t, then if this check passes for *all* of the $2t + 1$ honest parties, we are guaranteed that $C(x, 0) = C'(x)$, and so $C(0, 0) = a \cdot b$. Thus, each party checks that $C(\alpha_i, 0) = C'(\alpha_i)$, and if not it broadcasts a complaint. If there are more than t complaints, then it is clear that the dealer is corrupted. However, even when there are less than t complaints the dealer can be corrupted, and so the parties need to unequivocally verify each complaint.

The way the parties verify whether or not a complaint is false is as follows. The parties evaluate each one of the polynomials $D_1, \ldots, D_t$, A, B, and C on the point of the complaining party P_k. This is done by using the functionality $\widetilde{F}^k_{eval}$ (see Section 3.3.3). Observe that all of the polynomials $D_1, \ldots, D_t, C$ are bivariate and of degree-t, and so the bivariate $\widetilde{F}^k_{eval}$ can be used.

In contrast, $A(x)$ and $B(x)$ are only univariate polynomials and so $\widetilde{F}_{extend}$ (see Section 3.3.2) is first used in order to distribute bivariate polynomial $A(x, y)$ and $B(x, y)$ that fit $A(x)$ and $B(x)$, respectively. Following this, $\widetilde{F}^k_{eval}$ can also be used for $A(x, y)$ and $B(x, y)$. Finally, after the parties receive all of the shares of the complaining party, they can check whether the complaint is true or false. In case of a true complaint, the parties reconstruct the original shares and set their output to be $a \cdot b$. See Protocol 3.3.8 for a full specification. The protocol is in the $(\widetilde{F}_{VSS}, \widetilde{F}^1_{eval}, \ldots, \widetilde{F}^n_{eval}, \widetilde{F}_{extend})$-hybrid model. From here on, we write $\widetilde{F}_{eval}$-hybrid model to refer to all n functionalities $\widetilde{F}^1_{eval}, \ldots, \widetilde{F}^n_{eval}$.

Theorem 3.3.9 *Let $t < n/3$. Then, Protocol 3.3.8 is t-secure the $\widetilde{F}^{mult}_{VSS}$ functionality in the $(\widetilde{F}_{VSS}, \widetilde{F}_{eval}, \widetilde{F}_{extend})$-hybrid model, in the presence of a static malicious adversary.*

Proof: The proof is based on the proof of Theorem 2.6.14. We again separate between an honest dealer and a corrupted dealer.

Case 1 – the Dealer P_1 is Honest

Since the dealer is honest, the simulator does not send any input to the trusted party computing $\widetilde{F}^{mult}_{VSS}$, and so it receives directly from the trusted party the shares $A(\alpha_i), B(\alpha_i)$ and the polynomials $\langle C(x, \alpha_i), C(\alpha_i, y)\rangle$, for every $i \in I$.

The simulator chooses t degree-t polynomials $D_1(x), \ldots, D_t(x)$ as the simulator in Theorem 2.6.14. Then, the simulator chooses the degree-t bivariate polynomials $D^{\mathcal{S}}_1(x, y), \ldots, D^{\mathcal{S}}_t(x, y)$ uniformly at random under the constraint that $D^{\mathcal{S}}_i(x, 0) = D_i(x)$ for every $i = 1, \ldots, t$. We will show that these polynomials has the same distribution has in the real execution. In order to simulate complaints, observe that no honest party broadcasts a complaint. In addition, for every complaint of a dishonest party, the complaint resolution phase can be carried out since the simulator has all the points of the complaining party. We now describe the simulator.

The simulator $\mathcal{S}$.

1. *$\mathcal{S}$ invokes the adversary $\mathcal{A}$ with the auxiliary input z.*
2. *Recall that all honest parties send their inputs $A(\alpha_j), B(\alpha_j)$ to the trusted party computing $\widetilde{F}^{mult}_{VSS}$. Then, $\widetilde{F}^{mult}_{VSS}$ reconstructs the polynomials $A(x), B(x)$ and chooses a random polynomial $C(x, y)$. It then sends the simulator the values $\{A(\alpha_i), B(\alpha_i)\}_{i\in I}$ and the shares $\{C(x, \alpha_i), C(\alpha_i, y)\}_{i\in I}$.*
3. *$\mathcal{S}$ chooses $t-1$ random degree-t polynomials $D^{\mathcal{S}}_2(x), \ldots, D^{\mathcal{S}}_t(x)$. In addition, it chooses random degree-t polynomial $D^{\mathcal{S}}_1(x)$ under the constraint that:*

$$D^{\mathcal{S}}_1(\alpha_i) = (\alpha_i)^{-1} \cdot \left(A'(\alpha_i) \cdot B'(\alpha_i) - C(\alpha_i, 0) - \sum_{\ell=2}^{t} (\alpha_i)^\ell \cdot D^{\mathcal{S}}_\ell(\alpha_i) \right)$$

   for every $i \in I$.
4. *$\mathcal{S}$ chooses the bivariate polynomials $D^{\mathcal{S}}_1(x, y), \ldots, D^{\mathcal{S}}_t(x, y)$ under the constraint that $D^{\mathcal{S}}_j(x, 0) = D^{\mathcal{S}}_j(x)$ for every $j = 1, \ldots, t$.*

PROTOCOL 3.3.8 (Computing $\widetilde{F}_{VSS}^{mult}$ in the $(\widetilde{F}_{VSS}, F_{eval}, \widetilde{F}_{extend})$-hybrid model)

Input:

1. The dealer P_1 holds two degree-t polynomials A and B.
2. Each party P_i holds a pair of shares a_i and b_i such that $a_i = A(\alpha_i)$ and $b_i = B(\alpha_i)$.

Common input: A field description $\mathbb{F}$ and n distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$. **Aiding ideal functionality initialization:** Upon invocation, the trusted party computing the (fictitiously corruption-aware) functionality $\widetilde{F}_{VSS}$, and the corruption-aware functionality $\widetilde{F}_{extend}$ and F_{eval} receives the set of corrupted parties I.

The protocol:

1. *Dealing phase:*
   (a) The dealer P_1 defines the degree-$2t$ polynomial $D(x) = A(x) \cdot B(x)$;
   Denote $D(x) = a \cdot b + \sum_{k=1}^{2t} d_k \cdot x^k$.
   (b) P_1 chooses t^2 values $\{r_{\ell,j}\}$ uniformly and independently at random from $\mathbb{F}$, where $\ell = 1, \ldots, t$, and $j = 0, \ldots, t-1$. For every $\ell = 1, \ldots, t$, the dealer defines the polynomial $D_\ell(x)$: $D_\ell(x) = \sum_{m=0}^{t-1} r_{\ell,m} \cdot x^m + \left(d_{\ell+t} - \sum_{m=\ell+1}^{t} r_{m,t+\ell-m}\right) \cdot x^t$.
   (c) P_1 computes the polynomial: $C'(x) = D(x) - \sum_{\ell=1}^{t} x^\ell \cdot D_\ell(x)$.
   (d) P_1 chooses t random degree-t bivariate polynomials $D_1(x,y), \ldots, D_t(x,y)$ under the constraint that $D_\ell(x,0) = D_\ell(x)$ for every $\ell = 1, \ldots, t$. In addition, it chooses a random bivariate polynomial $C(x,y)$ of degree-t under the constraint that $C(x,0) = C'(x)$.
   (e) P_1 invokes the $\widetilde{F}_{VSS}$ functionality as dealer with the following inputs: $C(x,y)$, and $D_\ell(x,y)$ for every $\ell = 1, \ldots, t$.
2. Each party P_i works as follows:
   (a) If any of the shares it receives from $\widetilde{F}_{VSS}$ equal $\perp$ then P_i proceeds to the *reject phase.*
   (b) P_i computes $c'_i \overset{\text{def}}{=} a_i \cdot b_i - \sum_{k=1}^{t} (\alpha_i)^k \cdot D_k(\alpha_i, 0)$. If $C(\alpha_i, 0) \neq c'_i$, then P_i broadcasts $(\mathsf{complaint}, i)$; note that $C(\alpha_i, y)$ is part of P_i's output from $\widetilde{F}_{VSS}$ with $C(x,y)$.
   (c) If any party P_k broadcast $(\mathsf{complaint}, k)$ then go to the *complaint resolution phase.*
3. *Complaint resolution phase:*
   (a) P_1 chooses two random bivariate polynomials $A(x,y)$, $B(x,y)$ of degree t under the constraint that $A(x,0) = A(x)$ and $B(x,0) = B(x)$.
   (b) The parties invoke the $\widetilde{F}_{extend}$ functionality twice, where P_1 inserts $A(x,y)$, $B(x,y)$ and each party inserts a_i, b_i. If any one of the outputs is $\perp$ (in which case all parties receive $\perp$), P_i proceeds to *reject phase.*
   (c) The parties run the following for every $(\mathsf{complaint}, k)$ message:
      i. Run $t+3$ invocations of $\widetilde{F}_{eval}^{k}$, with each party P_i inputting its shares of $A(x,y)$, $B(x,y)$, $D_1(x,y), \ldots, D_t(x,y)$, $C(x,y)$, respectively.
      Let $A(\alpha_k, y), B(\alpha_k, y), D_1(\alpha_k, y), \ldots, D_t(\alpha_k, y), C(\alpha_k, y)$ be the resulting shares (we ignore the dual shares $S(x, \alpha_k)$ for each polynomial).
      ii. If: $C(\alpha_k, 0) \neq A(\alpha_k, 0) \cdot B(\alpha_k, 0) - \sum_{\ell=1}^{t} \alpha_k^\ell D_\ell(\alpha_k, 0)$, proceed to the *reject phase.*
4. *Reject phase:*
   (a) Every party P_i sends a_i, b_i to all P_j. Party P_i defines the vector of values $\vec{a} = (a_1, \ldots, a_n)$ that it received, where $a_j = 0$ if it was not received at all. P_i sets $A'(x)$ to be the output of Reed-Solomon decoding on $\vec{a}$. Do the same for $B'(x)$.
   (b) Every party P_i sets $C(x, \alpha_i) = C(\alpha_i, y) = A'(0) \cdot B'(0)$; a constant polynomial.
5. *Outputs:* Every party P_i outputs $C(x, \alpha_i), C(\alpha_i, y)$. Party P_1 outputs $(A(x), B(x), C(x,y))$.

5. *$\mathcal{S}$ simulates the $\widetilde{F}_{VSS}$ invocations (Step 1e of Protocol 3.3.8), and gives the adversary for every $i \in I$ the bivariate shares $\langle D_1^{\mathcal{S}}(x,\alpha_i), D_1^{\mathcal{S}}(\alpha_i,y)\rangle, \ldots, \langle D_t^{\mathcal{S}}(x,\alpha_i), D_t^{\mathcal{S}}(\alpha_i,y)\rangle$. In addition, it gives the bivariate shares $\langle C(x,\alpha_i), C(\alpha_i,y)\rangle$ that it has received from $\widetilde{F}_{VSS}^{mult}$.*

6. *If the adversary $\mathcal{A}$ broadcast a complaint message, for some $i \in I$, then the protocol proceed to the complaint resolution phase (Step 3 in Protocol 3.3.8).*

   $\mathcal{S}$ selects two (arbitrary) degree-t univariate polynomials $\hat{A}(x), \hat{B}(x)$ under the constraint that $\hat{A}(\alpha_i) = A(\alpha_i)$ and $\hat{B}(\alpha_i) = B(\alpha_i)$ for every $i \in I$. Then, it chooses two degree-t bivariate polynomials $\widetilde{A}(x,y)$, $\widetilde{B}(x,y)$ uniformly at random, under the constraints that $\widetilde{A}(x,0) = \hat{A}(x)$ and $\widetilde{B}(x,0) = \hat{B}(x)$.

   It then sends the bivariate shares $\langle \widetilde{A}(x,\alpha_i), \widetilde{A}(\alpha_i,y)\rangle, \langle \widetilde{B}(x,\alpha_i), \widetilde{B}(\alpha_i,y)\rangle$ for every $i \in I$, as the outputs of the $\widetilde{F}_{extend}$ functionality.

7. *For every $k \in I$ for which $\mathcal{A}$ instructs a corrupted party P_k to broadcast a $(\mathsf{complaint}, k)$ message, $\mathcal{S}$ simulates the complaint resolution phase by simulating P_i receiving outputs $(\widetilde{A}(x,\alpha_k), \widetilde{A}(\alpha_k,y))$, $(\widetilde{B}(x,\alpha_k), \widetilde{B}(\alpha_k,y))$, $(C(x,\alpha_k), C(\alpha_k,y))$, $(D_1^{\mathcal{S}}(x,\alpha_k), D_1^{\mathcal{S}}(\alpha_k,y))$, $\ldots$, $(D_t^{\mathcal{S}}(x,\alpha_k), D_t^{\mathcal{S}}(\alpha_k,y))$ from $\widetilde{F}_{eval}^k$, for every $i \in I$ (Step 3(c)i in Protocol 3.3.8).*

8. *$\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.*

We prove that for every $I \subseteq [n]$ where $1 \notin I$, for every $z \in \{0,1\}^*$ and all vectors of inputs $\vec{x}$:

$$\left\{\mathrm{IDEAL}_{\widetilde{F}_{VSS}^{mult}, \mathcal{S}(z), I}(\vec{x})\right\} \equiv \left\{\mathrm{HYBRID}_{\pi, \mathcal{A}(z), I}^{\widetilde{F}_{VSS}, \widetilde{F}_{extend}, \widetilde{F}_{eval}}(\vec{x})\right\}$$

We begin by showing that the output of the honest parties is distributed identically in the ideal world and the hybrid world. Then, we show that the view of the corrupted parties is distributed identically, when the output of the honest parties is given.

The honest parties' outputs. We analyze the distribution of the output of the honest parties. Let $A(x), B(x)$ be the two polynomials that are defined implicitly from the points of the honest parties inputs. Then, in the ideal world the trusted party chooses $C(x,y)$ uniformly at random from $\mathcal{B}^{A(0)\cdot B(0),t}$. The output of the honest parties fully defines these polynomials.

In the protocol execution, we have that the honest dealer chooses $D_1^{\mathcal{S}}(x), \ldots, D_t^{\mathcal{S}}(x)$ as instructed. Thus, it is clear that the constant term of $C(x,y)$ is $A(0)\cdot B(0)$ and the honest parties never complaint. In addition, relying on the $\widetilde{F}_{eval}$, $\widetilde{F}_{extend}$ functionalities, in the complaint resolution phase the honest parties never accept a (false) complaint. Therefore, each honest party outputs the bivariate share $\langle C(x,\alpha_i), C(\alpha_i,y)\rangle$.

We now show that the polynomial $C(x,y)$ is uniformly distributed in $\mathcal{B}^{A(0)B(0),t}$. The dealer has chosen $C(x,y)$ uniformly at random under the constraint that $C(x,0) = C'(x)$, and thus it is enough to show that $C'(x)$ is distributed uniformly at random in $\mathcal{P}^{A(0)B(0),t}$. However, this follows immediately from Theorem 2.6.14. We conclude that the output of the honest parties is distributed identically in the real and ideal executions.

The adversary's view. We now show that the view of the adversary is distributed identically in the real protocol and ideal executions, given the honest parties' outputs. Fixing the honest parties outputs, determine the polynomials $A'(x), B'(x), C(x,y)$.

The view of the adversary in the first round of an execution of the real protocol consists of the polynomials:

$$\left\{\left\langle D_1^{\mathcal{S}}(x,\alpha_i), D_1^{\mathcal{S}}(\alpha_i,y)\right\rangle, \ldots, \left\langle D_t^{\mathcal{S}}(x,\alpha_i), D_t^{\mathcal{S}}(\alpha_i,y)\right\rangle, \left\langle C(x,\alpha_i), C(\alpha_i,y)\right\rangle\right\}_{i\in I}$$

that are received from the invocations of $\widetilde{F}_{VSS}$ (Step 1e in Protocol 3.3.8). Furthermore, in case any corrupted party broadcast a complaint, in addition to the above, the adversary's view consists of the polynomials:

$$\left\{\left\langle \widetilde{A}(x,\alpha_i), \widetilde{A}(\alpha_i,y)\right\rangle, \left\langle \widetilde{B}(x,\alpha_i), \widetilde{B}(\alpha_i,y)\right\rangle\right\}_{i\in I}$$

which are the output of the $\widetilde{F}_{extend}$ functionality. Without loss of generality, we assume that the adversary always outputs a complaint message, for some i and thus we consider the distribution after the executions of $\widetilde{F}_{extend}$.

From Theorem 2.6.14, and since the univariate polynomials $D_1(x), \ldots, D_t(x)$ are chosen similarly in both theorems (i.e., in Protocol 2.6.15 and Protocol 3.3.8, and in the respective simulators of Theorems 2.6.14 and 3.3.9), we conclude that all the shares on the univariate polynomials $\{D_1(\alpha_i), \ldots, D_t(\alpha_i)\}_{i\in I}$ (and $\{D_1^{\mathcal{S}}(\alpha_i), \ldots, D_t^{\mathcal{S}}(\alpha_i)\}_{i\in I}$) are distributed identically in the real and ideal executions, even when conditioning on the fixed polynomials $A(x), B(x), C(x,0)$, and also when conditioning on the whole bivariate polynomial $C(x,y)$ (since $C(x,y)$ is chosen uniformly at random under the constraint that $C(x,0) = C(x)$, and therefore the view is independent of it).

Now, in the ideal execution, for every ℓ the simulator chooses bivariate polynomials $D_\ell^{\mathcal{S}}(x,y)$ uniformly and independently at random under the constraint that $D_\ell^{\mathcal{S}}(x,0) = D_\ell(x)$, and in addition, it chooses $\widetilde{A}(x,y), \widetilde{B}(x,y)$ under the constraint that $\widetilde{A}(x,0) = \hat{A}(x)$ and $\widetilde{B}(x,0) = \hat{B}(x)$. In the real execution, the dealer chooses the bivariate polynomials in a similar way: For every ℓ it chooses $D_\ell(x,y)$ uniformly at random under the constraint that $D_\ell(x,0) = D_\ell(x)$ and in addition, it chooses $A(x,y), B(x,y)$ uniformly at random under the constraint that $A(x,0) = A(x)$ and $B(x,0) = B(x)$. As we have seen above, all the polynomials in the simulation and in the real executions agree on the points $\{\alpha_i\}_{i\in I}$ (i.e., $D_\ell^{\mathcal{S}}(\alpha_i,0) = D_\ell(\alpha_i,0)$ for every $\ell = 1, \ldots, t$, $A(\alpha_i,0) = \hat{A}(\alpha_i,0)$ and $B(\alpha_i,0) = \hat{B}(\alpha_i,0)$). Moreover, Claim 2.5.4 tells as that the shares of corrupted partes of two bivariate polynomials, where each polynomial is based on some univariate polynomial that agree on the points of corrupted parties, are distributed identically. That is, for instance, the bivariate shares of the two polynomials $D_1(x,y)$ and $D_1^{\mathcal{S}}(x,y)$, the bivariate polynomial are chosen based on univariate polynomials $D_1(x,0)$ and $D_1^{\mathcal{S}}(x,0)$ that agree on all point $\{\alpha_i\}_{i\in I}$. This is true also for $D_2, \ldots, D_t$ (and $D_2^{\mathcal{S}}, \ldots, D_t^{\mathcal{S}}$), and also the outputs of $\widetilde{F}_{extend}$. Therefore, using Claim 2.5.4, we conclude that:

$$\begin{aligned}&\left\{(D_1^{\mathcal{S}}(x,\alpha_i), D_1^{\mathcal{S}}(\alpha_i,y)) \ldots, (D_t^{\mathcal{S}}(x,\alpha_i), D_t^{\mathcal{S}}(\alpha_i,y)), (\widetilde{A}(x,\alpha_i), \widetilde{A}(\alpha_i,y)), (\widetilde{B}(x,\alpha_i), \widetilde{B}(\alpha_i,y))\right\}_{i\in I} \equiv \\ &\quad \left\{(D_1(x,\alpha_i), D_1(\alpha_i,y)) \ldots, (D_t(x,\alpha_i), D_t(\alpha_i,y)), (A(x,\alpha_i), A(\alpha_i,y)), (B(x,\alpha_i), B(\alpha_i,y))\right\}_{i\in I}\end{aligned}$$

where both distributions are conditioned on the output of the honest parties. This concludes that the view of the adversary is the same conditioned on the output of the honest parties.

Case 2 - the Dealer P_1 is Corrupted

This case is simpler. Loosely speaking, security holds because (a) the dealer receives no messages from the honest parties (unless there are complaints in which case it learn nothing it did not know), and (b) any deviation by a corrupted dealer from the prescribed instructions is unequivocally detected in the verify phase via the $\widetilde{F}_{eval}$ invocations. We now describe the simulator.

The simulator $\mathcal{S}$.

1. *$\mathcal{S}$ invokes $\mathcal{A}$ with the auxiliary input z.*
2. *Recall that the honest parties send their inputs to the trusted party computing $\widetilde{F}^{mult}_{VSS}$, and the later reconstruct the polynomials $A(x)$ and $B(x)$. $\mathcal{S}$ receives $A(x), B(x)$ from $\widetilde{F}^{mult}_{VSS}$ (Step 4a in Functionality 3.3.7).*
3. *$\mathcal{S}$ receives the polynomials $C(x,y), D_1(x,y), \ldots, D_t(x,y)$ that $\mathcal{A}$ instructs the corrupted dealer to use in the $\widetilde{F}_{VSS}$ invocations (Step 1e in Protocol 3.3.8).*
4. *If $\deg(C) > t$ or if $\deg(D_\ell) > t$ for some $\ell = 1, \ldots, t$, then $\mathcal{S}$ proceeds to step 9.*
5. *For every $j \notin I$, such that $C(\alpha_k, 0) \neq A(\alpha_k) \cdot B(\alpha_k) - \sum_{m=1}^{t} \alpha_k^m \cdot D_m(\alpha_k, 0)$, simulator $\mathcal{S}$ simulates P_k broadcasting $(\mathsf{complaint}, k)$.*
6. *In case any party (either the simulator or the adversary $\mathcal{A}$ itself) has broadcast a $\mathsf{complaint}$, $\mathcal{S}$ receives from the adversary the bivariate polynomials $\widetilde{A}(x,y), \widetilde{B}(x,y)$. It then checks that $\widetilde{A}(x,0) = A(x)$ and $\widetilde{B}(x,0) = B(x)$. If the above does not hold, it sends $\perp$ as the output from the $\widetilde{F}_{extend}$ functionality, and proceeds to Step 9.*
7. *For every $(\mathsf{complaint}, k)$ message that was broadcast by a corrupted party P_k, the simulator $\mathcal{S}$ generates the results of the $\widetilde{F}_{eval}$ executions: It simply sends the necessary bivariate shares on $\widetilde{A}, \widetilde{B}, C, D_1^{\mathcal{S}}, \ldots, D_t^{\mathcal{S}}$. If there exists a k such that $C(\alpha_k, 0) \neq A'(\alpha_k, 0) \cdot B'(\alpha_k, 0) - \sum_{\ell=1}^{t} (\alpha_k)^\ell \cdot D_\ell^{\mathcal{S}}(\alpha_k, 0)$, then $\mathcal{S}$ proceeds to Step 9.*
8. *If $\mathcal{S}$ reaches this points, then it sends $C(x,y)$, as obtained from $\mathcal{A}$ above, to $\widetilde{F}^{mult}_{VSS}$.*
9. *Simulating reject:*
   (a) *$\mathcal{S}$ sends $\hat{C}(x,y) = x^{t+1}$ to the trusted party computing $\widetilde{F}^{mult}_{VSS}$ (i.e., $\mathcal{S}$ sends a polynomial $\hat{C}$ such that $\deg(\hat{C}) > t$).*
   (b) *$\mathcal{S}$ simulates every honest party P_j broadcasting $a_j = A(\alpha_j)$ and $b_j = B(\alpha_j)$ as in the reject phase (Step 4 in Protocol 3.3.8).*
   (c) *$\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.*

Since the simulator receives from $\widetilde{F}^{mult}_{VSS}$ the polynomials $A(x), B(x)$ it therefore can compute all actual inputs of the honest parties. Therefore, the view generated by the simulator is identical to the view in the actual execution. We now show that the output of the honest parties is distributed identically in the real and ideal, given the view of the corrupted parties. This follows from the same arguments as the proof of Theorem 2.6.14. If there is no reject, then from the exact same arguments it must hold that $C'(x)$ is a degree-t polynomial with constant term ab. Thus, using $\widetilde{F}_{VSS}$, the polynomial $C(x, y)$ is a degree-t bivariate polynomial that hides ab . On the other hand, if there is reject, then clearly in both executions the parties output the constant polynomial $A(0) \cdot B(0)$.

■

3.3.5 The $\widetilde{F}_{mult}$ Functionality and its Implementation

We are finally ready to show how to securely compute the product of shared values, in the presence of malicious adversaries.

The functionality. The functionality is similar to that of the F_{mult} functionality, with the only difference that the invariant of the protocol is that now the value on each wire is hidden by *bivariate* polynomial instead of a univariate polynomial. Therefore, the inputs of the parties are now shares on two degree-t bivariate polynomials $A(x, y), B(x, y)$, and the goal is to compute shares on some random degree-t bivariate polynomial $C(x, y)$ such that $C(0, 0) = A(0, 0) \cdot B(0, 0)$. We note that the corrupted parties are able to influence the output and determine the shares of the corrupted parties in the output bivariate polynomial $C(x, y)$. Therefore, we let the adversary to choose its output (i.e., its output shares), and the functionality chooses the output polynomial $C(x, y)$ to agree with the shares of the adversary.

In order to simplify the proof of security, and in order to avoid proving some non-trivial statements regarding distributions of bivariate polynomials, we define an ideal functionality that is *"less secure"* than the real protocol. In particular, the real protocol enable the adversary to influence $|I|$ bivariate shares on the output polynomial $C(x, y)$. However, we allow the ideal adversary to influence exactly t shares on the output polynomial $C(x, y)$. Of course, in case $|I| = t$ this makes no difference, but it does give the ideal adversary more ability in case where $|I| < t$. However, this formulation of functionality suffices for our needs in the overall BGW protocol, and simplifies immensely the proof of security.

FUNCTIONALITY 3.3.10 (Functionality $\widetilde{F}_{mult}$ for emulating a multiplication gate)

$\widetilde{F}_{mult}$ receives a set of indices $I \subseteq [n]$ and works as follows:

1. The $\widetilde{F}_{mult}$ functionality receives the inputs of the honest parties $\{\langle f_j^a(x), g_j^a(y)\rangle, \langle f_j^b(x), g_j^b(y)\rangle\}_{j \notin I}$. Let $A(x,y), B(x,y)$ be the unique degree-t bivariate polynomials determined by these, respectively. (If such polynomials do not exist then no security is guaranteed; see Footnote 2.)
2. $\widetilde{F}_{mult}$ sends $\{\langle A(x,\alpha_i), A(\alpha_i,y)\rangle, \langle B(x,\alpha_i), B(\alpha_i,y)\rangle\}_{i\in I}$ to the (ideal) adversary.
3. $\widetilde{F}_{mult}$ receives a degree-t bivariate polynomial $H(x,y)$ from the (ideal) adversary. If the adversary sends a polynomial of higher degree, truncate it to polynomial of degree-t in both variables.
4. $\widetilde{F}_{mult}$ defines a degree-t bivariate polynomial $C(x,y)$ such that:
   (a) $C(0,0) = A(0,0) \cdot B(0,0)$,
   (b) Let $\mathcal{T}$ be a set of size exactly t indices such that $I \subseteq \mathcal{T}$, where $\mathcal{T}$ is fully determined from I; Then, for every $i \in \mathcal{T}$, set $C(x,\alpha_i) = H(x,\alpha_i)$ and $C(\alpha_i,y) = H(\alpha_i,y)$.

   (such a degree-t polynomial always exists from Claim 3.2.1.)
5. *Output:* The functionality $\widetilde{F}_{mult}$ sends the polynomial $\langle C(x,\alpha_j), C(\alpha_j,y)\rangle$ to every honest party P_j ($j \notin I$).

   (There is no more output for the adversary.)

The protocol idea. The protocol idea is similar to the implementation in the original BGW, with the following major difference: The parties do not need to subshare their shares, since these are *already* shared. As input, each party P_i holds the bivariate shares $\langle A(x,\alpha_i), A(\alpha_i,y)\rangle$, $\langle B(x,\alpha_i), B(\alpha_i,y)\rangle$, where $A(x,y)$, $B(x,y)$ are degree-t bivariate polynomials with constant terms a, b, respectively. Thus, the polynomial $A(0,y)$ is a degree-t univariate polynomial with constant term $A(0,0) = a$. In addition, $B(0,y)$ is a degree-t bivariate polynomial $B(0,\alpha_i)$, and so $A(0,y) \cdot B(0,y)$ is a degree-$2t$ polynomial with the desired constant term ab. As was shown in [51], there exist constants $\gamma_1, \ldots, \gamma_n$ such that:

$$ab = \gamma_1 \cdot A(0,\alpha_1) \cdot B(0,\alpha_1) + \ldots + \gamma_n \cdot A(0,\alpha_n) \cdot B(0,\alpha_n) \tag{3.3.2}$$

We refer to the values $A(0,\alpha_i), B(0,\alpha_i)$ as the actual shares of party P_i, and those values are the constant terms of the polynomials $A(x,\alpha_i), B(x,\alpha_i)$, respectively, which are part of the shares of P_i. Observe that each party P_j holds the values $A(\alpha_j,\alpha_i)$ and $B(\alpha_j,\alpha_i)$. Therefore, P_i can use the $\widetilde{F}_{VSS}^{mult}$-functionality, and to distribute some random degree-t bivariate polynomial $C_i(x,y)$ with constant term $A(0,\alpha_i)\cdot B(0,\alpha_i)$. Since P_i distributes it using $\widetilde{F}_{VSS}^{mult}$, all parties are able to verify that the constant term of C_i is correct. After each party distributes the product of its shares, we have n bivariate polynomials $C_i(x,y)$ that are shared among the parties, where the constant term of each polynomial is $A(0,\alpha_i) \cdot B(0,\alpha_i)$, respectively. Now, let:

$$C(x,y) \stackrel{\text{def}}{=} \gamma_1 \cdot C_1(x,y) + \ldots + \gamma_n \cdot C_n(x,y)$$

observe that this is a bivariate polynomial of degree-t in both variables, its constant term is ab, and each party can compute its share on this polynomial using the same linear combination on its shares of $C_1, \ldots, C_n$. We now formally describe the protocol.

PROTOCOL 3.3.11 (Computing $\widetilde{F}_{mult}$ in the $\widetilde{F}^{mult}_{VSS}$-hybrid model)

- **Input:** Each party P_i holds $(\langle f_i^a(x), g_i^a(y)\rangle, \langle f_i^b(x), g_i^b(y)\rangle)$ for some bivariate polynomials $A(x,y), B(x,y)$ of degree t, which hide a, b, respectively. (If not all the points lie on a single degree-t bivariate polynomial, then no security guarantees are obtained. See Footnote 2.)
- **Common input:** The description of a field $\mathbb{F}$ and n distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$. In addition, the parties have constants $\gamma_1, \ldots, \gamma_n$ which are the first row of the inverse of the Vandemonde matrix (see [51]).
- **Aiding ideal functionality initialization:** Upon invocation, the trusted party computing the corruption-aware functionality $\widetilde{F}^{mult}_{VSS}$ receives the set of corrupted parties I.
- **The protocol:**
  1. For every $i = 1, \ldots, n$, the parties invoke the $\widetilde{F}^{mult}_{VSS}$ functionality as follows:
     (a) *Inputs:* In the ith invocation, party P_i plays the dealer. We recall that all parties hold shares on the univariate polynomials $A(x, \alpha_i) = f_i^a(x)$ and $B(x, \alpha_i) = f_i^b(x)$. Specifically, each party P_j sends $\widetilde{F}^{mult}_{VSS}$ their shares $g_j^a(\alpha_i) = A(\alpha_j, \alpha_i) = f_i^a(\alpha_j)$ and $g_j^b(\alpha_i) = B(\alpha_j, \alpha_i) = f_i^b(\alpha_j)$.
     (b) *Outputs:* The functionality $\widetilde{F}^{mult}_{VSS}$ chooses a random degree-t bivariate polynomial $C_i(x,y)$ with constant term $f_i^a(0) \cdot f_i^b(0) = A(0, \alpha_i) \cdot B(0, \alpha_i)$. Every party P_j receives the bivariate shares $\langle C_i(x, \alpha_j), C_i(\alpha_j, y)\rangle$, for every $1 \le j \le n$.
  2. At this stage, each party P_i holds polynomials $C_1(x, \alpha_i), \ldots, C_n(x, \alpha_i)$ and $C_1(\alpha_i, y), \ldots, C_n(\alpha_i, y)$. Then, it locally computes $C(x, \alpha_i) = \sum_{j=1}^n \gamma_j \cdot C_j(x, \alpha_i)$, and $C(\alpha_i, y) = \sum_{j=1}^n \gamma_j \cdot C_j(\alpha_i, y)$.
- **Output:** Each party P_i outputs $\langle C(x, \alpha_i), C(\alpha_i, y)\rangle$.

Theorem 3.3.12 *Let $t < n/3$. Then, Protocol 3.3.11 is t-secure for the $\widetilde{F}_{mult}$ functionality in the $\widetilde{F}^{mult}_{VSS}$-hybrid model, in the presence of a static malicious adversary.*

Proof: We start with the specification of the simulator $\mathcal{S}$.

The simulator $\mathcal{S}$.

1. *$\mathcal{S}$ invokes the adversary $\mathcal{A}$ with the auxiliary input z.*
2. *Recall that the honest parties send $\widetilde{F}_{mult}$ their input bivariate shares, and that the latter reconstructs the bivariate polynomials $A(x,y)$ and $B(x,y)$. Then, it sends $\mathcal{S}$ the bivariate shares $\{\langle A(x, \alpha_i), A(\alpha_i, y)\rangle, \langle B(x, \alpha_i), B(\alpha_i, y)\rangle\}_{i \in I}$.*
3. *For every $j \notin I$, $\mathcal{S}$ simulates the $\widetilde{F}^{mult}_{VSS}$ invocation where the honest party P_j is dealer (Step 1 in Protocol 3.3.11):*
   (a) $\mathcal{S}$ chooses a random polynomial $C_j(x,y) \in_R \mathcal{B}^{0,t}$, and hands $\mathcal{A}$ the univariate shares $\{A(\alpha_i, \alpha_j), B(\alpha_i, \alpha_j)\}_{i \in I}$ (as the correct inputs of the corrupted parties), and the output polynomials $\{\langle C_j(x, \alpha_i), C_j(\alpha_i, y)\rangle\}_{i \in I}$ (as the output shares of the chosen polynomial $C_j(x,y)$).

4. *For every $i \in I$, $\mathcal{S}$ simulates the $\widetilde{F}_{VSS}^{mult}$ invocation where the corrupted party P_i is the dealer (Step 1 in Protocol 3.3.11):*

   (a) *$\mathcal{S}$ hands the adversary $\mathcal{A}$ the bivariate shares $\langle A(x, \alpha_i), A(\alpha_i, y)\rangle, \langle B(x, \alpha_i), B(\alpha_i, y)\rangle$ as if incoming from $\widetilde{F}_{VSS}^{mult}$ (Step 4a of Functionality 3.3.7).*

   (b) *$\mathcal{S}$ receives from $\mathcal{A}$ the bivariate polynomial $C_i(x, y)$ that $\mathcal{A}$ sends to $\widetilde{F}_{VSS}^{mult}$ (Step 4b of Functionality 3.3.7).*

   (c) *$\mathcal{S}$ checks that $C_i(x, y)$ is of degree-t in both variables, and that $C_i(0, 0) = A(0, \alpha_i) \cdot B(0, \alpha_i)$. If one of this checks fails, then $\mathcal{S}$ resets $C_i(x, y) = A(0, \alpha_i) \cdot B(0, \alpha_i)$.*

   (d) *$\mathcal{S}$ hands $\mathcal{A}$ the polynomial $C_i(x, y)$ and the bivariate shares $\{\langle C_i(x, \alpha_k), C_i(\alpha_k, y)\rangle\}_{k\in I}$ (Step **??** of Functionality 3.3.7).*

5. *For every $i \in I$, the simulator $\mathcal{S}$ computes $H(x, y) = \sum_{j=1}^{n} \gamma_j \cdot C_j(x, y)$ and sends H to the $\widetilde{F}_{mult}$ functionality. Recall that at this stage, $\widetilde{F}_{VSS}^{mult}$ chooses a polynomial $C(x, y)$ that agree with the univariate shares of the corrupted parties in H and with $H(0, 0) = A(0, 0) \cdot B(0, 0)$, and sends the outputs to the honest parties.*

6. *$\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.*

The proof is similar to the proof of Theorem 2.6.18. The only difference between the simulation with $\mathcal{S}$ and $\mathcal{A}$, and the real execution of Protocol 3.3.11 with $\mathcal{A}$ is due to the fact that for every honest party $j \notin I$, $\mathcal{S}$ chooses the polynomial $C_i(x, y)$ to have constant term 0 instead of $A(0, \alpha_j) \cdot B(0, \alpha_j)$ (which it does not know). We therefore present a fictitious simulation, where the simulator gets as auxiliary information the values $A(0, \alpha_j) \cdot B(0, \alpha_j)$ for every $j \notin I$. Then, we show that an execution of the fictitious simulation and the real execution are identically distributed, and we also show that execution of the fictitious simulation and the original simulation are identically distributed. This implies what we have to prove.

A fictitious simulator. Let $\mathcal{S}'$ be exactly the same as $\mathcal{S}$, except that it receives for input the values $A(0, \alpha_j) \cdot B(0, \alpha_j)$ for every $j \notin I$. Then, instead of choosing $C_j(x) \in_R \mathcal{B}^{0,t}$, it chooses $C_j(x) \in_R \mathcal{B}^{A(0,\alpha_j)\cdot B(0,\alpha_j),t}$. We stress that $\mathcal{S}'$ runs in the ideal model with the same trusted party running $\widetilde{F}_{mult}$ as $\mathcal{S}$, and the honest parties receive output as specified by $\widetilde{F}_{mult}$ when running with the ideal adversary $\mathcal{S}$ or $\mathcal{S}'$.

The original and fictitious simulations. We begin by showing that the joint output of the adversary and honest parties is identical in the original and alternative simulations. That is,

$$\left\{\text{IDEAL}_{\widetilde{F}_{mult},\mathcal{S}(z),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n, z\in\{0,1\}^*} \equiv \left\{\text{IDEAL}_{\widetilde{F}_{mult},\mathcal{S}'(z'),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n, z\in\{0,1\}^*}$$

where z' contains the same z as $\mathcal{A}$ receives, together with the $A(0, \alpha_j) \cdot B(0, \alpha_j)$ values. The proof for this is identical to the analogue claim in the proof of Theorem 2.6.18, where here we use Corollary 3.2.2 to show that the bivariate shares $\{C_j(x, \alpha_i), C(\alpha_i, y)\}_{i\in I; j\notin I}$ when $C_j(0, 0) = 0$ are identically distributed to the bivariate shares $\{C_j(x, \alpha_i), C(\alpha_i, y)\}_{i\in I; j\notin I}$ when $C_j(0, 0) = A(0, \alpha_j) \cdot B(0, \alpha_j)$ (both are univariate polynomial that are chosen uniformly at random except for the constant term).

The fictitious simulation and a protocol execution. We now proceed to show that the joint output of the adversary and honest parties are identical in a protocol execution and in the alternative simulation.:

$$\left\{\text{IDEAL}_{\widetilde{F}_{mult},\mathcal{S}'(z'),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n, z\in\{0,1\}^*} \equiv \left\{\text{HYBRID}^{\widetilde{F}^{mult}_{VSS}}_{\pi,\mathcal{A}(z),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n, z\in\{0,1\}^*} .$$

Similarly to the proof of Theorem 2.6.18, we compare the ideal execution of $\mathcal{S}'$ with $\widetilde{F}_{mult}$ to an ideal execution of yet another simulator $\hat{\mathcal{S}}$ with a new functionality $\hat{F}_{mult}$, defined as follows:

- $\hat{F}_{mult}$ is the same as $\widetilde{F}_{mult}$ except that instead of receiving the bivariate polynomial $H(x,y)$ from the ideal adversary $\hat{\mathcal{S}}$, and choosing the resulting polynomial $C(x,y)$ as in Step 4b, it receives all the polynomials $C_1(x,y),\ldots,C_n(x,y)$ and defines $H(x,y) = \sum_{k=1}^{n} \gamma_k C_k(x,y)$.
- $\hat{\mathcal{S}}$ is the same as $\mathcal{S}'$ except that instead of sending the bivariate polynomial $H(x,y)$ to the trusted party, it sends all polynomials $C_1(x,y),\ldots,C_n(x,y)$ that it defined in the (fictitious) simulation.

It is easy to see that the joint output of $\hat{S}$ and the honest parties in an ideal execution with $\hat{F}_{mult}$ is identically to the joint distribution of $\mathcal{A}$ and the honest parties in the real execution. This is due to the fact that the simulator has all the correct inputs of the honest parties, plays exactly as the honest parties in the real executions, and acts exactly as described in all the $\widetilde{F}^{mult}_{VSS}$ executions.

It remains to show that the joint output of $\hat{\mathcal{S}}$ and the honest parties in an ideal execution with $\hat{F}_{mult}$ is identically to the joint distribution of $\mathcal{S}$ and the honest parties in an ideal execution with $\widetilde{F}_{mult}$. Since the fictitious simulator $\mathcal{S}'$ knows all the correct shares of the honest parties, from Eq. (3.3.2) it holds that:

$$ab = \gamma_1 \cdot A(0,\alpha_i) \cdot B(0,\alpha_i) + \ldots + \gamma_n \cdot A(0,\alpha_n) \cdot B(0,\alpha_n) \ ,$$

Thus, $H(0,0) = \sum_{k=1}^{n} \gamma_k C_k(0,0) = ab$. Therefore, the polynomial $H(x,y)$ that is sent by $\mathcal{S}'$ to $\widetilde{F}_{mult}$ uniquely defines the polynomial $C(x,y)$, which is the exactly the same polynomial as in $\hat{\mathcal{S}}$ with $\hat{F}_{mult}$. This completes the proof. ■

3.3.6 Wrapping Things Up – Perfectly-Secure Protocol

We now briefly describe the overall protocol for any function $f(x_1,\ldots,x_n) = (y_1,\ldots,y_n)$. As the original BGW protocol, the protocol consists of three phases, the input sharing phase, the circuit emulation phase and the output reconstruction phase.

In the input sharing phase, each party P_i distributes its input x_i using the $\widetilde{F}_{VSS}$ functionality. At the end of the input sharing phase, the value of each input wire is hidden using bivariate sharing. The parties then proceed to the circuit emulation phase, where the parties emulate the computation of the circuit C gate-by-gate, by maintaining the invariant that the intermediate value on each wire is hidden using bivariate sharing. As in BGW, we assume that the circuit consists of three types of gates: addition gates, multiplication-by-a-constant gates and multiplication gates.

It is easy to see that addition gates, and multiplication-by-a-constant gates can be computed without any interaction. When the parties reach a multiplication gate, the parties invoke the

$\widetilde{F}_{mult}$ functionality, to receive shares of a bivariate polynomials that hides the multiplication $A(0,0) \cdot B(0,0)$.

In the output stage of the original protocol of BGW, each party receive n points of a polynomial of degree-t, with at most t errors. Since $n \geq 3t+1$, using the Reed-Solomon error correction decoding, each party can correct t errors and reconstruct the polynomial of the output. In the bivariate case there are two possibilities for reconstructing the output. The first, is reducing it to the univariate case, by having each party use the value $F(0, \alpha_i)$, and refer to this value as the univariate share of the polynomial $F(0, y)$. The second option is to use the redundant information and to eliminate the need for error correcting. Each party simply sends its bivariate shares $\langle F(x, \alpha_i), F(\alpha_i, y) \rangle$. Then, the parties computing the output can simply take a subset of $2t+1$ polynomials that agree with $2t+1$ polynomials, reconstruct $F(x, y)$ and output $F(0,0)$.

Efficiency analysis. In short, our protocol utilizing the bivariate properties costs up to $O(n^5)$ field elements in private channels, together with $O(n^4)$ field elements in broadcast per multiplication gate in the case of malicious behavior. We remark that when no parties actively cheat, the protocol requires $O(n^4)$ field elements in private channels and no broadcast at all.

Protocol	**Optimistic Cost**	**Dishonest Cost**
$\widetilde{\boldsymbol{F}}_{\boldsymbol{VSS}}$:	$O(n^2)$ over pt-2-pt No broadcast	$O(n^2)$ over pt-2-pt $O(n^2)$ broadcast
$\widetilde{\boldsymbol{F}}_{\boldsymbol{extend}}$:	$O(n^2)$ over pt-2-pt No broadcast	$O(n^2)$ over pt-2-pt $O(n^2)$ broadcast
$\widetilde{\boldsymbol{F}}_{\boldsymbol{eval}}$:	$O(n^2)$ over pt-2-pt No broadcast	$O(n^2)$ over pt-2-pt No broadcast
$\widetilde{\boldsymbol{F}}_{\boldsymbol{VSS}}^{\boldsymbol{mult}}$:	$O(n^3)$ over pt-2-pt No broadcast	$O(n^4)$ over pt-2-pt $O(n^3)$ broadcast
$\widetilde{\boldsymbol{F}}_{\boldsymbol{mult}}$:	$O(n^4)$ over pt-2-pt No broadcast	$O(n^5)$ over pt-2-pt $O(n^4)$ broadcast
BGW:	$O(\lvert C\rvert \cdot n^4)$ over pt-2-pt No broadcast	$O(\lvert C\rvert \cdot n^5)$ over pt-2-pt $O(\lvert C\rvert \cdot n^4)$ broadcast

Part II

Complete Fairness in Secure Two-Party Computation

Chapter 4

A Full Characterization of Functions that Imply Fair Coin-Tossing

4.1 Introduction

The well-known impossibility result of Cleve [34] showed that the coin-tossing functionality, where two parties receive the same random bit, cannot be computed with complete fairness. This result implies that it is impossible to securely compute with fairness any function that can be used to toss a coin fairly. In this chapter, we focus on the class of deterministic Boolean functions with finite domain, and we ask for which functions in this class is it possible to information-theoretically toss an unbiased coin, given a protocol for securely computing the function with fairness. We provide a *complete characterization* of the functions in this class that imply and do not imply fair coin tossing. This characterization extends our knowledge of which functions cannot be securely computed with fairness.

The criterion. Intuitively, the property that we define over the function's truth table relates to the question of whether or not it is possible for one party to singlehandedly change the probability that the output of the function is 1 (or 0) based on how it chooses its input. In order to explain the criterion, we give two examples of functions that imply coin-tossing, meaning that a fair secure protocol for computing the function implies a fair secure protocol for coin tossing. We discuss how each of the examples can be used to toss a coin fairly, and this in turn will help us to explain the criterion. The functions are given below:

(a)

	y_1	y_2	y_3
x_1	0	1	1
x_2	1	0	0
x_3	0	0	1

(b)

	y_1	y_2	y_3
x_1	1	0	0
x_2	0	1	0
x_3	0	0	1

Consider function (a), and assume that there exists a fair protocol for this function. We show how to toss a fair coin using a single invocation of the protocol for f. Before doing so, we observe that the output of a single invocation of the function can be expressed by multiplying the truth-table matrix of the function by probability vectors. Specifically, assume that party P_1 chooses input x_i with probability p_i, for $i = 1, 2, 3$ (thus $p_1 + p_2 + p_3 = 1$ since it must choose some input); likewise, assume that P_2 chooses input y_i with probability q_i. Now, let M_f be the "truth table" of the function, meaning that $M_f[i,j] = f(x_i, y_j)$. Then, the output of the invocation of f upon the inputs chosen by the parties equals 1 with probability exactly $(p_1, p_2, p_3) \cdot M_f \cdot (q_1, q_2, q_3)^T$.

We are now ready to show how to toss a coin using f. First, note that there are two complementary rows; these are the rows specified by inputs x_1 and x_2. This means that if P_1 chooses one of the inputs in $\{x_1, x_2\}$ uniformly at random, then no matter what distribution over the inputs (corrupted) P_2 uses, the result is a uniformly chosen coin. In order to see this, observe that when we multiply the vector $(\frac{1}{2}, \frac{1}{2}, 0)$ (the distribution over the input of P_1) with the matrix M_f, the result is the vector $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$. This means that no matter what input P_2 will choose, or what distribution over the inputs it may use, the output is 1 with probability 1/2 (formally, the output is 1 with probability $\frac{1}{2} \cdot q_1 + \frac{1}{2} \cdot q_2 + \frac{1}{2} \cdot q_3 = \frac{1}{2}$ because $q_1 + q_2 + q_3 = 1$). This means that if P_1 is honest, then a corrupted P_2 cannot bias the output. Likewise, there are also two complementary columns (y_1 and y_3), and thus, if P_2 chooses one of the inputs in $\{y_1, y_3\}$ uniformly at random, then no matter what distribution over the inputs (a possibly corrupted) P_1 uses, the result is a uniform coin.

In contrast, there are no two complementary rows or columns in the function (b). However, if P_1 chooses one of the inputs $\{x_1, x_2, x_3\}$ uniformly at random (i.e., each input with probability one third), then no matter what distribution P_2 will use, the output is 1 with probability 1/3. Similarly, if P_2 chooses a uniformly random input, then no matter what P_1 does, the output is 1 with the same probability. Therefore, a single invocation of the function f in which the honest party chooses the uniform distribution over its inputs results in a coin that equals 1 with probability exactly $\frac{1}{3}$, irrespective of what the other party inputs. In order to obtain an unbiased coin that equals 1 with probability $\frac{1}{2}$ the method of von-Neumann [97] can be used. This method works by having the parties use the function f to toss two coins. If the resulting coins are different (i.e, 01 or 10), then they output the result of the first invocation. Otherwise, they run the protocol again. This yields a coin that equals 1 with probability $\frac{1}{2}$ since the probability of obtaining 01 equals the probability of obtaining 10. Thus, conditioned on the results being different, the probability of outputting 0 equals the probability of outputting 1.

The criterion is a simple generalization of the examples shown above. Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0,1\}$ be a function, and let M_f be the truth table representation as described above. If there exist two probability vectors[1], $\mathbf{p} = (p_1, \ldots, p_\ell)$, $\mathbf{q} = (q_1 \ldots, q_m)$ such that $\mathbf{p} \cdot M_f$ and $M_f \cdot \mathbf{q}^T$ are both vectors that equal δ everywhere, for some $0 < \delta < 1$. Observe that if such probability vectors exist, then the function implies the coin-tossing functionality as we described above. Specifically, P_1 chooses its input according to distribution $\mathbf{p}$, and P_2 chooses its inputs according to the distribution $\mathbf{q}$. The result is then a coin that equals 1 with probability δ. Using the method of von-Neumann, this can be used to obtain a uniformly distributed coin. We conclude:

Theorem 4.1.1 (informal) *Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0,1\}$ be a function that satisfies the aforementioned criterion. Then, the existence of a protocol for securely computing f with complete fairness implies the existence of a fair coin tossing protocol.*

An immediate corollary of this theorem is that any such function cannot be securely computed with complete fairness, or this contradicts the impossibility result of Cleve [34].

[1] $\mathbf{p} = (p_1, \ldots, p_k)$ is a probability vector if $p_i \geq 0$ for every $1 \leq i \leq k$, and $\sum_{i=1}^{k} p_i = 1$.

The more interesting and technically challenging part of our work is a proof that the criterion is tight. That is, we prove the following theorem:

Theorem 4.1.2 (informal) *Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0, 1\}$ be a function that does* not *satisfy the aforementioned criterion. Then, there exists an exponential-time adversary that can bias the outcome of every coin-tossing protocol that uses ideal and fair invocations of f.*

This result has a number of ramifications. Most notably, it provides a focus of our attention for the question of fairness in two-party secure computation. Specifically, the only functions that can potentially be securely computed with fairness are those for which the property does not hold. These functions have the property that one of the parties can partially influence the outcome of the result singlehandedly, a fact that is used inherently in the protocol of [58] for the function with an embedded XOR. This does not mean that all functions of this type can be fairly computed. However, it provides a good starting point (and indeed, as we will see in Chapter 5, a large subset of these functions can be computed fairly). In addition, our results define the set of functions for which Cleve's impossibility result suffices for proving that they cannot be securely computed with fairness. Given that no function other than those implying coin tossing has been ruled out since Cleve's initial result, understanding exactly what is included in this impossibility is of importance.

On fail-stop adversaries. Our main results above consider the case of malicious adversaries. In addition, we explore the fail-stop adversary model where the adversary follows the protocol like an honest party, but can halt early. This model is of interest since the impossibility result of Cleve [34] for achieving fair coin tossing holds also for fail-stop adversaries. In order to prove theorems regarding the fail-stop model, we first provide a definition of security with complete fairness for fail-stop adversaries that follows the real/ideal simulation paradigm. Surprisingly, this turns out not to be straightforward and we provide two natural formulations that are very different regarding feasibility. The formulations differ regarding the ideal-world adversary/simulator. The question that arises is whether or not the simulator is allowed to use a different input to the prescribed one. In the semi-honest model (which differs only in the fact that the adversary cannot halt early), the standard formulation is to not allow the simulator to change the prescribed input, whereas in the malicious model the simulator is always allowed to change the prescribed input. We therefore define two fail-stop models. In the first, called "fail-stop1", the simulator is allowed to either send the trusted party computing the function the prescribed input of the party or an abort symbol $\perp$, but nothing else. In the second, called "fail-stop2", the simulator may send any input that it wishes to the trusted party computing the function. (Note, however, that if there was no early abort then the prescribed input must be used because such an execution is identical to an execution between two honest parties.)

Observe that in the first model, the honest party is guaranteed to receive the output on the prescribed inputs, unless it receives abort. In addition, observe that any protocol that is secure in the presence of malicious adversaries is secure also in the fail-stop2 model. However, this is not true of the fail-stop1 model (this is due to the fact that the simulator in the ideal model for the case of malicious adversaries is more powerful than in the fail-stop2 ideal model since the former can send any input whereas the latter can only send the prescribed input or $\perp$).

We remark that Cleve's impossibility result holds in both models, since the parties do not have inputs in the coin-tossing functionality, and therefore there is no difference in the ideal-worlds of the models in this case. In addition, the protocols of [58] that are secure for malicious adversaries are secure for fail-stop2 (as mentioned above, this is immediate), but are *not* secure for fail-stop1.

We show that in the fail-stop1 model, it is impossible to securely compute with complete fairness any function containing an embedded XOR. We show this by constructing a coin-tossing protocol from any such function, that is secure in the fail-stop model. Thus, the only functions that can potentially be securely computed with fairness are those with no embedded XOR but with an embedded OR (if a function has neither, then it is trivial and can be computed unconditionally and fairly); we note that there are very few functions with this property. We conclude that in the fail-stop1 model, fairness cannot be achieved for almost all non-trivial functions. We remark that [58] presents secure protocols that achieve complete fairness for functions that have no embedded XOR; however, they are not secure in the fail-stop1 model, as mentioned.

Regarding the fail-stop2 model, we prove an analogous result to Theorem 4.1.2. In the proof of Theorem 4.1.2, the adversary that we construct changes its input in one of the invocations of f and then continues honestly. Thus it is malicious and not fail-stop2. Nevertheless, we show how the proof can be modified in order to hold for the fail-stop2 model as well. We then show how we can modify the proof to hold in the fail-stop2 model as well.

These extensions for fail-stop adversaries deepen our understanding regarding the feasibility of obtaining fairness. Specifically, any protocol that achieves fairness for any non-trivial function (or at least any function that has an embedded XOR) must have the property that the simulator can send any input in the ideal model. Stated differently, the input that is effectively used by a corrupted party cannot be somehow committed, thereby preventing this behaviour. This also explains why the protocols of [58] have this property, and is a key ingredient for our results in Chapter 5.

Open questions. In this work we provide an almost complete characterization regarding what functions imply and do not imply coin tossing. Our characterisation is not completely tight since the impossibility result of Theorem 4.1.2 only holds in the information-theoretic setting; this is due to the fact that the adversary needs to carry out inefficient computations. Thus, it is conceivable that coin tossing can be achieved computationally from some such functions. It is important to note, however, that any function that does not fulfil our criterion implies oblivious transfer (OT). Thus, any protocol that uses such a function has access to OT and all that is implied by OT (e.g., commitments, zero knowledge, and so on). Thus, any such computational construction would have to be inherently nonblack-box in some sense. Our work also only considers finite functions (where the size of the domain is not dependent on the security parameter); extensions to other function classes, including non-Boolean functions, is also of interest.

The main open question left by our work in this chapter is to characterize which functions for which the criterion does not hold can be securely computed with complete fairness. In the next chapter, we focus on this question and show that for a large subset of these functions – complete fairness *is* possible. As we will see, our overall characterization is not complete, and there exist functions that do not imply fair coin-tossing, and we do not know whether they are possible to compute fairly. Observe that in order to show that these functions (or some subset of these functions) cannot be securely computed with complete fairness, a new impossibility result must be proven. In particular, it will not be possible to reduce the impossibility to Cleve [34] since these functions do not imply coin tossing.

4.2 Definitions and Preliminaries

In the following, we present security definitions for protocols. This is similar to security definitions appear in Section 2.2. However, here we adapt the definitions to hold in the computational settings, i.e., when the parties and the adversary are computationally bounded and assumed to be probabilistic polynomial time.

We let κ denote the security parameter. A function $\mu(\cdot)$ is *negligible* if for every positive polynomial $p(\cdot)$ and all sufficiently large κ, it holds that $\mu(\kappa) < 1/p(\kappa)$. A *distribution ensemble* $X = \{X(a,\kappa)\}_{a\in\mathcal{D},\kappa\in\mathbb{N}}$ is an infinite sequence of random variables indexes by $a \in \mathcal{D}$ and $\kappa \in \mathbb{N}$. In the context of secure computation, κ is the security parameter and $\mathcal{D}$ denotes the domain of the parties' input. Two distribution ensembles $X = \{X(a,\kappa)\}_{a\in\mathcal{D},\kappa\in\mathbb{N}}$ and $Y = \{Y(a,\kappa)\}_{a\in\mathcal{D},\kappa\in\mathbb{N}}$ are *computationally indistinguishable*, denoted $X \overset{c}{\equiv} Y$, if for every non-uniform polynomial-time-algorithm D there exists a negligible function $\mu(\cdot)$ such that for every κ and every $a \in \mathcal{D}$:

$$|\Pr[D(X(a,\kappa)) = 1] - \Pr[D(Y(a,\kappa)) = 1]| \leq \mu(\kappa)$$

We consider binary deterministic functions over a finite domain; i.e., functions $f : X \times Y \to \{0,1\}$ where $X, Y \subset \{0,1\}^*$ are finite sets. Throughout, we will denote $X = \{x_1, \ldots, x_\ell\}$ and $Y = \{y_1, \ldots, y_m\}$, for constants $m, \ell \in \mathbb{N}$.

4.2.1 Secure Two-Party Computation with Fairness – Malicious Adversaries

We now define what it means for a protocol to be *secure with complete fairness.* Our definition follows the standard definition of [27, 53], except for the fact that we require complete fairness even though we are in the two-party setting. We consider active adversaries (malicious), who may deviate from the protocol in an arbitrary manner, and static corruptions. Let π be a two party protocol for computing a function $f : X \times Y \to \{0,1\}$ over a finite domain. In this thesis, we restrict our attention to functions that return the same output to both parties. We briefly describe the real execution and the ideal execution.

Execution in the ideal model. An ideal execution involves parties P_1 and P_2, an adversary $\mathcal{S}$ who has corrupted one of the parties, and the trusted party. An ideal execution for the computation of f proceeds as follows:

Inputs: P_1 and P_2 hold inputs $x \in X$, and $y \in Y$, respectively; the adversary $\mathcal{S}$ receives the security parameter 1^κ and an auxiliary input z.

Send inputs to trusted party: The honest party sends its input to the trusted party. The corrupted party controlled by $\mathcal{S}$ may send any value of its choice. Denote the pair of inputs sent to the trusted party by (x', y').

Trusted party sends outputs: If $x' \notin X$, the trusted party sets x' to be the default value x_1; likewise if $y' \notin Y$ the trusted party sets $y' = y_1$. Next, the trusted party computes $f(x', y')$ and sends the result to P_1 and P_2.

Outputs: The honest party outputs whatever it was sent by the trusted party, the corrupted party outputs nothing and $\mathcal{S}$ outputs an arbitrary function of its view.

We denote by $\text{IDEAL}_{f,\mathcal{S}(z)}(x, y, \kappa)$ the random variable consisting of the output of the adversary and the output of the honest party following an execution in the ideal model as described above.

Execution in the real model. In the real execution, the parties P_1 and P_2 interact, where one is corrupted and therefore controlled by the real-world adversary $\mathcal{A}$. In this case, the adversary $\mathcal{A}$ gets the inputs of the corrupted party and sends all messages on behalf of this party, using an arbitrary strategy. The honest party follows the instructions of π. Let $\text{REAL}_{\pi,\mathcal{A}(z)}(x, y, \kappa)$ be the random variable consisting of the view of the adversary and the output of the honest party, following an execution of π where P_1 begins with input x, P_2 with input y, the adversary has auxiliary input z, and all parties have security parameter 1^κ.

Security by emulation. Informally, a real protocol π is secure if any "attack" carried out by a real adversary $\mathcal{A}$ on π can be carried out by the ideal adversary $\mathcal{S}$ in the ideal model. This is formalized by requiring that $\mathcal{S}$ can simulate the real-model outputs while running in the ideal model.

Definition 4.2.1 *Protocol π* securely computes f with complete fairness in the presence of malicious adversaries *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that:*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z)}(x, y, \kappa)\right\}_{x\in X, y\in Y, z\in\{0,1\}^*, \kappa\in\mathbb{N}} \overset{c}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z)}(x, y, \kappa)\right\}_{x\in X, y\in Y, z\in\{0,1\}^*, \kappa\in\mathbb{N}} .$$

Protocol π computes f with statistical security *if for every $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that the* IDEAL *and* REAL *distributions are statistically close.*

4.2.2 Secure Two-Party Computation without Fairness (Security with Abort)

The following is the standard definition of secure computation *without fairness.* This definition is needed for Chapter 5 and is unnecessary for this chapter; however, it is given here for the sake of clarity and context.

Security without fairness is formalized by changing the ideal world and allowing the adversary to learn the output alone. This implies that the case where the adversary learns the output without the honest party is not considered as a breach of security since this is allowed in the ideal world. We modify the ideal world as follows:

Inputs: As previously.

Send inputs to trusted party: As previously.

Trusted party sends output to corrupted party: If $x' \notin X$, the trusted party sets x' to be the default value x_1; likewise if $y' \notin Y$ the trusted party sets $y' = y_1$. Next, the trusted party computes $f(x', y') = (f_1(x', y'), f_2(x', y')) = (w_1, w_2)$ and sends w_i to the the corrupted party P_i (i.e., to the adversary $\mathcal{A}$).

Adversary decides whether to abort: After receiving its output, the adversary sends either $\mathsf{proceed}$ or abort message to the trusted party. If its sends $\mathsf{proceed}$ to the trusted party, the trusted party sends w_j to the honest party P_j. If it sends abort, then it sends the honest party P_j the special symbol $\perp$.

Outputs: As previously.

We denote by $\text{IDEAL}^{\mathsf{abort}}_{f,\mathcal{S}(z)}(x, y, \kappa)$ the random variable consisting of the output of the adversary and the output of the honest party following an execution in the ideal model with abort as described above. The following definition is the analogue to Definition 4.2.1 and formalizes security with abort:

Definition 4.2.2 *Protocol π* securely computes f with abort in the presence of malicious adversaries *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that:*

$$\left\{\text{IDEAL}^{\mathsf{abort}}_{f,\mathcal{S}(z)}(x, y, \kappa)\right\}_{x\in X, y\in Y, z\in\{0,1\}^*, \kappa\in\mathbb{N}} \overset{\mathsf{c}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z)}(x, y, \kappa)\right\}_{x\in X, y\in Y, z\in\{0,1\}^*, \kappa\in\mathbb{N}} .$$

We remark that in the case of two-party computation, for any functionality f there exists a protocol π that securely computes it with security-with-abort [54, 53]. This is true also for reactive functionalities, that is, functionalities that consist of several stages, where in each stage the parties can give inputs and get outputs from the current stage of f, and the function f stores some state between its stages.

4.2.3 Hybrid Model and Composition

The hybrid model combines both the real and ideal models. Specifically, an execution of a protocol π in the g-hybrid model, for some functionality g, involves the parties sending normal messages to each other (as in the real model) and, in addition, having access to a trusted party computing g. We assume that the invocations of g occur sequentially, that is, there are no concurrent execution of g. The composition theorems of [27] imply that if π securely computes some functionality f in the g-hybrid model, and a protocol ρ securely computes g, then the protocol π^ρ (where every ideal call of g is replaced with an execution of ρ) securely-computes f in the real world.

Function implication. In this chapter, we study whether or not a function g "implies" the coin-tossing functionality. We now formally define what we mean by "function implication". Our formulation uses the notion of g-hybrid model defined above, where here we assume that the trusted party that computing g in the hybrid model computes it with complete fairness. That is, in each invocation of the function g, both parties receive the outputs from g simultaneously. We are now ready for the definition of function implication.

Definition 4.2.3 *Let $f : X \times Y \to Z$ and $g : X' \times Y' \to Z'$ be functions. We say that* function g implies function f in the presence of malicious adversaries *if there exists a protocol that securely computes f with complete fairness in the g-hybrid model, in the presence of static malicious adversaries (where g is computed according to the ideal world with fairness). We say that g* information-theoretically *implies f if the above holds with statistical security.*

Note that if f can be securely computed with fairness (under some assumption), then every function g computationally implies f. Thus, this is only of interest for functions f that either cannot be securely computed with fairness, or for which this fact is not known.

4.2.4 Coin-Tossing Definitions

The coin-tossing functionality. We define the coin-tossing functionality simply by $f_{\text{CT}}(\lambda, \lambda) = (U_1, U_1)$, where λ denotes the empty input and U_1 denotes the uniform distribution over $\{0, 1\}$. That is, the functionality receives no input, chooses a uniformly chosen bit and gives the both parties the same bit. This yields the following definition:

Definition 4.2.4 (Coin-Tossing by Simulation) *A protocol π is a* secure coin-tossing protocol via simulation *if it securely computes f_{CT} with complete fairness in the presence of malicious adversaries.*

The above definition provides very strong simulation-based guarantees, which is excellent for our positive results. However, when proving impossibility, it is preferable to rule out even weaker, non-simulation based definitions. We now present a weaker definition where the guarantee is that the honest party outputs an unbiased coin, irrespective of the cheating party's behaviour. However, we stress that since our impossibility result only holds with respect to an all-powerful adversary (as discussed in the introduction), our definition is stronger than above since it requires security in the presence of any adversary, and not just polynomial-time adversaries.

Notations. Denote by $\langle P_1, P_2 \rangle$ a two party protocol where both parties act honestly. Let $\text{OUTPUT}\langle P_1, P_2 \rangle$ denote the output of parties in an honest execution (if the outputs of the parties is not consistent, this predicate returns $\perp$). For $\ell \in \{1, 2\}$, let $\text{OUTPUT}_\ell\langle P_1^*, P_2^* \rangle$ denote the output of party P_ℓ^* in an execution of P_1^* with P_2^*. In some cases, we also specify the random coins that the parties use in the execution; $\langle P_1(r_1), P_2(r_2) \rangle$ denotes an execution where P_1 acts honestly and uses random tape r_1 and P_2 acts honestly and uses random tape r_2. Let $r(n)$ be a polynomial that bounds the number of rounds of the protocol π, and let $c(n)$ be an upper bound on the length of the random tape of the parties. Let Uni denote the uniform distribution over $\{0, 1\}^{c(n)} \times \{0, 1\}^{c(n)}$. We are now ready to define a coin-tossing protocol:

Definition 4.2.5 (information-theoretic coin-tossing protocol) *A polynomial-time protocol $\pi = \langle P_1, P_2 \rangle$ is an* unbiased coin-tossing protocol, *if the following hold:*

1. Agreement: *There exists a negligible function $\mu(\cdot)$ such that for every κ it holds that:*

$$\Pr_{r_1, r_2 \leftarrow \mathsf{Uni}} \Big[\text{OUTPUT}_1\langle P_1(r_1), P_2(r_2) \rangle \neq \text{OUTPUT}_2\langle P_1(r_1), P_2(r_2) \rangle \Big] \leq \mu(\kappa) \ .$$

2. No bias: *For every adversary $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$ such that for every $b \in \{0,1\}$ and every $\kappa \in \mathbb{N}$:*

$$\Pr\left[\textsc{output}_1\langle P_1, \mathcal{A}\rangle = b\right] \leq \frac{1}{2} + \mu(\kappa) \quad \text{and} \quad \Pr\left[\textsc{output}_2\langle \mathcal{A}, P_2\rangle = b\right] \leq \frac{1}{2} + \mu(\kappa) \ .$$

Observe that both requirements together guarantee that two honest parties will output the same uniformly distributed bit, except with negligible probability. That is, for every $b \in \{0,1\}$:

$$\left| \Pr_{r_1, r_2 \leftarrow \mathsf{Uni}} \left[\textsc{output}\langle P_1(r_1), P_2(r_2)\rangle = b\right] - \frac{1}{2} \right| \leq \mu(\kappa) \tag{4.2.1}$$

4.3 The Criterion

In this section we define the criterion, and explore its properties. We start with the definition of δ-balanced functions.

4.3.1 δ-Balanced Functions

A vector $\mathbf{p} = (p_1, \ldots, p_k)$ is a probability vector if $\sum_{i=1}^{k} p_i = 1$, and for every $1 \leq i \leq k$ it holds that $p_i \geq 0$. Let $\mathbf{1}_k$ be the all one vector of size k. In addition, for a given function $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0,1\}$, let M_f denote the matrix defined by the truth table of f. That is, for every $1 \leq i \leq \ell$, $1 \leq j \leq m$, it holds that $M_f[i,j] = f(x_i, y_j)$.

Informally, a function is balanced if there exist probabilities over the inputs for each party that determine the probability that the output equals 1, irrespective of what input the other party uses. Assume that P_1 chooses its input according to the probability vector $(p_1, \ldots, p_\ell)$, meaning that it uses input x_i with probability p_i, for every $i = 1, \ldots, \ell$, and assume that party P_2 uses the jth input y_j. Then, the probability that the output equals 1 is obtained by multiplying $(p_1, \ldots, p_\ell)$ with the jth column of M_f. Thus, a function is balanced on the left, or with respect to P_1, if when multiplying $(p_1, \ldots, p_\ell)$ with M_f the result is a vector with values that are all equal. Formally:

Definition 4.3.1 *Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0,1\}$ be a function, and let $0 \leq \delta_1, \delta_2 \leq 1$ be constants. We say that f is δ_1-*left-balanced *if there exists a probability vector $\mathbf{p} = (p_1, \ldots, p_\ell)$ such that:*

$$(p_1, \ldots, p_\ell) \cdot M_f = \delta_1 \cdot \mathbf{1}_m = (\delta_1, \ldots, \delta_1) \ .$$

*Likewise, we say that the function f is δ_2-*right-balanced *if there exists a probability vector $\mathbf{q} = (q_1, \ldots, q_m)$ such that:*

$$M_f \cdot (q_1, \ldots, q_m)^T = \delta_2 \cdot \mathbf{1}_\ell^T \ .$$

*If f is δ_1-left-balanced and δ_2-right-balanced, we say that f is (δ_1, δ_2)-balanced. If $\delta_1 = \delta_2$, then we say that f is δ-*balanced*, where $\delta = \delta_1 = \delta_2$. We say that f is* strictly δ-balanced *if $\delta_1 = \delta_2$ and $0 < \delta < 1$.*

Note that a function may be δ_2-right-balanced for some $0 \leq \delta_2 \leq 1$ but not left balanced. For example, consider the function defined by the truth table $M_f \overset{\text{def}}{=} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$. This function is right balanced for $\delta_2 = \frac{1}{2}$ by taking $\mathbf{q} = (\frac{1}{2}, \frac{1}{2}, 0)$. However, it is not left-balanced for any δ_1 because for every probability vector $(p_1, p_2) = (p, 1-p)$ it holds that $(p_1, p_2) \cdot M_{\widetilde{f}} = (p, 1-p, 1)$, which is not balanced for any p. Likewise, a function may be δ_2-right-balanced, but not left balanced.

We now prove a simple but somewhat surprising proposition, stating that if a function is (δ_1, δ_2)-balanced, then δ_1 and δ_2 must actually equal each other. Thus, any (δ_1, δ_2)-balanced function is actually δ-balanced.

Proposition 4.3.2 *Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0, 1\}$ be a (δ_1, δ_2)-balanced function for some constants $0 \leq \delta_1, \delta_2 \leq 1$. Then, $\delta_1 = \delta_2$, and so f is δ-balanced.*

Proof: Under the assumption that f is (δ_1, δ_2)-balanced, we have that there exist probability vectors $\mathbf{p} = (p_1, \ldots, p_\ell)$ and $\mathbf{q} = (q_1, \ldots, q_m)$ such that $\mathbf{p} \cdot M_f = \delta_1 \cdot \mathbf{1}_m$ and $M_f \cdot \mathbf{q}^T = \delta_2 \cdot \mathbf{1}_\ell^T$. Observe that since $\mathbf{p}$ and $\mathbf{q}$ are probability vectors, it follows that for every constant c we have $\mathbf{p} \cdot (c \cdot \mathbf{1}_\ell^T) = c \cdot (\mathbf{p} \cdot \mathbf{1}_\ell^T) = c$; likewise $(c \cdot \mathbf{1}_m) \cdot \mathbf{q}^T = c$. Thus,

$$\mathbf{p} \cdot M_f \cdot \mathbf{q}^T = \mathbf{p} \cdot \Big(M_f \cdot \mathbf{q}^T\Big) = \mathbf{p} \cdot \Big(\delta_2 \cdot \mathbf{1}_\ell^T\Big) = \delta_2$$

and

$$\mathbf{p} \cdot M_f \cdot \mathbf{q}^T = \Big(\mathbf{p} \cdot M_f\Big) \cdot \mathbf{q}^T = \Big(\delta_1 \cdot \mathbf{1}_m\Big) \cdot \mathbf{q}^T = \delta_1,$$

implying that $\delta_1 = \delta_2$. ■

Note that a function can be both δ_2-right-balanced and δ_2'-right-balanced for some $\delta_2 \neq \delta_2'$. For example, consider the function M_f, which was defined above. This function is δ_2-right-balanced for every $1/2 \leq \delta_2 \leq 1$ (by multiplying with the probability vector $(1-\delta_2, 1-\delta_2, 2\delta_2 - 1)^T$ from the right). Nevertheless, in cases where a function is δ_2-right-balanced for multiple values, Proposition 4.3.2 implies that the function cannot be left-balanced for *any* δ_1. Likewise, if a function is δ_1-left balanced for more than one value of δ_1, it cannot be right-balanced.

4.3.2 The Criterion

The criterion for determining whether or not a function implies coin-tossing is simply the question of whether the function is *strictly* δ-balanced for some δ. Formally:

Property 4.3.3 *A function $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0, 1\}$ is* strictly balanced *if it is δ-balanced for some $0 < \delta < 1$.*

Observe that if M_f has a monochromatic row (i.e., there exists an input x such that for all y_i, y_j it holds that $f(x, y_i) = f(x, y_j)$), then there exists a probability vector $\mathbf{p}$ such that $\mathbf{p} \cdot M_f = 0 \cdot \mathbf{1}_m$ or $\mathbf{p} \cdot M_f = 1 \cdot \mathbf{1}_m$; likewise for a monochromatic column. Nevertheless, we stress that the existence of such a row and column does not imply f is strictly balanced since it is required that δ be strictly between 0 and 1, and not equal to either.

4.3.3 Exploring the δ-Balanced Property

In this section we prove some technical lemmas regarding the property that we will need later in the proof. First, we show that if a function f is not left-balanced for any $0 \leq \delta \leq 1$ (resp. not right balanced), then it is *not close* to being balanced. More precisely, it seems possible that a function f can be not δ-balanced, but is only negligibly far from being balanced (i.e., there may exist some probability vector $\mathbf{p} = \mathbf{p}(\kappa)$ (that depends on the security parameter κ) such that all the values in the vector $\mathbf{p} \cdot M_f$ are at most negligibly far from δ, for some $0 \leq \delta \leq 1$). In the following claim, we show that this situation is impossible. Specifically, we show that if a function is not δ balanced, then there exists some *constant* $c > 0$, such that for any probability vector $\mathbf{p}$, there is a distance of at least c between two values in the vector $\mathbf{p} \cdot M_f$. This holds also for probability vectors that are functions of the security parameter κ (as can be the case in our setting of secure protocols).

Lemma 4.3.4 *Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0, 1\}$ be a function that is not left balanced for any $0 \leq \delta_1 \leq 1$ (including $\delta_1 = 0, 1$). Then, there exists a constant $c > 0$, such that for any probability vector $\mathbf{p}$, it holds that:*

$$\max_i(\delta_1, \ldots, \delta_m) - \min_i(\delta_1, \ldots, \delta_m) \geq c$$

where $(\delta_1, \ldots, \delta_m) = \mathbf{p} \cdot M_f$, and M_f is the matrix representation of the function f.

Proof: Let P^ℓ be the set of all probability vectors of size ℓ. That is, $\mathrm{P}^\ell \subseteq [0,1]^\ell$ (which itself is a subset of $\mathbb{R}^\ell$), and each vector sums up to one. P^ℓ is a closed and bounded space. Therefore using the Heine-Borel theorem, P^ℓ is a compact space.

We start by defining a function $\phi : \mathrm{P}^\ell \to [0, 1]$ as follows:

$$\phi(\mathbf{p}) = \max_i(\mathbf{p} \cdot M_f) - \min_i \cdot(\mathbf{p} \cdot M_f)$$

Clearly, the function $\mathbf{p} \cdot M_f$ (where M_f is fixed and $\mathbf{p}$ is the variable) is a continuous function. Moreover, the maximum (resp. minimum) of a continuous function is itself a continuous function. Therefore, from composition of continuous functions we have that the function ϕ is continuous. Using the extreme value theorem (a continuous function from a compact space to a subset of the real numbers attains its maximum and minimum), there exists some probability vector $\mathbf{p}_{\min}$ for which for all $\mathbf{p} \in \mathrm{P}^\ell$, $\phi(\mathbf{p}_{\min}) \leq \phi(\mathbf{p})$. Since f is not δ-balanced, $\mathbf{p}_{\min} \cdot M_f \neq \delta \cdot \mathbf{1}_m$ for any $0 \leq \delta \leq 1$, and so $\phi(\mathbf{p}_{\min}) > 0$. Let $c \overset{\text{def}}{=} \phi(\mathbf{p}_{\min})$. This implies that for any probability vector $\mathbf{p}$, we have that $\phi(\mathbf{p}) \geq \phi(\mathbf{p}_{\min}) = c$. That is:

$$\max_i(\delta_1, \ldots, \delta_m) - \min_i(\delta_1, \ldots, \delta_m) \geq c \tag{4.3.1}$$

where $(\delta_1, \ldots, \delta_m) = \mathbf{p} \cdot M_f$. We have proven this for *all* probability vectors of size ℓ. Thus, it holds also for every probability vector $\mathbf{p}(\kappa)$ that is a function of κ, and for all κ's (this is true since for every κ, $\mathbf{p}(\kappa)$ defines a concrete probability vector for which Eq. (4.3.1) holds). ■

A similar claim can be stated and proven for the case where f is not right balanced.

0-balance and 1-balanced functions. In our proof that a function that is not strictly-balanced does not imply the coin-tossing functionality, we deal separately with functions that are 0-balanced or 1-balanced (since they are balanced, but not strictly balanced). We now prove a property that will be needed for this case. Specifically, we show that a function f is 1-left-balanced (resp. 0-left-balanced) if and only if the matrix M_f contains the all 1 row (resp., the all 0 row). A similar argument holds for the case where the function is 1-right-balanced (or 0-right-balanced) and the all 1 column (or all 0 column).

Lemma 4.3.5 *Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0, 1\}$ be a function. The function f is left-1-balanced if and only if the matrix M_f contains the all one row.*

Proof: We first show that if there exists a probability vector $\mathbf{p} = (p_1, \ldots, p_\ell)$ for which $\mathbf{p} \cdot M_f = \mathbf{1}_m$, then M_f contains the all-one row. Let $\mathbf{p}$ be a probability vector for which $\mathbf{p} \cdot M_f = \mathbf{1}_m$. Since $\mathbf{p}$ is a probability vector (meaning that its sum is 1), we have that: $\langle \mathbf{p}, \mathbf{1}_\ell \rangle = 1$. Denote by C_i the ith column of the matrix M_f. Since $\mathbf{p} \cdot M_f = \mathbf{1}_m$, it follows that for every i it holds that $\langle \mathbf{p}, C_i \rangle = 1$. Combining this with the fact that $\langle \mathbf{p}, \mathbf{1}_\ell \rangle = 1$, we have that $\langle \mathbf{p}, \mathbf{1}_\ell - C_i \rangle = 0$.

Let $\mathbf{p} = (p_1, \ldots, p_\ell)$, and $C_i = (a_1, \ldots, a_\ell)$. We have that: $\sum_{k=1}^{\ell} p_k \cdot (1 - a_k) = 0$. As all values are non-negative it must be that if $p_k \neq 0$ then $a_k = 1$ (otherwise the result cannot be 0). Since this is true for any column C_i, we conclude that for every k such that $p_k \neq 0$, the kth *row* is the all one vector. Since $\mathbf{p}$ is a probability vector, there exists at least one k such that $p_k \neq 0$, implying that there exists an all-one row.

For the other direction, let k be the index of the all one row in the matrix M_f. Then, clearly for the probability vector e_k (the kth elementary vector, namely, all the indices are zero except for the kth index which equals 1), the product $e_k \cdot M_f = (1, \ldots, 1) = \mathbf{1}_m$, and so the matrix is left-1-balanced. ■

Using Proposition 4.3.2, we conclude that if a function f is 1-left-balanced but not left-balanced for any other δ_1, then it is either 1-right-balanced as well, or not right-balanced for any $0 \leq \delta_2 \leq 1$. This observation will be used in the proof.

4.4 Strictly-Balanced Functions Imply Coin Tossing

In this section, we show that any function f that is strictly δ-balanced can be used to fairly toss a coin. Intuitively, this follows from the well known method of Von Neumann [97] for obtaining an unbiased coin toss from a biased one. Specifically, given a coin that is heads with probability ϵ and tails with probability $1 - \epsilon$, Von Neumann showed that you can toss a coin that is heads with probability exactly $1/2$ by tossing the coin twice in each phase, and stopping the first time that the pair is either heads-tails or tails-heads. Then, the parties output heads if the pair is heads-tails, and otherwise they output tails. This gives an unbiased coin because the probability of heads-tails equals the probability of tails-heads (namely, both probabilities equal $\epsilon \cdot (1 - \epsilon)$). Now, since the function f is strictly δ-balanced it holds that if party P_1 chooses its input via the probability vector $(p_1, \ldots, p_\ell)$ then the output will equal 1 with probability δ, irrespective of what input is used by P_2; likewise if P_2 chooses its input via $(q_1, \ldots, q_m)$ then the output will be 1 with probability δ irrespective of what P_1 does. This yields a coin that equals 1 with probability δ and thus Von Neumann's method can be applied to securely implement the coin-tossing functionality f_{CT} defined in Section 4.2.4. We stress that if one of the parties aborts early and refuses to participate, then the other party proceeds by itself (essentially, tossing a coin with probability δ until it concludes). We have the following theorem:

Theorem 4.4.1 *Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0,1\}$ be a strictly-balanced function for some constant $0 < \delta < 1$, as in Property 4.3.3. Then, f information-theoretically and computationally implies the coin-tossing functionality f_{CT} with malicious adversaries.*

Proof Sketch: The proof of this theorem follows from the aforementioned intuition, and is quite straightforward. We therefore just sketch the details here. First, we define a δ-biased coin tossing functionality that outputs a coin to both parties that equals 1 with probability δ and equals 0 with probability $1-\delta$. As we have described, this functionality can be securely computed using a single invocation of f, where the parties P_1 and P_2 choose their inputs via the probability vectors $\mathbf{p}$ and $\mathbf{q}$, respectively, that are guaranteed to exist by Property 4.3.3. The simulation of this protocol is trivial since neither party can do anything to change the probability of the output of the call to f being 1, and this probability is exactly the probability that the coin tossing functionality outputs 1. Note that if one of the parties does not participate at all (i.e., sends no input), then by the security definition of fairness the honest party still receives output and so this is the same as above. That is, if one of the parties does not participate, then the other essentially continues with the protocol by itself until either 01 or 10 is obtained.

Next, we obtain a fair unbiased coin toss by having the parties invoke the δ-biased coin tossing functionality twice and repeating until two different coins are obtained. When this happens they output the result of the first coin in the two coin tosses. If they do not terminate within $\frac{\kappa}{2\delta(1-\delta)}$ repetitions, then they output $\perp$ and halt. Since the probability that they terminate in any given attempt is $\delta(1-\delta) + (1-\delta)\delta = 2\delta(1-\delta)$ (because they halt after receiving either 10 or 01), it follows that they output $\perp$ with probability only

$$\Big(1 - 2\delta(1-\delta)\Big)^{\frac{\kappa}{2\delta(1-\delta)}} < e^{-\kappa}$$

which is negligible. The simulator works as follows. It receives the bit $b \in \{0,1\}$ from the trusted party computing the unbiased coin tossing functionality, and then runs the δ-biased coin tossing functionality like in the protocol playing the trusted party computing f and giving the corrupted party the expected outputs, until the first pair with different coins is received (or until $\frac{\kappa}{2\delta(1-\delta)}$ attempts were made). When this happens, the simulator hands the adversary the outputs of the δ-biased coin tossing that the corrupted party receives to be b and then $1-b$ (and thus the output is supposed to be b). Since the probability of receiving b and then $1-b$ is the same as the probability of receiving $1-b$ and then b, this view generated by the simulator is identical to that seen by the adversary in a real execution. The only difference between the real and ideal distributions is if too many attempts are made, since in the former case the honest party outputs $\perp$ whereas in the latter case it always outputs a bit. This completes the proof.

■

Application to fairness. Cleve [34] showed that there does not exist a protocol that securely computes the fair coin-tossing functionality in the plain model. Since any strictly-balanced function f implies the coin-tossing functionality, a protocol for f implies the existence of a protocol for coin-tossing. We therefore conclude:

Corollary 4.4.2 *Let $f : \{x_1, \dots, x_\ell\} \times \{y_1, \dots, y_m\} \to \{0, 1\}$ be a strictly-balanced function. Then, there does not exist a protocol that securely computes f with fairness.*

4.5 Unbalanced Functions Do Not Information-Theoretically Imply Coin Tossing

We now show that any function f that is *not* strictly-balanced (for all δ) does *not* information-theoretically imply the coin-tossing functionality. Stated differently, there does not exist a protocol for fairly tossing an unbiased coin in the f-hybrid model, with statistical security. Observe that in the f-hybrid model, it is possible for the parties to simultaneously exchange information, in some sense, since both parties receive output from calls to f at the same time. Thus, Cleve-type arguments [34] that are based on the fact that one party must know more than the other party at some point do not hold. We prove our result by showing that for every protocol there exists an unbounded malicious adversary that can bias the result. Our unbounded adversary needs to compute probabilities, which can actually be approximated given an $\mathcal{NP}$-oracle. Thus, it is possible to interpret our technical result also as a black-box separation, if desired.

As we have mentioned in the introduction, although we prove an "impossibility result" here, the implication is the opposite. Specifically, our proof that an unbalanced[2] f cannot be used to toss an unbiased coin implies that it may be possible to securely compute such functions with fairness.

Recall that a function is not strictly balanced if it is not δ-balanced for any $0 < \delta < 1$. We treat the case that the function is not δ-balanced at all separately from the case that it *is* δ-balanced but for $\delta = 0$ or $\delta = 1$. In the proof we show that in both of these cases, such a function cannot be used to construct a fair coin tossing protocol.

In the f-hybrid model, the parties may invoke the function f in two different "directions". Specifically, in some of the invocations P_1 selects the first input of f (i.e., x) and P_2 selects the second input of f (i.e., y), while in other invocation the roles may be reversed (with P_1 selecting the y input and P_2 selecting the x input). The question of which party plays which role is determined by the protocol. For convenience, we assume that P_1 always selects the first input, and P_2 always selects the second input, but the parties have access to two ideal functions f and f^T. Thus, calls to f in the protocol where P_2 selects the first input and P_1 selects the second input are modeled by a call to f^T where P_1 selects the first input (which is the second input of f) and P_2 selects the second input (which is the first input of f).

Theorem 4.5.1 *Let $f : \{x_1, \dots, x_\ell\} \times \{y_1, \dots, y_m\} \to \{0, 1\}$ be a function that is* not *left-balanced, for any $0 \leq \delta_1 \leq 1$. Then, f does not information-theoretically imply the coin-tossing functionality with security in the presence of malicious adversaries.*

Before proceeding to the proof, we provide high level intuition. We begin by observing that if f does not contain an embedded OR (i.e., inputs x_0, x_1, y_0, y_1 such that $f(x_0, y_0) =$

[2]Note, that the name unbalanced is a bit misleading as the complement of not being strictly balanced also includes being 1 or 0-balanced.

$f(x_1, y_0) = f(x_0, y_1) \neq f(x_1, y_1)$) and an embedded XOR (i.e., inputs x_0, x_1, y_0, y_1 such that $f(x_0, y_0) = f(x_1, y_1) \neq f(x_0, y_1) = f(x_1, y_0)$), then it is trivial and can be computed by simply having one party send the output to the other. This is because such a function depends only on the input of one party. Thus, by [34], it is impossible to fairly toss an unbiased coin in the f-hybrid model, since this is the same as fairly tossing an unbiased coin in the plain model. Thus, we consider only functions f that have an embedded OR or an embedded XOR.

In addition, we can consider coin-tossing protocols that consist of calls to f and f^T only, and no other messages. This is due to the fact that we can assume that any protocol consists of rounds, where each round is either an invocation of f (or f^T) or a message consisting of a single bit being sent from one party to the other. Since f has an embedded OR or an embedded XOR, messages of a single bit can be sent by invoking f. This is because in both cases there exist inputs x_0, x_1, y_0, y_1 such that $f(x_1, y_0) \neq f(x_1, y_1)$ and $f(x_0, y_1) \neq f(x_1, y_1)$. Thus, in order for P_2 to send P_1 a bit, the protocol can instruct the parties to invoke f where P_1 always inputs x_1, and P_2 inputs y_0 or y_1 depending on the bit that it wishes to send; likewise for P_1. Thus, any non-trivial function f enables "bit transmission" in the above sense. Observe that if the sender is malicious and uses an incorrect input, then this simply corresponds to sending an incorrect bit in the original protocol. In addition, since both parties receive the same output from f, in case where the receiver is malicious and uses incorrect input (i.e., in the above example P_1 does not use x_1), the sender can detect it and can be interpreted as an abort. Therefore, we can assume that a protocol consists of rounds, where in each round there is an invocation of f (where P_1 selects the x input and P_2 selects the y input) followed by an invocation of f^T (where here P_1 selects the y input and P_2 selects the x input).

Intuition. The fact that f is not balanced implies that in any single invocation of f, one party is able to have some effect on the output by choosing its input appropriately. That is, if the function is non-balanced on the left then the party on the right can use an input not according to the prescribed distribution in the protocol, and this will change the probability of the output of the invocation being 1 (for example). However, it may be possible that the ability to somewhat influence the output in individual invocations is not sufficient to bias the overall computation, due to the way that the function calls are composed. Thus, in the proof we need to show that an adversary is in fact capable of biasing the overall protocol. We demonstrate this by showing that there exist crucial invocations where the ability to bias the outcome in these invocation suffice for biasing the overall outcome. Then, we show that such invocations are always reached in any execution of the protocol, and that the adversary can (inefficiently) detect when such an invocation has been reached and can (inefficiently) compute which input it needs to use in that invocation in order to bias the output.

We prove the above by considering the execution tree of the protocol, which is comprised of calls to f and the flow of the computation based on the output of f in each invocation (i.e., the parties proceed left in the tree if the output of f in this invocation is 0; and right otherwise). Observe that a path from the root of the tree to a leaf-node represents a protocol execution. We show that in *every* path from the root to a leaf, there exists at least one node with the property that influencing the output of the single invocation of that node yields a bias in the final outcome. In addition, we describe the strategy of the adversary to detect such a node and choose its input for that node in order to obtain a bias.

In more detail, for every node v in the execution tree of the protocol, the adversary calculates (in an inefficient manner) the probability that the output of the computation equals 1, assuming that v is reached in the execution. Observe that the probability of obtaining 1 at the root node is at most negligibly far from 1/2 (since it is a secure coin-tossing protocol), and that the probability of obtaining 1 at a leaf node is either 1 or 0, depending on whether the output at the given leaf is 1 or 0 (the way that we define the tree is such that the output is fully determined by the leaf). Using a pigeon-hole like argument, we show that on every path from the root to a leaf there must be at least one node where the probability of outputting a 1 given that this node is reached is significantly different than the probability of outputting 1 given that the node's child on the path is reached. We further show that this difference implies that the two children of the given node yield significantly different probabilities of outputting 1 (since the probability of outputting 1 at a node v is the weighted-average of outputting 1 at the children, based on the probability of reaching each child according to the protocol). This implies that in every protocol execution, there exists an invocation of f where the probability of outputting 1 in the entire protocol is significantly different if the output of this invocation of f is 0 or 1. Since f is not balanced, it follows that for any distribution used by the honest party to choose its input for this invocation, there exist two inputs that the corrupted party can use that result in significantly different probabilities of obtaining 1. In particular, at least one of these probabilities is *significantly different from the probability of obtaining 1 in this call when both parties are honest and follow the protocol.*[3] Thus, the adversary can cause the output of the entire execution to equal 1 with probability significantly different to 1/2, which is the probability when both parties play honestly.

The above description does not deal with question of whether the output will be biased towards 0 or 1. In fact we design two adversaries, one that tries to bias the output towards 0 and the other towards 1. Then we show that at least one of these adversaries will be successful (see Footnote 3 for an explanation as to why only one of the adversaries may be successful). The two adversaries are similar and very simple. They search for the node on the path of the execution where the bias can be created and there make their move. In all nodes until and after that node they behave honestly (i.e., choose inputs for the invocations of f according to the input distribution specified by the protocol). We analyze the success of the adversaries and show that at least one of them biases the output with noticeable probability.

Proof of Theorem 4.5.1: Assume that π is a coin tossing protocol that implements the coin-tossing functionality in the f-hybrid model. That is, when both parties are honest, they output the same uniformly distributed bit, except with negligible probability. We construct an exponential-time malicious adversary who biases the outcome, in contradiction to the assumption that π is a fair coin-tossing protocol.

As described above we assume that π consists only of invocations of f and f^T. Let $r(\kappa)$ be the number of rounds in π and let $c(\kappa)$ be an upper bound on the number of random coins used by each party in the protocol. We assume that the function is not left-balanced and therefore construct the adversary $\mathcal{A}$ that controls P_2.

[3]Observe that one of these probabilities may be the same as the probability of obtaining 1 in an honest execution, in which case choosing that input will not result in any bias. Thus, the adversary may be able to bias the output of the entire protocol towards 1 or may be able to bias the output of the entire protocol towards 0, but not necessarily both.

Simplifying assumptions. We first prove the theorem under three simplifying assumptions regarding the protocol π; we show how to remove these assumptions later. The first assumption is that the protocol π only makes calls to f and not to f^T (i.e., in all calls to f, party P_1 sends x and party P_2 sends y and the output is $f(x, y)$). The second assumption is that honest parties always agree on the same output bit (i.e., the probability of them outputting different coins or aborting is zero). The third assumption is that the output of the parties is a deterministic function of the public transcript of the protocol alone.

The transcript tree and execution probabilities. We define a binary tree $\mathcal{T}$ of depth $r(\kappa)$ that represents the transcript of the execution. Each node in the tree indicates an invocation of f, and the left and right edges from a node indicate invocations of f where the output was 0 and 1, respectively.

We define the information that is saved in the internal nodes and leaves. Given a pair of random tapes (r_1, r_2) used by P_1 and P_2, respectively, it is possible to generate a full transcript of an honest execution of the protocol that contains the inputs that are sent to f in each round by both parties, the output of f in that round and the output of the parties at the end of the execution. Thus, we write on each (internal) node $v \in \mathcal{T}$ all the pairs of random tapes that reach this node, and the inputs that are sent to the trusted party for the associated call to f for those given tapes.

The leaves of the tree contain the output of the execution. Note that by the second and third simplifying assumption on protocol π described above, each leaf determines a single output (for both parties and for all random tapes reaching the leaf).

Given the distribution of the transcripts on the nodes of the tree we define two values for each node $v \in \mathcal{T}$. The first value which we denote by p_v is the probability that the parties output 1 at the end of the computation conditioned on the execution reaching the node v. This is simply computed as the ratio between the number of random tapes that reach the node v and result in an output 1 and all random tapes that reach v. The second value, denoted by γ^v, is the probability that the output of the invocation of f in this node is 1 (when both parties play honestly). This is calculated by computing the probabilities over the inputs of the parties in the invocation, as specified by the protocol and based on the random tapes in the node v. Denote the distributions over the inputs of P_1 and P_2 in the leaf by $(\alpha_1^v, \ldots, \alpha_m^v)$ and $(\beta_1^v, \ldots, \beta_\ell^v)$, respectively (i.e., in leaf v, P_1 chooses x_i with probability α_i^v, and P_2 chooses y_j with probability β_j^v). It holds that the probability that the parties receive 1 as output in the invocation of f at node v is $\gamma^v = (\alpha_1^v, \ldots, \alpha_m^v) \cdot M_f \cdot (\beta_1^v, \ldots, \beta_\ell^v)^T$.

The adversaries. Let $\epsilon = \epsilon(\kappa) = \frac{1/2 - \mu(\kappa)}{r(\kappa)}$, where $\mu(\cdot)$ is from Eq. (4.2.1); i.e., the probability that P_1 and P_2 output 1 (or 0) is $1/2 \pm \mu(\kappa)$ (this means that p_v where v is the root node of the tree $\mathcal{T}$ equals $1/2 \pm \mu(\kappa)$). Denote the two children of $v \in \mathcal{T}$ by $left(v)$ and $right(v)$ and they are reached by output 0 and 1 in the call to f respectively. We construct two adversaries controlling P_2: one which attempts to bias the outcome towards 1 and the other which attempts to bias the outcome towards 0. The adversaries search for a node $v \in \mathcal{T}$ for which the difference in the probability of outputting 1 at v and the probability of outputting 1 at one of its children is at least ϵ. The adversaries behave honestly until they reach such a node, at which point they have the opportunity to bias the outcome by deviating from the protocol in the invocation at this node. In the remaining nodes they again act honestly.

Adversary $\mathcal{A}^1$ controlling P_2 (biasing towards 1):
Let v be the current node in the tree $\mathcal{T}$ (starting at the root) and $\boldsymbol{\alpha}^v = (\alpha_1^v, \ldots, \alpha_n^v)$ be the input distribution for P_1 (as explained above, this can be computed). Compute $(\delta_1^v, \ldots, \delta_\ell^v) = \boldsymbol{\alpha}^v \cdot M_f$. Let i and k be indices such that $\delta_i^v = \max_{1 \le j \le \ell}\{\delta_j^v\}$ and $\delta_k^v = \min_{1 \le j \le \ell}\{\delta_j^v\}$. In the *first* node v in the execution for which $|p_{right(v)} - p_v| \ge \epsilon$ or $|p_{left(v)} - p_v| \ge \epsilon$, act according to the following:

1. If $p_{right(v)} \ge p_v + \epsilon$ or $p_{left(v)} \le p_v - \epsilon$ then send input y_i in this invocation of f (this increases the probability of reaching node $right(v)$ because the probability of obtaining 1 in this invocation is δ_i^v which is *maximal*).
2. If $p_{left(v)} \ge p_v + \epsilon$ or $p_{right(v)} \le p_v - \epsilon$ then send input y_k in this invocation of f (this increases the probability of reaching node $left(v)$ because the probability of obtaining 1 in this invocation is δ_k^v which is *minimal*).

In all other nodes act honestly.

The adversary $\mathcal{A}^0$ controlling P_2 (biasing towards 0): The adversary is the same as $\mathcal{A}^1$ with the following differences:

- Step 1: If $p_{right(v)} \ge p_v + \epsilon$ or $p_{left(v)} \le p_v - \epsilon$: then send input y_k in this invocation of f (where δ_k^v is minimal).
- Step 2: If $p_{left(v)} \ge p_v + \epsilon$ or $p_{right(v)} \le p_v - \epsilon$: then send input y_i in this invocation of f (where δ_i^v is maximal).

First, we show that in any honest execution there exists a node v for which $|p_v - p_{right(v)}| \ge \epsilon$ or $|p_v - p_{left(v)}| \ge \epsilon$. Then, we show that once the execution reaches such a node v, one of the adversaries succeeds in biasing the outcome.

The set $\mathcal{V}$. Let $\mathcal{V}$ be the set of all nodes v in $\mathcal{T}$ that have two children, for which $|p_v - p_{right(v)}| \ge \epsilon$ or $|p_v - p_{left(v)}| \ge \epsilon$, and v is the first node in the path between the root and v that satisfies the condition. In order to see that in any execution, the adversaries reachs such a node, we show that for every pair of random coins $(r_1, r_2) \in \mathsf{Uni}$ there exists a node $v \in \mathcal{V}$ such that an honest execution with (r_1, r_2) reaches v. Fix (r_1, r_2). Note that any pair of random coins define a unique path $u_1, \ldots, u_{r(\kappa)}$ from the root u_1 to a leaf $u_{r(\kappa)}$.

We start with the case where $u_{r(\kappa)}$ defines output 1. From Eq. (4.2.1), we have that $p_{u_1} \le 1/2 + \mu(\kappa)$. In addition, $p_{u_{r(\kappa)}} = 1$ since the output is 1. Therefore, we have that:

$$\frac{\sum_{i=1}^{r(\kappa)-1}\left(p_{u_{i+1}} - p_{u_i}\right)}{r(\kappa)} = \frac{p_{u_{r(\kappa)}} - p_{u_1}}{r(\kappa)} \ge \frac{1 - \frac{1}{2} - \mu(\kappa)}{r(\kappa)} = \frac{\frac{1}{2} - \mu(\kappa)}{r(\kappa)} = \epsilon.$$

Thus, the average of differences is greater than or equal to ϵ, which means that there exists an i such that $p_{u_{i+1}} - p_{u_i} \ge \epsilon$. Moreover, this u_i must have two children since otherwise $p_{u_i} = p_{u_{i+1}}$. Since $u_{i+1} \in \{right(u_i), left(u_i)\}$, we conclude that there exists a node u_i such that either $p_{right(u_i)} \ge p_{u_i} + \epsilon$ or $p_{left(u_i)} \ge p_{u_i} + \epsilon$.

The second case is where $u_{r(\kappa)}$ defines output 0, and $p_{u_1} \ge 1/2 - \mu(\kappa)$. Similarly to the above, we have that:

$$\frac{\sum_{i=1}^{r(\kappa)-1}\left(p_{u_{i+1}}-p_{u_i}\right)}{r(\kappa)} = \frac{p_{u_{r(\kappa)}}-p_{u_1}}{r(\kappa)} \leq \frac{0-\frac{1}{2}+\mu(\kappa)}{r(\kappa)} = -\epsilon,$$

which implies that there exists an i such that $p_{u_{i+1}} - p_{u_i} \leq -\epsilon$. Moreover, this u_i must have two children, otherwise we must have that $p_{u_i} = p_{u_{i+1}}$. Since $u_{i+1} \in \{right(u_i), left(u_i)\}$, we conclude that there exists a node u_i such that either $p_{right(u_i)} \leq p_{u_i} - \epsilon$ or $p_{left(u_i)} \leq p_{u_i} - \epsilon$.

We conclude that in every execution of the protocol, the parties reach a node $v \in \mathcal{V}$.

We know that in every node $v \in \mathcal{V}$, the difference between $p_{left(v)}$ or $p_{right(v)}$ and p_v is large. We now show that this implies that the difference between $p_{right(v)}$ and $p_{left(v)}$ themselves is large. This is the basis for the ability of the adversary to bias the output. Let $v \in \mathcal{V}$ be a fixed node, let $\boldsymbol{\alpha}^v$ be the distribution over the input of the honest P_1 to f at node v, and let $\boldsymbol{\beta}^v$ be the distribution over the input of the honest P_2 to f at node v (as described above, these distributions over the inputs are fully defined and inefficiently computable). Let $\gamma^v = \boldsymbol{\alpha}^v \cdot M_f \cdot \boldsymbol{\beta}^v$ be the probability that the honest parties obtain output 1 in this node (when sending inputs according to the distributions). By the definition of p_v and γ^v, we have that:

$$p_v = \gamma^v \cdot p_{right(v)} + (1-\gamma^v) \cdot p_{left(v)}.$$

This is because with probability γ^v the honest parties proceed to $right(v)$ and with probability $1-\gamma^v$ they proceed to $left(v)$. Thus, the probability of outputting 1 at v is as above. We now prove:

Claim 4.5.2 *For every node $v \in \mathcal{V}$:*

1. $0 < \gamma^v < 1$.
2. *Only one of the following holds: $|p_v - p_{right(v)}| \geq \epsilon$ or $|p_v - p_{left(v)}| \geq \epsilon$ but not both.*
3. *If $p_{right(v)} \geq p_v + \epsilon$, then $p_{right(v)} \geq p_{left(v)} + \epsilon$.*
4. *If $p_{left(v)} \geq p_v + \epsilon$, then $p_{left(v)} \geq p_{right(v)} + \epsilon$.*
5. *If $p_{right(v)} \leq p_v - \epsilon$, then $p_{right(v)} \leq p_{left(v)} - \epsilon$.*
6. *If $p_{left(v)} \leq p_v - \epsilon$, then $p_{left(v)} \leq p_{right(v)} - \epsilon$.*

Proof: For Item 1, recall that the probability γ^v can be computed as the number of random tapes that reach the right child of v over the number of random tapes that are consistent with v. Therefore, if $\gamma^v = 0$, it means that no execution reaches the right child of v. Similarly, in case $\gamma^v = 1$, we have that there is no left child of v. In any case, both cases are in contradiction the the fact that $v \in \mathcal{V}$, which means above the other properties, that the node v has two children.

For Item 2, assume that both $|p_v - p_{right(v)}| \geq \epsilon$ and $|p_v - p_{left(v)}| \geq \epsilon$ hold simultaneously. This means that:

$$\begin{aligned} -\left(p_v - p_{right(v)}\right) &\leq \epsilon \leq p_v - p_{right(v)} \\ -\left(p_v - p_{left(v)}\right) &\leq \epsilon \leq p_v - p_{left(v)} \end{aligned}$$

When we multiply the first equation by γ^v and the second by $1-\gamma^v$ (both are not zero, according to the first item), we obtain that:

$$\begin{array}{rcl} -\gamma^v \cdot \left(p_v - p_{right(v)}\right) \leq & \gamma^v \cdot \epsilon & \leq \gamma^v \cdot \left(p_v - p_{right(v)}\right) \\ -(1-\gamma^v) \cdot \left(p_v - p_{left(v)}\right) \leq & (1-\gamma^v) \cdot \epsilon & \leq (1-\gamma^v) \cdot \left(p_v - p_{left(v)}\right) \end{array}$$

Summing the above two equations together, and recalling that $p_v = \gamma^v \cdot p_{right(v)} + (1-\gamma^v) \cdot p_{left(v)}$, we have that:

$$0 = -p_v + p_v \leq \epsilon \leq p_v - p_v = 0$$

and so $\epsilon = 0$, in contradiction to the fact that: $\epsilon = \frac{1/2 - \mu(\kappa)}{r(\kappa)} > 0$.

For Item 3, assume that $p_{right(v)} \geq p_v + \epsilon$. We have:

$$\begin{array}{rcl} p_{right(v)} & \geq & p_v + \epsilon \\ p_{right(v)} & \geq & \gamma^v \cdot p_{right(v)} + (1-\gamma^v) \cdot p_{left(v)} + \epsilon \\ (1-\gamma^v) \cdot p_{right(v)} & \geq & (1-\gamma^v) \cdot p_{left(v)} + \epsilon \\ p_{right(v)} & \geq & p_{left(v)} + \dfrac{\epsilon}{1-\gamma^v} \geq p_{left(v)} + \epsilon \end{array}$$

where the last inequality holds since $0 < 1 - \gamma^v < 1$.

The other items are derived through similar computations. ■

Analyzing the adversary's success. We are now ready to show that at least one of the adversaries $\mathcal{A}^1$ or $\mathcal{A}^0$ succeeds in biasing the output.

We start with some notation. For a given node v and a bit $b \in \{0,1\}$, denote by $\mathsf{out}^b_{|v}(P_1, P_2^*)$ the probability that an honest party P_1 outputs the bit b in an execution with P_2^*, conditioned on the event that the execution reached node v. Note that in an execution between two honest parties it holds that :

$$\begin{array}{rcl} \mathsf{out}^1_{|v}(P_1, P_2) & = & \Pr_{(r_1,r_2) \leftarrow \mathsf{Uni}}\left[\textsc{output}\langle P_1(r_1), P_2(r_2)\rangle = 1 \;\middle|\; v \text{ is reached}\right] \\ & = & p_v = \gamma^v \cdot p_{right(v)} + (1-\gamma^v) \cdot p_{left(v)}. \end{array}$$

Similarly, $\mathsf{out}^0_{|v}(P_1, P_2) = 1 - p_v$. Note also that when v is the root node, $\mathsf{out}^1_{|v}(P_1, P_2) = \frac{1}{2} \pm \mu(\kappa)$.

Let $v \in \mathcal{V}$. We consider two executions: in both executions P_1 is honest, while in the first execution P_2 is controlled by $\mathcal{A}^1$, and in the second execution P_2 is controlled by $\mathcal{A}^0$. We compute the probabilities $\mathsf{out}^1_{|v}(P_1, \mathcal{A}^1)$ and $\mathsf{out}^0_{|v}(P_1, \mathcal{A}^0)$ and compare these to the output when both parties are honest. Specifically, we show that for every $v \in \mathcal{V}$ there is a difference between the output probability at v when honest parties run, and when an honest P_1 runs with a dishonest P_2 (we actually consider the sum of the differences, and will later use this to show that at least one of $\mathcal{A}^0$ and $\mathcal{A}^1$ must succeed in biasing).

Claim 4.5.3 *There exists a constant $c > 0$ such that for every node $v \in \mathcal{V}$ it holds that:*

$$\left(\mathsf{out}^0_{|v}(P_1, \mathcal{A}^0) - \mathsf{out}^0_{|v}(P_1, P_2)\right) + \left(\mathsf{out}^1_{|v}(P_1, \mathcal{A}^1) - \mathsf{out}^1_{|v}(P_1, P_2)\right) \geq c \cdot \epsilon$$

Proof: Let $v \in \mathcal{V}$ and denote $\delta^v_{\max} = \max_{1 \leq i \leq \ell}\{\delta^v_i\}$ and $\delta^v_{\min} = \min_{1 \leq j \leq \ell}\{\delta^v_j\}$, where $\delta^v_1, \ldots, \delta^v_\ell$ are as computed by the adversaries. Since $v \in \mathcal{V}$, either $p_{right(v)} \geq p_v + \epsilon$, $p_{left(v)} \geq p_v + \epsilon$,

$p_{right}(v) \leq p_v - \epsilon$ or $p_{left(v)} \leq p_v - \epsilon$. We consider two cases:

Case 1: $p_{right(v)} \geq p_v + \epsilon$ or $p_{left(v)} \leq p_v - \epsilon$. In these cases, the adversary $\mathcal{A}^1$ sends y_i for which $\delta_i = \delta^v_{\max}$, while the adversary $\mathcal{A}^0$ sends y_k for which $\delta_k = \delta^v_{\min}$.

We now compute the difference between the probabilities that P_1 outputs 1 when it interacts with and honest P_2, and with the adversary $\mathcal{A}^1$, where in both executions we assume that v is reached.[4]

$$\begin{aligned}
&\mathsf{out}^1_{|v}(P_1, \mathcal{A}^1) - \mathsf{out}^1_{|v}(P_1, P_2)\\
&= \Pr_{(r_1,r_2)\leftarrow \mathsf{Uni}}\left[\textsc{output}\langle P_1(r_1), \mathcal{A}^1(r_2)\rangle = 1 \;\middle|\; v\right] - \Pr_{(r_1,r_2)\leftarrow \mathsf{Uni}}\left[\textsc{output}\langle P_1(r_1), P_2(r_2)\rangle = 1 \;\middle|\; v\right]\\
&= \delta^v_{\max} \cdot p_{right(v)} + (1-\delta^v_{\max}) \cdot p_{left(v)} - \left(\gamma^v \cdot p_{right(v)} + (1-\gamma^v)\cdot p_{left(v)}\right)\\
&= (\delta^v_{\max} - \gamma^v) \cdot p_{right(v)} + (\gamma^v - \delta^v_{\max}) \cdot p_{left(v)}\\
&= (\delta^v_{\max} - \gamma^v) \cdot \left(p_{right(v)} - p_{left(v)}\right)\\
&\geq (\delta^v_{\max} - \gamma^v) \cdot \epsilon\ .
\end{aligned}$$

where the last inequality follows from Claim 4.5.2 which states that $p_{right(v)} - p_{left(v)} \geq \epsilon$ in this case. Observe that when $\delta^v_{\max} = \gamma^v$, then there is no bias (i.e., the probability of obtaining output 1 may be the same as when P_2 is honest).

In a similar way, we compute the difference between the probabilities that P_1 outputs 0 when it interacts with an honest P_2, and with the adversary $\mathcal{A}^0$, conditioned on the event that the node $v \in \mathcal{V}$ is reached. We have

$$\begin{aligned}
&\mathsf{out}^0_{|v}(P_1, \mathcal{A}^0) - \mathsf{out}^0_{|v}(P_1, P_2)\\
&= \Pr_{(r_1,r_2)\leftarrow \mathsf{Uni}}\left[\textsc{output}\langle P_1(r_1), \mathcal{A}^0(r_2)\rangle = 0 \;\middle|\; v\right] - \Pr_{(r_1,r_2)\leftarrow \mathsf{Uni}}\left[\textsc{output}\langle P_1(r_1), P_2(r_2)\rangle = 0 \;\middle|\; v\right]\\
&= \left(1 - \Pr_{(r_1,r_2)\leftarrow \mathsf{Uni}}\left[\textsc{output}\langle P_1(r_1), \mathcal{A}^0(r_2)\rangle = 1 \;\middle|\; v\right]\right)\\
&\quad - \left(1 - \Pr_{(r_1,r_2)\leftarrow \mathsf{Uni}}\left[\textsc{output}\langle P_1(r_1), P_2(r_2)\rangle = 1 \;\middle|\; v\right]\right)\\
&= \Pr_{(r_1,r_2)\leftarrow \mathsf{Uni}}\left[\textsc{output}\langle P_1(r_1), P_2(r_2)\rangle = 1 \;\middle|\; v\right] - \Pr_{(r_1,r_2)\leftarrow \mathsf{Uni}}\left[\textsc{output}\langle P_1(r_1), \mathcal{A}^0(r_2)\rangle = 1 \;\middle|\; v\right]\\
&= \left(\gamma^v \cdot p_{right(v)} + (1-\gamma^v)\cdot p_{left(v)}\right) - \left(\delta^v_{\min} \cdot p_{right(v)} + (1-\delta^v_{\min}) \cdot p_{left(v)}\right)\\
&= (\gamma^v - \delta^v_{\min}) \cdot \left(p_{right(v)} - p_{left(v)}\right)\\
&\geq (\gamma^v - \delta^v_{\min}) \cdot \epsilon\ ,
\end{aligned}$$

where the last equality it true from Claim 4.5.2. Once again, when $\delta^v_{\min} = \gamma^v$, then there is no bias.

The crucial observation is that since f is *not* δ balanced, it holds that $\delta^v_{\min} \neq \delta^v_{\max}$ and thus

[4]Note that when we write $\textsc{output}\langle P_1(r_1), \mathcal{A}^1(r_2)\rangle$ we mean the output of P_1 when interacting with $\mathcal{A}^1$ with respective random tapes r_1 and r_2. Since $\mathcal{A}^1$ behaves according to the honest strategy except in the node $v \in \mathcal{V}$, we have that $\mathcal{A}^1$ uses the coins r_2 just like an honest P_2 except in node v. Moreover, when conditioning on the event that the node v is reached, we just write "v" for short, instead of "v is reached".

either $\delta^v_{\max} \neq \gamma^v$ or $\delta^v_{\min} \neq \gamma^v$ or both. We can compute the overall bias of the adversaries as:

$$\begin{aligned}\Big(\mathsf{out}^0_{|v}(P_1,\mathcal{A}^0) - \mathsf{out}^0_{|v}(P_1,P_2)\Big) + \Big(\mathsf{out}^1_{|v}(P_1,\mathcal{A}^1) - \mathsf{out}^1_{|v}(P_1,P_2)\Big)&\\ \geq\ (\delta^v_{\max} - \gamma^v)\cdot\epsilon + (\gamma^v - \delta^v_{\min})\cdot\epsilon = (\delta^v_{\max} - \delta^v_{\min})\cdot\epsilon\ .&\end{aligned}$$

Now, by the fact that f is *not* left-balanced, Claim 4.3.4 states that there exists a constant $c > 0$ such that for all probability vectors $\mathbf{p}$: $\max_i(\mathbf{p}\cdot M_f) - \min_i(\mathbf{p}\cdot M_f) \geq c$. In particular, this means that $\delta^v_{\max} - \delta^v_{\min} \geq c$. Therefore, we can conclude that:

$$\Big(\mathsf{out}^0_{|v}(P_1,\mathcal{A}^0) - \mathsf{out}^0_{|v}(P_1,P_2)\Big) + \Big(\mathsf{out}^1_{|v}(P_1,\mathcal{A}^1) - \mathsf{out}^1_{|v}(P_1,P_2)\Big) \geq (\delta^v_{\max} - \delta^v_{\min})\cdot\epsilon \geq c\cdot\epsilon\ .$$

Case 2: $\boldsymbol{p_{left(v)} \geq p_v + \epsilon}$ or $\boldsymbol{p_{right(v)} \leq p_v - \epsilon}$. In this case, $\mathcal{A}^1$ sends y_k for which $\delta_k = \delta^v_{\min}$, and $\mathcal{A}^0$ sends y_i for which $\delta_i = \delta^v_{\max}$. Using a similar calculation to the previous case, and using Claim 4.5.2 which states that $p_{left(v)} - p_{right(v)} \geq \epsilon$ in this case, we have:

$$\begin{aligned}&\mathsf{out}^1_{|v}(P_1,\mathcal{A}^1) - \mathsf{out}^1_{|v}(P_1,P_2)\\ &\quad= \Pr_{(r_1,r_2)\leftarrow\mathsf{Uni}}\Big[\text{OUTPUT}\langle P_1(r_1),\mathcal{A}^1(r_2)\rangle = 1\ \Big|\ v\Big] - \Pr_{(r_1,r_2)\leftarrow\mathsf{Uni}}\Big[\text{OUTPUT}\langle P_1(r_1),P_2(r_2)\rangle = 1\ \Big|\ v\Big]\\ &\quad= \delta^v_{\min}\cdot p_{right(v)} + (1-\delta^v_{\min})\cdot p_{left(v)} - \gamma^v\cdot p_{right(v)} - (1-\gamma^v)\cdot p_{left(v)}\\ &\quad= (\gamma^v - \delta^v_{\min})\cdot\big(p_{left(v)} - p_{right(v)}\big)\\ &\quad\geq (\gamma^v - \delta^v_{\min})\cdot\epsilon\ ,\end{aligned}$$

and

$$\begin{aligned}&\mathsf{out}^0_{|v}(P_1,\mathcal{A}^0) - \mathsf{out}^0_{|v}(P_1,P_2)\\ &\quad= \Pr_{(r_1,r_2)\leftarrow\mathsf{Uni}}\Big[\text{OUTPUT}\langle P_1(r_1),\mathcal{A}^0\rangle = 1\ \Big|\ v\Big] - \Pr_{(r_1,r_2)\leftarrow\mathsf{Uni}}\Big[\text{OUTPUT}\langle P_1(r_1),P_2(r_2)\rangle = 1\ \Big|\ v\Big]\\ &\quad= \gamma^v\cdot p_{right(v)} + (1-\gamma^v)\cdot p_{left(v)} - \delta^v_{\max}\cdot p_{right(v)} - (1-\delta^v_{\max})\cdot p_{left(v)}\\ &\quad= (\delta^v_{\max} - \gamma^v)\cdot\big(p_{left(v)} - p_{right(v)}\big)\\ &\quad\geq (\delta^v_{\max} - \gamma^v)\cdot\epsilon\ .\end{aligned}$$

Therefore, the overall bias is:

$$\begin{aligned}\Big(\mathsf{out}^0_{|v}(P_1,\mathcal{A}^0) - \mathsf{out}^0_{|v}(P_1,P_2)\Big) + \Big(\mathsf{out}^1_{|v}(P_1,\mathcal{A}^1) - \mathsf{out}^1_{|v}(P_1,P_2)\Big)&\\ \geq\ (\delta^v_{\max} - \gamma^v)\cdot\epsilon + (\gamma^v - \delta^v_{\min})\cdot\epsilon = (\delta^v_{\max} - \delta^v_{\min})\cdot\epsilon&\end{aligned}$$

Again, by the fact that f is *not* left-balanced, Claim 4.3.4 implies that for the same constant $c > 0$ as above it holds:

$$\Big(\mathsf{out}^0_{|v}(P_1,\mathcal{A}^0) - \mathsf{out}^0_{|v}(P_1,P_2)\Big) + \Big(\mathsf{out}^1_{|v}(P_1,\mathcal{A}^1) - \mathsf{out}^1_{|v}(P_1,P_2)\Big) \geq (\delta^v_{\max} - \delta^v_{\min})\cdot\epsilon \geq c\cdot\epsilon.$$

This completes the proof of Claim 4.5.3. ■

Computing the overall bias. Since in any execution exactly one node $v \in \mathcal{V}$ is reached, and since in any execution a node in $\mathcal{V}$ is reached, we have that for any party P_2^* (in particular, an honest P_2, the adversary $\mathcal{A}^0$ or the adversary $\mathcal{A}^1$), and for every bit $b \in \{0,1\}$:

$$\begin{aligned}
&\Pr_{(r_1,r_2)\leftarrow\mathsf{Uni}}\left[\textsc{output}\langle P_1(r_1), P_2^*(r_2)\rangle = b\right] \\
&\qquad = \sum_{v\in\mathcal{V}} \Pr\left[v \text{ is reached in } \langle P_1(r_1), P_2^*(r_2)\rangle\right] \cdot \mathsf{out}^b_{|v}(P_1(r_1), P_2^*(r_2))
\end{aligned}$$

We denote the probability above by $\mathsf{out}^b(P_1, P_2^*)$.

Note that since both adversaries act honestly until they reach a node $v \in \mathcal{V}$, the executions $\langle P_1, P_2^*\rangle$ (where $P_2^* \in \{P_2, \mathcal{A}^1, \mathcal{A}^0\}$) are the same up to the point when they reach the node v, and thus:

$$\begin{aligned}
\Pr_{(r_1,r_2)\leftarrow\mathsf{Uni}}\left[v \text{ is reached in } \langle P_1(r_1), P_2(r_2)\rangle\right] &= \Pr_{(r_1,r_2)\leftarrow\mathsf{Uni}}\left[v \text{ is reached in } \langle P_1(r_1), \mathcal{A}^1(r_2)\rangle\right] \\
&= \Pr_{(r_1,r_2)\leftarrow\mathsf{Uni}}\left[v \text{ is reached in } \langle P_1(r_1), \mathcal{A}^0(r_2)\rangle\right]
\end{aligned}$$

we therefore denote this equal value as reach_v. Moreover, since any execution reaches exactly one node $v \in \mathcal{V}$, we have: $\sum_{v\in\mathcal{V}} \mathsf{reach}_v = 1$.

Combining the above, we conclude that:

$$\begin{aligned}
&\left(\mathsf{out}^1(P_1,\mathcal{A}^1) - \mathsf{out}^1(P_1,P_2)\right) + \left(\mathsf{out}^0(P_1,\mathcal{A}^1) - \mathsf{out}^0(P_1,P_2)\right) \\
&\quad = \sum_{v\in\mathcal{V}} \mathsf{reach}_v \cdot \left[\left(\mathsf{out}^1_{|v}(P_1,\mathcal{A}^1) - \mathsf{out}^1_{|v}(P_1,P_2)\right) + \left(\mathsf{out}^0_{|v}(P_1,\mathcal{A}^0) - \mathsf{out}^0_{|v}(P_1,P_2)\right)\right] \\
&\quad \geq \sum_{v\in\mathcal{V}} \mathsf{reach}_v \cdot (c\cdot\epsilon) = c\cdot\epsilon
\end{aligned}$$

where the last inequality is true by Claim 4.5.3. Since π is a coin-tossing protocol, by Eq. (4.2.1) we have that:

$$\left|\mathsf{out}^0(P_1,P_2) - \frac{1}{2}\right| \leq \mu(\kappa) \quad \text{and} \quad \left|\mathsf{out}^1(P_1,P_2) - \frac{1}{2}\right| \leq \mu(\kappa)\ .$$

Thus, we have that:

$$\left(\mathsf{out}^1(P_1,\mathcal{A}^1) - \frac{1}{2}\right) + \left(\mathsf{out}^0(P_1,\mathcal{A}^0) - \frac{1}{2}\right) \geq c\cdot\epsilon - 2\mu(\kappa).$$

Since c is a constant, and $\epsilon = \epsilon(\kappa) = \frac{1/2-\mu(\kappa)}{r(\kappa)}$, we have that the sum of the biases is non-negligible, and therefore at least one of $\mathcal{A}^0$ and $\mathcal{A}^1$ succeeds in biasing the output of the coin-tossing protocol when running with an honest P_1.

The general case – removing the simplifying assumptions. In our proof above, we relied on three simplifying assumptions regarding the coin-tossing protocol:

1. The hybrid model allows invocations of the function f only, and not invocations of f^T.
2. Honest parties agree on the same output with probability 1.
3. The leaves of the tree fully define the output. Namely, the output of the parties are a deterministic function of the transcript of the protocol.

Item 1 is solved as follows. Instead of considering protocols where the only invocations are to the function f, we consider protocols where each round consists of an invocation of f, followed by an invocation of f^T. In the execution tree, each node at an even level indicates an invocation of f, whereas nodes at an odd level indicate executions of f^T. The variables $\gamma^v, left(v), right(v)$ are defined similarly as before.

As the analysis shows, at any execution there is a "jump" and there exists a node v for which $|p_{right(v)} - p_v|$ or $|p_{left(v)} - p_v|$ is greater than ϵ. This holds also for our case as well, where the tree consists nodes of f^T and f. It is easy to see that in the case that all the nodes v are in fact invocations of f, the same adversaries as before and the same analysis hold, and the adversaries can bias the result. However, in general, the "jump" may occur on executions of f^T. However, in such a case, the adversaries that we have presented control the "x-inputs" and therefore cannot cheat (since it may be that the function is not left-balanced but *is* right-balanced). Indeed, if all the "jumps" in the protocol are in nodes which are executions of f^T, the adversaries that we have proposed may not be able to bias the result at all.

We solve this problem by "splitting" the adversary $\mathcal{A}^1$ into two adversaries: $\mathcal{A}_1^1$ which controls P_1 and adversary $\mathcal{A}_2^1$ which controls P_2 (similarly, we "split" A^0 into $\mathcal{A}_1^0, \mathcal{A}_2^0$ who control P_1 and P_2, respectively.). The adversary $\mathcal{A}_2^1$ "attacks" the protocol exactly as $\mathcal{A}^1$, with the only restriction that it checks that the node v (for which there is a "jump") is an invocation of f. In case where v is an invocation of f^T, the adversary $\mathcal{A}_2^1$ controls the inputs of x and therefore cannot attack at this point. However, the adversary $\mathcal{A}_1^1$ controls the inputs of y at that node, and can perform the same attack as $\mathcal{A}^1$. Overall, the adversary $\mathcal{A}_2^1$ only attacks the protocol on nodes v which are invocations of f, whereas $\mathcal{A}_1^1$ attacks the protocol on nodes v which are invocations of f^T. The overall bias of the two adversaries $(\mathcal{A}_1^1,\mathcal{A}_2^1)$ in a protocol where the invocations are of f, f^T is exactly the same as the overall bias of the adversary $\mathcal{A}^1$ in a protocol where all the invocations are of the function f. Similarly, we split $\mathcal{A}^0$ into $(\mathcal{A}_1^0, \mathcal{A}_2^0)$, and the overall bias of $\mathcal{A}^0$ is essentially the same as the overall bias of $(\mathcal{A}_1^0, \mathcal{A}_2^0)$. Since the sum of the biases of $(\mathcal{A}^0, \mathcal{A}^1)$ is non-negligible, the sum of the biases of $(\mathcal{A}_1^0, \mathcal{A}_1^1, \mathcal{A}_2^0, \mathcal{A}_2^1)$ is non-negligible as well.

Item 2 can be dealt with in a straightforward manner. By the requirement on a secure coin-tossing function, the probability that two honest parties output different bits is at most negligible. In executions that reach leaves for which this event occurs, we can just assume that the adversary fails. Since such executions happen with negligible probability, this reduces the success probability of the adversary by at most a negligible amount.

We now turn to the Item 3. First we explain in what situations the general case might occur. Consider the case where there are two sets of random tapes that reach the same final node, where on one set the parties output 0 and on the other they output 1. This can happen if the final output is based on parties' full views, including their random tapes, rather than on the public transcript of the computation alone.

We now show that any protocol can be converted into a protocol where the transcript fully determines the output of both parties. Given a coin-tossing protocol π we build a protocol π' as follows: The parties invoke the protocol π and at the end of the computation, each party sends its random tape to the other party. In the protocol π', clearly each leaf fully determines the outputs of the parties, since each parties' entire view is in the public transcript. Furthermore, the requirement that two honest parties output the same bit except with negligible probability clearly holds also for π'. Finally, we claim that if there exists an adversary $\mathcal{A}'$ who succeeds in biasing the output of an honest party with non-negligible probability in π', then we can construct an adversary $\mathcal{A}$ who succeeds in biasing the output of an honest party with non-negligible probability in π. The adversary $\mathcal{A}$ simply invokes $\mathcal{A}'$, and aborts at the point where it is supposed to send its random tape. Since the output of the honest party is determined after the invocation of the protocol π and before the random tapes are sent as part of π', the output of the honest party when interacting with $\mathcal{A}$ is identical to its output when interacting with $\mathcal{A}'$. Thus, $\mathcal{A}$ also succeeds to bias the output of the honest party in π.

Given the above, we can transform a general π into π' for which the third simplifying assumption holds. Then, we can build the adversaries as proven for π', under this assumption. By what we have just shown, this implies the existence of an adversary who can bias the output of the honest party in π, as required. This concludes the proof. ■

Impossibility for 0 or 1-balanced functions. The above theorem proves impossibility for the case that the function is not balanced. As we have mentioned, we must separately deal with the case that the function *is* balanced, but not *strictly* balanced; i.e., the function is either 0-balanced or 1-balanced. The main difference in this case is that not all nodes which have significantly different probabilities in their two children can be used by the adversary to bias the outcome. This is due to the fact that the protocol may specify an input distribution for the honest party at such a node that forces the output to be either 0 or 1 (except with negligible probability), and so the "different child" is only reached with negligible probability. This can happen since the function *is* balanced with $\delta = 0$ or $\delta = 1$. Thus, there may be probability vectors for which $\delta_{\max} - \delta_{\min}$ is 0 or some negligible function. The proof therefore shows that this cannot happen too often, and the adversary can succeed enough to bias the output.

Theorem 4.5.4 *Let $f:\{x_1,\ldots,x_\ell\}\times\{y_1,\ldots,y_m\} \to \{0,1\}$ be a 0 or 1-balanced function. Then, f does not information-theoretically imply the coin-tossing protocol.*

Proof: By Claim 4.3.5, since f is 0 or 1-balanced, it contains the all-zero row and all-zero column, or the all-one row and the all-one column. Thus, if f is 0-balanced, an adversary can "force" the output of any given invocation of f to be 0, and so $\delta^v_{\min}$ (as defined in the proof of Claim 4.5.3) equals 0. Likewise, if f is 1-balanced, an adversary can force the output to be 1 and thus $\delta^v_{\max} = 1$. We use the same ideas as the proof of Theorem 4.5.1. We consider the transcript tree with the same simplifications as above (in particular, we consider protocols with invocations of f only, and no invocations of f^T). The transformation to the general case is the same, and we omit this part. In this proof, we set $\epsilon = \epsilon(\kappa) = \frac{1/4-\mu(\kappa)}{r(\kappa)}$. We start by rewriting the adversaries, and we write the differences in **bold**.

Adversary $\mathcal{A}^1$ controlling P_2 (biasing towards 1):
Let v be the current node in the tree $\mathcal{T}$ (starting at the root) and $\boldsymbol{\alpha}^v = (\alpha^v_1,\ldots,\alpha^v_n)$ be the input distribution for P_1 (as explained above, this can be computed). Compute $(\delta^v_1,\ldots,\delta^v_\ell) = \boldsymbol{\alpha}^v \cdot M_f$. Let i and k be indices such that $\delta^v_i = \max_{1\le j\le\ell}\{\delta^v_j\}$ and $\delta^v_k = \min_{1\le j\le\ell}\{\delta^v_j\}$ (denote $\delta^v_{\max} = \delta^v_i$ and $\delta^v_{\min} = \delta^v_k$). In the *first* node v in the execution with two children such that $|p_{right(v)} - p_v| \ge \epsilon$ or $|p_{left(v)} - p_v| \ge \epsilon$ **and $\boldsymbol{\epsilon \le \gamma^v \le 1-\epsilon}$** (checking also that $\epsilon \le \gamma^v \le 1-\epsilon$ is the only difference from previously), act according to the following:

If $p_{right(v)} \geq p_v + \epsilon$ or $p_{left(v)} \leq p_v - \epsilon$ then send input y_i in this invocation of f (this increases the probability of reaching node $right(v)$ because the probability of obtaining 1 in this invocation is δ_i which is *maximal*). If $p_{left(v)} \geq p_v + \epsilon$ or $p_{right(v)} \leq p_v - \epsilon$ then send input y_k in this invocation of f (this increases the probability of reaching node $left(v)$ because the probability of obtaining 0 in this invocation is δ_k which is *minimal*).

In all other nodes act honestly.

The adversary $\mathcal{A}^0$ controlling P_2 (biasing towards 0): The adversary is the same as $\mathcal{A}^1$ as defined above with the same differences as in the previous proof.

We now turn to the analysis. Let $\mathcal{V}$ denote the set of all nodes for which the adversaries do not act honestly and attempt to bias the result (i.e., the nodes that fulfill the conditions above). Observe that $\mathcal{V}$ is a subset of the set $\mathcal{V}$ defined in the proof of Theorem 4.5.1. We compute the sum of differences between the output probability at $v \in \mathcal{V}$ when honest parties run, and when an honest party P_1 runs with dishonest P_2. Like the proof of Theorem 4.5.1, we show that whenever the adversaries reach a node $v \in \mathcal{V}$, they succeed in biasing the result (the proof of this is almost the same as above). Then, we compute the probability that the execution reaches a node in $\mathcal{V}$. However, here, unlike the proof of Theorem 4.5.1, the probability of reaching a node $v \in \mathcal{V}$ is not 1.

The adversary's effect conditioned on reaching $\mathcal{V}$. We first obtain a bound on the sum of difference between the output probability at $v \in \mathcal{V}$ when honest parties run, and when an honest party P_1 runs with adversaries, $\mathcal{A}^1, \mathcal{A}^0$ respectively, where in both cases we are conditioning on the event that the execution reaches a node $v \in \mathcal{V}$. Since the attack is exactly the same as in the previous proof (Claim 4.5.3), we conclude that for any node $v \in \mathcal{V}$:

$$\Big(\mathsf{out}^0_{|v}(P_1, \mathcal{A}^0) - \mathsf{out}^0_{|v}(P_1, P_2)\Big) + \Big(\mathsf{out}^1_{|v}(P_1, \mathcal{A}^1) - \mathsf{out}^1_{|v}(P_1, P_2)\Big) \geq (\delta^v_{\max} - \delta^v_{\min}) \cdot \epsilon$$

We now compute a lower bound on the term $\delta^v_{\max} - \delta^v_{\min}$. In the above proof for an unbalanced function this was bounded by a constant c. In this case, where f is 0 or 1-balanced, Lemma 4.3.4 does not hold, and thus the proof is more involved.

First, consider the case that f is 1-balanced. As we have stated above, this implies that $\delta^v_{\max} = 1$ for every node v. We now show that $\delta^v_{\min} \leq 1 - \epsilon$. In order to see this, we note that $\delta^v_{\min} \leq \gamma^v$, for every node v (since γ^v is the "correct probability" of getting 1 and this is at least the minimum probability). Now, for every $v \in \mathcal{V}$, by the definition of $\mathcal{V}$ for this adversary we have that $\gamma^v \leq 1 - \epsilon$, and therefore we can conclude that $\delta^v_{\min} \leq 1 - \epsilon$, and so $\delta^v_{\max} - \delta^v_{\min} \geq \epsilon$. Next, consider the case that f is 0-balanced. In this case, $\delta^v_{\min} = 0$, but since $\delta^v_{\max} \geq \gamma^v$ (for the same reason that $\delta^v_{\min} \leq \gamma^v$) and since $\gamma^v \geq \epsilon$ by the definition of the adversary, we have that $\delta^v_{\max} \geq \epsilon$ and so once again $\delta^v_{\max} - \delta^v_{\min} \geq \epsilon$.

Combining the above, we have that for every $v \in \mathcal{V}$ the overall sum of the probability changes of the adversaries is:

$$\begin{aligned}\Big(\mathsf{out}^0_{|v}(P_1, \mathcal{A}^0) - \mathsf{out}^0_{|v}(P_1, P_2)\Big) &+ \Big(\mathsf{out}^1_{|v}(P_1, \mathcal{A}^1) - \mathsf{out}^1_{|v}(P_1, P_2)\Big) \\ &\geq (\delta^v_{\max} - \delta^v_{\min}) \cdot \epsilon \geq (1 - (1 - \epsilon)) \cdot \epsilon = \epsilon^2\end{aligned}$$

We remark that for the case where M_f is 0-balanced and not 1-balanced, $\delta^v_{\min} = 0$, but it follows that $\delta^v_{\max} \geq \gamma^v \geq \epsilon$, and so $\delta^v_{\max} - \delta^v_{\min} \geq \epsilon$, as well.

Computing the probability to reach $\mathcal{V}$. We have shown that when the parties reach nodes from $\mathcal{V}$, the adversaries succeed in biasing the result. Now, whenever the adversaries do not reach nodes from $\mathcal{V}$, they play honestly and therefore both parties are expected to output a fair coin. Therefore, if the execution reaches a node in $\mathcal{V}$ with non-negligible probability, this will imply non-negligible bias, as required. We now show that the probability that an execution reaches a node in $\mathcal{V}$ is at least $1/2$ (this is the key difference to the unbalanced case where *every* execution reached a node in $\mathcal{V}$).

Every pair of random coins (r_1, r_2) for the parties fully determine a path from the root to one of the leaves, and each leaf defines the output for both parties. For each node v in the tree $\mathcal{T}$, we consider its type according to the following:

(a) $|p_{right(v)} - p_v| < \epsilon$ and $|p_{left(v)} - p_v| < \epsilon$, or v has only one child

(b) $|p_{right(v)} - p_v| \geq \epsilon$ and $\epsilon \leq \gamma^v \leq 1 - \epsilon$

(c) $|p_{right(v)} - p_v| \geq \epsilon$ and $\gamma^v < \epsilon$

(d) $|p_{left(v)} - p_v| \geq \epsilon$ and $\epsilon \leq \gamma^v \leq 1 - \epsilon$

(e) $|p_{left(v)} - p_v| \geq \epsilon$ and $\gamma^v > 1 - \epsilon$

We first prove that these cases cover all possibilities. Specifically, we claim that it cannot hold that $|p_{right(v)} - p_v| \geq \epsilon$ and $\gamma^v > 1 - \epsilon$. Moreover, it cannot hold that $|p_{left(v)} - p_v| \geq \epsilon$ and $\gamma^v < \epsilon$. In order to see this, first note that since $p_v = \gamma^v \cdot p_{right(v)} + (1 - \gamma^v) \cdot p_{left(v)}$, we can write:

$$\left|p_{right(v)} - p_v\right| = \left|p_{right(v)} - \gamma^v \cdot p_{right(v)} - (1 - \gamma^v) \cdot p_{left(v)}\right| = (1 - \gamma^v) \cdot \left|p_{right(v)} - p_{left(v)}\right|, \tag{4.5.1}$$

and,

$$\left|p_{left(v)} - p_v\right| = \left|p_{left(v)} - \gamma^v \cdot p_{right(v)} - (1 - \gamma^v) \cdot p_{left(v)}\right| = \gamma^v \cdot \left|p_{left(v)} - p_{right(v)}\right|. \tag{4.5.2}$$

Now, $|p_{right(v)} - p_{left(v)}| \leq 1$ because they are both probabilities, and thus $|p_{right(v)} - p_v| \leq 1 - \gamma^v$. If $|p_{right(v)} - p_v| \geq \epsilon$ then this implies that $1 - \gamma^v \geq \epsilon$ and thus $\gamma^v \leq 1 - \epsilon$. Thus, it cannot be that $|p_{right(v)} - p_v| \geq \epsilon$ and $\gamma^v > 1 - \epsilon$, as required. Similarly, if $|p_{left(v)} - p_v| \geq \epsilon$ then $\gamma^v \geq \epsilon$ and thus it cannot be that $|p_{left(v)} - p_v| \geq \epsilon$ and $\gamma^v < \epsilon$.

Next, recall that the nodes in $\mathcal{V}$ are all of type (d) or (b). We now show that for any node of type (e), it must hold that $|p_v - p_{right(v)}| < \epsilon$. Similarly, for any node of type (c), it must hold that $|p_v - p_{left(v)}| < \epsilon$. We explain how we use this, immediately after the proof of the claim.

Claim 4.5.5 *Let v be a node in the tree $\mathcal{T}$ with two children. Then:*

1. *If $\gamma^v > 1 - \epsilon$, then $|p_{right(v)} - p_v| < \epsilon$.*

2. *If $\gamma^v < \epsilon$, then $|p_{left(v)} - p_v| < \epsilon$.*

Proof: For the first item, by Eq. (4.5.1) we have that $|p_{right(v)} - p_v| \leq 1 - \gamma^v$. In addition, if $\gamma^v > 1 - \epsilon$ then $1 - \gamma^v < \epsilon$. We conclude that $\left|p_{right(v)} - p_v\right| \leq 1 - \gamma^v < \epsilon$, as required. The second item is proven analogously using Eq. (4.5.2). ■

The claim above shows that for any node v of type (e) the right child has output probability that is close to the probability of the parent, and for any node of type (c) the left child has output probability that is close to the probability of the parent. This means that little progress towards the output is made in this invocation. Intuitively, it cannot be the case that little progress is made in *all* invocations. We will now show this formally.

Let $u_1, \ldots, u_{r(\kappa)}$ be a path in the tree $\mathcal{T}$ from the root u_1 to some leaf $u_{r(\kappa)}$. Recall that any execution defines a unique path from the root and one of the leaves. The next claim shows that there does not exist a path in the tree (associated with a pair of random tapes (r_1, r_2)) such that all the nodes in the path are of types (a), (e) and (c), and in all the nodes of type (e) the path proceeds to the right son and in all the nodes of type (c) the path proceeds to the left son.

Claim 4.5.6 *There does not exist a path $u_1, \ldots, u_{r(\kappa)}$ for which for every $i = 1, \ldots, r(\kappa) - 1$ one of the following conditions hold:*

1. $|p_{right(u_i)} - p_{u_i}| < \epsilon$ *and* $|p_{left(u_i)} - p_{u_i}| < \epsilon$, *or u_i has only one child* (i.e., u_i is type (a)).
2. $|p_{left(u_i)} - p_{u_i}| \geq \epsilon$, $\gamma^{u_i} > 1 - \epsilon$ *and* $u_{i+1} = right(u_i)$ (that is, u_i is type (e), and the path proceeds to the right son).
3. $|p_{right(u_i)} - p_{u_i}| \geq \epsilon$, $\gamma^{u_i} < \epsilon$ *and* $u_{i+1} = left(u_i)$. (that is, u_i is type (c), and the path proceeds to the left son).

Proof: Assume by contradiction that there exists a path $u_1, \ldots, u_{r(\kappa)}$ for which for every $i = 1, \ldots, r(\kappa) - 1$, one of the above conditions holds. Let u_i be a node. If the first item holds for u_i, then $|p_{u_{i+1}} - p_{u_i}| < \epsilon$ (observe that if u_i has just one child then $p_{u_{i+1}} = p_{u_i}$ and so $|p_{u_{i+1}} - p_{u_i}| < \epsilon$). If the second item holds for u_i, then this is a node of type (e) and the path proceeds to the right son. By Claim 4.5.5, this implies that $|p_{u_{i+1}} - p_{u_i}| < \epsilon$ (observe that the claim can be applied since the node u_i has two children by the fact that it is not of type (a)). Likewise, if the third item holds for u_i, then this is a node of type (c) and the party proceeds to the left son, and so by Claim 4.5.5 it holds that $|p_{u_{i+1}} - p_{u_i}| < \epsilon$. We conclude that for every $i = 1, \ldots, r(\kappa) - 1$, $|p_{u_{i+1}} - p_{u_i}| < \epsilon$, and so:

$$|p_{u_{r(\kappa)}} - p_{u_1}| \leq \sum_{i=1}^{r(\kappa)} |p_{u_{i+1}} - p_{u_i}| < \sum_{i=1}^{r(\kappa)} \epsilon = r(\kappa) \cdot \epsilon = r(\kappa) \cdot \frac{1/4 - \mu(\kappa)}{r(\kappa)} = 1/4 - \mu(\kappa).$$

Now, recall that $|p_{u_1} - 1/2| < \mu(\kappa)$ for some negligible function $\mu(\kappa)$. This implies that $p_{u_{r(\kappa)}}$ is non-negligibly far from the range $\left[\frac{1}{4}, \frac{3}{4}\right]$, in contradiction to the fact that it equals to 0 or 1. This completes the proof. ■

Recall that nodes in $\mathcal{V}$ are of type (b) and (d) only, and so if they are reached in an execution then the adversary succeeds in biasing the result. In addition, we have shown that it cannot be the case that an entire execution is made up of nodes of type (a) and nodes of type (e) where the execution when to the right and nodes of type (c) in which the execution went to the left. Thus, if no node in $\mathcal{V}$ is reached, it *must be the case* that a node of type (e) was reached and the execution went to the left, meaning that the output of the invocation of f was 0, or the execution reached a node of type (c) and the execution went to the right, meaning that the output of f was 1. We now show that the probability of this occurring is small. That is, we bound the following probability:

$$\Pr\left[\exists j \text{ for which } \mathsf{type}(u_j) = e : u_{j+1} = right(u_j) \quad \vee \quad \exists j \text{ for which } \mathsf{type}(u_j) = c : u_{j+1} = left(u_j)\right].$$

By the union bound, it is enough to compute the probability of each one of the terms. Observe that:

$$\Pr\left[u_{j+1} = left(u_j) \mid \mathsf{type}(u_j) = (e)\right] < \epsilon$$

because by the definition of type (e), $\gamma^{u_j} > 1 - \epsilon$. We therefore have:

$$\begin{aligned} &\Pr\left[\exists j \text{ for which } \mathsf{type}(u_j) = c : u_{j+1} = left(u_j)\right] \\ \leq \quad & \sum_{j \text{ s.t. } \mathsf{type}(u_j)=c} \Pr\left[u_{j+1} = left(u_j) \mid \mathsf{type}(u_j) = c\right] \\ < \quad & r(\kappa) \cdot \epsilon = r(\kappa) \cdot \frac{1/4 - \mu(\kappa)}{r(\kappa)} = 1/4 - \mu(\kappa). \end{aligned}$$

In a similar way, we conclude that the probability to reach a node of type (c) and to move to the right child is bounded by $1/4 - \mu(\kappa)$. Therefore, the probability of not reaching a node in $\mathcal{V}$ is bounded by $1/2 - 2\mu(\kappa)$.

Thus the probability of reaching a node in $\mathcal{V}$ is greater than 1/2. That is, we have that:

$$\sum_{v \in \mathcal{V}} \mathsf{reach}_v > \frac{1}{2}$$

where for every $v \in \mathcal{V}$, reach_v denotes the probability to reach the node v (which, as discussed in the proof of the unbalanced case, is the same in all executions of $\langle P_1, P_2^* \rangle$, where $P_2^* \in \{P_2, \mathcal{A}^0, \mathcal{A}^1\}$).

Concluding the proof. We have seen that the probability of reaching a node in $\mathcal{V}$ is greater than 1/2. Moreover, whenever the adversaries reach such a node, the sum of the differences between honest execution and execution with the adversaries is ϵ^2. Combining the above, we conclude that:

$$\begin{aligned} &\left(\mathsf{out}^1(P_1, \mathcal{A}^1) - \mathsf{out}^1(P_1, P_2)\right) + \left(\mathsf{out}^0(P_1, \mathcal{A}^1) - \mathsf{out}^0(P_1, P_2)\right) \\ = \quad & \sum_{v \in \mathcal{V}} \mathsf{reach}_v \cdot \left[\left(\mathsf{out}^1_{|v}(P_1, \mathcal{A}^1) - \mathsf{out}^1_{|v}(P_1, P_2)\right) + \left(\mathsf{out}^0_{|v}(P_1, \mathcal{A}^0) - \mathsf{out}^0_{|v}(P_1, P_2)\right)\right] \\ \geq \quad & \sum_{v \in \mathcal{V}} \mathsf{reach}_v \cdot \left(\epsilon^2\right) > \frac{1}{2} \cdot \epsilon^2 \end{aligned}$$

Similarly to the proof of Theorem 4.5.1, this implies that:

$$\left(\mathsf{out}^0(P_1, \mathcal{A}^0) - \frac{1}{2}\right) + \left(\mathsf{out}^1(P_1, \mathcal{A}^1) - \frac{1}{2}\right) > \frac{1}{2} \cdot \epsilon^2 - 2\mu(\kappa).$$

and so one of the adversaries succeeds in biasing the result with non-negligible probability. This concludes the proof. ■

Conclusion: Combining Theorems 4.4.1, 4.5.1 and 4.5.4, we obtain the following corollary:

Corollary 4.5.7 *Let $f : \{x_1, \dots, x_\ell\} \times \{y_1, \dots, y_m\} \to \{0, 1\}$ be a function.*

- *If f is strictly-balanced, then f implies the coin-tossing functionality (computationally and information-theoretically).*
- *If f is not strictly-balanced, then f does not information-theoretically imply the coin-tossing functionality in the presence of malicious adversaries.*

Impossibility in the OT-hybrid model. Our proof of impossibility holds in the information-theoretic setting only since the adversary must carry out computations that do not seem to be computable in polynomial-time. It is natural to ask whether or not the impossibility result still holds in the computational setting. We do not have an answer to this question. However, as a step in this direction, we show that the impossibility still holds if the parties are given access to an ideal oblivious transfer (OT) primitive as well as to the function f. That is, we prove the following:

Theorem 4.5.8 *Let $f : \{x_1, \dots, x_\ell\} \times \{y_1, \dots, y_m\} \to \{0, 1\}$ be a function. If f is not strictly-balanced, then the pair of functions (f, OT) do not information-theoretically imply the coin tossing functionality in the presence of malicious adversaries.*

Proof Sketch: In order to see that this is the case, first observe that if f has an embedded-OR then it implies oblivious transfer [69]. Thus, f can be used to obtain OT, and so the question of whether f implies coin tossing or (f, OT) imply coin tossing is the same. It thus remains to consider the case that f does not have an embedded OR but does have an embedded XOR (if it has neither then it is trivial and so clearly cannot imply coin tossing, as we have mentioned). We now show that in such a case f must be strictly balanced, and so this case is not relevant. Let x_1, x_2, y_1, y_2 be an embedded XOR in f; i.e., $f(x_1, y_1) = f(x_2, y_2) \neq f(x_1, y_2) = f(x_2, y_1)$. Now, if there exists a y_3 such that $f(x_1, y_3) = f(x_2, y_3)$ then f has an embedded OR. Thus, x_1 and x_2 must be complementary rows (as in example function (a) in the Introduction). Likewise, if there exists an x_3 such that $f(x_3, y_1) = f(x_3, y_2)$ then f has an embedded OR. Thus, y_1 and y_2 must be complementary columns. We conclude that f has two complementary rows and columns, and as we have shown in the Introduction, this implies that f is strictly balanced with $\delta = \frac{1}{2}$. ■

4.6 Fairness in the Presence of Fail-Stop Adversaries

In order to study the feasibility of achieving fair secure computation in the fail-stop model, we must first present a definition of security for this model. To the best of our knowledge, there is no simulation-based security definition for the fail-stop model in the literature. As we have mentioned in the introduction, there are two natural ways of defining security in this model, and it is not a priori clear which is the "correct one". We therefore define two models and study feasibility for both. In the first model, the ideal-model adversary/simulator must either send the party's prescribed input to the trusted party computing the function, or a special abort symbol

$\perp$, but nothing else. This is similar to the semi-honest model, except that $\perp$ can be sent as well. We note that if $\perp$ is sent, then both parties obtain $\perp$ for output and thus fairness is preserved.[5] This is actually a very strong requirement from the protocol since both parties either learn the prescribed output, or they both output $\perp$. In the second model, the ideal adversary can send any input that it wishes to the trusted party, just like a malicious adversary. We remark that if the real adversary does not abort a real protocol execution, then the result is the same as an execution of two honest parties and thus the output is computed from the prescribed inputs. This implies that the ideal adversary can really only send a different input in the case that the real adversary halts before the protocol is completed. As we have mentioned in the Introduction, the impossibility result of Cleve [34] for coin-tossing holds in the both models, since the parties have no input, and so for this functionality the models are identical.

4.6.1 Fail-Stop 1

In this section we define and explore the first fail-stop model. We proceed directly to define the ideal model:

Execution in the ideal world. An ideal execution involves parties P_1 and P_2, an adversary $\mathcal{S}$ who has corrupted one of the parties, and the trusted party. An ideal execution for the computation of f proceeds as follows:

Inputs: P_1 and P_2 hold inputs $x \in X$, and $y \in Y$, respectively; the adversary $\mathcal{S}$ receives the security parameter 1^n and an auxiliary input z.

Send inputs to trusted party: The honest party sends its input to the trusted party. The corrupted party controlled by $\mathcal{S}$ may send its prescribed input or $\perp$.

Trusted party sends outputs: If an input $\perp$ was received, then the trusted party sends $\perp$ to both parties. Otherwise, it computes $f(x, y)$ and sends the result to both parties.

Outputs: The honest party outputs whatever it was sent by the trusted party, the corrupted party outputs nothing and $\mathcal{S}$ outputs an arbitrary function of its view.

We denote by $\text{IDEAL}^{\textsf{f-stop-1}}_{f,\mathcal{S}(z)}(x, y, n)$ the random variable consisting of the output of the adversary and the output of the honest party following an execution in the ideal model as described above.

Security. The real model is the same as is defined in Section 2.2, except that we consider adversaries that are fail-stop only. This means that the adversary must behave exactly like an honest party, except that it can halt whenever it wishes during the protocol. We stress that its decision to halt or not halt, and where, may depend on its view. We are now ready to present the security definition.

Definition 4.6.1 (Security – fail-stop1) *Protocol* π securely computes f with complete fairness in the fail-stop1 model *if for every non-uniform probabilistic polynomial-time fail-stop adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial-time adversary*

[5] It is necessary to allow an explicit abort in this model since if the corrupted party does not participate at all then output cannot be computed. The typical solution to this problem, which is to take some default input, is not appropriate here because this means that the simulator can change the input of the corrupted party. Thus, such an early abort must result in output $\perp$.

$\mathcal{S}$ *in the ideal model such that:*

$$\left\{\text{IDEAL}^{\textsf{f-stop-1}}_{f,\mathcal{S}(z)}(x_1, x_2, n)\right\}_{x\in X, y\in Y, z\in\{0,1\}^*, n\in\mathbb{N}} \overset{c}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z)}(x, y, n)\right\}_{x\in X, y\in Y, z\in\{0,1\}^*, n\in\mathbb{N}} .$$

Exploring fairness in the fail-stop-1 model. We first observe that if a function contains an embedded XOR, then it cannot be computed fairly in this model.

Theorem 4.6.2 *Let* $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0, 1\}$ *be a function that contains an embedded XOR. Then,* f *implies the coin-tossing functionality and thus cannot be computed fairly.*

Proof: Assume that f contains an embedded XOR; i.e., there exist inputs x_1, x_2, y_1, y_2 such that $f(x_1, y_1) = f(x_2, y_2) \neq f(x_1, y_2) = f(x_2, y_1)$. We can easily construct a protocol for coin-tossing using f that is secure in the fail-stop model. Party P_1 chooses input $x \in \{x_1, x_2\}$ uniformly at random, P_2 chooses $y \in \{y_1, y_2\}$ uniformly at random, and the parties invoke the function f where P_1 inputs x and P_2 inputs y. In case the result of the invocation is $\perp$, the other party chooses its output uniformly at random.

Since the adversary is fail-stop1, it must follow the protocol specification or abort prematurely. In both cases, it is easy to see that the honest party outputs an unbiased coin. Formally, for any given fail-stop adversary $\mathcal{A}$ we can construct a simulator $\mathcal{S}$: $\mathcal{S}$ receives from the coin tossing functionality f_{CT} the bit b, and invokes the adversary $\mathcal{A}$. If $\mathcal{A}$ sends the trusted party computing f the symbol $\perp$, then $\mathcal{S}$ responds with $\perp$. Otherwise, (if $\mathcal{A}$ sends some real value - either x_1, x_2 if it controls P_1, or y_1, y_2 if it controls P_2), then $\mathcal{S}$ responds with the bit b that it received from f_{CT} as if it is the output of the ideal call to f. It is easy to see that the ideal and real distributions are identical. ■

As we have mentioned, if a function does not contain an embedded XOR or an embedded OR then it is trivial and can be computed fairly (because the output depends on only one of the party's inputs). It therefore remains to consider the feasibility of fairly computing functions that have an embedded OR but no embedded XOR. Gordon et. al [58] present a protocol for securely computing any function of this type with complete fairness, in the presence of a malicious adversary. However, the security of their protocol relies inherently on the ability of the simulator to send the trusted party an input that is not the corrupted party's prescribed input. Thus, their protocol seems not to be secure in this model.

The problem of securely computing functions that have an embedded OR but no embedded XOR therefore remains open. We remark that there are very few functions of this type, and any such function has a very specific structure, as discussed in [58].

4.6.2 Fail-Stop 2

In this section we define and explore the second fail-stop model. In this case, the ideal adversary can send any value it wishes to the trusted party (and the output of the honest party is determined accordingly). It is easy to see that in executions where the real adversary does not abort the output is the same as between two honest parties. Thus, the ideal adversary is forced to send the prescribed input of the party in this case. Observe that the ideal model here is identical to the ideal model for the case of malicious adversaries. Thus, the only difference between this definition and the definition of security for malicious adversaries is the quantification over the real adversary; here we quantify only over fail-stop real adversaries. Otherwise, all is the same.

Definition 4.6.3 (Security – fail-stop2) *Protocol π* securely computes f with complete fairness in the fail-stop2 model *if for every non-uniform probabilistic polynomial-time fail-stop adversary $\mathcal{A}$ in the real world, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that:*

$$\left\{\mathrm{IDEAL}_{f,\mathcal{S}(z)}(x,y,n)\right\}_{x\in X,y\in Y,z\in\{0,1\}^*,n\in\mathbb{N}} \overset{\mathrm{c}}{\equiv} \left\{\mathrm{REAL}_{\pi,\mathcal{A}(z)}(x,y,n)\right\}_{x\in X,y\in Y,z\in\{0,1\}^*,n\in\mathbb{N}} .$$

In the g-hybrid-model for fail-stop2 adversaries, where the parties have access to a trusted party computing function g for them, a corrupted party may provide an incorrect input to an invocation of g as long as it halts at that point. This may seem arbitrary. However, it follows naturally from the definition since a secure fail-stop2 protocol is used to replace the invocations of g in the real model. Thus, if a fail-stop adversary can change its input as long as it aborts in the real model, then this capability is necessary also for invocations of g in the g-hybrid model.

Exploring fairness in the fail-stop-2 model. In the following we show that the malicious adversaries that we constructed in the proofs of Theorem 4.5.1 and Theorem 4.5.4 can be modified to be *fail-stop2*. We remark that the adversaries that we constructed did not abort during the protocol execution, but rather continued after providing a "different" input in one of the f invocations. Thus, they are not fail-stop2 adversaries. In order to prove the impossibility for this case, we need to modify the adversaries so that they *halt* at the node v for which they can bias the outcome of the invocation (i.e., a node v for which v's children in the execution tree have significantly different probabilities for the output of the entire execution equalling 1). Recall that in this fail-stop model, the adversary is allowed to send a different input than prescribed in the invocation at which it halts; thus, this is a valid attack strategy.

We prove impossibility in this case by considering two possible cases, relating to the potential difference between the honest party outputting 1 at a node when the other party aborts at that node but until then was fully honest, or when the other party continues honestly from that node (to be more exact, we consider the average over all nodes).

1. First, assume that there is a noticeable difference between an abort after fully honest behavior and a fully honest execution. In this case, we construct a fail-stop adversary who plays fully honestly until an appropriate node where such a difference occurs and then halts. (In fact, such an adversary is even of the fail-stop1 type).

2. Next, assume that there is *no noticeable difference* between an abort after fully honest behavior and a fully honest execution. Intuitively, this means that continuing honestly or halting makes no difference. Thus, if we take the malicious adversaries from Section 4.5 and modify them so that they halt immediately after providing malicious input (as required in the fail-stop2 model), then we obtain that there is no noticeable difference between the original malicious adversary and the fail-stop2 modified adversary. We remark that this is not immediate since the difference in this case is between aborting and not aborting without giving any malicious input. However, as we show, if there is no difference when honest inputs are used throughout, then this is also no difference when a malicious input is used.

We conclude that one of the two types of fail-stop2 adversaries described above can bias any protocol.

Theorem 4.6.4 *Let $f:\{x_1,\ldots,x_\ell\}\times\{y_1,\ldots,y_m\}\rightarrow\{0,1\}$ be a function that is* not *δ-balanced, for any $0<\delta<1$. Then, f does not information-theoretically imply the coin-tossing protocol in the fail-stop2 model.*

Proof: We follow the proofs of Theorem 4.5.4 and use the same ideas. Recall that the adversaries play honestly until they reach a node in $\mathcal{V}$. Then, the adversaries send an input value to the invocation of f not according to the protocol, but continue to play honestly afterward. In our case, we define similar adversaries: the adversaries play honestly until they reach a node in $\mathcal{V}$. Then, they send an input value not according to the protocol and abort. This is allowed in our model: the adversaries play honestly and according to the protocol until they reach a node for which they quit. In the single invocation of f for which they abort they send a different value to that specified by the protocol. This is allowed since in the fail-stop2 model the ideal simulator is allowed to send any input it desires in the case of an early abort.

We recall that the adversaries reach a node in the set $\mathcal{V}$ with probability greater than 1/2. We now analyze whether they succeed to bias the result when the execution reaches a node in $\mathcal{V}$.

Aborting vs. continuing honestly. At each node v of the transcript tree $\mathcal{T}$, we write in addition to the probability of outputting 1 when both parties act honestly (i.e., p_v), the probability that the output of the honest party is 1 if the execution is terminated in the node v (i.e., the "default output" upon abort). We denote this probability by q_v. Now, recall the definition of the set $\mathcal{V}$: the set of nodes for which the adversary does not act honestly and changes the input. As mentioned above, the adversary act honestly until it reaches a node in $\mathcal{V}$. Then, in this node it aborts and so may send an incorrect value to the invocation of f, and the honest party receives 0 or 1 from the invocation according the the inputs that were sent. In all the following invocations of f, the honest party receives $\perp$ and thus it chooses its output according to the probability $q_{right(v)}$ (if the execution proceeded to the right child of v in the last invocation of f) or $q_{left(v)}$ (if the execution proceeded to the left child). Recall that γ^v denotes the probability of receiving output 1 from the invocation of f at the node v in an honest execution, and thus this is the probability that the execution proceeds to the right child of v. Moreover, recall that in both proofs, each node $v\in\mathcal{V}$ has two children. We have the following claim:

Claim 4.6.5 *Let $\mathcal{V}$ be as above. Then, there exists a negligible function $\mu'(\cdot)$ such that:*

$$\sum_{v\in\mathcal{V}}\left(|p_{left(v)}-q_{left(v)}|+|p_{right(v)}-q_{right(v)}|\right)\cdot\mathsf{reach}_v\leq\mu'(\kappa)$$

Proof: Assume in contradiction that the claim does not hold. Then, there exists a non-negligible function $\omega(\kappa)$ such that:

$$\sum_{v\in\mathcal{V}}\left(|p_{left(v)}-q_{left(v)}|+|p_{right(v)}-q_{right(v)}|\right)\cdot\mathsf{reach}_v\geq\omega(\kappa)$$

We construct an adversary that succeed to bias the coin with some non-negligible probability.

Let $\mathcal{V}' = \{left(v), right(v) \mid v \in \mathcal{V}\}$, i.e., all the children of nodes in $\mathcal{V}$, and define the sets $\mathcal{V}'_p = \{v' \in \mathcal{V}' \mid p_{v'} \geq q_{v'}\}$ and $\mathcal{V}'_q = \{v' \in \mathcal{V}' \mid p_{v'} < q_{v'}\}$. Note that $\mathcal{V}'_p$ and $\mathcal{V}'_q$ constitute a partitioning of the set $\mathcal{V}'$. We now consider two adversaries $\mathcal{B}^0$ and $\mathcal{B}^1$ that work as follows: the adversary $\mathcal{B}^0$ aborts at each node $v' \in \mathcal{V}_p$, and $\mathcal{B}^1$ aborts at each node for which $v' \in \mathcal{V}_q$ (that is, both adversaries send $\perp$ to the invocation of f in the node v'), but play completely honestly until that point (these adversaries never send "incorrect" input in an invocation of f).

Let $v' \in \mathcal{V}_p$. The difference in the probability that an honest party outputs 0 in an honest execution and when interacting with the adversary $\mathcal{B}^0$, conditioned on the event that the node v' is reached, is:

$$\begin{aligned}\mathsf{out}^0_{|v'}(P_1, \mathcal{B}^0) - \mathsf{out}^0_{|v'}(P_1, P_2) &= \Pr_{(r_1,r_2)\leftarrow \mathsf{Uni}}\Big[\textsc{output}\langle P_1(r_1), \mathcal{B}^0(r_2)\rangle = 0 \;\Big|\; v' \text{ is reached}\Big] \\ &\quad - \Pr_{(r_1,r_2)\leftarrow \mathsf{Uni}}\Big[\textsc{output}\langle P_1(r_1), P_2(r_2)\rangle = 0 \;\Big|\; v' \text{ is reached}\Big] \\ &= (1 - q_{v'}) - (1 - p_{v'}) = p_{v'} - q_{v'} \ .\end{aligned}$$

Recall that the adversary $\mathcal{B}^1$ plays honestly when it reaches a node $v' \in \mathcal{V}'_p$, and so for such nodes $v' \in \mathcal{V}'_p$,

$$\mathsf{out}^0_{|v'}(P_1, \mathcal{B}^1) - \mathsf{out}^0_{|v'}(P_1, P_2) = 0.$$

Next, consider a node $v' \in \mathcal{V}'_q$. The difference in the probability that an honest party outputs 1 in an honest execution and when interacting with adversary $\mathcal{B}^1$, conditioned on the event that v' is reached, is:

$$\begin{aligned}\mathsf{out}^1_{|v'}(P_1, \mathcal{B}^1) - \mathsf{out}^1_{|v'}(P_1, P_2) &= \Pr_{(r_1,r_2)\leftarrow \mathsf{Uni}}\Big[\textsc{output}\langle P_1(r_1), \mathcal{B}^1(r_2)\rangle = 1 \;\Big|\; v' \text{ is reached}\Big] \\ &\quad - \Pr_{(r_1,r_2)\leftarrow \mathsf{Uni}}\Big[\textsc{output}\langle P_1(r_1), P_2(r_2)\rangle = 1 \;\Big|\; v' \text{ is reached}\Big] \\ &= q_{v'} - p_{v'}\end{aligned}$$

On the other hand, when $v' \in \mathcal{V}'_q$, the adversary $\mathcal{B}^0$ plays honestly, and so for such $v' \in \mathcal{V}'_q$:

$$\mathsf{out}^1_{|v'}(P_1, \mathcal{B}^1) - \mathsf{out}^1_{|v'}(P_1, P_2) = 0.$$

Now, recall that $\mathcal{V}' = \{left(v), right(v) \mid v \in \mathcal{V}\}$. At a node $v \in \mathcal{V}$, since both adversaries behave honestly, we move to the right child with probability γ^v and to the left child with probability $1 - \gamma^v$. Therefore, for every $v \in \mathcal{V}$, for every $P_2^* \in \{P_2, \mathcal{B}^0, \mathcal{B}^1\}$ and for every $b \in \{0, 1\}$ it holds that:

$$\mathsf{out}^b_{|v}(P_1, P_2^*) = \gamma^v \cdot \textsc{output}^b_{|right(v)}(P_1, P_2^*) + (1 - \gamma^v) \cdot \textsc{output}^b_{|left(v)}(P_1, P_2^*),$$

and so, for each one of the adversaries $\mathcal{B} \in \{\mathcal{B}^0, \mathcal{B}^1\}$:

$$\begin{aligned}\mathsf{out}^b_{|v}(P_1, \mathcal{B}) - \mathsf{out}^b_{|v}(P_1, P_2) &= \gamma^v \cdot \Big(\textsc{output}^b_{|right(v)}(P_1, \mathcal{B}) - \textsc{output}^b_{|right(v)}(P_1, P_2)\Big) \\ &\quad + (1 - \gamma^v) \cdot \Big(\textsc{output}^b_{|left(v)}(P_1, \mathcal{B}) - \textsc{output}^b_{|left(v)}(P_1, P_2)\Big)\end{aligned}$$

Recall that for every $v \in \mathcal{V}$, by the definition of $\mathcal{V}$ it holds that $\epsilon \leq \gamma^v \leq 1 - \epsilon$, and thus

$\gamma^v \geq \epsilon$ and $1-\gamma^v \geq \epsilon$. Combining the above, for every $v \in \mathcal{V}$ it holds that:

$$\begin{aligned}
&\Big(\mathsf{out}^1_{|v}(P_1,\mathcal{B}^1) - \mathsf{out}^1_{|v}(P_1,P_2)\Big) + \Big(\mathsf{out}^0_{|v}(P_1,\mathcal{B}^0) - \mathsf{out}^0_{|v}(P_1,P_2)\Big)\\
&\qquad = \gamma^v \cdot \left|p_{right(v)} - q_{right(v)}\right| + (1-\gamma^v)\cdot\left|p_{left(v)} - q_{left(v)}\right|\\
&\qquad \geq \epsilon \cdot \left(\left|p_{right(v)} - q_{right(v)}\right| + \left|p_{left(v)} - q_{left(v)}\right|\right)
\end{aligned}$$

and therefore we conclude:

$$\begin{aligned}
&\big(\mathsf{out}^1(P_1,\mathcal{B}^1) - \mathsf{out}^1(P_1,P_2)\big) + \big(\mathsf{out}^0(P_1,\mathcal{B}^0) - \mathsf{out}^0(P_1,P_2)\big)\\
&= \sum_{v\in\mathcal{V}} \Big[\Big(\mathsf{out}^1_{|v}(P_1,\mathcal{B}^1) - \mathsf{out}^1_{|v}(P_1,P_2)\Big) + \Big(\mathsf{out}^0_{|v}(P_1,\mathcal{B}^0) - \mathsf{out}^0_{|v}(P_1,P_2)\Big)\Big]\cdot \mathsf{reach}_v\\
&\geq \epsilon\cdot\sum_{v\in\mathcal{V}}\left(\left|p_{right(v)} - q_{right(v)}\right| + \left|p_{left(v)} - q_{left(v)}\right|\right)\cdot\mathsf{reach}_v \geq \epsilon\cdot\omega(\kappa)
\end{aligned}$$

which is non-negligible, in contradiction to out assumption that the coin-tossing protocol is secure. We therefore conclude that there exists a negligible function $\mu'(\cdot)$ such that:

$$\sum_{v\in\mathcal{V}}\left(\left|p_{right(v)} - q_{right(v)}\right| + \left|p_{left(v)} - q_{left(v)}\right|\right)\cdot \Pr\left[v \text{ is reached}\right] \leq \mu'(\kappa).$$

■

We now consider the bias of the modified fail-stop adversaries $\mathcal{A}^0,\mathcal{A}^1$ as in the proof of Theorem 4.5.1, conditioned on the event that the adversaries have reached a node $v \in \mathcal{V}$. We have 2 cases:

Case 1: $p_{right(v)} \geq p_v + \epsilon$ or $p_{left(v)} \leq p_v - \epsilon$: Observe that in the fail-stop case, since the adversary aborts in some node $v \in \mathcal{V}$, the honest party outputs 1 with probability $q_{left(v)}$ or $q_{right(v)}$ and not $p_{left(v)}$ or $p_{right(v)}$. We have that:

$$\begin{aligned}
&\mathsf{out}^1_{|v}(P_1,\mathcal{A}^1) - \mathsf{out}^1_{|v}(P_1,P_2) = (\delta^v_{\max} - \gamma^v)\cdot\left(q_{right(v)} - q_{left(v)}\right)\\
&\quad = (\delta^v_{\max} - \gamma^v)\cdot\left(q_{right(v)} - p_{right(v)} + p_{right(v)} - q_{left(v)} + p_{left(v)} - p_{left(v)}\right)\\
&\quad = (\delta^v_{\max} - \gamma^v)\cdot\left((p_{right(v)} - p_{left(v)}) + (q_{right(v)} - p_{right(v)}) + (p_{left(v)} - q_{left(v)})\right)\\
&\quad \geq (\delta^v_{\max} - \gamma^v)\cdot\left((p_{right(v)} - p_{left(v)}) - |q_{right(v)} - p_{right(v)}| - |p_{left(v)} - q_{left(v)}|\right).
\end{aligned}$$

In a similar way:

$$\begin{aligned}
&\mathsf{out}^0_{|v}(P_1,\mathcal{A}^0) - \mathsf{out}^0_{|v}(P_1,P_2) = (\gamma^v - \delta^v_{\min})\cdot\left(q_{right(v)} - q_{left(v)}\right)\\
&\quad \geq (\gamma^v - \delta^v_{\min})\cdot\left((p_{right(v)} - p_{left(v)}) - |q_{right(v)} - p_{right(v)}| - |p_{left(v)} - q_{left(v)}|\right).
\end{aligned}$$

Summing up the differences in probabilities we derive:

$$\begin{aligned}
&\Big(\mathsf{out}^1_{|v}(P_1,\mathcal{A}^1) - \mathsf{out}^1_{|v}(P_1,P_2)\Big) + \Big(\mathsf{out}^0_{|v}(P_1,\mathcal{A}^0) - \mathsf{out}^0_{|v}(P_1,P_2)\Big)\\
&\quad \geq (\delta^v_{\max} - \delta^v_{\min})\cdot\left(p_{right(v)} - p_{left(v)}\right) - (\delta^v_{\max} - \delta^v_{\min})\cdot\left(|q_{right(v)} - p_{right(v)}| + |p_{left(v)} - q_{left(v)}|\right)
\end{aligned}$$

Now, recall that in the proof of Theorem 4.5.1, we showed that $\left(p_{right(v)} - p_{left(v)}\right) \geq \epsilon$. In addition, in the case that f is not balanced, there exists a constant $c > 0$ such that $\delta^v_{\max} - \delta^v_{\min} \geq c$, and in the case that f is 1 or 0-balanced it holds that $\delta^v_{\max} - \delta^v_{\min} \geq \epsilon$ (as shown in the proof of Theorem 4.5.4). In all cases, there exists a non-negligible value Δ such that $\delta^v_{\max} - \delta^v_{\min} \geq \Delta$. Moreover, since $\delta^v_{\max}, \delta^v_{\min}$ are both probabilities it trivially holds that $\delta^v_{\max} - \delta^v_{\min} \leq 1$. Combining all the above, we have:

$$\begin{aligned}\Big(\mathsf{out}^1_{|v}(P_1, \mathcal{A}^1) - \mathsf{out}^1_{|v}(P_1, P_2)\Big) + \Big(\mathsf{out}^0_{|v}(P_1, \mathcal{A}^0) - \mathsf{out}^0_{|v}(P_1, P_2)\Big) \\ \geq \quad \Delta \cdot \epsilon - \big(|q_{right(v)} - p_{right(v)}| + |p_{left(v)} - q_{left(v)}|\big)\end{aligned}$$

Case 2: $p_{left(v)} \geq p_v + \epsilon$ or $p_{right(v)} \leq p_v - \epsilon$: With similar calculations we derive:

$$\begin{aligned}\Big(\mathsf{out}^1_{|v}(P_1, \mathcal{A}^1) - \mathsf{out}^1_{|v}(P_1, P_2)\Big) + \Big(\mathsf{out}^0_{|v}(P_1, \mathcal{A}^0) - \mathsf{out}^0_{|v}(P_1, P_2)\Big) \\ \geq \quad \Delta \cdot \epsilon - \big(|p_{left(v)} - q_{left(v)}| + |p_{right(v)} - q_{right(v)}|\big)\end{aligned}$$

where, Δ is the same as above.

Overall bias. We have computed the differences in the probability that an honest party outputs 1 when interacting with an honest P_2 or with one of the adversaries, conditioned on the event that a node $v \in \mathcal{V}$ is reached. We are now ready to compute the overall bias.

$$\begin{aligned}&\big(\mathsf{out}^1(P_1, \mathcal{A}^1) - \mathsf{out}^1(P_1, P_2)\big) + \big(\mathsf{out}^0(P_1, \mathcal{A}^0) - \mathsf{out}^0(P_1, P_2)\big) \\ &= \sum_{v \in \mathcal{V}} \mathsf{reach}_v \cdot \Big[\Big(\mathsf{out}^1_{|v}(P_1, \mathcal{A}^1) - \mathsf{out}^1_{|v}(P_1, P_2)\Big) + \Big(\mathsf{out}^0_{|v}(P_1, \mathcal{A}^1) - \mathsf{out}^0_{|v}(P_1, P_2)\Big)\Big] \\ &\geq \sum_{v \in \mathcal{V}} \mathsf{reach}_v \cdot \big[\Delta \cdot \epsilon - \big(|p_{left(v)} - q_{left(v)}| + |p_{right(v)} - q_{right(v)}|\big)\big] \\ &\geq \sum_{v \in \mathcal{V}} \mathsf{reach}_v \cdot \Delta \cdot \epsilon - \sum_{v \in \mathcal{V}} \big(|p_{left(v)} - q_{left(v)}| + |p_{right(v)} - q_{right(v)}|\big) \cdot \mathsf{reach}_v \\ &\geq \frac{1}{2} \cdot \Delta \cdot \epsilon - \mu'(\kappa)\end{aligned}$$

where the last inequality is true from Claim 4.6.5. We therefore conclude that the sum of the biases:

$$\left(\mathsf{out}^1(P_1, \mathcal{A}^1) - \frac{1}{2}\right) + \left(\mathsf{out}^0(P_1, \mathcal{A}^0) - \frac{1}{2}\right)$$

is non-negligible, as required. ■

Chapter 5

Towards Characterizing Complete Fairness

In the previous chapter, we discovered what functions are ruled out using the impossibility result of Cleve. In this chapter, we focus on the only known possibility result in complete fairness - the protocol of Gordon, Hazay, Katz and Lindell [58] - and discover what functions can be computed using this protocol. We show that many more functions can be computed fairly than what has been though previously. We also present positive results for asymmetric Boolean functions (where the parties do not necessarily receive the same output), and functions with non-binary outputs.

5.1 Introduction

The work of Gordon, Hazay, Katz and Lindell (GHKL) [58] provides a fair protocol that computes some specific non-trivial function. The work also presents a generalization of this protocol that may potentially be used to compute a large class of functions. It also shows how to construct a (rather involved) set of equations for a given function, that indicates whether the function can be computed fairly using this protocol.

In more detail, [58] considered a particular simple (but non-trivial) 3×2 function, and showed that this function can be computed fairly by constructing a protocol for this function and a simulator for this protocol. Then, in [58] it was shown that this protocol can be generalized, by identifying certain values and constants that can be parameterized, and parameterized the simulator also in a similar way. In addition, [58] showed how to examine whether the (parameterized) simulator succeeds to simulate the protocol, by constructing a set of equations, where its solution is in fact, the actual parameters that the simulator has to use. The set of equations that [58] presented is rather involved, relies heavily on the actual parameters of the real protocol rather than properties of the computed function, and in particular it is hard to decide for a given function whether it can be computed fairly using this protocol, or not.

The protocol of [58] is ground-breaking and completely changed our perception regarding fairness. The fact that *something* non-trivial can be computed fairly is surprising, and raises many interesting questions. For instance, are there many functions that can be computed fairly, or only a few? Which functions can be computed fairly? Which functions can be computed using this generalized GHKL protocol? What property distinguishes these functions from the functions that are impossible to compute fairly?

We focus on the general protocol of GHKL (or, framework), and explore which functions can be computed using this protocol. Surprisingly, it turns out that many functions can be computed fairly, in contrast to what was previously thought. In particular, we show that almost all functions with distinct domain sizes (i.e., functions $f : X \times Y \to \{0, 1\}$ with $|X| \neq |Y|$) can be computed with complete fairness using this protocol.

Intuition. We present some intuition before proceeding to our results in more detail. The most important and acute point is to understand what distinguishes functions that can be computed fairly from functions that cannot. Towards this goal, let us reconsider the impossibility result of Cleve. This result shows that fair coin-tossing is impossible by constructing concrete adversaries that *bias* and *influence* the output of the honest party in any protocol implementing coin-tossing. We believe that such adversaries can be constructed for any protocol computing any function, and not specific to coin-tossing. In any protocol, one party can better predict the outcome than the other, and abort the execution if it is not satisfied with the result. Consequently, it has a concrete ability to *influence* the output of the honest party by aborting prematurely. Of course, a fair protocol should limit and decrease this ability to the least possible, but in general, this phenomenon cannot be totally eliminated and cannot be prevented.

So if this is the case, how do fair protocols exist? The answer to this question does not lie in the real execution but rather in the ideal process: *the simulator can simulate this influence in the ideal execution.* In some sense, for some functions, the simulator has the ability to significantly influence the output of the honest party in the ideal execution and therefore the bias in the real execution is not considered a breach of security. This is due to the fact that in the malicious setting the simulator has an ability that is crucial in the context of fairness: it can *choose* what input it sends to the trusted party. Indeed, the protocol of GHKL uses this switching-input ability in the simulation, and as pointed out in Chapter 4 (Theorem 4.6.2), once we take away this advantage from the simulator – every function that contains an embedded XOR cannot be computed fairly, and fairness is almost always impossible.

Therefore, the structure of the function plays an essential role in the question of whether a function can be computed fairly or not. This is because this structure reflects the "power" and the "freedom" that the simulator has in the ideal world and how it can influence the output of the honest party. The question of whether a function can be computed fairly is related to the amount of "power" the simulator has in the ideal execution. Intuitively, the more freedom that the simulator has, the more likely that the function can be computed fairly.

A concrete example. We demonstrate this "power of the simulator" on two functions. The first is the XOR function, which is impossible to compute by a simple implication of Cleve's result. The second is the specific function for which GHKL has proved to be possible (which we call "the GHKL function"). The truth tables of the functions are given in Figure 5.1.

(a)

	y_1	y_2
x_1	0	1
x_2	1	0

(b)

	y_1	y_2
x_1	0	1
x_2	1	0
x_3	1	1

Figure 5.1: (a) The XOR function – impossible, (b) The GHKL function – possible

What is the freedom of the simulator in each case? Consider the case where P_1 is corrupted (that is, we can assume that P_1 is the first to receive an output, and thus it is "harder" to simulate). In the XOR function, let p be the probability that the simulator sends the input x_1 to the trusted party, and let $(1-p)$ be the probability that it sends x_2. Therefore, the output of P_2 in the ideal execution can be represented as $(q_1, q_2) = p \cdot (0,1) + (1-p) \cdot (1,0) = (1-p, p)$, which means that if P_2 inputs y_1, then it receives 1 with probability $1-p$, and if it uses input y_2, then it receives 1 with probability p. We call this vector *"the output distribution vector"* for P_2, and the set of all possible output distribution vectors reflects the freedom that the simulator has in the ideal execution. In the XOR function, this set is simply $\{(1-p, p) \mid 0 \leq p \leq 1\}$, which gives the simulator one degree of freedom. Any increment of the probability in the first coordinate must be balanced with an equivalent decrement in the second coordinate, and vice versa. In some sense, the output of the honest party in case its input is y_1 is correlated to its output in case of input y_2.

On the other hand, consider the case of the GHKL function. Assume that the simulator chooses x_1 with probability p_1, x_2 with probability p_2 and x_3 with probability $1-p_1-p_2$. Then, all the output vector distributions are of the form:

$$(q_1, q_2) = p_1 \cdot (0,1) + p_2 \cdot (1,0) + (1 - p_1 - p_2) \cdot (1,1) = (1 - p_1, 1 - p_2) .$$

This gives the simulator two degrees of freedom, which is significantly more power.

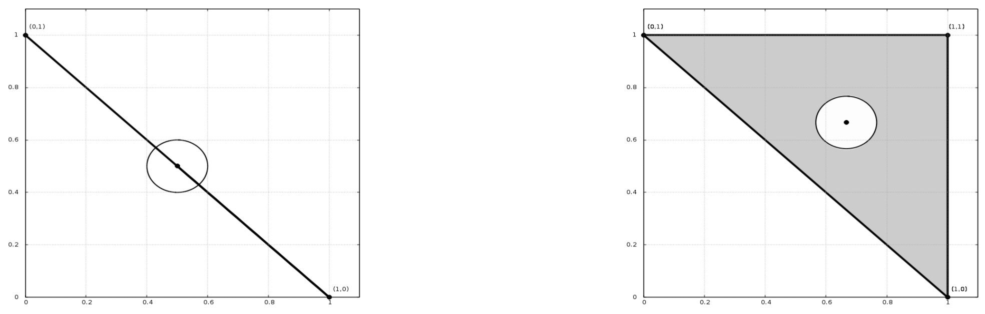
Geometrically, we can refer to the rows of the truth table as points in $\mathbb{R}^2$, and so in the XOR function we have the two points $(0,1)$ and $(1,0)$. All the output distribution vectors are of the form $p \cdot (0,1) + (1-p) \cdot (1,0)$ which is exactly the line segment between these two points (geometric object of dimension 1). In the GHKL function, all the output distribution vectors are the triangle between the points $(0,1), (1,0)$ and $(1,1)$, which is a geometric object of dimension 2 (a full dimensional object in $\mathbb{R}^2$).

The difference between these two geometric objects already gives a perception for the reason why the XOR function is impossible to compute, whereas the GHKL function is possible, as the simulator has significantly more options in the latter case. However, we provide an additional refinement. At least on the intuitive level, fix some output distribution vector of the honest party (q_1, q_2). Assume that there exists a real-world adversary that succeeds in biasing the output and obtain output distribution vector (q_1', q_2') that is at most ϵ-far from (q_1, q_2). In the case of the XOR function, this results in points that are not on the line, and therefore this adversary cannot be simulated. In contrast, in the case of the GHKL function, these points are still in the triangle, and therefore this adversary can be simulated.

In Figure 5.2, we show the geometric objects defined by the XOR and the GHKL functions. The centers of the circles are the output distribution of honest executions, and the circuits represent the possible biases in the real execution. In (a) there exist small biases that are invalid points, whereas in (b) all small biases are valid points that can be simulated.

5.1.1 Our Results

For a given function $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0,1\}$, we consider its geometric representation as the convex-hull of ℓ points over $\mathbb{R}^m$, where the jth coordinate of the ith point is simply $f(x_i, y_j)$. We say that this geometric object is of full dimension in $\mathbb{R}^m$, if it cannot be embedded in any subspace of dimension smaller than m (or, any subspace that is isomorphic to $\mathbb{R}^{m-1}$). We then prove that any function that its geometric representation is of full dimension


(a) The potential output distribution vectors of the XOR function: a line segment between $(0,1)$ and $(1,0)$.

(b) The potential output distribution vectors of the GHKL function: the triangle between $(0,1)$, $(1,0)$ and $(1,1)$.

Figure 5.2: The geometric objects defined by the XOR function (a) and the GHKL function (b).

can be computed with complete fairness. We prove the following theorem:

Theorem 5.1.1 (informal) *Let $f : X \times Y \to \{0,1\}$ be a function. Assuming the existence of an Oblivious Transfer, if the geometric object defined by f is of full-dimension, then the function can be computed with complete fairness.*

For the proof, we use the extended GHKL protocol (with some concrete set of parameters). Moreover, the proof uses tools from convex geometry. We find the connection between the problem of fairness and convex geometry very appealing.

On the other hand, we show that if the function is not full dimensional, and satisfies some additional requirements (that are almost always satisfied in functions with $|X| = |Y|$), then the function *cannot* be computed using the protocol of [58].

We then proceed to the class of asymmetric functions where the parties do not necessarily get the same output, and the class of non-binary output. Interestingly, the GHKL protocol can be extended to these classes of functions. We show:

Theorem 5.1.2 (informal) *Assuming the existence of an Oblivious Transfer:*

1. *There exists a large class of asymmetric Boolean functions that can be computed with complete fairness.*
2. *For any finite range Σ, there exists a large class of functions $f : X \times Y \to \Sigma$ that can be computed with complete-fairness.*

For the non-binary case, we provide a general criterion that holds only for functions for which $|X| > (|\Sigma|-1)\cdot|Y|$, that is, when the ratio between the domain sizes is greater than $|\Sigma|-1$. This, together with the results in the binary case, may refer to an interesting relationship between the size of the domains and possibility of fairness. This is the first time that a fair protocol is constructed for both non-binary output, and asymmetric Boolean functions. This shows that fairness is not restricted to a very specific and particular type of functions, but rather a property that under certain circumstances can be achieved. Moreover, it shows the power that is concealed in the GHKL protocol alone.

Open problems. Our work is an important step towards a full characterization of fairness of finite domain functions. The main open question is to finalize this characterization. In addition, can our results be generalized to functions with infinite domains (domains with sizes that depend on the security parameter)? Finally, in the non-binary case, we have a positive result only when the ratio between the domain sizes is greater than $|\Sigma| - 1$. A natural question is whether fairness be achieved in any other case, or for any other ratio.

5.2 Definitions and Preliminaries

We cover the necessary definitions for our analysis, and we cover the mathematical background that is needed for our results.

Notations. In most of this chapter, we consider binary deterministic functions over a finite domain; i.e., functions $f : X \times Y \to \{0,1\}$ where $X, Y \subset \{0,1\}^*$ are finite sets. Throughout the chapter, we denote $X = \{x_1, \ldots, x_\ell\}$ and $Y = \{y_1, \ldots, y_m\}$, for constants $\ell, m \in \mathbb{N}$. Let M_f be the $\ell \times m$ matrix that represents the function, i.e., a matrix whose entry position (i,j) is $f(x_i, y_j)$. For $1 \leq i \leq \ell$, let X_i denote the ith row of M_f, and for $1 \leq j \leq m$ let Y_j denote the jth column of M_f. A vector $\mathbf{p} = (p_1, \ldots, p_\ell)$ is a *probability vector* if $p_i \geq 0$ for every $1 \leq i \leq \ell$ and $\sum_{i=1}^{\ell} p_i = 1$. As a convention, we use **bold**-case letters to represent a vector (e.g., $\mathbf{p}$, $\mathbf{q}$), and sometimes we use upper-case letters (e.g., X_i, as above). We denote by $\mathbf{1}_k$ (resp. $\mathbf{0}_k$) the all one (resp. all zero) vector of size k. We work in the Euclidian space $\mathbb{R}^m$, use the Euclidian norm $||x|| = \sqrt{\langle x, x \rangle}$ and the distance function as $d(x,y) = ||x - y||$.

5.2.1 Secure Computation – Definitions

We refer the reader to Section 4.2 for definitions of computationally-indistinguishability, definition of secure computation with fairness (Definition 4.2.1), and secure computation without fairness (Definition 4.2.2). These are merely the standard definitions of [27, 53]. In addition, we use standard O notation, and let poly denote a polynomial function.

Hybrid model and composition. In Section 4.2.3 we defined the g-hybrid model, and mentioned the (sequential) composition theorem of [27]. In this Chapter, we consider the case where g is a reactive functionality, which means that the functionality g stores some state between consecutive executions. We assume that the protocol π^g that uses the functionality g does not contain any messages between the parties, and all the communication between them is performed via the functionality g. Recall that any functionality (even reactive one) can be computed with security-with-abort, and therefore there exists a protocol ρ that securely computes g (with no fairness). Since there are no messages in π^g we can apply the sequential composition theorem of [27] and obtain the following Proposition:

Proposition 5.2.1 *Let g be a reactive functionality, let π be a protocol that securely computes f with complete fairness in the g-hybrid model (where g is computed according to the ideal world with abort), and assumes that π contains no communication between the parties rather than queries of g and that a single execution of g occurs at the same time. Then, there exists a protocol Π that securely computes f with complete fairness in the plain model.*

5.2.2 Mathematical Background

Our characterization is based on the geometric representation of the function f. In the following, we provide the necessary mathematical background, and link it to the context of cryptography whenever possible. Most of the following mathematical definitions are taken from [93, 62].

Output vector distribution and convex combination. We now analyze the "power of the simulator" in the ideal execution. The following is an inherent property of the concrete function and the ideal execution, and is true for any protocol computing the function. Let $\mathcal{A}$ be an adversary that corrupts the party P_1, and assume that the simulator $\mathcal{S}$ chooses its input according to some distribution $\mathbf{p} = (p_1, \ldots, p_\ell)$. That is, the simulator sends an input x_i with probability p_i, for $1 \leq i \leq \ell$. Then, the length m vector $\mathbf{q} = (q_{y_1}, \ldots, q_{y_m}) \overset{\text{def}}{=} \mathbf{p} \cdot M_f$ represents the *output distribution vector* of the honest party P_2. That is, in case the input of P_2 is y_j for some $1 \leq j \leq m$, then it gets 1 with probability q_{y_j}.

Convex combination. The output distribution vector is in fact a convex combination of the rows $\{X_1, \ldots, X_\ell\}$ of the matrix M_f. That is, when the simulator uses $\mathbf{p}$, the output vector distribution of P_2 is:

$$\mathbf{p} \cdot M_f = (p_1, \ldots, p_\ell) \cdot M_f = p_1 \cdot X_1 + \ldots + p_\ell \cdot X_\ell \ .$$

A convex combination of points $X_1, \ldots, X_\ell$ in $\mathbb{R}^m$ is a linear combination of the points, where all the coefficients (i.e., $(p_1, \ldots, p_\ell)$) are non-negative and sum up to 1.

Convex hull. The set of all possible output distributions vectors that the simulator can produce in the ideal execution is:

$$\{\mathbf{p} \cdot M_f \mid \mathbf{p} \text{ is a probability vector}\} \ .$$

In particular, this set reflects the "freedom" that the simulator has in the ideal execution. This set is, in fact, the *convex hull* of the row vectors $X_1, \ldots, X_\ell$, and is denoted as $\mathbf{conv}(\{X_1, \ldots, X_\ell\})$. That is, for a set $S = \{X_1, \ldots, X_\ell\}$, $\mathbf{conv}(S) = \left\{\sum_{i=1}^{\ell} p_i \cdot X_i \mid 0 \leq p_i \leq 1, \sum_{i=1}^{m} p_i = 1\right\}$. The convex-hull of a set of points is a convex set, which means that for every $X, Y \in \mathbf{conv}(S)$, the line segment between X and Y also lies in $\mathbf{conv}(S)$, that is, for every $X, Y \in \mathbf{conv}(S)$ and for every $0 \leq \lambda \leq 1$, it holds that $\lambda \cdot X + (1 - \lambda) \cdot Y \in \mathbf{conv}(S)$.

Geometrically, the convex-hull of two (distinct) points in $\mathbb{R}^2$ is the line-segment that connects them. The convex-hull of three points in $\mathbb{R}^2$ may be a line (in case all the points lie on a single line), or a triangle (in case where all the points are collinear). The convex-hull of 4 points may be a line, a triangle, or a parallelogram. In general, the convex-hull of k points in $\mathbb{R}^2$ may define a convex polygon of at most k vertices. In $\mathbb{R}^3$, the convex-hull of k points can be either a line, a triangle, a tetrahedron, a parallelepiped, etc.

Affine-hull and affine independence. A subset B of $\mathbb{R}^m$ is an affine subspace if $\lambda \cdot \mathbf{a} + \mu \cdot \mathbf{b} \in B$ for every $\mathbf{a}, \mathbf{b} \in B$ and $\lambda, \mu \in \mathbb{R}$ such that $\lambda + \mu = 1$. For a set of points $S = \{X_1, \ldots, X_\ell\}$, its affine hull is defined as $\mathbf{aff}(S) = \left\{\sum_{i=1}^{\ell} \lambda_i \cdot X_i \mid \sum_{i=1}^{\ell} \lambda_i = 1\right\}$, which is similar to convex hull, but without the additional requirement for non-negative coefficients. The set of points

$X_1, \ldots, X_\ell$ in $\mathbb{R}^m$ is affinely independent if $\sum_{i=1}^{\ell} \lambda_i X_i = \mathbf{0}_m$ holds with $\sum_{i=1}^{\ell} \lambda_i = 0$ only if $\lambda_1 = \ldots = \lambda_\ell = 0$. In particular, it means that one of the points is in the affine hull of the other points. It is easy to see that the set of points $\{X_1, \ldots, X_\ell\}$ is affinely independent if and only if the set $\{X_2 - X_1, \ldots, X_\ell - X_1\}$ is a linearly independent set. As a result, any $m+2$ points in $\mathbb{R}^m$ are affine dependent, since any $m+1$ points in $\mathbb{R}^m$ are linearly dependent. In addition, it is easy to see that the points $\{X_1, \ldots, X_\ell\}$ over $\mathbb{R}^m$ is affinely independent if and only if the set of points $\{(X_1, 1), \ldots, (X_\ell, 1)\}$ over $\mathbb{R}^{m+1}$ is linearly independent.

Affine-dimension and affine-basis. If the set $S = \{X_1, \ldots, X_\ell\}$ over $\mathbb{R}^m$ is affinely independent, then $\mathbf{aff}(S)$ has dimension $\ell - 1$, and we write $\dim(\mathbf{aff}(S)) = \ell - 1$. In this case, S is the affine basis for $\mathbf{aff}(S)$. Note that an affine basis for an m-dimensional affine space has $m+1$ elements. As we will see, the affine dimension of ℓ row vectors $X_1, \ldots, X_\ell$ plays an essential role in our characterization.

Linear hyperplane. A linear hyperplane in $\mathbb{R}^m$ is a $(m-1)$-dimensional affine-subspace of $\mathbb{R}^m$. The linear hyperplane can be defined as all the points $X = (x_1, \ldots, x_m)$ which are the solutions of a linear equation:

$$a_1 x_1 + \ldots a_m x_m = b \ ,$$

for some constants $\mathbf{a} = (a_1, \ldots, a_m) \in \mathbb{R}^m$ and $b \in \mathbb{R}$. We denote this hyperplane by:

$$\mathcal{H}(\mathbf{a}, b) \stackrel{\text{def}}{=} \{X \in \mathbb{R}^m \mid \langle X, \mathbf{a} \rangle = b\} \ .$$

Throughout the chapter, for short, we will use the term hyperplane instead of linear hyperplane. It is easy to see that indeed this is an affine-subspace. In $\mathbb{R}^1$, a hyperplane is a single point, in $\mathbb{R}^2$ it is a line, in $\mathbb{R}^3$ it is a plane and so on. We remark that for any m affinely independent points in $\mathbb{R}^m$ there exists a *unique* hyperplane that contains all of them (and infinitely many in case they are not affinely independent). This is a simple generalization of the fact that for any distinct 2 points there exists a single line that passes through them, for any 3 (collinear) points there exists a single plane that contains all of them, etc.

Convex polytopes. Geometrically, a full dimensional convex polytope in $\mathbb{R}^m$ is the convex-hull of a finite set S where $\dim(\mathbf{aff}(S)) = m$. Polytopes are familiar objects: in $\mathbb{R}^2$, we get *convex polygons* (a triangle, a parallelogram etc.). In $\mathbb{R}^3$, we get *convex polyhedra* (a tetrahedron, a parallelepiped etc.). Convex polytopes play an important role in solutions of linear programming.

In addition, a special case of polytope is simplex. If the set S is affinely independent of cardinality $m+1$, then $\mathbf{conv}(S)$ is an m-dimensional *simplex* (or, m-simplex). For $m = 2$, this is simply a triangle, whereas in $m = 3$ we get a tetrahedron. A simplex in $\mathbb{R}^m$ consists of $m+1$ *facets*, which are themselves simplices of lower dimensions. For instance, a tetrahedron (which is a 3-simplex) consists of 4 facets, which are themselves triangles (2-simplex).

5.3 The Protocol of Gordon, Hazay, Katz and Lindell [58]

In the following, we give a high level overview of [58]. We also present its simulation strategy, and the set of equations that indicates whether a given function can be computed with this protocol, which is the important part for our discussion. We also generalize somewhat the protocol by adding additional parameters, which adds some flexibility to the protocol and may potentially compute more functions, and we also represent it a bit differently than the original construction, which is merely a matter of taste.

5.3.1 The Protocol

Assume the existence of an online dealer (a reactive functionality that can be replaced using standard secure computation that is secure-with-abort). The parties invoke this online-dealer and send it their respective inputs $(x, y) \in X \times Y$. The online dealer computes values $a_1, \dots, a_R$ and $b_1, \dots, b_R$ (we will see later how they are defined). In round i, the dealer sends party P_1 the value a_i and afterward it sends b_i to P_2. At each point of the execution, each party can halt the online-dealer, preventing the other party from receiving its value in that round. In such a case, the other party is instructed to halt and output the last value it has received from the dealer. For instance, if P_1 aborts in round i after it learns a_i and prevents from P_2 to learn b_i, P_2 halts and outputs b_{i-1}.

The values $(a_1, \dots, a_R), (b_1, \dots, b_R)$ are generated by the dealer in the following way: The dealer first chooses a round i^* according to geometric distribution with parameter α. In each round $i < i^*$, the parties receive bits (a_i, b_i), that depend on their respective inputs solely and uncorrelated to the input of the other party. In particular, for party P_1 the dealer computes $a_i = f(x_i, \hat{y})$ for some random $\hat{y}$, and for P_2 it computes $b_i = f(\hat{x}, y_j)$ for some random $\hat{x}$. As we will see, we will choose $\hat{y}$ uniformly from Y, whereas $\hat{x}$ will be chosen according to some distribution X_{real}, which is a parameter for the protocol[1]. For every round $i \geq i^*$, the parties receive the correct output $a_i = b_i = f(x_i, y_j)$. Note that if we set $R = \alpha^{-1} \cdot \omega(\ln \kappa)$, then $i^* < R$ with overwhelming probability, and so correctness holds.

The full specification consists of the definition of the online-dealer, which is the F_{dealer} functionality and is formally defined in Functionality 5.3.2. In addition, we present the protocol in the F_{dealer}-hybrid model in Protocol 5.3.3. Algorithm 5.3.1 is needed for both the F_{dealer} functionality and the protocol.

$\mathsf{RandOut}_1(x)$ – **Algorithm for** P_1:	$\mathsf{RandOut}_2(y)$ – **Algorithm for** P_2:
Input: An input $x \in X$.	**Input:** An input $y \in Y$.
	Parameter: Distribution X_{real}.
Output: Choose $\hat{y} \leftarrow Y$ uniformly and output $f(x, \hat{y})$.	**Output:** Choose $\hat{x} \leftarrow X$ according to distribution X_{real} and output $f(\hat{x}, y)$.

ALGORITHM 5.3.1 (Default output algorithms – $\mathsf{RandOut}_1(x)$, $\mathsf{RandOut}_2(y)$**)**

[1]This is the generalization of the protocol of GHKL that we have mentioned, which adds some flexibility, and will make the proof of security a bit simpler. We note that a similar distribution Y_{real} could have been added as well, but as we will see, this modification is not helpful and will just add complication.

FUNCTIONALITY 5.3.2 (The (reactive) online-dealer functionality – F_{dealer})

- **Inputs:** P_1 sends x as its input, Y sends y.
- **Parameters:** (α, R), where $0 < \alpha < 1$ and R is the number of rounds.
- **The functionality:**
  1. **Check inputs:** If $x \notin X$ or $y \notin Y$ then send abort to both parties and halt.
  2. **Preliminary phase:**
     1. Choose i^* according to a geometric distribution with parameter α.
     2. Define values $(a_1, \ldots, a_R)$, $(b_1, \ldots, b_R)$, where:
        (a) For $i = 1$ to $i^* - 1$: set $(a_i, b_i) = (\mathsf{RandOut}_1(x), \mathsf{RandOut}_2(y))$.
        (b) For $i = i^*$ to R: set $(a_i, b_i) = (f(x, y), f(x, y))$.
  3. **The online phase:**
     1. In every round $i = 1$ to R:
        (a) Upon receiving proceed from P_2, send a_i to P_1.
        (b) Upon receiving proceed from P_1, send b_i to P_2.

        Upon receiving abort from any one of the parties at any point of the execution, send abort to both parties and halt.

PROTOCOL 5.3.3 (Computing f in the F_{dealer}-hybrid model)

- **Inputs:** P_1 holds x, P_2 holds y.
- **Parameters:** Some value $0 < \alpha < 1$, and security parameter κ.
- **The protocol:**
  1. Set $R = \alpha^{-1} \cdot \omega(\log \kappa)$.
  2. P_1 computes $a_0 = \mathsf{RandOut}_1(x)$, P_2 computes $b_0 = \mathsf{RandOut}_2(y)$.
  3. The parties invoke the F_{dealer} functionality. P_1 sends as input x, P_2 sends y.
  4. For every round $i = 1$ to R:
     (a) P_2 sends proceed to F_{dealer}, and P_1 receives a_i from F_{dealer}. If P_1 receives abort from F_{dealer}, it halts and outputs a_{i-1}.
     (b) P_1 sends proceed to F_{dealer}, and P_2 receives b_i from F_{dealer}. If P_2 receives abort from F_{dealer}, it halts and outputs b_{i-1}.
- **Output:** If all R iterations have been run, party P_1 outputs a_R and party P_2 outputs b_R.

5.3.2 Security

Since P_2 is the second to receive an output, it is easy to simulate an adversary that corrupts P_2. If the adversary aborts before i^*, then it has not obtained any information about the input of P_1. If the adversary aborts at or after i^*, then in the real execution the honest party P_1 already receives the correct output $f(x_i, y_j)$, and fairness is obtained. Therefore, the protocol is secure with respect to corrupted P_2, *for any function* f.

The case of corrupted P_1 is more delicate, and defines some requirements from f. Intuitively, if the adversary aborts before i^*, then the outputs of both parties are uncorrelated, and no one gets any advantage. If the adversary aborts after i^*, then both parties receive the correct output and fairness is obtained. The worst case then occurs when P_1 aborts exactly in iteration i^*, as P_1 has then learned the correct value of $f(x_i, y_j)$ while P_2 has not. Since the simulator has to give P_1 the true output if it aborts at i^*, it sends the trusted party the *true* input x_i in round i^*. As a result, P_2 in the ideal execution learns the correct output $f(x_i, y_j)$ in round i^*, unlike the real execution where it outputs a random value $f(\hat{x}, y_j)$. In [58] , this problem is overcome in a very elegant way: in order to balance this advantage of the honest party in the ideal execution in case the adversary aborts at i^*, the simulator chooses a random value $\hat{x}$ *different* from the way it is chosen in the real execution in case the adversary abort *before* i^* (that is, according to a different distribution than the one the dealer uses in the real execution). The calculations show that, overall, the output distribution of the honest party is distributed identically in the real and ideal executions. This balancing is possible only sometimes, and depends on the actual function f that is being evaluated.

In more detail, in the real execution the dealer before i^* chooses b_i as $f(\hat{x}, y_j)$, where $\hat{x}$ is chosen according to some distribution X_{real}. In the ideal execution, in case the adversary sends x to the simulated online-dealer, aborts in round $i < i^*$ upon viewing some a_i, the simulator chooses the input $\tilde{x}$ it sends to the trusted party according to distribution X_{ideal}^{x,a_i}. Then, define $Q^{x,a_i} = X_{ideal}^{x,a_i} \cdot M_f$, the output distribution vector of the honest party P_2 in this case. In fact, the protocol and the simulation define the output distribution vectors Q^{x,a_i}, and simulation is possible only if the corresponding X_{ideal}^{x,a_i} exists, which depends on the function f being computed.

We now present the exact requirements from the output distribution vectors Q^{x,a_i}. In the proof sketch below, we do not present the simulator nor a full proof of security; we just give a perception for the reason why the vectors Q^{x,a_i} are defined in such a way. The full proof can be found in Appendix C and [58].

The output distributions vectors $\boldsymbol{Q^{x,a}}$. Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0,1\}$. Fix X_{real}, and let U_Y denote the uniform distribution over Y. For every $x \in X$, denote by p_x the probability that $a_i = 1$ before i^*. Similarly, for every $y_j \in Y$, let p_{y_j} denote the probability $b_i = 1$ before i^*. That is: $p_x \stackrel{\text{def}}{=} \Pr_{\hat{y} \leftarrow U_Y}[f(x,\hat{y}) = 1]$, and $p_{y_j} \stackrel{\text{def}}{=} \Pr_{\hat{x} \leftarrow X_{real}}[f(\hat{x}, y_j) = 1]$. For every $x \in X$, $a \in \{0,1\}$, define the row vectors $Q^{x,a} = (q_{y_1}^{x,a}, \ldots, q_{y_m}^{x,a})$ indexed by $y_j \in Y$ as follows:

$$q_{y_j}^{x,0} \stackrel{\text{def}}{=} \begin{cases} p_{y_j} & \text{if } f(x,y_j) = 1 \\ p_{y_j} + \frac{\alpha \cdot p_{y_j}}{(1-\alpha)\cdot(1-p_x)} & \text{if } f(x,y_j) = 0 \end{cases}, \quad q_{y_j}^{x,1} \stackrel{\text{def}}{=} \begin{cases} p_{y_j} + \frac{\alpha \cdot (p_{y_j} - 1)}{(1-\alpha)\cdot p_x} & \text{if } f(x,y_j) = 1 \\ p_{y_j} & \text{if } f(x,y_j) = 0 \end{cases} \tag{5.3.1}$$

If for every $x \in X, a \in \{0,1\}$, there exists a probability vector $X_{ideal}^{x,a}$ such that $X_{ideal}^{x,a} \cdot M_f = Q^{x,a}$ then the simulator succeeds to simulate the protocol. We therefore have the following theorem:

Theorem 5.3.4 *Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0,1\}$ and let M_f be as above. If there exist probability vector X_{real} and a parameter $0 < \alpha < 1$ (where $\alpha^{-1} \in O(\mathsf{poly}(\kappa))$), such that for every $x \in X$, $a \in \{0,1\}$, there exists a probability vector $X_{ideal}^{x,a}$ for which:*

$$X_{ideal}^{x,a} \cdot M_f = Q^{x,a} \ ,$$

then the protocol securely computes f with complete fairness.

Proof Sketch: The full proof, including the case where P_2 is corrupted, appears in Appendix C. Here, we consider the case where P_1 is corrupted. Let $\mathcal{A}$ be the adversary that corrupts it, and let $\mathcal{S}$ be the simulator mentioned above. Let x be the input the adversary sends to the simulated F_{dealer} functionality. We recall that in case the adversary aborts exactly at i^*, the simulator

$\mathcal{S}$ sends the input x to the trusted party, and so both parties receive $f(x,y)$, unlike the real execution. Moreover, in case the adversary has aborted in round $i < i^*$, upon viewing a in round i, the simulator $\mathcal{S}$ chooses input $\hat{x}$ according to distribution $X_{ideal}^{x,a}$. The full specification of the simulator appears in the full proof.

We show that the joint distribution of the view of the adversary and the output of the honest party is distribution identically in the hybrid and the ideal executions. This is done easily in the case where the adversary aborts some round $i > i^*$ (and thus, both parties receive the correct output $f(x,y)$), and is given in the full proof of this theorem. Now, we consider the case where $i \leq i^*$. In the full proof, we show that the view of the adversary until round i (i.e., of the first $i-1$ rounds) is distributed identically in both executions. Thus, all that is left to show is that the view of the adversary in the last round and the output of the honest party are distributed identically in both executions, and this is what we show here in this proof sketch. That is, we show that for every $(a,b) \in \{0,1\}^2$,

$$\begin{aligned}\Pr\left[(\text{VIEW}^i_{\mathsf{hyb}}, \text{OUTPUT}_{\mathsf{hyb}}) = (a,b) \ \mid \ i \leq i^*\right] \\ = \ \Pr\left[(\text{VIEW}^i_{\mathsf{ideal}}, \text{OUTPUT}_{\mathsf{ideal}}) = (a,b) \ \mid \ i \leq i^*\right] \qquad (5.3.2)\end{aligned}$$

where $(\text{VIEW}^i_{\mathsf{hyb}}, \text{OUTPUT}_{\mathsf{hyb}})$ denotes the view of the adversary in round i (i.e., the round where it has aborted), and the output of the honest party in the hybrid execution. $\text{VIEW}^i_{\mathsf{ideal}}, \text{OUTPUT}_{\mathsf{ideal}})$ denote the outputs in the ideal execution.

We now show that these probabilities are equal. First, observe that:

$$\Pr\left[i = i^* \mid i \leq i^*\right] = \alpha \qquad \text{and} \qquad \Pr\left[i < i^* \mid i \leq i^*\right] = 1 - \alpha \ .$$

We now show that Eq. (5.3.2) holds, by considering all possible values for (a,b).

Case $f(x,y) = 0$. We compute the probability that the outputs of the parties are $(0,0)$ in the real execution, when the adversary $\mathcal{A}$ aborts in round $i \leq i^*$. With probability α, we get that $i = i^*$. In this case, the output of the adversary P_1 is always the correct output 0. On the other hand, the output of the honest party P_2 is chosen independently according to $\mathsf{RandOut}_2(y)$. With probability $1-\alpha$, we have that $i < i^*$. In this case *both* parties get uncorrelated values, chosen according to $\mathsf{RandOut}_1(x), \mathsf{RandOut}_2(y)$. Thus, the probability that we get $(0,0)$ is:

$$\Pr\left[(\text{VIEW}^i_{\mathsf{hyb}}, \text{OUTPUT}_{\mathsf{hyb}}) = (0,0) \ \mid \ i \leq i^*\right] = \alpha \cdot 1 \cdot (1-p_y) + (1-\alpha) \cdot (1-p_x) \cdot (1-p_y) \ .$$

On the other hand, in the ideal execution, with probability α, *both* parties receive the true output $f(x,y)$. With probability $1-\alpha$, we have that $i < i^*$. Thus, the simulator sends the adversary the value 0 with probability $(1-p_x)$, and in case it aborts, it chooses the input $\hat{x}$ according to distribution $X_{ideal}^{x,0}$. Since $Q^{x,0} = X_{ideal}^{x,0} \cdot M_f$, the probability that the honest party receives 0 is exactly $1 - q_y^{x,0}$. We therefore have that:

$$\Pr\left[(\text{VIEW}^i_{\mathsf{ideal}}, \text{OUTPUT}_{\mathsf{ideal}}) = (a,b) \ \mid \ i \leq i^*\right] = \alpha + (1-\alpha) \cdot (1-p_x) \cdot (1-q_y^{x,0}) \ .$$

Similarly, we compute the above for all possible outputs $(a,b) \in \{0,1\}^2$ and obtain:

output (a,b)	real	ideal
$(0,0)$	$(1-\alpha)\cdot(1-p_x)\cdot(1-p_y)+\alpha\cdot(1-p_y)$	$(1-\alpha)\cdot(1-p_x)\cdot(1-q_y^{x,0})+\alpha$
$(0,1)$	$(1-\alpha)\cdot(1-p_x)\cdot p_y+\alpha\cdot p_y$	$(1-\alpha)\cdot(1-p_x)\cdot q_y^{x,0}$
$(1,0)$	$(1-\alpha)\cdot p_x\cdot(1-p_y)$	$(1-\alpha)\cdot p_x\cdot(1-q_y^{x,1})$
$(1,1)$	$(1-\alpha)\cdot p_x\cdot p_y$	$(1-\alpha)\cdot p_x\cdot q_y^{x,1}$

Therefore, we get the following constraints:

$$q_y^{x,0}=p_y+\frac{\alpha\cdot p_y}{(1-\alpha)\cdot(1-p_x)} \quad \text{and} \quad q_y^{x,1}=p_y\ ,$$

which are satisfied according to our assumption in the theorem.

Case $f(x,y)=1$. This is similar to the previous case. We have:

output (a,b)	real	ideal
$(0,0)$	$(1-\alpha)\cdot(1-p_x)\cdot(1-p_y)$	$(1-\alpha)\cdot(1-p_x)\cdot(1-q_y^{x,0})$
$(0,1)$	$(1-\alpha)\cdot(1-p_x)\cdot p_y$	$(1-\alpha)\cdot(1-p_x)\cdot q_y^{x,0}$
$(1,0)$	$(1-\alpha)\cdot p_x\cdot(1-p_y)+\alpha\cdot(1-p_y)$	$(1-\alpha)\cdot p_x\cdot(1-q_y^{x,1})$
$(1,1)$	$(1-\alpha)\cdot p_x\cdot p_y+\alpha\cdot p_y$	$(1-\alpha)\cdot p_x\cdot q_y^{x,1}+\alpha$

and we again get the following constraints:

$$q_y^{x,0}=p_y \quad \text{and} \quad q_y^{x,1}=p_y+\frac{\alpha\cdot(p_y-1)}{(1-\alpha)\cdot p_x}\ .$$

However, since $X_{ideal}^{x,a}\cdot M_f=Q^{x,a}$, the above constraints are satisfied. ■

An alternative formulation for Theorem 5.3.4, is to require that for every x,a, the points $Q^{x,a}$ be in $\mathbf{conv}(\{X_1,\ldots,X_\ell\})$, where X_i is the ith row of M_f. Moreover, observe that in order to decide whether a function can be computed using the protocol, there are 2ℓ linear systems that should be satisfied, with m constraints each, and with $2\ell^2$ variables overall. This criterion depends heavily on some parameters of the protocols (like p_{x_i},p_{y_j}) rather than on properties of the function. We are interested in a simpler and easier way to validate criterion.

5.4 Our Criteria

5.4.1 Possibility of Full-Dimensional Functions

In this section, we show that any function that defines a full-dimensional geometric object can be computed using the protocol of [58]. The formal definition for this notion is as follows:

Definition 5.4.1 (full-dimensional function) *Let $f:\{x_1,\ldots,x_\ell\}\times\{y_1,\ldots,y_m\}\to\{0,1\}$ be a function, and let $X_1,\ldots,X_\ell$ be the ℓ rows of M_f over $\mathbb{R}^m$. If $\dim(\mathbf{aff}(\{X_1,\ldots,X_\ell\}))=m$, then we say that f is* a full-dimensional function.

Recall that for a set of points $S=\{X_1,\ldots,X_\ell\}\in\mathbb{R}^m$, if $\dim(\mathbf{aff}(S))=m$ then the convex-hull of the points defines a full-dimensional convex polytope. Thus, intuitively, the simulator has enough power to simulate the protocol. Recall that a basis for an affine space of dimension m has cardinality $m+1$, and thus we must have that $\ell > m$. Thus, we assume without loss of generality that this holds (and consider the transposed function $f^T : \{y_1, \ldots, y_m\} \times \{x_1, \ldots, x_\ell\} \to \{0, 1\}$, defined as $f^T(y, x) = f(x, y)$, otherwise). Overall, our property inherently holds only if $\ell \neq m$.

Alternative Representation

Before we prove that any full-dimensional function can be computed fairly, we give a different representation for this definition. This strengthens our understanding of this property, and is also related to the balanced property defined in Chapter 4. We have:

Claim 5.4.2 *Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0, 1\}$ be a function, let M_f be as above and let $S = \{X_1, \ldots, X_\ell\}$ be the rows of M_f (ℓ points in $\mathbb{R}^m$). The following are equivalent:*

1. The function is right-unbalanced with respect to arbitrary vectors.
That is, for every non-zero $\mathbf{q} \in \mathbb{R}^m$ and any $\delta \in \mathbb{R}$ it holds that: $M_f \cdot \mathbf{q}^T \neq \delta \cdot \mathbf{1}_\ell$.

2. The rows of the matrix do not lie on the same hyperplane.
That is, for every non-zero $\mathbf{q} \in \mathbb{R}^m$ and any $\delta \in \mathbb{R}$, there exists a point X_i such that $X_i \notin \mathcal{H}(\mathbf{q}, \delta)$. Alteratively, $\mathbf{conv}(\{X_1, \ldots, X_\ell\}) \not\subseteq \mathcal{H}(\mathbf{q}, \delta)$.

3. The function is full-dimensional.
There exists a subset of $\{X_1, \ldots, X_\ell\}$ of cardinality $m + 1$, that is affinely independent. *Thus,* $\dim(\mathbf{aff}(\{X_1, \ldots, X_\ell\})) = m$.

Proof:
<u>**¬1 ⇔ ¬2**</u>: By contradiction, if there exists $\mathbf{q}, \delta$ such that $M_f \cdot \mathbf{q}^T = \delta \cdot \mathbf{1}_\ell$, then for every row of the matrix M_f, it holds that $\langle X_i, \mathbf{q}\rangle = \delta$ and so $X_i \in \mathcal{H}(\mathbf{q}, \delta)$. This also implies that for any point in $\mathbf{conv}(\{X_1, \ldots, X_\ell\})$, the point is in $\mathcal{H}(\mathbf{q}, \delta)$. This is because any point in $\mathbf{conv}(\{X_1, \ldots, X_\ell\})$ can be represented as $\mathbf{a} \cdot M_f$ where $\mathbf{a}$ is a probability vector, and thus we have that

$$\langle \mathbf{a} \cdot M_f, \mathbf{q}\rangle = \mathbf{a} \cdot M_f \cdot \mathbf{q}^T = \mathbf{a} \cdot \delta \cdot \mathbf{1}_\ell = \delta \ ,$$

since $\mathbf{a}$ is a probability vector and sum-up to 1. We therefore have that $\mathbf{conv}(\{X_1, \ldots, X_\ell\}) \subseteq \mathcal{H}(\mathbf{q}, \delta)$. For the reverse direction, if such $\mathcal{H}(\mathbf{q}, \delta)$ exists that contains all the rows of the matrix, then clearly $M_f \cdot \mathbf{q}^T = \delta \cdot \mathbf{1}_\ell^T$ in contradiction.

<u>**3 ⇒ 2**</u>: Since the affine dimension of $\{X_1, \ldots, X_\ell\}$ is m, then it cannot lie in a single hyperplane (an affine subspace of dimension $m - 1$). Thus, this implication trivially holds.

<u>**2 ⇒ 3**</u>: We first claim that there exists a subset of $m+1$ points $S' = \{\hat{X}_1, \ldots, \hat{X}_{m+1}\}$ that does not lie on the same hyperplane, for any hyperplane. This set can be found iteratively, where we start with a single point, and add some other distinct point (such must exist otherwise we found a hyperplane that contains all the points). Then we look for another point that does not lie on the same line that is defined by the 2 points that we have (again, such must exist from the same reason as above). We then look for a fourth point that does not lie on the same plane that contains all the three points that we have, and so on. At the end of this process, we get the set $S' = \{\hat{X}_1, \ldots, \hat{X}_{m+1}\}$, which are $m + 1$ points that do not lie on the same hyperplane, for any hyperplane. We now claim that S' is affinely independent. Take the first m points, which define

some hyperplane $\mathcal{H}(\mathbf{q}, \delta)$. Since $X_{m+1} \notin \mathcal{H}(\mathbf{q}, \delta)$ and since $\mathcal{H}(\mathbf{q}, \delta) = \mathbf{aff}(\{X_1, \ldots, X_m\})$ then $X_{m+1} \notin \mathbf{aff}(\{X_1, \ldots, X_m\})$, and therefore the points $X_1, \ldots, X_{m+1}$ are affine independent.

■

From Alternative 1, checking whether a function is full-dimensional can be done very efficiently. Giving that $\ell > m$, all we have to do is to verify that the only possible solution $\mathbf{q}$ to the linear system $M_f \cdot \mathbf{q}^T = \mathbf{0}_\ell^T$ is the trivial one, and the there is no solution $\mathbf{q}$ to the linear system $M_f \cdot \mathbf{q}^T = \mathbf{1}_\ell^T$. This implies that the function is unbalanced for every $\delta \in \mathbb{R}$.

The Proof of Possibility

We now show that any function that is full-dimensional can be computed with complete fairness, using the protocol of [58]. The proof for this Theorem is geometrical. Recall that by Theorem 5.3.4, we need to show that there exists a solution for some set of equations. In our proof here, we show that such a solution exists without solving the equations explicitly. We show that all the points $Q^{x,a}$ that the simulator needs (by Theorem 5.3.4) are in the convex-hull of the rows $\{X_1, \ldots, X_\ell\}$, and therefore there exist probability vectors $X_{ideal}^{x,a}$ as required. We show this in two steps. First, we show that all the points are very "close" to some point $\mathbf{c}$, and therefore, all the points are inside the Euclidian ball centered at $\mathbf{c}$ for some small radius ϵ (defined as $B(\mathbf{c}, \epsilon) \stackrel{\text{def}}{=} \{Z \in \mathbb{R}^m \mid d(Z, \mathbf{c}) \leq \epsilon\}$). Second, we show that this whole ball is embedded inside the convex-polytope that is defined by the rows of the function, which implies that all the points $Q^{x,a}$ are in the convex-hull and simulation is possible.

In more detail, fix some distribution X_{real} for which the point $\mathbf{c} = (p_{y_1}, \ldots, p_{y_m}) = X_{real} \cdot M_f$ is inside the convex-hull of the matrix. Then we observe that by adjusting α, all the points $Q^{x,a}$ that we need are very "close" to this point $\mathbf{c}$. This is because each coordinate $q_{y_j}^{x,a}$ is exactly p_{y_j} plus some term that is multiplied by $\alpha/(1-\alpha)$, and therefore we can control its distance from p_{y_j} (see Eq. (5.3.1)). In particular, if we choose $\alpha = 1/\ln\kappa$, then for all sufficiently large κ's the distance between $Q^{x,a}$ and $\mathbf{c}$ is smaller than any constant. Still, for $\alpha = 1/\ln\kappa$, the number of rounds of the protocol is $R = \alpha^{-1} \cdot \omega(\ln\kappa) = \ln\kappa \cdot \omega(\ln\kappa)$, and thus asymptotically remains unchanged.

All the points $Q^{x,a}$ are close to the point $\mathbf{c}$. This implies that they all lie in the m-dimensional Euclidian ball of some constant radius $\epsilon > 0$ centered at $\mathbf{c}$. Moreover, since the function is of full-dimension, the convex-hull of the function defines a full-dimensional convex polytope, and therefore this ball is embedded in this polytope. We prove this by showing that the center of the ball $\mathbf{c}$ is "far" from each facet of the polytope, using the separation theorems of closed convex sets. As a result, all the points that are "close" to $\mathbf{c}$ (i.e., our ball) are still "far" from each facet of the polytope, and thus they are inside it. As an illustration, consider again the case of the GHKL function in Figure 5.2, in the Introduction to t his chapter (where here, the circle represents the ball and the center of the circle is the point $\mathbf{c}$). We conclude that all the points that the simulator needs are in the convex-hull of the function, and therefore the protocol can be simulated.

Before we proceed to the full proof formally, we give an additional definition and an important claim that will be helpful for our proof. For a set $F \subseteq \mathbb{R}^m$ and a point $\mathbf{p} \in \mathbb{R}^m$, we define the distance between $\mathbf{p}$ and F to be the minimal distance between $\mathbf{p}$ and a point in F, that is: $d(\mathbf{p}, F) = \min\{d(\mathbf{p}, \mathbf{f}) \mid \mathbf{f} \in F\}$. The following claim states that if a point is not on a closed convex set, then there exists a constant distance between the point and the convex set. We use this claim to show that the point $\mathbf{c}$ is far enough from each one of the facets of the polytope (and therefore the ball centered in $\mathbf{c}$ is in the convex). This Claim is a simple implication of the separation theorems for convex sets, see [93]. We have:

Claim 5.4.3 *Let $\mathcal{C}$ be a closed convex subset of $\mathbb{R}^m$, and let $\mathbf{a} \in \mathbb{R}^m$ such that $\mathbf{a} \notin \mathcal{C}$. Then there exists a constant $\epsilon > 0$ such that $d(\mathbf{a}, \mathcal{C}) > \epsilon$ (that is, for every $Z \in \mathcal{C}$ it holds that $d(\mathbf{a}, Z) > \epsilon$).*

We are now ready for the main theorem of this section:

Theorem 5.4.4 *Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0, 1\}$ be a Boolean function. If f is of full-dimension, then f can be computed with complete fairness.*

Proof: Since f is full-dimensional, there exists a subset of $m+1$ rows that is affinely independent. Let $S' = \{X_1, \ldots, X_{m+1}\}$ be this subset of rows. We now choose parameters for the GHKL protocol, such that $\mathbf{c}$ will be inside the simplex that is defined by S'. For this, we can simply define X_{real} to be the uniform distribution over S' (the ith position of X_{real} is 0 if $X_i \notin S'$, and $1/(m+1)$ if $X_i \in S'$). We also set $\alpha = 1/\ln \kappa$, and these are all the parameters that are needed for the real protocol.

Let $\mathbf{c} = (p_{y_1}, \ldots, p_{y_m}) = X_{real} \cdot M_f$, the output distribution vector that is correspond to X_{real}. Consider the set of points $\{Q^{x,a}\}_{x \in X, a \in \{0,1\}}$ that are needed for the simulation as in Eq. (5.3.1). The next claim shows that all these points are close to $\mathbf{c}$, and in the m-dimensional ball $B(\mathbf{c}, \epsilon)$ for some small $\epsilon > 0$. That is:

Claim 5.4.5 *For every constant $\epsilon > 0$, for every $x \in X, a \in \{0,1\}$, and for all sufficiently large κ's it holds that:*

$$Q^{x,a} \in B(\mathbf{c}, \epsilon) \ .$$

Proof: Fix ϵ. Since $\alpha = 1/\ln \kappa$, for every constant $\delta > 0$ and for all sufficiently large κ's, it holds that $\alpha/(1-\alpha) < \delta$. We show that for every x, a, it holds that $d(Q^{x,a}, \mathbf{c}) \leq \epsilon$, and thus $Q^{x,a} \in B(\mathbf{c}, \epsilon)$.

Recall the definition of $Q^{x,0}$ as in Eq. (5.3.1): If $f(x, y_j) = 1$ then $q^0_{y_j} = p_{y_j}$, and thus $|p_{y_j} - q^0_{y_j}| = 0$. In case $f(x, y_j) = 1$, for $\delta = \epsilon(1-p_x)/\sqrt{m}$ and for all sufficiently large κ's, it holds that:

$$\left| p_{y_j} - q^{x,0}_{y_j} \right| = \left| p_{y_j} - p_{y_j} - \frac{\alpha}{1-\alpha} \cdot \frac{p_{y_j}}{(1-p_x)} \right| \leq \frac{\alpha}{1-\alpha} \cdot \frac{1}{(1-p_x)} \leq \frac{\delta}{(1-p_x)} = \frac{\epsilon}{\sqrt{m}} \ .$$

Therefore, for all sufficiently large κ's, $\left| p_{y_j} - q^{x,0}_{y_j} \right| \leq \epsilon/\sqrt{m}$ irrespectively to whether $f(x, y_j)$ is 1 or 0. Similarly, for all sufficiently large κ's, it holds that: $\left| p_{y_j} - q^{x,1}_{y_j} \right| \leq \epsilon/\sqrt{m}$. Overall, for every $x \in X$, $a \in \{0,1\}$ we have that the distance between the points $Q^{x,a}$ and $\mathbf{c}$ is:

$$d(Q^{x,a}, \mathbf{c}) = \sqrt{\sum_{j=1}^{m} \left(q^{x,b}_{y_j} - p_{y_j} \right)^2} \leq \sqrt{\sum_{j=1}^{m} \left(\frac{\epsilon}{\sqrt{m}} \right)^2} \leq \epsilon \ ,$$

and therefore $Q^{x,a} \in B(\mathbf{c}, \epsilon)$. ∎

We now show that this whole ball is embedded inside the simplex of S'. That is:

Claim 5.4.6 *There exists a constant $\epsilon > 0$ for which $B(\mathbf{c}, \epsilon) \subset \mathbf{conv}(S')$.*

Proof: Since $S' = \{X_1, \ldots, X_{m+1}\}$ is an affinely independent set of cardinality $m+1$, $\mathbf{conv}(S')$ is a simplex. Recall that $\mathbf{c}$ is a point in the simplex (since it assigns 0 to any row that is not in S'), and so $\mathbf{c} \in \mathbf{conv}(S')$. We now show that for every *facet* of the simplex, there exists a constant distance between the point $\mathbf{c}$ and the facet. Therefore, there exists a small ball around $\mathbf{c}$ that is "far" from each facet of the simplex, and inside the simplex.

For every $1 \leq i \leq m+1$, the ith facet of the simplex is the set $F_i = \mathbf{conv}(S' \setminus \{X_i\})$, i.e., the convex set of the vertices of the simplex without the vertex X_i. We now show that $\mathbf{c} \notin F_i$, and therefore, using Claim 5.4.3, $\mathbf{c}$ is ϵ-far from F_i, for some small $\epsilon > 0$.

In order to show that $\mathbf{c} \notin F_i$, we show that $\mathbf{c} \notin \mathcal{H}(\mathbf{q}, \delta)$, where $\mathcal{H}(\mathbf{q}, \delta)$ is a hyperplane that contains F_i. That is, let $\mathcal{H}(\mathbf{q}, \delta)$ be the unique hyperplane that contains all the points $S' \setminus \{X_i\}$ (these are m affinely independent points and therefore there is a unique hyperplane that contains all of them). Recall that $X_i \notin \mathcal{H}(\mathbf{q}, \delta)$ (otherwise, S' is affinely dependent). Observe that $F_i = \mathbf{conv}(S' \setminus \{X_i\}) \subset \mathcal{H}(\mathbf{q}, \delta)$, since each point X_i is in the hyperplane, and the hyperplane is an affine set. We now show that since $X_i \notin \mathcal{H}(\mathbf{q}, \delta)$, then $\mathbf{c} \notin \mathcal{H}(\mathbf{q}, \delta)$ and therefore $\mathbf{c} \notin F_i$.

Assume by contradiction that $\mathbf{c} \in \mathcal{H}(\mathbf{q}, \delta)$. We can write:

$$\delta = \langle \mathbf{c}, \mathbf{q} \rangle = \Big\langle \sum_{j=1}^{m+1} \frac{1}{m+1} \cdot X_j, \mathbf{q} \Big\rangle = \frac{1}{m+1} \langle X_i, \mathbf{q} \rangle + \frac{1}{m+1} \sum_{j \neq i} \langle X_j, \mathbf{q} \rangle = \frac{1}{m+1} \langle X_i, \mathbf{q} \rangle + \frac{m}{m+1} \cdot \delta$$

and so, $\langle X_i, \mathbf{q} \rangle = \delta$, which implies that $X_i \in \mathcal{H}(\mathbf{q}, \delta)$, in contradiction.

Since $\mathbf{c} \notin F_i$, and since F_i is a closed[2] convex, we can apply Claim 5.4.3 to get the existence of a constant $\epsilon_i > 0$ such that $d(\mathbf{c}, F_i) > \epsilon_i$.

Now, let $F_1, \ldots, F_{m+1}$ be the facets of the simplex. We get the existence of $\epsilon_1, \ldots, \epsilon_{m+1}$ for each facet as above. Let $\epsilon = \min\{\epsilon_1, \ldots, \epsilon_{m+1}\}/2$, and so for every i, we have: $d(\mathbf{c}, F_i) > 2\epsilon$.

Consider the ball $B(\mathbf{c}, \epsilon)$. We show that any point in this ball is of distance at least ϵ from each facet F_i. Formally, for every $\mathbf{b} \in B(\mathbf{c}, \epsilon)$, for every facet F_i it holds that: $d(\mathbf{b}, F_i) > \epsilon$. This can be easily derived from the triangle inequality, where for every $\mathbf{b} \in B(\mathbf{c}, \epsilon/2)$:

$$d(\mathbf{c}, \mathbf{b}) + d(\mathbf{b}, F_i) \geq d(\mathbf{c}, F_i) > 2\epsilon \ ,$$

and so $d(\mathbf{b}, F_i) > \epsilon$ since $d(\mathbf{b}, \mathbf{c}) \leq \epsilon$.

Overall, all the points $\mathbf{b} \in B(\mathbf{c}, \epsilon)$ are of distance at least ϵ from each facet of the simplex, and inside the simplex. This shows that $B(\mathbf{c}, \epsilon) \subset \mathbf{conv}(S')$. ■

In conclusion, there exists a constant $\epsilon > 0$ for which it holds that $B(\mathbf{c}, \epsilon) \subset \mathbf{conv}(S') \subseteq \mathbf{conv}(\{X_1, \ldots, X_\ell\})$. Moreover, for all $x \in X, a \in \{0,1\}$ and for all sufficiently large κ's, it holds that $Q^{x,a} \in B(\mathbf{c}, \epsilon)$. Therefore, the requirements of Theorem 5.3.4 are satisfied, and the protocol securely computes f with complete fairness. ■

On the Number of Full-Dimensional Functions

We count the number of functions that are full dimensional. Recall that a function with $|X| = |Y|$ cannot be full-dimensional, and we consider only functions where $|X| \neq |Y|$. Interest-

[2]The convex-hull of a finite set S of vectors in $\mathbb{R}^m$ is a compact set, and therefore is closed (see [93, Theorem 15.4]).

ingly, the probability that a random function with distinct domain sizes is full-dimensional tends to 1 when $|X|, |Y|$ grow. Thus, almost always, a random function with distinct domain sizes can be computed with complete fairness(!). The answer to the frequency of full-dimensional functions within the class of Boolean functions with distinct sizes relates to a beautiful problem in combinatorics and linear algebra, that has received careful attention: Estimating the probability that a random Boolean matrix of size $m \times m$ is singular. Denote this probability by P_m. The answer to our question is $1 - P_m$, and is even larger when the difference between $|X|$ and $|Y|$ increases (see Claim 5.4.7 below).

The value of P_m is conjectured to be $(1/2 + o(1))^m$. Recent results [72, 68, 98] are getting closer to this conjecture by showing that $P_m \leq (1/\sqrt{2} + o(1))^m$, which is roughly the probability to have two identical or compliments rows or columns. Observe that this is a negligible function. Since our results hold only for the case of *finite* domain, it is noteworthy that P_m is small already for very small dimensions m. For instance, $P_{10} < 0.29$, $P_{15} < 0.047$ and $P_{30} < 1.6 \cdot 10^{-6}$ (and so $> 99.999\%$ of the 31×30 functions can be computed fairly). See more experimental results in [96]. The following Claim is based on [101, Corollary 14]:

Claim 5.4.7 *With a probability that tends to* 1 *when* $|X|, |Y|$ *grow, a random function with* $|X| \neq |Y|$ *is full-dimensional.*

Proof: Our question is equivalent to the following: What is the probability that the convex-hull of $m + 1$ (or even more) random 0/1-points in $\mathbb{R}^m$ is an m-dimensional simplex?

Recall that P_m denotes the probability that a random m vectors of size m are linearly dependent. Then, the answer to our question is simply $1 - P_m$. This is because with very high probability our $m + 1$ points will be distinct, we can choose the first point X_1 arbitrarily, and the rest of the points $S = \{X_2, \ldots, X_{m+1}\}$ uniformly at random. With probability $1 - P_m$, the set S is linearly independent, and so it linearly spans X_1. It is easy to see that this implies that $\{X_2 - X_1, \ldots, X_{m+1} - X_1\}$ is a linearly independent set, and thus $\{X_1, \ldots, X_{m+1}\}$ is affinely-independent set. Overall, a random set $\{X_1, \ldots, X_{m+1}\}$ is affinely independent with probability $1 - P_m$. ■

5.4.2 Functions that Are Not Full-Dimensional

A Negative Result

We now consider the case where the functions are not full-dimensional. This includes the limited number of functions for which $|X| \neq |Y|$, and *all* functions with $|X| = |Y|$. In particular, for a function that is not full-dimensional, all the rows of the function lie in some hyperplane (a $(m-1)$-dimensional subspace of $\mathbb{R}^m$), and all the columns of the matrix lie in a different hyperplane (in $\mathbb{R}^\ell$). We show that under additional requirements, the protocol of [58] cannot be simulated for any choice of parameters, with respect to the specific simulation strategy defined in the proof of Theorem 5.3.4.

We have the following Theorem:

Theorem 5.4.8 *Let* $f, M_f, \{X_1, \ldots, X_\ell\}$ *be as above, and let* $\{Y_1, \ldots, Y_m\}$ *be the columns of* M_f. *Assume that the function is not full-dimensional, that is, there exist non-zero* $\mathbf{p} \in \mathbb{R}^\ell$, $\mathbf{q} \in \mathbb{R}^m$ *and some* $\delta_1, \delta_2 \in \mathbb{R}$ *such that:*

$$X_1, \ldots, X_\ell \in \mathcal{H}(\mathbf{q}, \delta_2) \qquad \text{and} \qquad Y_1, \ldots, Y_m \in \mathcal{H}(\mathbf{p}, \delta_1) \ .$$

Assume that, in addition, $\mathbf{0}_\ell, \mathbf{1}_\ell \notin \mathcal{H}(\mathbf{p}, \delta_1)$ and $\mathbf{0}_m, \mathbf{1}_m \notin H(\mathbf{q}, \delta_2)$. Then the function f cannot be computed using the GHKL protocol, for any choice of parameters (α, X_{real}), with respect to the specific simulation strategy used in Theorem 5.3.4.

Proof: We first consider the protocol where P_1 plays the party that inputs $x \in X$ and P_2 inputs $y \in Y$ (that is, P_2 is the second to receive output, exactly as GHKL protocol is described in Section 5.3). Fix any X_{real}, α, and let $\mathbf{c} = (p_{y_1}, \ldots, p_{y_m}) = X_{real} \cdot M_f$. First, observe that $\mathbf{conv}(\{X_1, \ldots, X_\ell\}) \subseteq \mathcal{H}(\mathbf{q}, \delta_2)$, since for any point $Z \in \mathbf{conv}(\{X_1, \ldots, X_\ell\})$, we can represent Z as $\mathbf{a} \cdot M_f$ for some probability vector $\mathbf{a}$. Then we have that $\langle Z, \mathbf{q}\rangle = \langle \mathbf{a} \cdot M_f, \mathbf{q}\rangle = \mathbf{a} \cdot \delta_2 \cdot \mathbf{1}_\ell = \delta_2$ and so $Z \in \mathcal{H}(\mathbf{q}, \delta_2)$. Now, assume by contradiction that the set of equations is satisfied. This implies that $Q^{x,a} \in \mathcal{H}(\mathbf{q}, \delta_2)$ for every $x \in X$, $a \in \{0,1\}$, since $Q^{x,a} \in \mathbf{conv}(\{X_1, \ldots, X_\ell\}) \subseteq \mathcal{H}(\mathbf{q}, \delta_2)$.

Let $\circ$ denote the entrywise product over $\mathbb{R}^m$, that is, for $Z = (z_1, \ldots, z_m)$, $W = (w_1, \ldots, w_m)$, the point $Z \circ W$ is defined as $(z_1 \cdot w_1, \ldots, z_m \cdot w_m)$. Recall that $\mathbf{c} = (p_{y_1}, \ldots, p_{y_m})$. We claim that for every X_i, the point $\mathbf{c} \circ X_i$ is also in the hyperplane $\mathcal{H}(\mathbf{q}, \delta_2)$. This trivially holds if $X_i = \mathbf{1}_m$. Otherwise, recall the definition of $Q^{x_i,0}$ (Eq. (5.3.1)):

$$q_{y_j}^{x_i,0} \overset{\text{def}}{=} \begin{cases} p_{y_j} & \text{if } f(x_i, y_j) = 1 \\ p_{y_j} + \frac{\alpha \cdot p_{y_j}}{(1-\alpha)\cdot(1-p_{x_i})} & \text{if } f(x_i, y_j) = 0 \end{cases},$$

Since $X_i \neq \mathbf{1}_m$, it holds that $p_{x_i} \neq 1$. Let $\gamma = \frac{\alpha}{(1-\alpha)\cdot(1-p_{x_i})}$. We can write $Q^{x,0}$ as follows:

$$Q^{x,0} = (1+\gamma) \cdot \mathbf{c} - \gamma \cdot (\mathbf{c} \circ X_i) \ .$$

Since for every i, the point $Q^{x_i,0}$ is in the hyperplane $\mathcal{H}(\mathbf{q}, \delta_2)$, we have:

$$\delta_2 = \langle Q^{x,0}, \mathbf{q}\rangle = \langle (1+\gamma)\cdot\mathbf{c} - \gamma\cdot(\mathbf{c}\circ X_i), \mathbf{q}\rangle = (1+\gamma)\cdot\langle \mathbf{c}, \mathbf{q}\rangle - \gamma\cdot\langle \mathbf{c}\circ X_i, \mathbf{q}\rangle = (1+\gamma)\cdot\delta_2 - \gamma\cdot\langle \mathbf{c}\circ X_i, \mathbf{q}\rangle$$

and thus, $\langle \mathbf{c} \circ X_i, \mathbf{q}\rangle = \delta_2$ which implies that $\mathbf{c} \circ X_i \in \mathcal{H}(\mathbf{q}, \delta_2)$.

We conclude that all the points $(\mathbf{c} \circ X_1), \ldots, (\mathbf{c} \circ X_\ell)$ are in the hyperplane $\mathcal{H}(\mathbf{q}, \delta_2)$. Since all the points $Y_1, \ldots, Y_m$ are in $\mathcal{H}(\mathbf{p}, \delta_1)$, it holds that $\mathbf{p} \cdot M_f = \delta_1 \cdot \mathbf{1}_m$. Thus, $\sum_{i=1}^{\ell} p_i \cdot X_i = \delta_1 \cdot \mathbf{1}_m$, which implies that:

$$\begin{aligned} \sum_{i=1}^{\ell} p_i \cdot \delta_2 &= \sum_{i=1}^{\ell} p_i \cdot \Big\langle \mathbf{c} \circ X_i, \mathbf{q} \Big\rangle = \Big\langle \sum_{i=1}^{\ell} p_i \cdot (\mathbf{c} \circ X_i), \mathbf{q} \Big\rangle = \Big\langle \mathbf{c} \circ (\sum_{i=1}^{\ell} p_i \cdot X_i), \mathbf{q} \Big\rangle = \langle \mathbf{c} \circ (\delta_1 \cdot \mathbf{1}_m), \mathbf{q}\rangle \\ &= \delta_1 \cdot \langle \mathbf{c}, \mathbf{q}\rangle = \delta_1 \cdot \delta_2 \end{aligned}$$

and thus it must hold that either $\sum_{i=1}^{\ell} p_i = \delta_1$ or $\delta_2 = 0$, which implies that $\mathbf{1} \in \mathcal{H}(\mathbf{p}, \delta_1)$ or $\mathbf{0} \in \mathcal{H}(\mathbf{q}, \delta_2)$, in contradiction to the additional requirements.

The above shows that the protocol does not hold when the P_1 party is the first to receive output. We can change the roles and let P_2 be the first to receive an output (that is, we can use the protocol to compute f^T). In such a case, we will get that it must hold that $\sum_{i=1}^{m} q_i = \delta_2$ or $\delta_1 = 0$, again, in contradiction to the assumptions that $\mathbf{1} \notin \mathcal{H}(\mathbf{q}, \delta_2)$ and $\mathbf{0} \notin \mathcal{H}(\mathbf{p}, \delta_1)$. ■

This negative result does not rule out the possibility of these functions using some other protocol. However, it rules out the only known possibility result that we have for fairness. Moreover, incorporating this with the characterization of coin-tossing (Chapter 4), there exists a large set of functions for which the only possibility result does not hold, and the only impossibility result does not hold either. Moreover, this class of functions shares a similar (but yet distinct) algebraic structure with the class of functions that implies fair coin-tossing. See more in Subsection 5.4.3.

A possible generalization of [58]. In Section 5.3, we generalized the protocol such that the algorithm $\mathsf{RandOut}_2(y)$ chooses $\hat{x}$ according to some distribution X_{real} and not just uniformly at random as in [58]. Similarly, we could have generalized the protocol for the $\mathsf{RandOut}_1(x)$ algorithm as well, while defining some distribution Y_{real}, and parameterized the protocol with this additional distribution (in the same sense as we parameterized X_{real}). By using this modification, the protocol may potentially compute more functions. However, by a simple adjustment of the proof of Theorem 5.4.8, as long as Y_{real} is "valid" (in a sense that $p_{x_i} = 1$ if and only if $X_i = \mathbf{1}_m$ and $p_{x_i} = 0$ if and only if $X_i = \mathbf{0}_m$, otherwise the output distribution vectors $Q^{x,a}$ are not well-defined), this generalization does not help, and the negative result holds for this generalization also.

Our theorem does not hold in cases where either $\mathbf{0}_\ell \in \mathcal{H}(\mathbf{p}, \delta_1)$ or $\mathbf{1}_\ell \in \mathcal{H}(\mathbf{p}, \delta_1)$ (likewise, for $\mathcal{H}(\mathbf{q}, \delta_2)$). These two requirements are in some sense equivalent. This is because the alphabet is not significant, and we can switch between the two symbols 0 and 1. Thus, if for some function f the hyperplane $\mathcal{H}(\mathbf{p}, \delta_1)$ passes through the origin $\mathbf{0}$, the corresponding hyperplane for the function $\bar{f}(x,y) = 1 - f(x,y)$ passes through $\mathbf{1}$ and vice versa. Feasibility of fairness for f and $\bar{f}$ is equivalent.

On the Number of Functions that Satisfy the Additional Requirements

We now count on the number of functions with $|X| = |Y|$ that satisfy these additional requirements, that is, define hyperplanes that do not pass through the origin $\mathbf{0}$ and the point $\mathbf{1}$. As we have seen in Theorem 5.4.8, these functions cannot be computed with complete fairness using the protocol of [58]. As we will see, only a negligible amount of functions with $|X| = |Y|$ do not satisfy these additional requirements. Thus, our characterization of [58] is almost tight: Almost all functions with $|X| \neq |Y|$ can be computed fairly, whereas almost all functions with $|X| = |Y|$ cannot be computed using the protocol of [58]. We have the following Claim:

Claim 5.4.9 *With a probability that tends to* 0 *when* $|X|, |Y|$ *grow, a random function with* $|X| = |Y|$ *defines hyperplanes that pass through the points* $\mathbf{0}$ *or* $\mathbf{1}$.

Proof: Let $m = |X| = |Y|$. Recall that P_m denotes the probability that random m vectors of size m are linearly dependent. Moreover, by Claim 5.4.7, the probability that a random set $\{X_1, \ldots, X_{m+1}\}$ is affinely independent with probability $1 - P_m$, even when one of the points is chosen arbitrarily.

Thus, with probability P_m, the set $\{X_1, \ldots, X_m, \mathbf{1}\}$ where $X_1, \ldots, X_m$ are chosen at random is affinely dependent. In this case, the hyperplane defined by $\{X_1, \ldots, X_m\}$ contains the point $\mathbf{1}$. Similarly, the set $\{X_1, \ldots, X_m, \mathbf{0}\}$ is affinely dependent with the same probability P_m. Overall, using union-bound, the probability that the hyperplane of random points $X_1, \ldots, X_m$ contains the points $\mathbf{1}$ or $\mathbf{0}$ is less than or equal to $2 \cdot P_m$. From similar arguments, the probability that the hyperplane that is defined by the columns of the matrix contains either $\mathbf{1}$ or $\mathbf{0}$ is $2 \cdot P_m$, and therefore the overall probability is $4 \cdot P_m$. ■

Functions with Monochromatic Input

We consider a limited case where the above requirements do not satisfy, that is, functions that are not full-dimensional but define hyperplanes that pass through $\mathbf{0}$ or $\mathbf{1}$. For this set of functions, the negative result does not apply. We now show that for some subset in this class, fairness is possible. Our result here does not cover all functions in this subclass.

Assume that a function contains a "monochromatic input", that is, one party has an input that causes the same output irrespectively to the input of the other party. For instance, P_2 has input y_j such that for every $x \in X$: $f(x, y_j) = 1$. In this case, the point $\mathbf{1}_\ell$ is one of the columns of the matrix, and therefore, the hyperplane $\mathcal{H}(\mathbf{p}, \delta_1)$ must pass through it. We show that in this case we can ignore this input and consider the "projected" $m \times (m-1)$ function f' where we remove the input y_j. This latter function may now be full-dimensional, and the existence of a protocol for f' implies the existence of a protocol for f. Intuitively, this is because when P_2 uses y_j, the real-world adversary P_1 cannot bias its output since it is always 1. We have:

Claim 5.4.10 *Let $f : X \times Y \to \{0,1\}$, and assume that M_f contains the all-one (resp. all-zero) column. That is, there exists $y \in Y$ such that for every $\hat{x} \in X$, $f(\hat{x}, y) = 1$ (resp. $f(\hat{x}, y) = 0$).*

If the function $f' : X \times Y' \to \{0,1\}$, where $Y' = Y \setminus \{y\}$ is full-dimensional, then f can be computed with complete-fairness.

Proof: Assume that the function contains the all one column, and that it is obtained by input y_m (i.e., the mth column is the all-one column). Let $X_1, \ldots, X_m$ be the rows of M_f, and let X'_i be the rows over $\mathbb{R}^{m-1}$ without the last coordinate, that is, $X_i = (X'_i, 1)$. Consider the "projected" function $f' : \{x_1, \ldots, x_m\} \times \{y_1, \ldots, y_{m-1}\} \to \{0,1\}$ be defined as $f'(x, y) = f(x, y)$, for every x, y in the range (we just remove y_m from the possible inputs of P_2). The rows of $M_{f'}$ are $X'_1, \ldots, X'_m$.

Now, since f' is of full-dimensional, the function f' can be computed using the GHKL protocol. Let $X^{x,a}_{ideal}$ be the solutions for equations of Theorem 5.3.4 for the function f'. It can be easily verified that $X^{x,a}_{ideal}$ are the solutions for equations for the f function as well, since for every x, a, the first $m-1$ coordinates of $Q^{x,a}$ are the same as f', and the last coordinate of $Q^{x,a}$ is always 1. For $Q^{x,0}$ it holds immediately, for $Q^{x,1}$ observe that $p_{y_m} = 1$ no matter what X_{real} is, and thus $p_{y_j} + \frac{\alpha \cdot (p_{y_j} - 1)}{(1-\alpha) \cdot p_x} = 1 + 0 = 1$). Therefore, $X^{x,a}_{ideal}$ are the solutions for f as well, and Theorem 5.3.4 follows for f as well. ■

The above implies an interesting criterion that is easy to verify:

Proposition 5.4.11 *Let $f : \{x_1, \ldots, x_m\} \times \{y_1, \ldots, y_m\} \to \{0,1\}$ be a function. Assume that f contains the all-one column, and that M_f is of full rank. Then the function f can be computed with complete fairness.*

Proof: Let $X_1, \ldots, X_m$ be the rows of M_f, and assume that the all-one column is the last one (i.e., input y_m). Consider the points $X'_1, \ldots, X'_m$ in $\mathbb{R}^{m-1}$, where for every i, $X_i = (X'_i, 1)$ (i.e., X'_i is the first $m-1$ coordinates of X_i). Since M_f is of full-rank, the rows $X_1, \ldots, X_m$ are linearly independent, which implies that m points $X'_1, \ldots, X'_m$ in $\mathbb{R}^{m-1}$ are affinely independent. We therefore can apply Claim 5.4.10 and fairness in f is possible. ■

Finally, from simple symmetric properties, almost *always* a random matrix that contains the all one row / vector is of full rank, in the sense that we have seen in Claims 5.4.7 and 5.4.9. Therefore, almost always a random function that contains a monochromatic input can be computed with complete fairness.

Functions with no embedded XOR. [58] presents a totally different and simpler protocol for handling functions that do not contain an embedded XOR (i.e., the Boolean AND/OR functions, and the greater-than function). An immediate Corollary from Proposition 5.4.11 is that *all* the functions that can be computed using this simpler protocol, can also be computed using the second generalized protocol. This is because all functions that do not contain an embedded XOR (or their complement function) have both full-rank and monochromatic input, and therefore can be computed using the generalized protocol by Proposition 5.4.11. This shows that indeed, the only known possibility result that we have for fairness is Protocol 5.3.3.

However, this first totally different protocol gives an answer to an interesting question regarding the round-complexity of fair protocols. In particular, [58] gives a lower bound of $\omega(\log \kappa)$ for the round complexity of protocol for computing functions that contain embedded XOR. The simpler protocol for handling functions with no embedded XOR has round complexity that is linear in $|X|, |Y|$ and is *independent* of the security parameter κ.

5.4.3 Conclusion: Symmetric Boolean Functions with Finite Domain

We overview all the known results in complete fairness for symmetric Boolean functions with finite domain, and we link our results to the balanced property of Chapter 4.

The Characterization

A full-dimensional function is an important special case of this unbalanced property, as was pointed out in Claim 5.4.2. Combining the above characterization of Chapter 4 with this chapter, we get the following Theorem:

Theorem 5.4.12 *Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0, 1\}$, and let M_f be the corresponding matrix representing f as above. Then:*

1. **Balanced with respect to probability vectors (Chapter 4):**
If there exist probability vectors $\mathbf{p} = (p_1, \ldots, p_\ell), \mathbf{q} = (q_1, \ldots, q_m)$ and a constant $0 < \delta < 1$ such that: $\mathbf{p} \cdot M_f = \delta \cdot \mathbf{1}_m$ and $M_f \cdot \mathbf{q}^T = \delta \cdot \mathbf{1}_\ell^T$.
Then, the function f implies fair coin-tossing, and is impossible to compute fairly.
2. **Balanced with respect to arbitrary vectors, but not balanced with respect to probability vectors:**
If there exist two non-zero vectors $\mathbf{p} = (p_1, \ldots, p_\ell) \in \mathbb{R}^\ell$, $\mathbf{q} = (q_1, \ldots, q_m) \in \mathbb{R}^m$, $\delta_1, \delta_2 \in \mathbb{R}$, such that: $\mathbf{p} \cdot M_f = \delta_1 \cdot \mathbf{1}_m$ and $M_f \cdot \mathbf{q}^T = \delta_2 \cdot \mathbf{1}_\ell^T$
then we say that the function is balanced with respect to arbitrary vectors. Then, the function does not (information-theoretically) imply fair-coin tossing (Corollary 4.5.7). Moreover:
   (a) *If δ_1 and δ_2 are non-zero, $\sum_{i=1}^{\ell} p_i \neq \delta_1$ and $\sum_{i=1}^{m} q_i \neq \delta_2$, then the function f cannot be computed using the GHKL protocol (Theorem 5.4.8).*
   (b) *Otherwise: this case is left not characterized. For a subset of this subclass, we show possibility (Proposition 5.4.10).*
3. **Unbalanced with respect to arbitrary vectors:**
If for every non-zero $\mathbf{p} = (p_1, \ldots, p_\ell) \in \mathbb{R}^\ell$ and any $\delta_1 \in \mathbb{R}$ it holds that $\mathbf{p} \cdot M_f \neq \delta_1 \cdot \mathbf{1}_m$, **OR** *for every non-zero $\mathbf{q} = (q_1, \ldots, q_m) \in \mathbb{R}^m$ and any $\delta_2 \in \mathbb{R}$ it holds that $M_f \cdot \mathbf{q}^T \neq \delta_2 \cdot \mathbf{1}_\ell^T$, then f can be computed with complete fairness (Theorem 5.4.4).*

We remark that, in general, if $|X| \neq |Y|$ then almost always a random function is in subclass 3. Moreover, if $|X| = |Y|$, then only a negligible amount of functions are in subclass 2b, and thus only a negligible amount of functions are left not characterized.

If a function is balanced with respect to arbitrary vectors (i.e., the vector may contain negative values), then all the rows of the function lie in the hyperplane $\mathcal{H}(\mathbf{q}, \delta_2)$, and all the columns lie in the hyperplane $\mathcal{H}(\mathbf{p}, \delta_1)$. Observe that $\delta_1 = 0$ if and only if $\mathcal{H}(\mathbf{p}, \delta_1)$ passes through the origin, and $\sum_{i=1}^{\ell} p_i = \delta_1$ if and only if $\mathcal{H}(\mathbf{p}, \delta_1)$ passes through the all one point $\mathbf{1}$. Thus, the requirements of subclass 2a are a different formalization of the requirements of Theorem 5.4.8. Likewise, the requirements of subclass 3 are a different formalization of Theorem 5.4.4, as was proven in Claim 5.4.2.

Examples

We give a few examples for interesting functions and practical tasks that can be computed with complete-fairness.

- **Set membership.** Assume some finite possible elements Ω, and consider the task of set-membership: P_1 holds $S \subseteq \Omega$, P_2 holds some elements $\omega \in \Omega$, and the parties wish to find out (privately) whether $\omega \in S$. The number of possible inputs for P_1 is $|\Omega|$, whereas the number of possible inputs for P_2 is $|P(\Omega)| = 2^{|\Omega|}$, and the truth table for this function contains all possible Boolean vectors of length-$|\Omega|$ (see Figure 5.3 for the case of $\Omega = \{a, b\}$).

- **Private evaluation of Boolean function.** Let $\mathcal{F} = \{g \mid g : \Omega \to \{0, 1\}\}$ be the family of all Boolean functions with $|\Omega|$ inputs. Assume that P_1 holds some function $g \in \mathcal{F}$ and P_2 holds some input $y \in \Omega$. The parties wish to privately learn $g(y)$. This task has the exact same truth-table as the set-membership functionality, i.e., it contains all possible Boolean vectors of length-$|\Omega|$ (in this case, each vector represents a possible function g).

- **Private matchmaking.** Assume that P_1 holds some set of preferences, while P_2 holds a profile. The parties wish to learn (privately) whether there is a matching between them. In fact, this task is a special case of the *subset-equal* functionality. that is, P_1 holds some $A \subseteq \Omega$, P_2 holds $B \subseteq \Omega$, and the parties wish to learn whether $A \subseteq B$. Although the possible inputs for both parties is $2^{|\Omega|}$ (i.e., $|X| = |Y|$), the truth-table for this functionality satisfies Proposition 5.4.11, contains monochromatic row and can be computed using the protocol. See Figure 5.3.

- **Set disjointness.** The set-disjointness is another functionality that is feasible although $|X| = |Y|$, and is defined as follows: P_1 holds $A \subseteq \Omega$, P_2 holds $B \subseteq \Omega$, and the parties learn whether $A \cap B = \emptyset$. In fact, the possibility of this function is easily derived from the possibility of $A \subseteq B$, using the following observation:

$$A \cap B = \emptyset \iff A \subseteq \overline{B}$$

	a	b
$\emptyset$	0	0
$\{a\}$	1	0
$\{b\}$	0	1
$\{a,b\}$	1	1

(a) The set-membership functionality

	$\emptyset$	$\{a\}$	$\{b\}$	$\{a,b\}$
$\emptyset$	1	1	1	1
$\{a\}$	0	1	0	1
$\{b\}$	0	0	1	1
$\{a,b\}$	0	0	0	1

(b) The functionality $A \subseteq B$

Figure 5.3: The set-membership and $A \subseteq B$ functionalities with $\Omega = \{a, b\}$

5.5 Extensions: Asymmetric Functions and Non-Binary Outputs

5.5.1 Asymmetric Functions

We now move to a richer class of functions, and consider asymmetric Boolean functions where the parties do not necessarily get the same output. We consider functions $f(x,y) = (f_1(x,y), f_2(x,y))$, where each f_i, $i \in \{1,2\}$ is defined as: $f_i : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0,1\}$. Interestingly, our result here shows that if the function f_2 is of full-dimension, then f can be computed fairly, irrespectively to the function f_1. This is because simulating P_1 is more challenging (because it is the first to receive an output) and the simulator needs to assume the rich description of f_2 in order to be able to bias the output of the honest party P_2. On the other hand, since P_2 is the second to receive an output, simulating P_2 is easy and the simulator does not need to bias the output of P_1.

The protocol. We first revise the protocol of [58]. The $\mathsf{RandOut}_1(x)$ algorithm is changed such the function that is being evaluated is f_1, and the algorithm $\mathsf{RandOut}_2(y)$ is modified such that the function that is being evaluated is f_2. Step 2.2.b in the F_{dealer} functionality (Functionality 5.3.2) is modified such that each party receives an output according to its respective output, that is:

Step 2.2.b: *For $i = i^*$ to R: set $(a_i, b_i) = (f_1(x,y), f_2(x,y))$.*

Analysis. The proof for the case where P_2 follows exactly along the lines of the proof of Claim C.1.1. We now turn to corrupted P_1. Here, we again define the probability p_x for every $x \in X$ as $\Pr_{\hat{y} \leftarrow U_Y}[f_1(x,\hat{y}) = 1]$. Then, for every $y_j \in Y$, we define $p_{y_j} = \Pr_{\hat{x} \leftarrow X_{real}}[f_2(\hat{x}, y_j) = 1]$. This time, $(p_{y_1}, \ldots, p_{y_m}) = X_{real} \cdot M_{f_2}$, where M_{f_2} represents the function f_2.

The case where P_1 is corrupted is very similar to the symmetric case; however, the constraints and the vectors $Q^{x,a}$ are different. Using the same calculations as the symmetric case, we get the following vectors $Q^{x,0} = (q^{x,0}_{y_1}, \ldots, q^{x,0}_{y_m})$ and $Q^{x,1} = (q^{x,1}_{y_1}, \ldots, q^{x,1}_{y_m})$:

$$q^{x,0}_{y_j} \stackrel{\text{def}}{=} \begin{cases} p_{y_j} + \frac{\alpha \cdot p_{y_j}}{(1-\alpha)\cdot(1-p_x)} & \text{if } f(x,y_j) = (0,0) \\ p_{y_j} + \frac{\alpha \cdot (p_{y_j}-1)}{(1-\alpha)\cdot(1-p_x)} & \text{if } f(x,y_j) = (0,1) \\ p_{y_j} & \text{if } f(x,y_j) = (1,0) \\ p_{y_j} & \text{if } f(x,y_j) = (1,1) \end{cases},$$

$$q_{y_j}^{x,1} \stackrel{\text{def}}{=} \begin{cases} p_{y_j} & \text{if } f(x,y_j) = (0,0) \\ p_{y_j} & \text{if } f(x,y_j) = (0,1) \\ p_{y_j} + \frac{\alpha \cdot p_{y_j}}{(1-\alpha)\cdot p_x} & \text{if } f(x,y_j) = (1,0) \\ p_{y_j} + \frac{\alpha\cdot(p_{y_j}-1)}{(1-\alpha)\cdot p_x} & \text{if } f(x,y_j) = (1,1) \end{cases} .$$

Observe that if $f_1(x_i, y_j) = 0$ it implies that $p_{x_i} \neq 1$, and therefore $Q^{x_i,0}$ is well-defined (i.e., $p_{x_i} = 1$ if and only of the row X_i is the all one row). Similarly, if $f_1(x_i, y_j) = 1$ it implies that $p_{x_i} \neq 0$, and therefore $Q^{x,1}$ is well-defined. Overall, we get:

Theorem 5.5.1 *Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0,1\} \times \{0,1\}$, where $f = (f_1, f_2)$. Let M_{f_2} be the matrix that represents f_2. If there exist X_{real}, $0 < \alpha < 1$ such that $\alpha^{-1} \in O(\mathsf{poly}(\kappa))$, such that for every $x \in X$, $a \in \{0,1\}$, there exists a probability vector $X_{ideal}^{x,a}$ such that:*

$$X_{ideal}^{x,a} \cdot M_{f_2} = Q^{x,a} ,$$

then Protocol 5.3.3 securely computes f with complete fairness.

Proof: The case of P_2 is corrupted follows exactly the same as the case in the proof of Theorem C.2.1. All that is left is just what are the requirements from $f(x, y_j)$ in each one of the possible outputs $(0,0), (0,1), (1,0), (1,1)$. The cases of $(0,0)$ and $(1,1)$ are exactly the same as in the proof of Theorem C.2.1, where the case of $(0,0)$ corresponds to the symmetric case of $f(x, y_j) = 0$, and the case of $(1,1)$ corresponds to the symmetric case $f(x, y_j) = 1$. This defines the following requirements (exactly as Equation 5.3.1):

$$q_{y_j}^{x,0} \stackrel{\text{def}}{=} \begin{cases} p_{y_j} & \text{if } f(x,y_j) = (1,1) \\ p_{y_j} + \frac{\alpha\cdot p_{y_j}}{(1-\alpha)\cdot(1-p_x)} & \text{if } f(x,y_j) = (0,0) \end{cases} \qquad q_{y_j}^{x,1} \stackrel{\text{def}}{=} \begin{cases} p_{y_j} + \frac{\alpha\cdot(p_{y_j}-1)}{(1-\alpha)\cdot p_x} & \text{if } f(x,y_j) = (1,1) \\ p_{y_j} & \text{if } f(x,y_j) = (0,0) \end{cases}$$

We now turn to the cases where the outputs are distinct.

The case where $\boldsymbol{f(x, y_j) = (0,1)}$. That is, we consider the case where $f_1(x, y_j) = 0$, whereas $f_2(x, y_j) = 1$. We get the following four possibilities:

output (a,b)	REAL	IDEAL
$(0,0)$	$(1-\alpha)\cdot(1-p_x)\cdot(1-p_y) + \alpha\cdot(1-p_y)$	$(1-\alpha)\cdot(1-p_x)\cdot(1-q_y^{x,0})$
$(0,1)$	$(1-\alpha)\cdot(1-p_x)\cdot p_y + \alpha\cdot p_y$	$(1-\alpha)\cdot(1-p_x)\cdot q_y^{x,0} + \alpha$
$(1,0)$	$(1-\alpha)\cdot p_x\cdot(1-p_y)$	$(1-\alpha)\cdot p_x\cdot(1-q_y^{x,1})$
$(1,1)$	$(1-\alpha)\cdot p_x\cdot p_y$	$(1-\alpha)\cdot p_x\cdot q_y^{x,1}$

The only difference from the case of $f(x, y_j) = (0,0)$ is that in the ideal execution, the $+\alpha$ is obtained in the second row instead of the first row. The probabilities are equal if and only if the following hold:

$$q_y^{x,0} = p_y + \frac{\alpha\cdot(p_y - 1)}{(1-\alpha)\cdot(1-p_x)} \qquad \text{and} \qquad q_y^{x,1} = p_y .$$

The case where $\boldsymbol{f(x, y) = (1,0)}$. Similarly to the above, we obtain:

output (a,b)	REAL	IDEAL
$(0,0)$	$(1-\alpha)\cdot(1-p_x)\cdot(1-p_y)$	$(1-\alpha)\cdot(1-p_x)\cdot(1-q_y^{x,0})$
$(0,1)$	$(1-\alpha)\cdot(1-p_x)\cdot p_y$	$(1-\alpha)\cdot(1-p_x)\cdot q_y^{x,0}$
$(1,0)$	$(1-\alpha)\cdot p_x\cdot(1-p_y)+\alpha\cdot(1-p_y)$	$(1-\alpha)\cdot p_x\cdot(1-q_y^{x,1})+\alpha$
$(1,1)$	$(1-\alpha)\cdot p_x\cdot p_y+\alpha\cdot p_y$	$(1-\alpha)\cdot p_x\cdot q_y^{x,1}$

The only difference from the case of $f(x,y_j)=(1,1)$ is that in the ideal execution, the $+\alpha$ is obtained in the second row instead of the fourth row, which result in the following constraints:

$$q_y^{x,0}=p_y \qquad \text{and} \qquad q_y^{x,1}=p_y+\frac{\alpha\cdot p_y}{(1-\alpha)\cdot p_x}\ .$$

■

Similarly to the case of single output, the above implies that:

Corollary 5.5.2 *Let $f:\{x_1,\ldots,x_\ell\}\times\{y_1,\ldots,y_m\}\to\{0,1\}\times\{0,1\}$, where $f=(f_1,f_2)$. If f_2 is a full-dimensional function, then f can be computed with complete fairness.*

Proof: It is easy to see that for every constant $\epsilon>0$, for every $x\in X$ and $a\in\{0,1\}$, and for all sufficiently large κ's, it holds that $Q^{x,a}\in B(\mathbf{c},\epsilon)$, where $\mathbf{c}=(p_{y_1},\ldots,p_{y_m})$. As we saw, in case f_2 is of full-dimension, we can embed $\mathbf{c}$ inside the convex and there exists a constant $\epsilon>0$ such that $B(\mathbf{c},\epsilon)\subset\mathbf{conv}(\{X_1^{(2)},\ldots,X_\ell^{(2)}\})$, where $X_i^{(2)}$ is the ith row of f_2. The corollary follows. ■

5.5.2 Functions with Non-Binary Output

Until now, all the known possibility results for fairness dealt with the case of binary output. We now extend the results to the case of non-binary output. Let $\Sigma=\{0,\ldots,k-1\}$ be an alphabet for some finite $k>0$ (the alphabet may be arbitrary, we use $[0,k-1]$ for the sake of convenience), and consider functions $f:\{x_1,\ldots,x_\ell\}\times\{y_1,\ldots,y_m\}\to\Sigma$.

The protocol is exactly the GHKL protocol presented in Section 5.3, where here a_i,b_i are elements in Σ and not just bits. We just turn to the analysis. The proof of corrupted P_2 again follows along the lines of the proof of Claim C.1.1. The difficult part is corrupted P_1. Again, all the claims follow exactly in the same way and we just need to compare the output distributions of the parties in case the adversary aborts before or at i^*. We analyze this in the following.

Fix some X_{real}, and let U_Y denote the uniform distribution over Y. For any symbol $\sigma\in\Sigma$, and for every $x\in X$, denote by $p_x(\sigma)$ the probability that $\mathsf{RandOut}_1(x)$ is σ. Similarly, for every $y_j\in Y$, let $p_{y_j}(\sigma)$ denote the probability that the output of $\mathsf{RandOut}_2(y_j)$ is σ. That is:

$$p_x(\sigma)\stackrel{\text{def}}{=}\Pr_{\hat{y}\leftarrow U_Y}[f(x,\hat{y})=\sigma] \qquad \text{and} \qquad p_{y_j}(\sigma)\stackrel{\text{def}}{=}\Pr_{\hat{x}\leftarrow X_{real}}[f(\hat{x},y_j)=\sigma]\ .$$

Observe that $\sum_{\sigma\in\Sigma}p_x(\sigma)=1$ and $\sum_{\sigma\in\Sigma}p_{y_j}(\sigma)=1$.

For every $\sigma\in\Sigma$, we want to present the vector $(p_{y_1}(\sigma),\ldots,p_{y_m}(\sigma))$ as a function of X_{real} and M_f, as we did in the binary case (where there we just had: $(p_{y_1},\ldots,p_{y_m})=X_{real}\cdot M_f$). However, here it is a bit more complicated than the binary case. Therefore, for any $\sigma\in\Sigma$, we define the binary matrix M_f^σ as follows:

$$M_f^\sigma(i,j)=\begin{cases}1 & \text{if } f(x_i,y_j)=\sigma\\ 0 & \text{otherwise}\end{cases}\ .$$

Then, for every $\sigma \in \Sigma$, it holds that $(p_{y_1}(\sigma), \ldots, p_{y_m}(\sigma)) = X_{real} \cdot M_f^{\sigma}$. Moreover, it holds that: $\sum_{\sigma \in \Sigma} M_f^{\sigma} = \mathbf{J}_{\ell \times m}$, where $\mathbf{J}_{\ell \times m}$ is the all one matrix with sizes $\ell \times m$. In the binary case, M_f is in fact M_f^1, and there was no need to consider the matrix M_f^0, for a reason that we will see below (intuitively, since in the binary case, $p_{y_j}(0) + p_{y_j}(1) = 1$).

The output distribution vectors. Similarly as in the binary case, in case the adversary sends x to the simulated F_{dealer} and aborts in round $i < i^*$ upon receiving some symbol $a \in \Sigma$, the simulator chooses its input $\hat{x}$ according to some distribution $X_{ideal}^{x,a}$. For each such a vector $X_{ideal}^{x,a}$, we define $|\Sigma|$ points $Q^{x,a}(\sigma_1), \ldots, Q^{x,a}(\sigma_k)$ in $\mathbb{R}^m$, where each vector $Q^{x,a}(b)$ (for $b \in \Sigma$) is obtained by: $Q^{x,a}(b) = X_{ideal}^{x,a} \cdot M_f^b$.

For a given $x \in X, y_j \in Y$ and $a, b \in \Sigma$, we define the requirements from each vector $Q^{x,a}(b) = (q_{y_1}^{x,a}(b), \ldots, q_{y_m}^{x,a}(b))$. We have:

$$q_{y_j}^{x,a}(b) \stackrel{\text{def}}{=} \begin{cases} p_{y_j}(b) + \frac{\alpha}{(1-\alpha)} \cdot \frac{(p_{y_j}(b)-1)}{p_x(a)} & \text{if } f(x,y_j) = a = b \\ p_{y_j}(b) + \frac{\alpha}{(1-\alpha)} \cdot \frac{p_{y_j}(b)}{p_x(a)} & \text{if } f(x,y_j) = a \neq b \\ p_{y_j}(b) & \text{if } f(x,y_j) \neq a \end{cases} \tag{5.5.1}$$

We then require that for the *same* distribution $X_{ideal}^{x,a}$ it holds that $X_{ideal}^{x,a} \cdot M_f^b = Q^{x,a}(b)$ for every $b \in \Sigma$, *simultaneously.* We have the following Theorem:

Theorem 5.5.3 *Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \Sigma$ be a function. If there exists a parameter $0 < \alpha < 1$, $\alpha^{-1} \in O(\mathsf{poly}(\kappa))$ and a distribution X_{real}, such that for every $x \in X$, for every $a \in \Sigma$ and for every $b \in \Sigma$, it holds that:*

$$X_{ideal}^{x,a} \cdot M_f^b = Q^{x,a}(b) \quad ,$$

Then the function f can be computed with complete fairness.

Proof: The simulator for P_1 is exactly the same as in Theorem 5.3.4. All that is left is to show that the outputs are distributed identically in case the adversary aborts before or at i^*.

The real execution. We now consider the output distribution where the adversary halts in round i for which $i < i^*$. In such a case, both parties output independent outputs according to $\mathsf{RandOut}_1(x)$ and $\mathsf{RandOut}_2(y_j)$, where (x, y_j) are the inputs sent to the F_{dealer}. As a result, they receive output (a, b) with probability $(1-\alpha) \cdot p_x(a) \cdot p_{y_j}(b)$, where $(1-\alpha)$ is the probability that $i < i^*$ given that $i \leq i^*$. In case $i = i^*$ (which given $i \leq i^*$, it happens with probability α), the adversary P_1 learns the correct output $f(x, y_j)$, whereas P_2 gets an output according to $\mathsf{RandOut}_2(y_j)$. Overall, we have that:

$$\begin{aligned} &\Pr\left[(\text{VIEW}_{\mathsf{hyb}}^i, \text{OUTPUT}_{\mathsf{hyb}}) = (a,b) \;\mid\; i \leq i^*\right] \\ &= \begin{cases} (1-\alpha) \cdot p_x(a) \cdot p_{y_j}(b) + \alpha \cdot 1 \cdot p_{y_j}(b) & \text{if } f(x,y_j) = a = b \\ (1-\alpha) \cdot p_x(a) \cdot p_{y_j}(b) + \alpha \cdot 1 \cdot p_{y_j}(b) & \text{if } f(x,y_j) = a \neq b \\ (1-\alpha) \cdot p_x(a) \cdot p_{y_j}(b) & \text{if } f(x,y_j) \neq a \end{cases} \end{aligned}$$

(We differentiate between the first two cases although they are equal here, since they are different in the ideal.)

The ideal execution. In case the adversary aborts at i^*, the simulator sends the true input x to the trusted party and *both* parties receive the correct output $f(x,y_j)$. On the other hand, in case the adversary aborts in round $i < i^*$ after sending some $x \in X$ to the simulated F_{dealer} and upon receiving a symbol $a \in \Sigma$, the simulator chooses an input $\hat{x}$ according to distribution $X_{ideal}^{x,a}$, and the output of the honest party is determined accordingly, where we denote by $q_{y_j}^{x,a}(b)$ the probability that the output of the honest party P_2 is b in this case. Therefore, we have that:

$$\Pr\left[(\text{VIEW}^i_{\mathsf{ideal}}, \text{OUTPUT}_{\mathsf{ideal}}) = (a,b) \;\mid\; i \leq i^*\right]$$
$$= \begin{cases} (1-\alpha)\cdot p_x(a)\cdot q_{y_j}^{x,a}(b) + \alpha & \text{if } f(x,y_j) = a = b \\ (1-\alpha)\cdot p_x(a)\cdot q_{y_j}^{x,a}(b) & \text{if } f(x,y_j) = a \neq b \\ (1-\alpha)\cdot p_x(a)\cdot q_{y_j}^{x,a}(b) & \text{if } f(x,y_j) \neq a \end{cases}$$

Therefore, if Eq. (5.5.1) holds, then we have for every $(a,b) \in \Sigma^2$:

$$\Pr\left[(\text{VIEW}^i_{\mathsf{hyb}}, \text{OUTPUT}_{\mathsf{hyb}}) = (a,b) \;\mid\; i \leq i^*\right] = \Pr\left[(\text{VIEW}^i_{\mathsf{ideal}}, \text{OUTPUT}_{\mathsf{ideal}}) = (a,b) \;\mid\; i \leq i^*\right] .$$

This, combining with the rest of the proof of Theorem C.2.1 shows that the protocol can be simulated.

■

Relaxing the requirements. In the above Theorem, for the *same* distribution $X_{ideal}^{x,a}$, it is required that $X_{ideal}^{x,a} \cdot M_f^b = Q^{x,a}(b)$ for every $b \in \Sigma$ simultaneously. This requirement makes things much harder, since it involves convex combinations with the same coefficients ($X_{ideal}^{x,a}$) of *different* convex-hulls $M_f^{\sigma_1}, \ldots, M_f^{\sigma_k}$. We therefore want to encode all the requirements together for the same $X_{ideal}^{x,a}$. We do it in two steps. First, we show that it it is enough to consider equality with respect to $|\Sigma| - 1$ symbols in the alphabet, and the last one is a simple derivation of the others (this also explains why in the binary case, we did not consider the matrix M_f^0). Next, we show hot to encode all the requirements together into a single system.

Claim 5.5.4 *Let f, be as above and let $\sigma \in \Sigma$ be arbitrary. Then, if there exists $0 < \alpha < 1$ (such that $\alpha^{-1} \in O(\mathsf{poly}(\kappa))$), distribution X_{real} such that for every $x \in X$, for every $a \in \Sigma$ and for every $b \in \Sigma \setminus \{\sigma\}$*

$$X_{ideal}^{x,a} \cdot M_f^b = Q^{x,a}(b) ,$$

then it holds that $X_{ideal}^{x,a} \cdot M_f^\sigma = Q^{x,a}(\sigma)$ as well and thus f can be computed with complete fairness.

Proof: Fix σ, α, X_{real} and assume that for every $x \in X$, $a \in \Sigma$ there exists $X_{ideal}^{x,a}$ as above. We now show that the $X_{ideal}^{x,a} \cdot M_f^\sigma = Q^{x,a}(\sigma)$. The key observation is that the sum of all matrices $\sum_{b\in\Sigma} M_f^b$ is $\mathbf{J}_{\ell\times m}$ (the all-one matrix). Moreover, for every x, a, it holds that $\sum_b Q^{x,a}(b) = \mathbf{1}_m$. This can is derived by analyzing each coordinate $q_{y_j}^{x,a}$ separately as follows:

- **Case 1: $f(x,y_j) \neq a$.** In this case, $q_{y_j}^{x,a}(b) = p_{y_j}(b)$ for every $b \in \Sigma$. Therefore,

$$\sum_{b\in\Sigma} q_{y_j}^{x,a}(b) = \sum_{b\in\Sigma} p_{y_j}(b) = 1 .$$

- **Case 2: $f(x,y_j)=a$.** In this case, for every $b \neq a$, we have that:

$$q^{x,a}_{y_j}(b) = p_{y_j}(b) + \frac{\alpha}{(1-\alpha)} \cdot \frac{p_{y_j}(b)}{p_x(a)}$$

Observe that $\sum_{b\neq a} p_{y_j}(b) = 1 - p_{y_j}(a)$ (since they all sum-up to 1), and therefore we have:

$$\sum_{b\neq a} q^{x,a}_{y_j}(b) = \big(1-p_{y_j}(a)\big) + \frac{\alpha}{(1-\alpha)} \cdot \frac{1-p_{y_j}(a)}{p_x(a)} \ .$$

On the other hand, for $b=a$ we have: $q^{x,a}_{y_j}(a) = p_{y_j}(a) + \frac{\alpha}{(1-\alpha)} \cdot \frac{(p_{y_j}(a)-1)}{p_x(a)}$, and thus:

$$\begin{aligned}\sum_{b\in\Sigma} q^{x,a}_{y_j}(b) &= q^{x,a}_{y_j}(a) + \sum_{b\neq a} q^{x,a}_{y_j}(b) \\ &= \left(p_{y_j}(a) + \frac{\alpha}{(1-\alpha)} \cdot \frac{(p_{y_j}(a)-1)}{p_x(a)}\right) + \left(\big(1-p_{y_j}(a)\big) + \frac{\alpha}{(1-\alpha)} \cdot \frac{1-p_{y_j}(a)}{p_x(a)}\right) = 1 \ .\end{aligned}$$

Therefore, we can conclude that for every x,a it holds that: $Q^{x,a}(\sigma) = \mathbf{1}_m - \sum_{b\neq\sigma} Q^{x,a}(b)$. Thus, we conclude:

$$X^{x,a}_{ideal} \cdot M^{\sigma}_f = X^{x,a}_{ideal} \cdot \Big(\mathbf{J}_{\ell\times m} - \sum_{b\neq\sigma} M^b_f\Big) = \mathbf{1}_m - \sum_{b\neq\sigma} X^{x,a}_{ideal} \cdot M^b_f = \mathbf{1}_m - \sum_{b\neq\sigma} Q^{x,a}(b) = Q^{x,a}(\sigma) \ .$$

and the claim follows. ■

Encoding the requirements together. Let M_f denote the $\ell \times (|\Sigma|-1)\cdot m$ binary matrix defined as the concatenation of $M^{\sigma_1}_f || \ldots || M^{\sigma_{k-1}}_f$. Let $Q^{x,a}$ be the $(|\Sigma|-1)\cdot m$ vector which is a concatenation of $Q^{x,a} = (Q^{x,a}(\sigma_1), \ldots, Q^{x,a}(\sigma_{k-1}))$. Then we have that:

Corollary 5.5.5 *Let $f : \{x_1,\ldots,x_\ell\} \times \{y_1,\ldots,y_m\} \to \Sigma$, M_f, $Q^{x,a}$ be as above. If there exists a parameter $0<\alpha<1$ (for which $\alpha^{-1} \in O(\mathsf{poly}(\kappa))$), and a distribution X_{real}, such that for every $x \in X$, for every $a \in \Sigma$ it holds that:*

$$X^{x,a}_{ideal} \cdot M_f = Q^{x,a} \quad ,$$

then the function f can be computed with complete fairness.

As a result, if the rows of the matrix M_f define a full-dimensional object (this time, in $\mathbb{R}^{\ell\times(|\Sigma|-1)\cdot m}$), then the function can be computed fairly.

Corollary 5.5.6 *Let M_f be an $\ell \times ((|\Sigma|-1)\cdot m)$ matrix as above and let $X_1,\ldots,X_\ell$ be the rows of M_f. If there exists a subset of cardinality $(|\Sigma|-1)\cdot m+1$ of $\{X_1,\ldots,X_\ell\}$ that is affinely-independent, then f can be computed with complete fairness.*

Proof: Let $s = (|\Sigma|-1)\cdot m$. Fix $\alpha = 1/\ln(\kappa)$ and let X_{real} be the uniform distribution over the affinely independent rows in M_f (and assigns 0 to the others, if they exist). Similarly to the binary case, we observe that all the points $Q^{x,a}$ (this time - points in $\mathbb{R}^s$ and not $\mathbb{R}^m$) are in the Euclidian ball $B(\mathbf{c},\epsilon)$, where $\mathbf{c} = X_{real}\cdot M_f$. Moreover, since $\mathbf{conv}(\{X_1,\ldots,X_\ell\})$ defines an s-dimensional simplex in $\mathbb{R}^s$, the ball $B(\mathbf{c},\epsilon)$ is inside it, and therefore there exist probability vectors $X^{x,a}_{ideal}$ as above. ■

Alternative representation. We now write the above in a different form. Giving a function $f : X \times Y \to \Sigma$, let $\rho \in \Sigma$ be arbitrarily, and define $\Sigma_\rho = \Sigma \setminus \{\rho\}$. Define the *Boolean* function $f' : X \times Y^{\Sigma_\rho} \to \{0, 1\}$, where $Y^{\Sigma_\rho} = \{y_j^\sigma \mid y_j \in Y, \sigma \in \Sigma_\rho\}$, as follows:

$$f'(x, y_j^\sigma) = \begin{cases} 1 & \text{if } f(x, y_j) = \sigma \\ 0 & \text{otherwise} \end{cases}$$

Observe that $|Y^{\Sigma_\rho}| = (|\Sigma| - 1) \cdot |Y|$. It is easy to see that if f is full-dimensional, then the function f can be computed with complete-fairness. This provides a property that may be satisfied only when $|X|/|Y| > |\Sigma| - 1$.

An example. We give an example for a non-binary function that can be computed with complete-fairness. We consider the trinary alphabet $\Sigma = \{0, 1, 2\}$, and thus we consider a function of dimensions 5×2. We provide the trinary function f and the function f' that it is reduced to. Since the binary function f' is a full-dimensional function in $\mathbb{R}^4$, it can be computed fairly, and thus the trinary function f can be computed fairly as well. We have:

$\boldsymbol{f}$	y_1	y_2
x_1	0	1
x_2	1	0
x_3	1	1
x_4	2	0
x_5	1	2

$\Longrightarrow$

$\boldsymbol{f'}$	y_1^1	y_2^1	y_1^2	y_2^2
x_1	0	1	0	0
x_2	1	0	0	0
x_3	1	1	0	0
x_4	0	0	1	0
x_5	1	0	0	1

Appendix A

Full Specification of the BGW Protocol

In this Appendix, we provide a full specification of the BGW protocol, for both the semi-honest and malicious settings. We note that all these specifications of subprotocols and functionalities already appear in the body of this work (Chapter 2); however, a reader may find it beneficial and more convenient to read all the description centralized. In the electronic version of this thesis, the reader may also use the hyperlinks (appear in the **SEE:** label) to jump to the detailed explanation of each functionality and protocol.

A.1 The Protocol for Semi-Honest Adversaries

A.1.1 The Functionalities

FUNCTIONALITY A.1.1 (The degree-reduction functionality F_{reduce}^{deg})

SEE: Section 2.4.3

Let $h(x) = h_0 + \ldots + h_{2t}x^{2t}$ be a polynomial, and denote by $\mathsf{trunc}_t(h(x))$ the polynomial of degree t with coefficients $h_0, \ldots, h_t$. That is, $\mathsf{trunc}_t(h(x)) = h_0 + h_1 x + \ldots + h_t x^t$. Then:

$$F_{reduce}^{deg}(h(\alpha_1), \ldots, h(\alpha_n)) = (\hat{h}(\alpha_1), \ldots, \hat{h}(\alpha_n))$$

where $\hat{h}(x) = \mathsf{trunc}_t(h(x))$.

FUNCTIONALITY A.1.2 (The F_{rand}^{2t} functionality)

SEE: Section 2.4.3

$$F_{rand}^{2t}(\lambda, \ldots, \lambda) = (r(\alpha_1), \ldots, r(\alpha_n)),$$

where $r(x) \in_R \mathcal{P}^{0,2t}$ is random, and λ denotes the empty string.

FUNCTIONALITY A.1.3 (The F_{mult} for sharing a produce of shares)

SEE: Section 2.4.3.

$$F_{mult}\Big((f_a(\alpha_1), f_b(\alpha_1)), \ldots, (f_a(\alpha_n), f_b(\alpha_n))\Big) = \Big(f_{ab}(\alpha_1), \ldots, f_{ab}(\alpha_n)\Big)$$

where $f_a(x) \in \mathcal{P}^{a,t}$, $f_b(x) \in \mathcal{P}^{b,t}$, and $f_{ab}(x)$ is a *random* polynomial in $\mathcal{P}^{a \cdot b,t}$.

A.1.2 The Protocols

PROTOCOL A.1.4 (Computing F_{reduce}^{deg})

SEE: Claim 2.4.11

- **Inputs:** Each party P_i holds $h(\alpha_i)$.
- **The protocol:** The parties invoke the semi-honest BGW protocol (Protocol 2.4.1) for the linear functionality $\Big(\hat{h}(\alpha_1), \ldots, \hat{h}(\alpha_n)\Big)^T = A \cdot \Big(h(\alpha_1), \ldots, h(\alpha_n)\Big)^T$, where $A = V_{\vec{\alpha}} \cdot P_T \cdot V_{\vec{\alpha}}^{-1}$. ($V_{\vec{\alpha}}$ denote the Vandermonde matrix, and P_T is the linear projection of T).

PROTOCOL A.1.5 (Privately Computing F_{rand}^{2t})

SEE: Protocol 2.4.9

- **Input:** The parties do not have inputs for this protocol.
- **The protocol:**
  - Each party P_i chooses a random polynomial $q_i(x) \in_R \mathcal{P}^{0,2t}$. Then, for every $j \in \{1, \ldots, n\}$ it sends $s_{i,j} = q_i(\alpha_j)$ to party P_j.
  - Each party P_i receives $s_{1,i}, \ldots, s_{n,i}$ and computes $\delta_i = \sum_{j=1}^{n} s_{j,i}$.
- **Output:** Each party P_i outputs δ_i.

PROTOCOL A.1.6 (Computing F_{mult} in the $F_{rand}^{2t} - F_{reduce}^{deg}$-hybrid model)

SEE: Protocol 2.4.7.

- **Input:** Each party P_i holds values β_i, γ_i, such that $\mathsf{reconstruct}_{\vec{\alpha}}(\beta_1, \ldots, \beta_n) \in \mathcal{P}^{a,t}$ and $\mathsf{reconstruct}_{\vec{\alpha}}(\gamma_1, \ldots, \gamma_n) \in \mathcal{P}^{b,t}$ for some $a, b \in \mathbb{F}$.
- **The protocol:**
  1. Each party locally computes $s_i = \beta_i \cdot \gamma_i$.
  2. **Randomize:** Each party P_i sends λ to F_{rand}^{2t} (formally, it writes λ on its oracle tape for F_{rand}^{2t}). Let σ_i be the oracle response for party P_i.
  3. **Reduce the degree:** Each party P_i sends $(s_i + \sigma_i)$ to F_{reduce}^{deg}. Let δ_i be the oracle response for P_i.
- **Output:** Each party P_i outputs δ_i.

PROTOCOL A.1.7 (t-Private Computation in the F_{mult}-Hybrid Model)

SEE: Protocol 2.4.1.

- **Inputs:** Each party P_i has an input $x_i \in \mathbb{F}$.
- **Auxiliary input:** Each party P_i has an arithmetic circuit C over the field $\mathbb{F}$, such that for every $\vec{x} \in \mathbb{F}^n$ it holds that $C(\vec{x}) = f(\vec{x})$, where $f : \mathbb{F}^n \to \mathbb{F}^n$. The parties also have a description of $\mathbb{F}$ and distinct non-zero values $\alpha_1, \dots, \alpha_n$ in $\mathbb{F}$.
- **The protocol:**
  1. **The input sharing stage:** Each party P_i chooses a polynomial $q_i(x)$ uniformly from the set $\mathcal{P}^{x_i,t}$ of all polynomials of degree t with constant term x_i. For every $j \in \{1, \dots, n\}$, P_i sends party P_j the value $q_i(\alpha_j)$.

     Each party P_i records the values $q_1(\alpha_i), \dots, q_n(\alpha_i)$ that it received.
  2. **The circuit emulation stage:** Let $G_1, \dots, G_\ell$ be a predetermined topological ordering of the gates of the circuit. For $k = 1, \dots, \ell$ the parties work as follows:
     - *Case 1 – G_k is an addition gate:* Let β_i^k and γ_i^k be the shares of input wires held by party P_i. Then, P_i defines its share of the output wire to be $\delta_i^k = \beta_i^k + \gamma_i^k$.
     - *Case 2 – G_k is a multiplication-by-a-constant gate with constant c:* Let β_i^k be the share of the input wire held by party P_i. Then, P_i defines its share of the output wire to be $\delta_i^k = c \cdot \beta_i^k$.
     - *Case 3 – G_k is a multiplication gate:* Let β_i^k and γ_i^k be the shares of input wires held by party P_i. Then, P_i sends (β_i^k, γ_i^k) to the ideal functionality F_{mult} of Eq. (2.4.1) and receives back a value δ_i^k. Party P_i defines its share of the output wire to be δ_i^k.
  3. **The output reconstruction stage:** Let $o_1, \dots, o_n$ be the output wires, where party P_i's output is the value on wire o_i. For every $k = 1, \dots, n$, denote by $\beta_1^k, \dots, \beta_n^k$ the shares that the parties hold for wire o_k. Then, each P_i sends P_k the share β_i^k.

     Upon receiving all shares, P_k computes $\mathsf{reconstruct}_{\vec{\alpha}}(\beta_1^k, \dots, \beta_n^k)$ and obtains a polynomial $g_k(x)$ (note that $t + 1$ of the n shares suffice). P_k then defines its output to be $g_k(0)$.

A.2 The Protocol for Malicious Adversaries

A.2.1 The Functionalities

FUNCTIONALITY A.2.1 (The F_{VSS} functionality)

SEE: Functionality 2.5.5

$$F_{VSS}(q(x), \lambda, \dots, \lambda) = \begin{cases} (q(\alpha_1), \dots, q(\alpha_n)) & \text{if } \deg(q) \le t \\ (\perp, \dots, \perp) & \text{otherwise} \end{cases}$$

FUNCTIONALITY A.2.2 (The $\widetilde{F}_{VSS}$ functionality)

SEE: Functionality 2.5.6

$$\widetilde{F}_{VSS}(S(x,y),\lambda,\ldots,\lambda) = \begin{cases} ((f_1(x),g_1(y)),\ldots,(f_n(x),g_n(y))) & \text{if } \deg(S) \le t \\ (\perp,\ldots,\perp) & \text{otherwise} \end{cases},$$

where $f_i(x) = S(x,\alpha_i)$, $g_i(y) = S(\alpha_i,y)$.

FUNCTIONALITY A.2.3 (Functionality F^A_{mat} for matrix multiplication, with $A \in \mathbb{F}^{n\times m}$)

SEE: Functionality 2.6.4

The F^A_{mat}-functionality receives as input a set of indices $I \subseteq [n]$ and works as follows:

1. F^A_{mat} receives the inputs of the honest parties $\{g_j(x)\}_{j\notin I}$; if a polynomial $g_j(x)$ is not received or its degree is greater than t, then F^A_{mat} resets $g_j(x) = 0$.
2. F^A_{mat} sends shares $\{g_j(\alpha_i)\}_{j\notin I; i\in I}$ to the (ideal) adversary.
3. F^A_{mat} receives the corrupted parties' polynomials $\{g_i(x)\}_{i\in I}$ from the (ideal) adversary; if a polynomial $g_i(x)$ is not received or its degree is greater than t, then F^A_{mat} resets $g_i(x) = 0$.
4. F^A_{mat} computes $\vec{Y}(x) = (Y_1(x),\ldots,Y_m(x)) = (g_1(x),\ldots,g_n(x))\cdot A$.
5. (a) For every $j \notin I$, functionality F^A_{mat} sends party P_j the entire length-m vector $\vec{y} = \vec{Y}(0)$, together with P_j's shares $(g_1(\alpha_j),\ldots,g_n(\alpha_j))$ on the input polynomials.
   (b) In addition, functionality F^A_{mat} sends the (ideal) adversary its output: the vector of polynomials $\vec{Y}(x)$, and the corrupted parties' outputs ($\vec{y}$ together with $(g_1(\alpha_i),\ldots,g_n(\alpha_i))$, for every $i \in I$).

FUNCTIONALITY A.2.4 (Functionality $F^{subshare}_{VSS}$ for subsharing shares)

SEE: Functionality 2.6.7

$F^{subshare}_{VSS}$ receives a set of indices $I \subseteq [n]$ and works as follows:

1. $F^{subshare}_{VSS}$ receives the inputs of the honest parties $\{\beta_j\}_{j\notin I}$. Let $f(x)$ be the unique degree-t polynomial determined by the points $\{(\alpha_j,\beta_j)\}_{j\notin I}$.[1]
2. For every $j \notin I$, $F^{subshare}_{VSS}$ chooses a random degree-t polynomial $g_j(x)$ under the constraint that $g_j(0) = \beta_j = f(\alpha_j)$.
3. $F^{subshare}_{VSS}$ sends the shares $\{g_j(\alpha_i)\}_{j\notin I; i\in I}$ to the (ideal) adversary.
4. $F^{subshare}_{VSS}$ receives polynomials $\{g_i(x)\}_{i\in I}$ from the (ideal) adversary; if a polynomial $g_i(x)$ is not received or if $g_i(x)$ is of degree higher than t, then $F^{subshare}_{VSS}$ sets $g_i(x) = 0$.
5. *$F^{subshare}_{VSS}$ determines the output polynomials $g'_1(x),\ldots,g'_n(x)$:*
   (a) For every $j \notin I$, $F^{subshare}_{VSS}$ sets $g'_j(x) = g_j(x)$.

(b) For every $i \in I$, if $g_i(0) = f(\alpha_i)$ then $F_{VSS}^{subshare}$ sets $g_i'(x) = g_i(x)$. Otherwise it sets $g_i'(x) = f(\alpha_i)$, (i.e., $g_i'(x)$ is the constant polynomial equalling $f(\alpha_i)$ everywhere).

6. (a) For every $j \notin I$, $F_{VSS}^{subshare}$ sends the polynomial $g_j'(x)$ and the shares $(g_1'(\alpha_j), \ldots, g_n'(\alpha_j))$ to party P_j.

   (b) Functionality $F_{VSS}^{subshare}$ sends the (ideal) adversary the vector of polynomials $\vec{Y}(x) = (g_1(x), \ldots, g_n(x)) \cdot H^T$, where H is the parity-check matrix of the appropriate Reed-Solomon code (see below). In addition, it sends the corrupted parties' outputs $g_i'(x)$ and $(g_1'(\alpha_i), \ldots, g_n'(\alpha_i))$ for every $i \in I$.

FUNCTIONALITY A.2.5 (Functionality F_{eval}^k for evaluating a polynomial on α_k)

SEE: Functionality 2.6.10

F_{eval}^k receives a set of indices $I \subseteq [n]$ and works as follows:

1. The F_{eval}^k functionality receives the inputs of the honest parties $\{\beta_j\}_{j \notin I}$. Let $f(x)$ be the unique degree-t polynomial determined by the points $\{(\alpha_j, \beta_j)\}_{j \notin I}$. (If not all the points lie on a single degree-t polynomial, then no security guarantees are obtained; see Footnote 9.)
2. (a) For every $j \notin I$, F_{eval}^k sends the output pair $(f(\alpha_j), f(\alpha_k))$ to party P_j.

   (b) For every $i \in I$, F_{eval}^k sends the output pair $(f(\alpha_i), f(\alpha_k))$ to the (ideal) adversary, as the output of P_i.

FUNCTIONALITY A.2.6 (Functionality F_{VSS}^{mult} for sharing a product of shares)

SEE: Functionality 2.6.13

F_{VSS}^{mult} receives a set of indices $I \subseteq [n]$ and works as follows:

1. The F_{VSS}^{mult} functionality receives an input pair (a_j, b_j) from every honest party P_j ($j \notin I$). (The dealer P_1 also has polynomials $A(x), B(x)$ such that $A(\alpha_j) = a_j$ and $B(\alpha_j) = b_j$, for every $j \notin I$.)
2. F_{VSS}^{mult} computes the unique degree-t polynomials A and B such that $A(\alpha_j) = a_j$ and $B(\alpha_j) = b_j$ for every $j \notin I$ (if no such A or B exist of degree-t, then F_{VSS}^{mult} behaves differently as in Footnote 9).
3. If the dealer P_1 is honest ($1 \notin I$), then:

   (a) F_{VSS}^{mult} chooses a random degree-t polynomial C under the constraint that $C(0) = A(0) \cdot B(0)$.

   (b) *Outputs for honest:* F_{VSS}^{mult} sends the dealer P_1 the polynomial $C(x)$, and for every $j \notin I$ it sends $C(\alpha_j)$ to P_j.

   (c) *Outputs for adversary:* F_{VSS}^{mult} sends the shares $(A(\alpha_i), B(\alpha_i), C(\alpha_i))$ to the (ideal) adversary, for every $i \in I$.
4. If the dealer P_1 is corrupted ($1 \in I$), then:

   (a) F_{VSS}^{mult} sends $(A(x), B(x))$ to the (ideal) adversary.

   (b) F_{VSS}^{mult} receives a polynomial C as input from the (ideal) adversary.

   (c) If either $\deg(C) > t$ or $C(0) \neq A(0) \cdot B(0)$, then F_{VSS}^{mult} resets $C(x) = A(0) \cdot B(0)$; that is, the constant polynomial equalling $A(0) \cdot B(0)$ everywhere.

   (d) *Outputs for honest:* F_{VSS}^{mult} sends $C(\alpha_j)$ to P_j, for every $j \notin I$.
   (There is no more output for the adversary in this case.)

FUNCTIONALITY A.2.7 (Functionality F_{mult} for emulating a multiplication gate)

SEE: Functionality 2.6.16

F_{mult} receives a set of indices $I \subseteq [n]$ and works as follows:

1. The F_{mult} functionality receives the inputs of the honest parties $\{(\beta_j, \gamma_j)\}_{j \notin I}$. Let $f_a(x), f_b(x)$ be the unique degree-t polynomials determined by the points $\{(\alpha_j, \beta_j)\}_{j \notin I}$, $\{(\alpha_j, \gamma_j)\}_{j \notin I}$, respectively. (If such polynomials do not exist then no security is guaranteed; see Footnote 9.)
2. F_{mult} sends $\{(f_a(\alpha_i), f_b(\alpha_i))\}_{i \in I}$ to the (ideal) adversary.
3. F_{mult} receives points $\{\delta_i\}_{i \in I}$ from the (ideal) adversary (if some δ_i is not received, then it is set to equal 0).
4. F_{mult} chooses a random degree-t polynomial $f_{ab}(x)$ under the constraints that:
   (a) $f_{ab}(0) = f_a(0) \cdot f_b(0)$, and
   (b) For every $i \in I$, $f_{ab}(\alpha_i) = \delta_i$.

   (such a degree-t polynomial always exists since $|I| \leq t$).
5. The functionality F_{mult} sends the value $f_{ab}(\alpha_j)$ to every honest party P_j $(j \notin I)$.

A.2.2 The Protocols

PROTOCOL A.2.8 (Securely Computing F_{VSS} in the $\widetilde{F}_{VSS}$-hybrid model)

SEE: Protocol 2.5.7

- **Input:** The dealer $D = P_1$ holds a polynomial $q(x)$ of degree at most t (if not, then the honest dealer just aborts at the onset). The other parties $P_2, \ldots, P_n$ have no input.
- **Common input:** The description of a field $\mathbb{F}$ and n non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **The protocol:**
  1. The dealer selects a uniformly distributed bivariate polynomial $S(x, y) \in \mathcal{B}^{q(0),t}$, under the constraint that $S(0, z) = q(z)$.
  2. The parties invoke the $\widetilde{F}_{VSS}$ functionality where P_1 is dealer and inputs $S(x, y)$, each other party has no input.
- **Output** If the output of $\widetilde{F}_{VSS}$ is $(f_i(x), g_i(y))$, output $f_i(0)$. Otherwise, output $\perp$.

PROTOCOL A.2.9 (Securely Computing $\widetilde{F}_{VSS}$)

SEE: Protocol 2.5.9

- **Input:** The dealer $D = P_1$ holds a bivariate polynomial $S(x, y)$ of degree at most t in both variables (if not, then the honest dealer just aborts at the onset). The other parties $P_2, \ldots, P_n$ have no input.
- **Common input:** The description of a field $\mathbb{F}$ and n non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **The protocol:**

1. **Round 1 (send shares) – the dealer:**
   (a) For every $i \in \{1, \ldots, n\}$, the dealer defines the polynomials $f_i(x) \stackrel{\text{def}}{=} S(x, \alpha_i)$ and $g_i(y) \stackrel{\text{def}}{=} S(\alpha_i, y)$. It then sends to each party P_i the polynomials $f_i(x)$ and $g_i(y)$.
2. **Round 2 (exchange subshares) – each party P_i:**
   (a) Store the polynomials $f_i(x)$ and $g_i(y)$ that were received from the dealer. (If $f_i(x)$ or $g_i(y)$ is of degree greater than t then truncate it to be of degree t.)
   (b) For every $j \in \{1, \ldots, n\}$, send $f_i(\alpha_j)$ and $g_i(\alpha_j)$ to party P_j.
3. **Round 3 (broadcast complaints) – each party P_i:**
   (a) For every $j \in \{1, \ldots, n\}$, let (u_j, v_j) denote the values received from player P_j in Round 2 (these are supposed to be $u_j = f_j(\alpha_i)$ and $v_j = g_j(\alpha_i)$).
   If $u_j \neq g_i(\alpha_j)$ or $v_j \neq f_i(\alpha_j)$, then broadcast $\mathsf{complaint}(i, j, f_i(\alpha_j), g_i(\alpha_j))$.
   (b) If no parties broadcast a complaint, then every party P_i outputs $f_i(0)$ and halts.
4. **Round 4 (resolve complaints) – the dealer:** For every complaint message received, do the following:
   (a) Upon viewing a message $\mathsf{complaint}(i, j, u, v)$ broadcast by P_i, check that $u = S(\alpha_j, \alpha_i)$ and $v = S(\alpha_i, \alpha_j)$. (Note that if the dealer and P_i are honest, then it holds that $u = f_i(\alpha_j)$ and $v = g_i(\alpha_j)$.) If the above condition holds, then do nothing. Otherwise, broadcast $\mathsf{reveal}(i, f_i(x), g_i(y))$.
5. **Round 5 (evaluate complaint resolutions) – each party P_i:**
   (a) For every $j \neq k$, party P_i marks (j, k) as a joint complaint if it viewed two messages $\mathsf{complaint}(k, j, u_1, v_1)$ and $\mathsf{complaint}(j, k, u_2, v_2)$ broadcast by P_k and P_j, respectively, such that $u_1 \neq v_2$ or $v_1 \neq u_2$. If there exists a joint complaint (j, k) for which the dealer did not broadcast $\mathsf{reveal}(k, f_k(x), g_k(y))$ nor $\mathsf{reveal}(j, f_j(x), g_j(y))$, then go to Step 6 (and do not broadcast consistent). Otherwise, proceed to the next step.
   (b) Consider the set of $\mathsf{reveal}(j, f_j(x), g_j(y))$ messages sent by the dealer (truncating the polynomials to degree t if necessary as in Step 2a):
      i. If there exists a message in the set with $j = i$ then reset the stored polynomials $f_i(x)$ and $g_i(y)$ to the new polynomials that were received, and go to Step 6 (without broadcasting consistent).
      ii. If there exists a message in the set with $j \neq i$ and for which $f_i(\alpha_j) \neq g_j(\alpha_i)$ or $g_i(\alpha_j) \neq f_j(\alpha_i)$, then go to Step 6 (without broadcasting consistent).
   If the set of reveal messages does not contain a message that fulfills either one of the above conditions, then proceed to the next step.
   (c) Broadcast the message consistent.
6. **Output decision (if there were complaints) – each party P_i:** If at least $n - t$ parties broadcast consistent, output $(f_i(x), g_i(y))$. Otherwise, output $\perp$.

PROTOCOL A.2.10 (Securely computing F^A_{mat} in the F_{VSS}-hybrid model)

SEE: Protocol 2.6.5.

- **Inputs:** Each party P_i holds a polynomial $g_i(x)$.
- **Common input:** A field description $\mathbb{F}$, n distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$, and a matrix $A \in \mathbb{F}^{n \times m}$.
- **Aiding ideal functionality initialization:** Upon invocation, the trusted party computing the (fictitiously corruption-aware) functionality F_{VSS} is given the set of corrupted parties I.

- **The protocol:**

  1. Each party P_i checks that its input polynomial is of degree-t; if not, it resets $g_i(x) = 0$. It then invokes the F_{VSS} functionality as dealer with $g_i(x)$ as its private input.
  2. At the end of Step 1, each party P_i holds the values $g_1(\alpha_i), \ldots, g_n(\alpha_i)$. If any value equals $\perp$, then P_i replaces it with 0.
  3. Denote $\vec{x}^i = (g_1(\alpha_i), \ldots, g_n(\alpha_i))$. Then, each party P_i locally computes $\vec{y}^i = \vec{x}^i \cdot A$ (equivalently, for every $k = 1, \ldots, m$, each P_i computes $Y_k(\alpha_i) = \sum_{\ell=1}^{n} g_\ell(\alpha_i) \cdot a_{\ell,k}$ where $(a_{1,k}, \ldots, a_{n,k})^T$ is the kth column of A, and stores $\vec{y}^i = (Y_1(\alpha_i), \ldots, Y_m(\alpha_i))$).
  4. Each party P_i sends $\vec{y}^i$ to every P_j $(1 \leq j \leq n)$.
  5. For every $j = 1, \ldots, n$, denote the vector received by P_i from P_j by $\vec{\hat{Y}}(\alpha_j) = (\hat{Y}_1(\alpha_j), \ldots, \hat{Y}_m(\alpha_j))$. (If any value is missing, it replaces it with 0. We stress that different parties may hold different vectors if a party is corrupted.) Each P_i works as follows:
     – For every $k = 1, \ldots, m$, party P_i locally runs the Reed-Solomon decoding procedure (with $d = 2t + 1$) on the possibly corrupted codeword $(\hat{Y}_k(\alpha_1), \ldots, \hat{Y}_k(\alpha_n))$ to get the codeword $(Y_k(\alpha_1), \ldots, Y_k(\alpha_n))$; see Figure 2.1. It then reconstructs the polynomial $Y_k(x)$ and computes $y_k = Y_k(0)$.

- **Output:** P_i outputs $(y_1, \ldots, y_m)$ as well as the shares $g_1(\alpha_i), \ldots, g_n(\alpha_i)$.

PROTOCOL A.2.11 (Securely computing $F_{VSS}^{subshare}$ in the F_{mat}^{H}-hybrid model)

SEE: Protocol 2.6.8

- **Inputs:** Each party P_i holds a value β_i; we assume that the points (α_j, β_j) of the honest parties all lie on a single degree-t polynomial (see the definition of $F_{VSS}^{subshare}$ above and Footnote 9 therein).
- **Common input:** The matrix $H \in \mathbb{F}^{2t \times n}$ which is the parity-check matrix of the Reed-Solomon code.
- **The protocol:**

  1. Each party P_i chooses a random degree-t polynomial $g_i(x)$ under the constraint that $g_i(0) = \beta_i$
  2. The parties invoke the F_{mat}^{H} functionality (i.e., Functionality 2.6.4 for matrix multiplication with the transpose of the parity-check matrix H). Each party P_i inputs the polynomial $g_i(x)$ from the previous step, and receives from F_{mat}^{H} as output the shares $g_1(\alpha_i), \ldots, g_n(\alpha_i)$, the degree-$t$ polynomial $g_i(x)$ and the length $2t$ vector $\vec{s} = (s_1, \ldots, s_{2t}) = (g_1(0), \ldots, g_n(0)) \cdot H^T$. Recall that $\vec{s}$ is the syndrome vector of the possible corrupted codeword $\vec{\gamma} = (g_1(0), \ldots, g_n(0))$.
  3. Each party locally runs the Reed-Solomon decoding procedure using $\vec{s}$ only, and receives back an error vector $\vec{e} = (e_1, \ldots, e_n)$.
  4. For every k such that $e_k = 0$: each party P_i sets $g'_k(\alpha_i) = g_k(\alpha_i)$.
  5. For every k such that $e_k \neq 0$:
     (a) Each party P_i sends $g_k(\alpha_i)$ to every P_j.
     (b) Each party P_i receives $g_k(\alpha_1), \ldots, g_k(\alpha_n)$; if any value is missing, it sets it to 0. P_i runs the Reed-Solomon decoding procedure on the values to reconstruct $g_k(x)$.
     (c) Each party P_i computes $g_k(0)$, and sets $g'_k(\alpha_i) = g_k(0) - e_k$ (which equals $f(\alpha_k)$).

- **Output:** P_i outputs $g_i(x)$ and $g'_1(\alpha_i), \ldots, g'_n(\alpha_i)$.

PROTOCOL A.2.12 (Securely computing F^k_{eval} in the $F^{subshare}_{VSS}$-hybrid model)

SEE: Protocol 2.6.11.

- **Inputs:** Each party P_i holds a value β_i; we assume that the points (α_j, β_j) for every honest P_j all lie on a single degree-t polynomial f (see the definition of F^k_{eval} above and Footnote 9).
- **Common input:** The description of a field $\mathbb{F}$ and n distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **Aiding ideal functionality initialization:** Upon invocation, the trusted party computing the corruption-aware functionality $F^{subshare}_{VSS}$ receives the set of corrupted parties I.
- **The protocol:**
  1. The parties invoke the $F^{subshare}_{VSS}$ functionality with each party P_i using β_i as its private input. At the end of this stage, each party P_i holds $g'_1(\alpha_i), \ldots, g'_n(\alpha_i)$, where all the $g'_i(x)$ are of degree t, and for every i, $g'_i(0) = f(\alpha_i)$.
  2. Each party P_i locally computes: $Q(\alpha_i) = \sum_{\ell=1}^{n} \lambda_\ell \cdot g'_\ell(\alpha_i)$, where $(\lambda_1, \ldots, \lambda_n) = \vec{\alpha}_k \cdot V_{\vec{\alpha}}^{-1}$. Each party P_i sends $Q(\alpha_i)$ to all P_j.
  3. Each party P_i receives all the shares $\hat{Q}(\alpha_j)$ from each other party $1 \leq j \leq n$ (if any value is missing, replace it with 0). Note that some of the parties may hold different values if a party is corrupted. Then, given the possibly corrupted codeword $(\hat{Q}(\alpha_1), \ldots, \hat{Q}(\alpha_n))$, each party runs the Reed-Solomon decoding procedure and receives the codeword $(Q(\alpha_1), \ldots, Q(\alpha_n))$. It then reconstructs $Q(x)$ and computes $Q(0)$.
- **Output:** Each party P_i outputs $(\beta_i, Q(0))$.

PROTOCOL A.2.13 (Securely computing F^{mult}_{VSS} in the F_{VSS}-F_{eval}-hybrid model)

SEE: Protocol 2.6.15.

- **Input:**
  1. The dealer P_1 holds two degree-t polynomials A and B.
  2. Each party P_i holds a pair of shares a_i and b_i such that $a_i = A(\alpha_i)$ and $b_i = B(\alpha_i)$.
- **Common input:** A field description $\mathbb{F}$ and n distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **Aiding ideal functionality initialization:** Upon invocation, the trusted party computing the (fictitiously corruption-aware) functionality F_{VSS} and the corruption-aware functionality F_{eval} receives the set of corrupted parties I.
- **The protocol:**
  1. *Dealing phase:*
     (a) The dealer P_1 defines the degree-$2t$ polynomial $D(x) = A(x) \cdot B(x)$; denote $D(x) = a \cdot b + \sum_{\ell=1}^{2t} d_\ell \cdot x^\ell$.
     (b) P_1 chooses t^2 values $\{r_{k,j}\}$ uniformly and independently at random from $\mathbb{F}$, where $k = 1, \ldots, t$, and $j = 0, \ldots, t-1$.
     (c) For every $\ell = 1, \ldots, t$, the dealer P_1 defines the polynomial $D_\ell(x)$:
     $$D_\ell(x) = \left(\sum_{m=0}^{t-1} r_{\ell,m} \cdot x^m \right) + \left(d_{\ell+t} - \sum_{m=\ell+1}^{t} r_{m,t+\ell-m} \right) \cdot x^t.$$
     (d) P_1 computes the polynomial:
     $$C(x) = D(x) - \sum_{\ell=1}^{t} x^\ell \cdot D_\ell(x).$$

   (e) P_1 invokes the F_{VSS} functionality as dealer with input $C(x)$; each party P_i receives $C(\alpha_i)$.
   (f) P_1 invokes the F_{VSS} functionality as dealer with input $D_\ell(x)$ for every $\ell = 1, \ldots, t$; each party P_i receives $D_\ell(\alpha_i)$.
2. *Verify phase:* Each party P_i works as follows:
   (a) If any of the $C(\alpha_i), D_\ell(\alpha_i)$ values equals $\perp$ then P_i proceeds to the *reject phase* (note that if one honest party received $\perp$ then all did).
   (b) Otherwise, P_i computes $c'_i = a_i \cdot b_i - \sum_{\ell=1}^{t} (\alpha_i)^\ell \cdot D_\ell(\alpha_i)$. If $c'_i \neq C(\alpha_i)$ then P_i broadcasts $(\mathsf{complaint}, i)$.
   (c) If any party P_k broadcast $(\mathsf{complaint}, k)$ then go to the *complaint resolution phase.*
3. *Complaint resolution phase:* Run the following for every $(\mathsf{complaint}, k)$ message:
   (a) Run $t + 3$ invocations of F^k_{eval}: in each of the invocations each party P_i inputs the corresponding value $a_i, b_i, C(\alpha_i), D_1(\alpha_i), \ldots, D_t(\alpha_i)$.
   (b) Let $A(\alpha_k), B(\alpha_k), \tilde{C}(\alpha_k), \tilde{D}_1(\alpha_k), \ldots, \tilde{D}_t(\alpha_k)$ be the respective outputs that all parties receive from the invocations. Compute $\tilde{C}'(\alpha_k) = A(\alpha_k) \cdot B(\alpha_k) - \sum_{\ell=1}^{t} (\alpha_k)^\ell \cdot \tilde{D}_\ell(\alpha_k)$. (We denote these polynomials by $\tilde{C}, \tilde{D}_\ell, \ldots$ since if the dealer is not honest they may differ from the specified polynomials above.)
   (c) If $\tilde{C}(\alpha_k) \neq \tilde{C}'(\alpha_k)$, then proceed to the *reject phase.*
4. *Reject phase* (skip to the output if not explicitly instructed to run the reject phase):
   (a) Every party P_i broadcasts the pair (a_i, b_i). Let $\vec{a} = (a_1, \ldots, a_n)$ and $\vec{b} = (b_1, \ldots, b_n)$ be the broadcast values (where zero is used for any value not broadcast). Then, P_i computes $A'(x)$ and $B'(x)$ to be the outputs of Reed-Solomon decoding on $\vec{a}$ and $\vec{b}$, respectively.
   (b) Every party P_i sets $C(\alpha_i) = A'(0) \cdot B'(0)$.

- **Output:** Every party P_i outputs $C(\alpha_i)$.

PROTOCOL A.2.14 (Computing F_{mult} in the $(F^{subshare}_{VSS}, F^{mult}_{VSS})$-hybrid model)

SEE: Protocol 2.6.17.

- **Input:** Each party P_i holds a_i, b_i, where $a_i = f_a(\alpha_i)$, $b_i = f_b(\alpha_i)$ for some polynomials $f_a(x), f_b(x)$ of degree t, which hide a, b, respectively. (If not all the points lie on a single degree-t polynomial, then no security guarantees are obtained. See Footnote 9.)
- **Common input:** A field description $\mathbb{F}$ and n distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **Aiding ideal functionality initialization:** Upon invocation, the trusted party computing the corruption-aware functionalities $F^{subshare}_{VSS}$ and F^{mult}_{VSS} receives the set of corrupted parties I.
- **The protocol:**
  1. The parties invoke the $F^{subshare}_{VSS}$ functionality with each party P_i using a_i as its private input. Each party P_i receives back shares $A_1(\alpha_i), \ldots, A_n(\alpha_i)$, and a polynomial $A_i(x)$. (Recall that for every i, the polynomial $A_i(x)$ is of degree-t and $A_i(0) = f_a(\alpha_i) = a_i$.)
  2. The parties invoke the $F^{subshare}_{VSS}$ functionality with each party P_i using b_i as its private input. Each party P_i receives back shares $B_1(\alpha_i), \ldots, B_n(\alpha_i)$, and a polynomial $B_i(x)$.
  3. For every $i = 1, \ldots, n$, the parties invoke the F^{mult}_{VSS} functionality as follows:
     (a) *Inputs:* In the ith invocation, party P_i plays the dealer. All parties P_j $(1 \leq j \leq n)$ send F^{mult}_{VSS} their shares $A_i(\alpha_j), B_i(\alpha_j)$.

   (b) *Outputs:* The dealer P_i receives $C_i(x)$ where $C_i(x) \in_R \mathcal{P}^{A_i(0)\cdot B_i(0),t}$, and every party P_j $(1 \leq j \leq n)$ receives the value $C_i(\alpha_j)$.

4. At this stage, each party P_i holds values $C_1(\alpha_i), \ldots, C_n(\alpha_i)$, and locally computes $Q(\alpha_i) = \sum_{\ell=1}^{n} \lambda_\ell \cdot C_\ell(\alpha_i)$, where $(\lambda_1, \ldots, \lambda_n)$ is the first row of the matrix $V_{\vec{\alpha}}^{-1}$.

- **Output:** Each party P_i outputs $Q(\alpha_i)$.

PROTOCOL A.2.15 (t-Secure Computation of f in the (F_{mult}, F_{VSS})-Hybrid Model)

SEE: Protocol 2.7.1.

- **Inputs:** Each party P_i has an input $x_i \in \mathbb{F}$.
- **Common input:** Each party P_i holds an arithmetic circuit C over a field $\mathbb{F}$ of size greater than n, such that for every $\vec{x} \in \mathbb{F}^n$ it holds that $C(\vec{x}) = f(\vec{x})$, where $f : \mathbb{F}^n \to \mathbb{F}^n$. The parties also hold a description of $\mathbb{F}$ and distinct non-zero values $\alpha_1, \ldots, \alpha_n$ in $\mathbb{F}$.
- **Aiding ideal functionality initialization:** Upon invocation, the trusted parties computing the (fictitiously corruption-aware) functionality F_{VSS} and the corruption-aware functionality F_{mult} receive the set of corrupted parties I.
- **The protocol:**
  1. **The input sharing stage:**
     (a) Each party P_i chooses a polynomial $q_i(x)$ uniformly at random from the set $\mathcal{P}^{x_i,t}$ of degree-t polynomials with constant-term x_i. Then, P_i invokes the F_{VSS} functionality as dealer, using $q_i(x)$ as its input.
     (b) Each party P_i records the values $q_1(\alpha_i), \ldots, q_n(\alpha_i)$ that it received from the F_{VSS} functionality invocations. If the output from F_{VSS} is $\perp$ for any of these values, P_i replaces the value with 0.
  2. **The circuit emulation stage:** Let $G_1, \ldots, G_\ell$ be a predetermined topological ordering of the gates of the circuit. For $k = 1, \ldots, \ell$ the parties work as follows:
     - *Case 1 – G_k is an addition gate:* Let β_i^k and γ_i^k be the shares of input wires held by party P_i. Then, P_i defines its share of the output wire to be $\delta_i^k = \beta_i^k + \gamma_i^k$.
     - *Case 2 – G_k is a multiplication-by-a-constant gate with constant c:* Let β_i^k be the share of the input wire held by party P_i. Then, P_i defines its share of the output wire to be $\delta_i^k = c \cdot \beta_i^k$.
     - *Case 3 – G_k is a multiplication gate:* Let β_i^k and γ_i^k be the shares of input wires held by party P_i. Then, P_i sends (β_i^k, γ_i^k) to the ideal functionality F_{mult} and receives back a value δ_i^k. Party P_i defines its share of the output wire to be δ_i^k.
  3. **The output reconstruction stage:**
     (a) Let $o_1, \ldots, o_n$ be the output wires, where party P_i's output is the value on wire o_i. For every $i = 1, \ldots, n$, denote by $\beta_1^i, \ldots, \beta_n^i$ the shares that the parties hold for wire o_i. Then, each P_j sends P_i the share β_j^i.
     (b) Upon receiving all shares, P_i runs the Reed-Solomon decoding procedure on the possible corrupted codeword $(\beta_1^i, \ldots, \beta_n^i)$ to obtain a codeword $(\tilde{\beta}_1^i, \ldots, \tilde{\beta}_n^i)$. Then, P_i computes $\mathsf{reconstruct}_{\vec{\alpha}}(\tilde{\beta}_1^i, \ldots, \tilde{\beta}_n^i)$ and obtains a polynomial $g_i(x)$. Finally, P_i then defines its output to be $g_i(0)$.

Appendix B

Full Specification of the Efficient Perfectly-Secure Multiplication Protocol

In this Appendix, we provide a full specification of the efficient perfectly-secure multiplication protocol that was given in Chapter 3. This is similarly to the previous appendix that gives full specification of the BGW protocol (Chapter 2).

B.1 Functionalities

FUNCTIONALITY B.1.1 (The $\widetilde{F}_{VSS}$ functionality)

SEE: Functionality 2.5.6

$$\widetilde{F}_{VSS}(S(x,y),\lambda,\ldots,\lambda) = \begin{cases} ((f_1(x),g_1(y)),\ldots,(f_n(x),g_n(y))) & \text{if } \deg(S) \leq t \\ (\perp,\ldots,\perp) & \text{otherwise} \end{cases},$$

where $f_i(x) = S(x,\alpha_i)$, $g_i(y) = S(\alpha_i,y)$.

FUNCTIONALITY B.1.2 (The $\widetilde{F}_{extend}$ Functionality for Extending Shares)

SEE: Functionality 3.3.1

$\widetilde{F}_{extend}$ receives a set of indices $I \subseteq [n]$ and works as follows:

1. The $\widetilde{F}_{extend}$ functionality receives the shares of the honest parties $\{\beta_j\}_{j\notin I}$. Let $q(x)$ be the unique degree-t polynomial determined by the points $\{(\alpha_j,\beta_j)\}_{j\notin I}$. (If no such polynomial exists then no security is guaranteed[1].)
2. In case that the dealer is corrupted, $\widetilde{F}_{extend}$ sends $q(x)$ to the (ideal) adversary.
3. $\widetilde{F}_{extend}$ receives $S(x,y)$ from the dealer. Then, it checks that $S(x,y)$ is of degree-t in both variables *and* $S(x,0) = q(x)$.
4. If both condition holds, define the output of P_i to be the pair of univariate polynomials $\langle S(x,\alpha_i), S(\alpha_i,y)\rangle$. Otherwise, define the output of P_i to be $\perp$.

5. (a) $\widetilde{F}_{extend}$ sends the outputs to each honest party P_j $(j \notin I)$.
   (b) $\widetilde{F}_{extend}$ sends the output of each corrupted party P_i $(i \in I)$ to the (ideal) adversary.

FUNCTIONALITY B.1.3 (The Functionality $\widetilde{F}^{k}_{eval}$)

SEE: Functionality 3.3.4

$\widetilde{F}^{k}_{eval}$ receives a set of indices $I \subseteq [n]$ and works as follows:

1. The $\widetilde{F}^{k}_{eval}$ functionality receives from each honest party P_j the pair of degree-t polynomials $(f_j(x), g_j(y))$, for every $j \notin I$. Let $S(x, y)$ be the single bivariate polynomial with degree-t in both variables that satisfies $S(x, \alpha_j) = f_j(x)$, $S(\alpha_j, y) = g_j(y)$ for every $j \notin I$. (If no such $S(x, y)$ exists, then no security is guaranteed; see Footnote 2).
2. (a) For every $j \notin I$, F^{k}_{eval} sends the output pair $\langle S(x, \alpha_k), S(\alpha_k, y)\rangle$.
   (b) In addition, for every $i \in I$, F^{k}_{eval} sends the output pair $\langle S(x, \alpha_k), S(\alpha_k, y)\rangle$ to the (ideal) adversary.

FUNCTIONALITY B.1.4 (The $\widetilde{F}^{mult}_{VSS}$ functionality for sharing a product of shares)

SEE: Functionality 3.3.7

$\widetilde{F}^{mult}_{VSS}$ receives a set of indices $I \subseteq [n]$ and works as follows:

1. The $\widetilde{F}^{mult}_{VSS}$ functionality receives an input pair (a_j, b_j) from every honest party P_j $(j \notin I)$. The dealer P_1 has polynomials $A(x), B(x)$ such that $A(\alpha_j) = a_j$ and $B(\alpha_j) = b_j$, for every j.
2. $\widetilde{F}^{mult}_{VSS}$ computes the unique degree-t univariate polynomials A and B such that $A(\alpha_j) = a_j$ and $B(\alpha_j) = b_j$ for every $j \notin I$ (if no such A or B exist of degree-t, then $\widetilde{F}^{mult}_{VSS}$ behaves differently as in Footnote 2).
3. If the dealer P_1 is honest $(1 \notin I)$, then:
   (a) $\widetilde{F}^{mult}_{VSS}$ chooses a random degree-t bivariate polynomial $C(x, y)$ under the constraint that $C(0, 0) = A(0) \cdot B(0)$.
   (b) *Outputs for honest:* $\widetilde{F}^{mult}_{VSS}$ sends the dealer P_1 the polynomial $C(x, y)$, and for every $j \notin I$ it sends the bivariate shares $\langle C(x, \alpha_j), C(\alpha_j, y)\rangle$.
   (c) *Output for adversary:* For every $i \in I$, the functionality $\widetilde{F}^{mult}_{VSS}$ sends the univariate shares $A(\alpha_i), B(\alpha_i)$, and the bivariate shares $\langle C(x, \alpha_i), C(\alpha_i, y)\rangle$.
4. If the dealer P_1 is corrupted $(1 \in I)$, then:
   (a) $\widetilde{F}^{mult}_{VSS}$ sends $(A(x), B(x))$ to the (ideal) adversary.
   (b) $\widetilde{F}^{mult}_{VSS}$ receives a polynomial C as input from the (ideal) adversary.
   (c) If either $\deg(C) > t$ or $C(0, 0) \neq A(0) \cdot B(0)$, then $\widetilde{F}^{mult}_{VSS}$ resets $C(x, y) = A(0) \cdot B(0)$. That is, the constant polynomial equalling $A(0) \cdot B(0)$ everywhere.
   (d) *Output for honest:* $\widetilde{F}^{mult}_{VSS}$ sends the bivariate shares $\langle C(x, \alpha_j), C(\alpha_j, y)\rangle$ to P_j for every $j \notin I$. (There is no more output for the adversary in this case.)

FUNCTIONALITY B.1.5 (Functionality $\widetilde{F}_{mult}$ for emulating a multiplication gate)

SEE: Functionality 3.3.10

$\widetilde{F}_{mult}$ receives a set of indices $I \subseteq [n]$ and works as follows:

1. The $\widetilde{F}_{mult}$ functionality receives the inputs of the honest parties $\{\langle f_j^a(x), g_j^a(y)\rangle, \langle f_j^b(x), g_j^b(y)\rangle\}_{j\notin I}$. Let $A(x,y), B(x,y)$ be the unique degree-t bivariate polynomials determined by these, respectively. (If such polynomials do not exist then no security is guaranteed; see Footnote 2.)
2. $\widetilde{F}_{mult}$ sends $\{\langle A(x,\alpha_i), A(\alpha_i,y)\rangle, \langle B(x,\alpha_i), B(\alpha_i,y)\rangle\}_{i\in I}$ to the (ideal) adversary.
3. $\widetilde{F}_{mult}$ receives a degree-t bivariate polynomial $H(x,y)$ from the (ideal) adversary. If the adversary sends a polynomial of higher degree, truncate it to polynomial of degree-t in both variables.
4. $\widetilde{F}_{mult}$ defines a degree-t bivariate polynomial $C(x,y)$ such that:
   (a) $C(0,0) = A(0,0)\cdot B(0,0)$,
   (b) Let $\mathcal{T}$ be a set of size exactly t indices such that $I \subseteq \mathcal{T}$, where $\mathcal{T}$ is fully determined from I; Then, for every $i \in \mathcal{T}$, set $C(x,\alpha_i) = H(x,\alpha_i)$ and $C(\alpha_i,y) = H(\alpha_i,y)$.

   (such a degree-t polynomial always exists from Claim 3.2.1.)
5. *Output:* The functionality $\widetilde{F}_{mult}$ sends the polynomial $\langle C(x,\alpha_j), C(\alpha_j,y)\rangle$ to every honest party P_j ($j \notin I$).
   (There is no more output for the adversary.)

B.2 The Protocols

PROTOCOL B.2.1 (Securely Computing $\widetilde{F}_{extend}$ in the $\widetilde{F}_{VSS}$-Hybrid Model)

SEE: Protocol 3.3.2

- **Input:** The dealer P_1 holds a univariate polynomial of degree-t, $q(x)$, and a degree-t bivariate polynomial $S(x,y)$ that satisfies $S(x,0) = q(x)$. Each party P_i holds $q(\alpha_i)$.
- **Common input:** A field description $\mathbb{F}$ and n distinct non-zero elements $\alpha_1,\ldots,\alpha_n \in \mathbb{F}$.
- **Aiding ideal functionality initialization:** Upon invocation, the trusted party computing the (fictitiously corruption-aware) functionality $\widetilde{F}_{VSS}$ receives the set of corrupted parties I.
- **The protocol:**
  1. The parties invoke the $\widetilde{F}_{VSS}$ functionality, where P_1 (the dealer) uses the bivariate polynomial $S(x,y)$, and any other party inputs λ (the empty string).
  2. If $\widetilde{F}_{VSS}$ returns $\perp$, each outputs $\perp$ and halts.
  3. Otherwise, let $\langle S(x,\alpha_i), S(\alpha_i,y)\rangle$ be the output shares of $\widetilde{F}_{VSS}$. If $S(\alpha_i,0) \neq q(\alpha_i)$, then broadcast $\mathsf{complaint}(i)$.
- **Output:** If more than t parties broadcast $\mathsf{complaint}$, output $\perp$. Otherwise, output $\langle f_i(x), g_i(y)\rangle = \langle S(x,\alpha_i), S(\alpha_i,y)\rangle$.

PROTOCOL B.2.2 (Securely Computing $\widetilde{F}^{k}_{eval}$)

SEE: Protocol 3.3.5

- **Input:** Each party holds two degree-t polynomials $f_i(x), g_i(y)$.

  (It is assumed that there there exists a single bivariate polynomial $S(x, y)$ such that for every $i \in [n]$: $S(x, \alpha_i) = f_i(x)$ and $S(\alpha_i, y) = g_i(y)$. If such a polynomial does not exists, then no security is guaranteed; see the definition of the functionality).
- **Common input:** A field description $\mathbb{F}$ and n distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **The protocol:**
  1. Each party P_i sends to each party the values $(f_i(\alpha_k), g_i(\alpha_k))$ (which are the values $(g_k(\alpha_i), f_k(\alpha_i)) = (S(\alpha_k, \alpha_i), S(\alpha_i, \alpha_k))$, respectively).
  2. At the end of this stage each party holds the sequences $(\hat{S}(\alpha_1, \alpha_k), \ldots, \hat{S}(\alpha_n, \alpha_k))$ and $(\hat{S}(\alpha_k, \alpha_1), \ldots, \hat{S}(\alpha_k, \alpha_n))$, where each is a possibly corrupted codeword of distance at most-t from each one of the following codewords, respectively:
  $$\begin{aligned} S(\alpha_1, \alpha_k), \ldots, S(\alpha_n, \alpha_k)) &= (g_k(\alpha_1), \ldots, g_k(\alpha_n)), \\ S(\alpha_k, \alpha_1), \ldots, S(\alpha_k, \alpha_n)) &= (f_k(\alpha_1), \ldots, f_k(\alpha_n)) . \end{aligned}$$
  3. Using the Reed-Solomon decoding procedure, each party decodes the above codewords and reconstructs the polynomials $S(x, \alpha_k) = f_k(x)$ and $S(\alpha_k, y) = g_k(y)$.
- **Output:** Each party outputs $\langle S(x, \alpha_k), S(\alpha_k, y)\rangle = \langle f_k(x), g_k(y)\rangle$.

PROTOCOL B.2.3 (Computing $\widetilde{F}^{mult}_{VSS}$ in the $(\widetilde{F}_{VSS}, F_{eval}, \widetilde{F}_{extend})$-hybrid model)

SEE: Protocol 3.3.8
Input:

1. The dealer P_1 holds two degree-t polynomials A and B.
2. Each party P_i holds a pair of shares a_i and b_i such that $a_i = A(\alpha_i)$ and $b_i = B(\alpha_i)$.

Common input: A field description $\mathbb{F}$ and n distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$. **Aiding ideal functionality initialization:** Upon invocation, the trusted party computing the (fictitiously corruption-aware) functionality $\widetilde{F}_{VSS}$, and the corruption-aware functionality $\widetilde{F}_{extend}$ and F_{eval} receives the set of corrupted parties I.

The protocol:

1. *Dealing phase:*
   (a) The dealer P_1 defines the degree-$2t$ polynomial $D(x) = A(x) \cdot B(x)$;
   Denote $D(x) = a \cdot b + \sum_{k=1}^{2t} d_k \cdot x^k$.
   (b) P_1 chooses t^2 values $\{r_{\ell,j}\}$ uniformly and independently at random from $\mathbb{F}$, where $\ell = 1, \ldots, t$, and $j = 0, \ldots, t-1$. For every $\ell = 1, \ldots, t$, the dealer defines the polynomial $D_\ell(x)$: $D_\ell(x) = \sum_{m=0}^{t-1} r_{\ell,m} \cdot x^m + \left(d_{\ell+t} - \sum_{m=\ell+1}^{t} r_{m,t+\ell-m}\right) \cdot x^t$.
   (c) P_1 computes the polynomial: $C'(x) = D(x) - \sum_{\ell=1}^{t} x^\ell \cdot D_\ell(x)$.
   (d) P_1 chooses t random degree-t bivariate polynomials $D_1(x, y), \ldots, D_t(x, y)$ under the constraint that $D_\ell(x, 0) = D_\ell(x)$ for every $\ell = 1, \ldots, t$. In addition, it chooses a random bivariate polynomial $C(x, y)$ of degree-t under the constraint that $C(x, 0) = C'(x)$.

   (e) P_1 invokes the $\widetilde{F}_{VSS}$ functionality as dealer with the following inputs: $C(x, y)$, and $D_\ell(x, y)$ for every $\ell = 1, \ldots, t$.

2. Each party P_i works as follows:

   (a) If any of the shares it receives from $\widetilde{F}_{VSS}$ equal $\perp$ then P_i proceeds to the *reject phase.*

   (b) P_i computes $c'_i \overset{\text{def}}{=} a_i \cdot b_i - \sum_{k=1}^{t} (\alpha_i)^k \cdot D_k(\alpha_i, 0)$. If $C(\alpha_i, 0) \neq c'_i$, then P_i broadcasts $(\mathsf{complaint}, i)$; note that $C(\alpha_i, y)$ is part of P_i's output from $\widetilde{F}_{VSS}$ with $C(x, y)$.

   (c) If any party P_k broadcast $(\mathsf{complaint}, k)$ then go to the *complaint resolution phase.*

3. *Complaint resolution phase:*

   (a) P_1 chooses two random bivariate polynomials $A(x, y)$, $B(x, y)$ of degree t under the constraint that $A(x, 0) = A(x)$ and $B(x, 0) = B(x)$.

   (b) The parties invoke the $\widetilde{F}_{extend}$ functionality twice, where P_1 inserts $A(x, y)$, $B(x, y)$ and each party inserts a_i, b_i. If any one of the outputs is $\perp$ (in which case all parties receive $\perp$), P_i proceeds to *reject phase.*

   (c) The parties run the following for every $(\mathsf{complaint}, k)$ message:

      i. Run $t+3$ invocations of $\widetilde{F}^k_{eval}$, with each party P_i inputting its shares of $A(x, y)$, $B(x, y)$, $D_1(x, y), \ldots, D_t(x, y)$, $C(x, y)$, respectively.
      Let $A(\alpha_k, y), B(\alpha_k, y), D_1(\alpha_k, y), \ldots, D_t(\alpha_k, y), C(\alpha_k, y)$ be the resulting shares (we ignore the dual shares $S(x, \alpha_k)$ for each polynomial).

      ii. If: $C(\alpha_k, 0) \neq A(\alpha_k, 0) \cdot B(\alpha_k, 0) - \sum_{\ell=1}^{t} \alpha_k^\ell D_\ell(\alpha_k, 0)$, proceed to the *reject phase.*

4. *Reject phase:*

   (a) Every party P_i sends a_i, b_i to all P_j. Party P_i defines the vector of values $\vec{a} = (a_1, \ldots, a_n)$ that it received, where $a_j = 0$ if it was not received at all. P_i sets $A'(x)$ to be the output of Reed-Solomon decoding on $\vec{a}$. Do the same for $B'(x)$.

   (b) Every party P_i sets $C(x, \alpha_i) = C(\alpha_i, y) = A'(0) \cdot B'(0)$; a constant polynomial.

5. *Outputs:* Every party P_i outputs $C(x, \alpha_i), C(\alpha_i, y)$. Party P_1 outputs $(A(x), B(x), C(x, y))$.

PROTOCOL B.2.4 (Computing $\widetilde{F}_{mult}$ in the $\widetilde{F}^{mult}_{VSS}$-hybrid model)

SEE: Protocol 3.3.11

- **Input:** Each party P_i holds $(\langle f_i^a(x), g_i^a(y)\rangle, \langle f_i^b(x), g_i^b(y)\rangle)$ for some bivariate polynomials $A(x, y), B(x, y)$ of degree t, which hide a, b, respectively. (If not all the points lie on a single degree-t bivariate polynomial, then no security guarantees are obtained. See Footnote 2.)

- **Common input:** The description of a field $\mathbb{F}$ and n distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$. In addition, the parties have constants $\gamma_1, \ldots, \gamma_n$ which are the first row of the inverse of the Vandemonde matrix (see [51]).

- **Aiding ideal functionality initialization:** Upon invocation, the trusted party computing the corruption-aware functionality $\widetilde{F}^{mult}_{VSS}$ receives the set of corrupted parties I.

- **The protocol:**

  1. For every $i = 1, \ldots, n$, the parties invoke the $\widetilde{F}^{mult}_{VSS}$ functionality as follows:

     (a) *Inputs:* In the ith invocation, party P_i plays the dealer. We recall that all parties hold shares on the univariate polynomials $A(x, \alpha_i) = f_i^a(x)$ and $B(x, \alpha_i) = f_i^b(x)$. Specifically, each party P_j sends $\widetilde{F}^{mult}_{VSS}$ their shares $g_j^a(\alpha_i) = A(\alpha_j, \alpha_i) = f_i^a(\alpha_j)$ and $g_j^b(\alpha_i) = B(\alpha_j, \alpha_i) = f_i^b(\alpha_j)$.

(b) *Outputs:* The functionality $\widetilde{F}_{VSS}^{mult}$ chooses a random degree-t bivariate polynomial $C_i(x,y)$ with constant term $f_i^a(0) \cdot f_i^b(0) = A(0,\alpha_i) \cdot B(0,\alpha_i)$. Every party P_j receives the bivariate shares $\langle C_i(x,\alpha_j), C_i(\alpha_j,y)\rangle$, for every $1 \le j \le n$.

2. At this stage, each party P_i holds polynomials $C_1(x,\alpha_i),\ldots,C_n(x,\alpha_i)$ and $C_1(\alpha_i,y),\ldots,C_n(\alpha_i,y)$. Then, it locally computes $C(x,\alpha_i) = \sum_{j=1}^n \gamma_j \cdot C_j(x,\alpha_i)$, and $C(\alpha_i,y) = \sum_{j=1}^n \gamma_j \cdot C_j(\alpha_i,y)$.

- **Output:** Each party P_i outputs $\langle C(x,\alpha_i), C(\alpha_i,y)\rangle$.

Appendix C

Security Proof for Protocol 5.3.3

We now analyze the security of the protocol of [58]. All the analysis that we present in this section appears also in [58], and is given here for completeness.

First, when both parties are honest, they both receive an output except for some negligible probability. Recall that in the first step of the protocol, the number of rounds R is set as $\alpha^{-1} \cdot \omega(\log \kappa)$. Both parties learn the correct output unless $i^* > R$, which happens with probability $(1-\alpha)^R < e^{-\alpha \cdot R} \leq e^{-\omega(\log \kappa)}$, which is negligible in κ. Note that we can set α to be polynomially small (say, $1/\kappa$), and still get number of rounds which is polynomial ($\kappa \cdot \omega(\log \kappa)$). In our results, we set $\alpha = 1/\ln \kappa$.

C.1 Security with Respect to Corrupted P_2

In each round, P_1 is the first to receive a message from F_{dealer}. As a result, in round i^* the party P_2 gets an output *after* P_1 has already received its output. Therefore, simulating corrupted P_2 is easy: the simulator invokes the adversary and receives its input y to the simulated F_{dealer}. It then chooses the round i^* as F_{dealer}, and gives the adversary values according to $\mathsf{RandOut}_2(y)$ for each round until i^*. In case the round i^* is reached, the simulator sends y to the trusted party computing f, receives the output $f(x, y)$, and gives the adversary this value until round R. Clearly, if the adversary aborts after or at i^*, the honest party P_1 has already learned the correct output in the real execution and also in the ideal. If the adversary aborts before i^*, the output of the honest party P_1 is determined according to $\mathsf{RandOut}_1(x)$, that is, $f(x, \hat{y})$ where $\hat{y}$ is chosen uniformly at random. Therefore, in case the adversary aborts before i^*, the simulator chooses $\hat{y}$ according to the uniform distribution over Y and sends this input to the trusted party. Overall, the protocol can be simulated *for any function* f. the following Claim is proven in [58], and is given here for completeness:

Claim C.1.1 *For every function $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0,1\}$, for every set of distribution X_{real}, for every $0 < \alpha < 1$ such that $\alpha^{-1} \in O(\mathsf{poly}(\kappa))$, Protocol 5.3.3 securely computes f in the presence of malicious adversary corrupting P_2.*

Proof: Fix f, X_{real} and α. Let $\mathcal{A}$ be an adversary that corrupts P_2. We construct the simulator $\mathcal{S}$ as follows:

1. $\mathcal{S}$ invokes $\mathcal{A}$ on input y and with auxiliary input z.
2. $\mathcal{S}$ chooses $\hat{y} \leftarrow Y$ uniformly, as in algorithm $\mathsf{RandOut}_1$. This value will be sent to the trusted party as the input of $\mathcal{A}$ in case $\mathcal{A}$ aborts before i^*.
3. $\mathcal{S}$ receives y' from $\mathcal{A}$ as was sent to F_{dealer} (Step 3 in the protocol). It then verifies that $y' \in Y$, if not – it sends abort as response from F_{dealer}, sends $\hat{y}$ to the trusted party and halts.

4. $\mathcal{S}$ chooses i^* according to geometric distribution with parameter α.
5. For every $i = 1, \ldots, i^* - 1$:
   (a) If $\mathcal{S}$ receives proceed from $\mathcal{A}$, then it sets $b_i = \mathsf{RandOut}_2(y')$ and sends b_i to $\mathcal{A}$ as was given from F_{dealer}.
   (b) If $\mathcal{S}$ receives abort from $\mathcal{A}$, it sends to $\mathcal{A}$ the message abort as was given from F_{dealer}. In addition, it sends $\hat{y}$ to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs and halts.
6. In the simulated round $i = i^*$:
   (a) If $\mathcal{S}$ receives proceed from $\mathcal{A}$, then it sends y' to the trusted party computing f and receives back the output $b_{out} = f(x, y')$. It then gives b_{out} back to $\mathcal{A}$.
   (b) If $\mathcal{S}$ receives abort from $\mathcal{A}$, it gives back to $\mathcal{A}$ the message abort as was sent from F_{dealer}. Then it sends the default input $\hat{y}$ to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs and halts.
7. In the simulated rounds $i = i^* + 1$ to R:
   (a) If $\mathcal{S}$ receives proceed from $\mathcal{A}$, then it gives b_{out} to $\mathcal{A}$.
   (b) If $\mathcal{S}$ receives abort from $\mathcal{A}$, it sends abort to $\mathcal{A}$ as was given from F_{dealer}.
8. If $\mathcal{S}$ has not halted yet and $i^* > R$, then $\mathcal{S}$ sends $\hat{y}$ to the trusted party.
9. $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.

There is no communication between P_1 and P_2, and all the view of P_2 consists of the outputs of the F_{dealer} functionality to P_2. Moreover, P_2 only sends y' in the first round to F_{dealer}, and all the rest of the messages are either proceed or abort. Therefore, after sending y', all what $\mathcal{A}$ can do is either send proceed or to abort the interaction. Assume that $\mathcal{A}$ aborts at some round i. If $i = 0$, i.e., if $\mathcal{A}$ does not send the input y' to F_{dealer}, then in the real execution P_1 outputs $\mathsf{RandOut}_1(x)$ which is independent of y'. In the ideal, $\mathcal{S}$ chooses $\hat{y}$ uniformly from Y, and sends to the trusted party computing f the value $\hat{y}$, which determines the output of P_1 to be $f(x, \hat{y})$. This is exactly the same as the implementation of $\mathsf{RandOut}_1(x)$. In case $\mathcal{A}$ aborts at some round $i < i^*$, the view of P_2 consists of i independent invocations of $\mathsf{RandOut}_2(y')$, while the output of the honest P_1 consists of $\mathsf{RandOut}_1(x)$. $\mathcal{S}$ works exactly in the same way – for every round until i^*, it sends to P_2 a fresh output that depends only on the value y' – $\mathsf{RandOut}_2(y')$. In case it aborts, it sends $\hat{y}$ as we above, resulting the output of P_1 to distribute identically as in the real execution. In case $\mathcal{A}$ aborts at i^* or after i^* (i.e., in case $i^* \geq i$), P_1 has already learned the output $b_{out} = f(x, y')$ in the real execution. Therefore, $\mathcal{S}$ can send the true input y' to the trusted party, which determines the output of P_1 to be $f(x, y')$. It learns the output b_{out}, and gives this value to $\mathcal{A}$ as the outputs of F_{dealer}, exactly as in the real execution. ■

C.2 Security with Respect to Corrupted P_1

This case is more complex. Intuitively, the adversary does have an advantage in the real execution, in case it succeeds to predict correctly and aborts exactly at round i^*. In such a case, the corrupted party P_1 learns the correct output $f(x, y)$, whereas P_2 learns a value according to $\mathsf{RandOut}_2(y)$. However, as we will see, this advantage in the real execution can be simulated in the ideal execution under certain circumstances, for *some* functions f.

The output vector distributions $\boldsymbol{Q^{x,a}}$. Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0,1\}$. Fix X_{real}, and let U_Y denote the uniform distribution over Y. For every $x \in X$, denote by p_x the probability that the output of $\mathsf{RandOut}_1$ is 1 on the input x. Similarly, for every $y_j \in Y$, let p_{y_j} denote the probability that the output of $\mathsf{RandOut}_2$ is 1 on the input y_j. That is:

$$p_x \overset{\text{def}}{=} \Pr_{\hat{y} \leftarrow U_Y} [f(x, \hat{y}) = 1] \quad \text{and} \quad p_{y_j} \overset{\text{def}}{=} \Pr_{\hat{x} \leftarrow X_{real}} [f(\hat{x}, y_j) = 1]$$

For every $x \in X$, $a \in \{0,1\}$, define the m-dimensional row vectors $Q^{x,a} = (q^{x,a}_{y_1}, \ldots, q^{x,a}_{y_m})$ indexed by $y_j \in Y$ as follows:

$$q^{x,0}_{y_j} \overset{\text{def}}{=} \begin{cases} p_{y_j} & \text{if } f(x,y_j) = 1 \\ p_{y_j} + \frac{\alpha \cdot p_{y_j}}{(1-\alpha)\cdot(1-p_x)} & \text{if } f(x,y_j) = 0 \end{cases} \qquad q^{x,1}_{y_j} \overset{\text{def}}{=} \begin{cases} p_{y_j} + \frac{\alpha\cdot(p_{y_j}-1)}{(1-\alpha)\cdot p_x} & \text{if } f(x,y_j) = 1 \\ p_{y_j} & \text{if } f(x,y_j) = 0 \end{cases} \tag{C.2.1}$$

We have:

Theorem C.2.1 *Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_m\} \to \{0,1\}$ and let M_f be as above. If there exist probability vector X_{real}, parameter $0 < \alpha < 1$ such that $\alpha^{-1} \in O(\mathsf{poly}(\kappa))$, such that for every $x \in X$, $a \in \{0,1\}$, there exists a probability vector $X^{x,a}_{ideal}$ for which:*

$$X^{x,a}_{ideal} \cdot M_f = Q^{x,a} \;,$$

then Protocol 5.3.3 securely computes f with complete fairness.

Proof: We start with the description of the simulator $\mathcal{S}$. The simulator chooses i^* as F_{dealer}. If the adversary aborts after i^*, then both parties learn output and so the simulator sends the true input to the trusted party and fairness is obtained. If the adversary aborts i^*, then the simulator needs to give the adversary the true output $f(x,y)$. Thus, it sends the true input to the trusted party. However, by doing so the honest party learn the correct output unlike the real execution, when it outputs a random value according to $\mathsf{RandOut}_2(y)$. Thus, if the adversary aborts before i^*, the simulator chooses the input to send to the trusted party not according to X_{real}, but according to $X^{x,a}_{ideal}$. This should balance the advantage that the simulator gives the honest party in case the adversary aborts exactly at i^*.

The simulator $\mathcal{S}$.

1. $\mathcal{S}$ invokes $\mathcal{A}$ with input x and auxiliary input z.
2. When $\mathcal{A}$ sends x' to F_{dealer}, $\mathcal{S}$ checks that $x' \in X$ and sets $x = x'$. If $x' \notin X$, $\mathcal{S}$ chooses a default input $\hat{x} \in X$ according to the distribution X_{real}, sends $\hat{x}$ to the trusted party, outputs whatever $\mathcal{A}$ outputs and halts.
3. $\mathcal{S}$ chooses i^* according to geometric distribution with parameter α.
4. For every $i = 1$ to $i^* - 1$:
   (a) $\mathcal{S}$ chooses $a_i = \mathsf{RandOut}_1(x)$ and gives a_i to $\mathcal{A}$ as was sent from F_{dealer}.
   (b) If $\mathcal{A}$ sends abort back to F_{dealer}, then $\mathcal{S}$ chooses $\hat{x}$ according to the distribution X^{x,a_i}_{ideal}. It then sends $\hat{x}$ to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs and halts. If $\mathcal{A}$ sends $\mathsf{proceed}$, then $\mathcal{S}$ proceeds to the next iteration.
5. In round $i = i^*$:
   (a) $\mathcal{S}$ sends the input x to the trusted party computing f and receives $a_{out} = f(x,y)$.
   (b) $\mathcal{S}$ gives to $\mathcal{A}$ the value a_{out} as was sent from F_{dealer}.
6. In rounds $i = i^* + 1$ to R:
   (a) If $\mathcal{A}$ sends $\mathsf{proceed}$, $\mathcal{S}$ gives back to $\mathcal{A}$ the value a_{out} and proceeds to the next iteration.

7. If $\mathcal{S}$ has not halted yet and $i^* > R$, then $\mathcal{S}$ chooses $\hat{x}$ according to the distribution X_{real}, it sends $\hat{x}$ to the trusted party.

8. $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.

Let $\text{VIEW}_{\mathsf{hyb}}(x, y)$ denote the view of P_1 in an execution of the protocol with the adversary $\mathcal{A}$, where $\mathcal{A}$'s input is x, its auxiliary input is z and P_2's input is y. Let $\text{OUTPUT}_{\mathsf{hyb}}(x, y)$ be the output of P_2 in such an execution. Let $\text{VIEW}_{\mathsf{ideal}}(x, y)$ denote the view of the adversary in the ideal execution where the parties are invoked with the inputs (x, y), respectively, and the auxiliary input of the simulator is z. Let $\text{OUTPUT}_{\mathsf{ideal}}$ be the output of P_2 in the ideal execution.

Since $\mathcal{A}$ can only abort the execution, let i denote the round for which $\mathcal{A}$ has aborted. In case $\mathcal{A}$ does not abort during the execution of the protocol, then set $i = R+1$. Note that $\mathcal{A}$ may also abort *before* the invocation of F_{dealer}; in such a case we say that $i = 0$. We want to show that for every $(\vec{a}, b) \in \{0,1\}^{i+1}$, it holds that:

$$\begin{aligned} &\Pr\left[(\text{VIEW}_{\mathsf{hyb}}(x, y), \text{OUTPUT}_{\mathsf{hyb}}(x, y)) = (\vec{a}, b)\right] \\ &\quad = \Pr\left[(\text{VIEW}_{\mathsf{ideal}}(x, y), \text{OUTPUT}_{\mathsf{ideal}}(x, y)) = (\vec{a}, b)\right] \end{aligned} \tag{C.2.2}$$

If $\mathcal{A}$ aborts before the invocation of F_{dealer} (or has sent an invalid input to F_{dealer}), then the view of the adversary is empty, and the honest party P_2 outputs a value according to $\mathsf{RandOut}_2(y)$. The same happens in the ideal execution, since $\mathcal{S}$ sends in such a case a value $\hat{x}$ distributed according to X_{real}. In such a case, both terms of Eq. (C.2.2) are 0 in case $\vec{a}$ is not empty, and clearly both terms have the same probability for any value of b. Thus, we get that for every $b \in \{0,1\}$:

$$\begin{aligned} &\Pr\left[(\text{VIEW}_{\mathsf{hyb}}(x, y), \text{OUTPUT}_{\mathsf{hyb}}(x, y)) = (\lambda, b) \mid i = 0\right] \\ &\quad = \Pr\left[(\text{VIEW}_{\mathsf{ideal}}(x, y), \text{OUTPUT}_{\mathsf{ideal}}(x, y)) = (\lambda, b) \mid i = 0\right] . \end{aligned}$$

In case $i = R+1$ (i.e., the adversary does not abort), in the real the adversary sees values $a_1, \ldots, a_{i^*-1}$ that are independent to y, and all the values $a_{i^*}, \ldots, a_R$ are $f(x, y)$. In the ideal, we have the exact same thing. Therefore, Eq. (C.2.2) holds conditioning on the event $i = R+1$. Moreover, the above is true whenever the adversary aborts at a round $i > i^*$, and thus the equation holds also when we condition on the event $i > i^*$.

We remain with the case where $1 \leq i \leq i^*$. If $i = i^*$, in the real execution the output of P_2 is independent of x, and is determined according to $\mathsf{RandOut}_1(x)$. However, In the ideal execution, the simulator $\mathcal{S}$ queries the trusted party computing f on x, receives back $f(x, y)$, which determines the output of P_2 to be the correct output. Therefore, if $\mathcal{A}$ aborts exactly at i^*, in the real execution it learns the correct output while P_2 does not, and in the ideal execution *both* parties output the true output $f(x, y)$. Therefore, in order to simulate the protocol correctly, the simulator modifies the output of P_2 in case the adversary aborts before i^*, and chooses $\hat{x}$ according to some distribution X_{ideal}^{x,a_i} and not according to X_{real} as apparently expected. In the following, we show that if $X_{ideal}^{x,a_i} \cdot M_f = Q^{x,a_i}$, then Eq. (C.2.2) is satisfied.

Let $\vec{a} = (\vec{a}_{i-1}, a)$, and $\text{VIEW}_{\mathsf{hyb}}(x, y) = (\overrightarrow{\text{VIEW}}_{\mathsf{hyb}}^{i-1}, \text{VIEW}_{\mathsf{hyb}}^{i})$. We have that:

$$\begin{aligned} &\Pr\left[(\text{VIEW}_{\mathsf{hyb}}(x, y), \text{OUTPUT}_{\mathsf{hyb}}(x, y)) = (\vec{a}, b) \mid i \leq i^*\right] \\ &\quad = \Pr\left[(\text{VIEW}_{\mathsf{hyb}}^{i}, \text{OUTPUT}_{\mathsf{hyb}}) = (a, b) \mid \overrightarrow{\text{VIEW}}_{\mathsf{hyb}}^{i-1} = \vec{a}_{i-1},\ i \leq i^*\right] \cdot \Pr\left[\overrightarrow{\text{VIEW}}_{\mathsf{hyb}}^{i-1} = \vec{a}_{i-1},\ i \leq i^*\right] \\ &\quad = \Pr\left[(\text{VIEW}_{\mathsf{hyb}}^{i}, \text{OUTPUT}_{\mathsf{hyb}}) = (a, b) \mid i \leq i^*\right] \cdot \Pr\left[\overrightarrow{\text{VIEW}}_{\mathsf{hyb}}^{i-1} = \vec{a}_{i-1} \mid i \leq i^*\right] \end{aligned}$$

where the last equations is true since conditioning on the even that $i \leq i^*$, the random variables $(\text{VIEW}_{\mathsf{hyb}}^{i}, \text{OUTPUT}_{\mathsf{hyb}})$ are independent of $\overrightarrow{\text{VIEW}}_{\mathsf{hyb}}^{i-1}$. Similarly, write $\text{VIEW}_{\mathsf{ideal}}(x, y) = (\overrightarrow{\text{VIEW}}_{\mathsf{ideal}}^{i-1}, \text{VIEW}_{\mathsf{ideal}}^{i})$, we have:

$$\begin{aligned}
&\Pr\left[(\text{VIEW}_{\mathsf{ideal}}(x,y), \text{OUTPUT}_{\mathsf{ideal}}(x,y)) = (\vec{a}, b) \mid i \le i^*\right] \\
&\quad= \Pr\left[(\text{VIEW}^i_{\mathsf{ideal}}, \text{OUTPUT}_{\mathsf{ideal}}) = (a,b) \mid \overrightarrow{\text{VIEW}}^{i-1}_{\mathsf{ideal}} = \vec{a}_{i-1},\ i \le i^*\right] \cdot \Pr\left[\overrightarrow{\text{VIEW}}^{i-1}_{\mathsf{ideal}} = \vec{a}_{i-1},\ i \le i^*\right] \\
&\quad= \Pr\left[(\text{VIEW}^i_{\mathsf{ideal}}, \text{OUTPUT}_{\mathsf{ideal}}) = (a,b) \mid i \le i^*\right] \cdot \Pr\left[\overrightarrow{\text{VIEW}}^{i-1}_{\mathsf{ideal}} = \vec{a}_{i-1} \ \mid\ i \le i^*\right]
\end{aligned}$$

It is easy to see that for any values $\vec{a}_{i-1} \in \{0,1\}^{i-1}$:

$$\Pr\left[\overrightarrow{\text{VIEW}}^{i-1}_{\mathsf{ideal}} = \vec{a}_{i-1} \ \mid\ i \le i^*\right] = \Pr\left[\overrightarrow{\text{VIEW}}^{i-1}_{\mathsf{hyb}} = \vec{a}_{i-1} \ \mid\ i \le i^*\right]$$

since in both the simulation and the real execution, the adversary receives values according to $\mathsf{RandOut}_1(x)$.

It is left to show that for every $(a,b) \in \{0,1\}^2$, it holds that:

$$\Pr\left[(\text{VIEW}^i_{\mathsf{hyb}}, \text{OUTPUT}_{\mathsf{hyb}}) = (a,b) \ \mid\ i \le i^*\right] = \Pr\left[(\text{VIEW}^i_{\mathsf{ideal}}, \text{OUTPUT}_{\mathsf{ideal}}) = (a,b) \ \mid\ i \le i^*\right] \quad \text{(C.2.3)}$$

We have already seen it in the proof sketch in Section 5.3. We just give the high-level overview.

In case $f(x,y) = 0$. The probabilities are as follows:

output (a,b)	real	ideal
$(0,0)$	$(1-\alpha)\cdot(1-p_x)\cdot(1-p_y) + \alpha\cdot(1-p_y)$	$(1-\alpha)\cdot(1-p_x)\cdot(1-q_y^{x,0}) + \alpha$
$(0,1)$	$(1-\alpha)\cdot(1-p_x)\cdot p_y + \alpha\cdot p_y$	$(1-\alpha)\cdot(1-p_x)\cdot q_y^{x,0}$
$(1,0)$	$(1-\alpha)\cdot p_x\cdot(1-p_y)$	$(1-\alpha)\cdot p_x\cdot(1-q_y^{x,1})$
$(1,1)$	$(1-\alpha)\cdot p_x\cdot p_y$	$(1-\alpha)\cdot p_x\cdot q_y^{x,1}$

Therefore, we get the following constraints:

$$q_y^{x,0} = p_y + \frac{\alpha\cdot p_y}{(1-\alpha)\cdot(1-p_x)} \qquad \text{and} \qquad q_y^{x,1} = p_y\ ,$$

which are satisfied according to our assumption in the theorem.

In case $f(x,y) = 1$. Similarly, for the case of $f(x,y) = 1$, we have:

output (a,b)	real	ideal
$(0,0)$	$(1-\alpha)\cdot(1-p_x)\cdot(1-p_y)$	$(1-\alpha)\cdot(1-p_x)\cdot(1-q_y^{x,0})$
$(0,1)$	$(1-\alpha)\cdot(1-p_x)\cdot p_y$	$(1-\alpha)\cdot(1-p_x)\cdot q_y^{x,0}$
$(1,0)$	$(1-\alpha)\cdot p_x\cdot(1-p_y) + \alpha\cdot(1-p_y)$	$(1-\alpha)\cdot p_x\cdot(1-q_y^{x,1})$
$(1,1)$	$(1-\alpha)\cdot p_x\cdot p_y + \alpha\cdot p_y$	$(1-\alpha)\cdot p_x\cdot q_y^{x,1} + \alpha$

we again get the following constraints:

$$q_y^{x,0} = p_y \qquad \text{and} \qquad q_y^{x,1} = p_y + \frac{\alpha\cdot(p_y-1)}{(1-\alpha)\cdot p_x}\ .$$

However, since $X^{x,a}_{ideal}\cdot M_f = Q^{x,a}$, the above constraints are satisfied. ■

Bibliography

[1] Ittai Abraham, Danny Dolev, Rica Gonen, and Joseph Y. Halpern. Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In *PODC*, pages 53–62, 2006.

[2] Shashank Agrawal and Manoj Prabhakaran. On fair exchange, fair coins and fair sampling. In *CRYPTO (1)*, pages 259–276, 2013.

[3] Gilad Asharov. Towards characterizing complete fairness in secure two-party computation. In *TCC*, pages 291–316, 2014.

[4] Gilad Asharov, Ran Canetti, and Carmit Hazay. Towards a game theoretic view of secure computation. In *EUROCRYPT*, pages 426–445, 2011.

[5] Gilad Asharov and Yehuda Lindell. Utility dependence in correct and fair rational secret sharing. In *CRYPTO*, pages 559–576, 2009.

[6] Gilad Asharov and Yehuda Lindell. A full proof of the bgw protocol for perfectly-secure multiparty computation. *IACR Cryptology ePrint Archive*, 2011:136, 2011. An abbreviated version appeared as a book chapter in [7].

[7] Gilad Asharov and Yehuda Lindell. *The BGW Protocol for Perfectly-Secure Multiparty Computation*, pages 120 – 167. IOS Press, 2013. Chapter 5 in [90].

[8] Gilad Asharov, Yehuda Lindell, and Tal Rabin. Perfectly-secure multiplication for any $t < n/3$. In *CRYPTO*, pages 240–258, 2011.

[9] Gilad Asharov, Yehuda Lindell, and Tal Rabin. A full characterization of functions that imply fair coin tossing and ramifications to fairness. In *TCC*, pages 243–262, 2013.

[10] N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In *ACM Conference on Computer and Communications Security*, pages 7–17, 1997.

[11] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures (extended abstract). In *EUROCRYPT*, pages 591–606, 1998.

[12] Giuseppe Ateniese. Efficient verifiable encryption (and fair exchange) of digital signatures. In *ACM Conference on Computer and Communications Security*, pages 138–146, 1999.

[13] Feng Bao, Robert H. Deng, and Wenbo Mao. Efficient and practical fair exchange protocols with off-line ttp. In *IEEE Symposium on Security and Privacy*, pages 77–85, 1998.

[14] Donald Beaver. Multiparty protocols tolerating half faulty processors. In *CRYPTO*, pages 560–572, 1989.

[15] Donald Beaver. Foundations of secure interactive computing. In *CRYPTO*, pages 377–391, 1991.

[16] Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority (extended announcement). In *FOCS*, pages 468–473, 1989.

[17] Zuzana Beerliová-Trubíniová and Martin Hirt. Efficient multi-party computation with dispute control. In *TCC*, pages 305–328, 2006.

[18] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure mpc with linear communication complexity. In *TCC*, pages 213–230, 2008.

[19] Amos Beimel, Yehuda Lindell, Eran Omri, and Ilan Orlov. $1/p$-secure multiparty computation without honest majority and the best of both worlds. In *CRYPTO*, pages 277–296, 2011.

[20] Amos Beimel, Eran Omri, and Ilan Orlov. Protocols for multiparty coin toss with dishonest majority. In *CRYPTO*, pages 538–557, 2010.

[21] Michael Ben-Or, Oded Goldreich, Silvio Micali, and Ronald L. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.

[22] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[23] Manuel Blum. How to exchange (secret) keys. *ACM Trans. Comput. Syst.*, 1(2):175–193, 1983.

[24] Dan Boneh and Moni Naor. Timed commitments. In *CRYPTO*, pages 236–254, 2000.

[25] Ernest F. Brickell, David Chaum, Ivan Damgård, and Jeroen van de Graaf. Gradual and verifiable release of a secret. In *CRYPTO*, pages 156–166, 1987.

[26] Christian Cachin and Jan Camenisch. Optimistic fair secure computation. In *CRYPTO*, pages 93–111, 2000.

[27] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[28] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[29] Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. Adaptive versus non-adaptive security of multi-party protocols. *J. Cryptology*, 17(3):153–207, 2004.

[30] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In *EUROCRYPT*, pages 337–351, 2002.

[31] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.

[32] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19, 1988.

[33] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *FOCS*, pages 383–395, 1985.

[34] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, pages 364–369, 1986.

[35] Richard Cleve. Controlled gradual disclosure schemes for random bits and their applications. In *CRYPTO*, pages 573–588, 1989.

[36] Richard Cleve and Russell Impagliazzo. Martingales, collective coin flipping and discrete control processes (extended abstract), 1993. `http://www.cpsc.ucalgary.ca/~cleve/pubs/martingales.ps`.

[37] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *EUROCRYPT*, pages 316–334, 2000.

[38] Ivan Damgård. Practical and provably secure release of a secret and exchange of signatures. In *EUROCRYPT*, pages 200–217, 1993.

[39] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, pages 572–590, 2007.

[40] Yevgeniy Dodis, Pil Joong Lee, and Dae Hyun Yum. Optimistic fair exchange in a multi-user setting. In *Public Key Cryptography*, pages 118–133, 2007.

[41] Yevgeniy Dodis and Silvio Micali. Parallel reducibility for information-theoretically secure computation. In *CRYPTO*, pages 74–92, 2000.

[42] Shimon Even. Protocol for signing contracts. In *CRYPTO*, pages 148–153, 1981.

[43] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. In *CRYPTO*, pages 205–210, 1982.

[44] Shimon Even and Yacov Yacobi. Relations among public key signature schemes. *Technical Report #175, Technion Israel Institute of Technology, Computer Science Department*, 1980.

[45] P. Feldman. *Optimal Algorithms for Byzantine Agreement.* PhD thesis, Massachusetts Institute of Technology, 1988.

[46] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.

[47] Georg Fuchsbauer, Jonathan Katz, and David Naccache. Efficient rational secret sharing in standard communication networks. In *TCC*, pages 419–436, 2010.

[48] Juan A. Garay, Markus Jakobsson, and Philip D. MacKenzie. Abuse-free optimistic contract signing. In *CRYPTO*, pages 449–466, 1999.

[49] Juan A. Garay, Philip D. MacKenzie, Manoj Prabhakaran, and Ke Yang. Resource fairness and composability of cryptographic protocols. In *TCC*, pages 404–428, 2006.

[50] Juan A. Garay and Carl Pomerance. Timed fair exchange of standard signatures: [extended abstract]. In *Financial Cryptography*, pages 190–207, 2003.

[51] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In *PODC*, pages 101–111, 1998.

[52] Oded Goldreich. A simple protocol for signing contracts. In *CRYPTO*, pages 133–136, 1983.

[53] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[54] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[55] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In *CRYPTO*, pages 77–93, 1990.

[56] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *STOC*, pages 365–377, 1982.

[57] S. Dov Gordon. *On fairness in secure computation*. PhD thesis, University of Maryland, 2010.

[58] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete fairness in secure two-party computation. In *STOC*, pages 413–422., 2008. Extended full version available on: `http://eprint.iacr.org/2008/303`. Journal version: [59].

[59] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete fairness in secure two-party computation. *J. ACM*, 58(6):24, 2011.

[60] S. Dov Gordon, Yuval Ishai, Tal Moran, Rafail Ostrovsky, and Amit Sahai. On complete primitives for fairness. In *TCC*, pages 91–108, 2010.

[61] S. Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. In *EUROCRYPT*, pages 157–176, 2010.

[62] Branko Grünbaum. *Convex Polytopes : Second Edition Prepared by Volker Kaibel, Victor Klee, and Günter Ziegler (Graduate Texts in Mathematics)*. Springer, May 2003.

[63] Joseph Y. Halpern and Vanessa Teague. Rational secret sharing and multiparty computation: extended abstract. In *STOC*, pages 623–632, 2004.

[64] Martin Hirt and Ueli M. Maurer. Robustness for free in unconditional multi-party computation. In *CRYPTO*, pages 101–118, 2001.

[65] Martin Hirt, Ueli M. Maurer, and Bartosz Przydatek. Efficient secure multi-party computation. In *ASIACRYPT*, pages 143–161, 2000.

[66] Martin Hirt and Jesper Buus Nielsen. Robust multiparty computation with linear communication complexity. In *CRYPTO*, pages 463–482, 2006.

[67] Russell Impagliazzo and Moti Yung. Direct minimum-knowledge computations. In *CRYPTO*, pages 40–51, 1987.

[68] J. Komlòs J. Kahn and E. Szemerèdi. On the probability that a random ± 1-matrix is singular. *Journal of Amer. Math. Soc.*, 8:223–240, 1995.

[69] Joe Kilian. A general completeness theorem for two-party games. In *STOC*, pages 553–560, 1991.

[70] Gillat Kol and Moni Naor. Cryptography and game theory: Designing protocols for exchanging information. In *TCC*, pages 320–339, 2008.

[71] Gillat Kol and Moni Naor. Games for exchanging information. In *STOC*, pages 423–432, 2008.

[72] J. Komlòs. On the determinant of $(0, 1)$ matrices. *Studia Sci. Math. Hungar*, 2:7–21, 1967.

[73] Alptekin Küp**c**cü and Anna Lysyanskaya. Optimistic fair exchange with multiple arbiters. In *ESORICS*, pages 488–507, 2010.

[74] Alptekin Küp**c**cü and Anna Lysyanskaya. Usable optimistic fair exchange. In *CT-RSA*, pages 252–267, 2010.

[75] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. *SIAM J. Comput.*, 39(5):2090–2112, 2010.

[76] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[77] Andrew Y. Lindell. Legally-enforceable fairness in secure two-party computation. In *CT-RSA*, pages 121–137, 2008.

[78] Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *FOCS*, pages 394–403, 2003.

[79] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. Sequential composition of protocols without simultaneous termination. In *PODC*, pages 203–212, 2002.

[80] Michael Luby, Silvio Micali, and Charles Rackoff. How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin. In *FOCS*, pages 11–21, 1983.

[81] Anna Lysyanskaya and Nikos Triandopoulos. Rationality and adversarial behavior in multi-party computation. In *CRYPTO*, pages 180–197, 2006.

[82] R. J. McEliece and D. V. Sarwate. On sharing secrets and reed-solomon codes. *Commun. ACM*, 24(9):583–584, September 1981.

[83] Silvio Micali. Certified email with invisible post offices, 1997. Technical report; an invited presentation at the RSA '97 conference (1997).

[84] Silvio Micali. Simple and fast optimistic protocols for fair electronic exchange. In *PODC*, pages 12–19, 2003.

[85] Silvio Micali and Phillip Rogaway. Secure computation, unpublished manuscript, 1992. Preliminary verstion in *CRYPTO*, pages 392-404, 1991.

[86] Tal Moran, Moni Naor, and Gil Segev. An optimally fair coin toss. In *TCC*, pages 1–18, 2009.

[87] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[88] Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Optimal efficiency of optimistic contract signing. In *PODC*, pages 113–122, 1998.

[89] Benny Pinkas. Fair secure two-party computation. In *EUROCRYPT*, pages 87–105, 2003.

[90] Manoj Prabhakaran and Amit Sahai, editors. *Secure Multi-Party Computation.* IOS Press, 2013.

[91] Michael O. Rabin. How to exchange secrets with oblivious transfer. *Technical Report TR-81, Aiken Computation Lab, Harvard University*, 1981.

[92] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *STOC*, pages 73–85, 1989.

[93] Steven Roman. *Advanced Linear Algebra 3rd ed.* Graduate Texts in Mathematics 135. New York, NY: Springer. xviii, 2008.

[94] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[95] Umesh V. Vazirani and Vijay V. Vazirani. Trapdoor pseudo-random number generators, with applications to protocol design. In *FOCS*, pages 23–30, 1983.

[96] Thomas Voigt and Günter M. Ziegler. Singular 0/1-matrices, and the hyperplanes spanned by random 0/1-vectors. *Combinatorics, Probability and Computing*, 15(3):463–471, 2006.

[97] John von Neumann. Various Techniques Used in Connection with Random Digits. *J. Res. Nat. Bur. Stand.*, 12:36–38, 1951.

[98] Philip J. Wood. *On the probability that a discrete complex random matrix is singular.* PhD thesis, Rutgers University, New Brunswick, NJ, USA, 2009. AAI3379178.

[99] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982.

[100] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

[101] Günter M. Ziegler. Lectures on 0/1-polytopes. *Polytopes: Combinatorics and Computation, Vol. 29 of DMV Seminar, Birkhauser, Basel*, pages 1–40, 2000.