Perfectly-Secure Multiplication for any $t < n/3$

Gilad Asharov[1]*, Yehuda Lindell[1]*, and Tal Rabin[2]

[1] Bar-Ilan University
asharog@cs.biu.ac.il, lindell@biu.ac.il
[2] IBM T.J. Watson Research
talr@us.ibm.com

Abstract. In the setting of secure multiparty computation, a set of n parties with private inputs wish to jointly compute some functionality of their inputs. One of the most fundamental results of information-theoretically 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. The most technically challenging part of this result is a protocol for multiplying two shared values, with perfect security in the presence of up to $t < n/3$ malicious adversaries.

In this paper we provide a full specification of the BGW perfect multiplication protocol and prove its security. This includes one new step for the perfect multiplication protocol in the case of $n/4 \leq t < n/3$. As in the original BGW protocol, this protocol works whenever the parties hold univariate (Shamir) shares of the input values. In addition, we present a new multiplication protocol that utilizes *bivariate* secret sharing in order to achieve higher efficiency while maintaining a round complexity that is constant per multiplication. Both of our protocols are presented with full proofs of security.

1 Introduction

The groundbreaking BGW protocol [4] for perfectly-secure multiparty computation appeared over 20 years ago and has had a huge impact on our field; the importance of the result coupled with its elegant and ingenious techniques are the source of this great following. The BGW protocol enables a set of parties $P_1, \ldots, P_n$ to compute any functionality of their inputs while preserving security in the presence of up to $t < n/3$ malicious parties. The protocol is comprised of a few components; a method for verifiable secret sharing (VSS), a protocol for adding two secrets that are in shared form and a protocol for multiplying two secrets given in shared form. Despite its importance and the fact that over a thousand papers have built upon it, a full proof of its correctness has never appeared. In [1] we rectify this situation and present a proof of the BGW protocol, together with a new more efficient multiplication protocol presented here. In addition, we provide a full specification of the protocol. This includes a new

* Supported by the European Research Council as part of the ERC project LAST.

step that is needed for the case of $n/4 \leq t < n/3$. In this extended abstract we focus on the question of perfect multiplication, including the new step needed for the BGW protocol and our new more efficient protocol.

BGW perfect multiplication. The aim of the BGW multiplication protocol is to have a set of parties compute a sharing of the product $a \cdot b$, given a sharing of the individual values a and b.[3] Let $a_1, \ldots, a_n$ denote the parties' shares of a, and let $b_1, \ldots, b_n$ denote their shares of b. The protocol works according to the following steps:

1. Each party P_i shares its shares a_i and b_i with all other parties. This is carried out in a way (using error correcting codes) that prevents a corrupted P_i from sharing a value $a'_i \neq a_i$ or $b'_i \neq b_i$.
2. Next, each P_i needs to distribute shares of the product of its shares $a_i \cdot b_i$ as follows. We focus on a fixed party P_i, and let $A(x)$ and $B(x)$ be the respective polynomials for the sharing of a_i and b_i from the previous step.
   (a) Party P_i computes polynomials $D_1, \ldots, D_t$ 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 free coefficient $a_i \cdot b_i$. Note that since each polynomial D_ℓ is multiplied by x^ℓ, we have that the free coefficient of $C(x)$ is always $a_i \cdot b_i$ (i.e., $A(0) \cdot B(0)$). As shown in BGW, P_i can choose the polynomials $D_1, \ldots, D_t$ in a special way so as to cancel out all coefficients of degree higher than t, and to ensure that $C(x)$ is of degree-t exactly. We stress that if P_i uses "incorrect" polynomials, then $C(x)$ may be of degree higher than t.
   (b) Party P_i verifiably shares the polynomials $D_1, \ldots, D_t$ with all parties.
   (c) Each party computes its share of $C(x)$ based on its shares of a_i, b_i and the polynomials $D_1, \ldots, D_t$.
   (d) At this point, it is guaranteed that the parties hold shares of a polynomial with free coefficient $a_i \cdot b_i$ (as described in Step 2a above) and it remains to *verify that these shares define a polynomial of degree t* (and not a higher degree).
3. Once the above is completed for all P_i we have that all parties hold valid shares of all share products $a_1 \cdot b_1, \ldots, a_n \cdot b_n$. Given these subshares, it suffices for each party to carry out a local linear computation [14] with the result being that they obtain valid shares of $a \cdot b$, as required.

Verifying the degree of the polynomial. We examine how to carry out Step 2d above, that is, how to verify that the shares held by the parties define a degree-t polynomial rather than a polynomial of higher degree.

First we need to touch on a subtle point which is the source of the challenge of realizing the verification step. The question is what we mean when we say that the shares of the honest parties should define a polynomial of degree t (or less)? There is a clear distinction between two cases. The first is that given the set of $2t+1$ shares held by the honest parties, we wish to ensure that their shares all lie

[3] There are actually some subtleties in formally defining the multiplication functionality since the adversary can determine some of the points that the sharing polynomial goes through. Nevertheless, this is the basic idea.

on the same degree-t polynomial. If they do not, then we are willing to modify up to t of the honest parties' shares to achieve this goal. This is what typically happens in the verification step of VSS protocols; the dealer modifies the shares that it originally gave to some of the parties by broadcasting new shares. The second interpretation is that we need to verify that *all* of the $2t+1$ shares held by the honest parties at the onset lie on a single degree-t polynomial. If not, then they should be notified of this fact, and should not change their shares. This is the cause of some difficulty as it requires a mechanism to distinguish between honest and corrupt parties; in particular, to distinguish between a corrupt party who lies about its share and an honest party who has an incorrect share.

In the BGW multiplication step we are in the second case. The shares that the honest parties hold have been created via the computation in Step 2c. The correctness of the computation requires that the constant term of the polynomial defined by the honest parties' shares be $a_i \cdot b_i$. However, a corrupted P_i who does not share the D_ℓ's in an appropriate manner can cause the resulting polynomial $C(x)$ to be of degree higher than t. In this case, there are at least two subsets of honest parties of size $t+1$ such that the polynomials $f(x)$ and $f'(x)$ defined by their shares have different free coefficients. Thus at least one is not equal to $a_i \cdot b_i$.

We conclude that in order to carry out the verification required in the multiplication, we cannot use the verification strategy offered by known VSS protocols (in particular the one in BGW). This is because their strategy just guarantees that all parties output shares on a polynomial defined by *some* subset of $t+1$ of the honest parties' shares. Furthermore, any verification technique that provides only this guarantee cannot be used.

Therefore, a *new* verification protocol is needed that guarantees that the polynomial $C(x)$ is of degree-t *without* changing the value of the free coefficient of $C(x)$, i.e. by not changing the shares of the honest parties. Conceptually, this can be achieved by constructing a protocol that enables honest parties to prove that their share is incorrect, and by that prove that P_i has cheated. We stress that the VSS methodology does not achieve this property since in the case of inconsistencies the parties cannot know if the dealer or another party is cheating.

Our results. In this paper, we focus on perfect multiplication in the presence of up to $t < n/3$ malicious parties. We present two methods for carrying out the verification, along with a complete specification and proof of security of the resulting multiplication protocols. The first protocol works whenever the parties have univariate Shamir shares [19] of the input values. Thus, it does not depend on any specific properties of the VSS method used to initially share the values. Furthermore, it is close in flavor to the original protocol of BGW. The second protocol that we present utilizes the additional information given in a *bivariate* polynomial sharing (as used in the verifiable secret sharing of BGW and Feldman [4, 12]) in order to significantly improve the efficiency of each multiplication, while preserving a constant round complexity for a single multiplication. For example, we can completely eliminate the first step of the BGW multiplication protocol, which is to share the shares a_i and b_i. In addition to being more efficient, our resulting multiplication protocol is also significantly simpler. 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 first protocol (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.

In summary, our two protocols are incomparable. The first protocol is less efficient but works with *any* VSS of Shamir shares, and not necessarily with VSS that is based on bivariate techniques. The second protocol is simpler and more efficient but works only when the parties also have additional information on the shares that is a byproduct of the bivariate-based VSS protocol.

An additional important contribution of this paper is that we provide *full proofs of security* of all of our protocols and subprotocols, under the standard definitions of security following the ideal/real model paradigm [5, 11]. This includes the non-trivial definition of the ideal multiplication functionality and other subfunctionalities used. The full proof of the protocols in this paper together with a full proof of the entire BGW protocol (including the semi-honest case, the VSS protocol and more) appears in [1].

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 second protocol takes the first step by reducing the communication complexity of BGW and [8] while maintaining constant round complexity per multiplication.

Concurrent composition and adaptive security. Both of our protocols achieve *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 [18] 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 [6]. Second, it was shown in [7] that any protocol that is perfectly secure in the presence of malicious static adversaries under the definition of security of [10], is also secure in the presence of malicious adaptive corruptions. The additional requirements of the definition of [10] clearly hold for all BGW protocols and subprotocols. Thus, we obtain both adaptive security and universal composition for free.

Related work. We compare our second 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. [8]. 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. Further-

more, in the case that no parties actually cheat, the cost of [8] 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) 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 [15, 16, 2, 17, 9, 3]. This work culminated in linear communication complexity in [3], 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 protocol are actually higher; e.g., the round complexity of [15] 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.

2 Preliminaries and Tools

In this paper we will refer to a few functionalities which are described formally in the full version [1]. Here we give a brief description of these functionalities.

We use the following VSS functionality F_{VSS}. The dealer inputs a polynomial $f(x)$ of degree t, and the parties receive shares of that polynomial; i.e., party P_i receives $f(\alpha_i)$ where $\alpha_1, \ldots, \alpha_n$ are fixed elements in the finite field. The "verifiable" part is that if f is of degree greater than t, then the parties reject the dealer's shares and output $\perp$. Observe that the secret $s = f(0)$ is only implicitly defined in the functionality; it is however well defined.

The second functionality which we need is for sub-sharing of shares, denoted $F_{VSS}^{subshare}$. 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.[4] 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, and involves n invocations of VSS plus the transmission of $O(n^3)$ field elements over private channels. A full discussion of the complexity and the solution from BGW appear in the full version.

We denote by $I \subset [n]$ the indices of the (up to t) corrupted parties.

[4] If not all of the points lie on a single degree-t polynomial, then no security guarantees are obtained. Formally, this is achieved by defining that in this case the functionality sends the inputs of the honest parties to the corrupted parties, and sets the output of the honest parties to be whatever the adversary desires. In this way, any protocol is secure in this "bad case". From now on we just ignore this case, since our functionalities are used only when this property is fulfilled.

3 Verifying that a Shared Polynomial is of Degree t

As discussed in the introduction, in order to complete the BGW multiplication protocol we need a subprotocol that enables the parties to verify that the shares held by *all* the honest parties for $C(x)$, as computed in Step 2a of the BGW perfect multiplication described above, lie on the same degree-t polynomial. That is, the parties all hold shares of $A(x)$, $B(x)$, $D_1(x), \ldots, D_t(x)$ and they wish to verify that their shares of $A(x) \cdot B(x) - \sum_{\ell=1}^{t} x^\ell \cdot D_\ell(x)$ define a degree-t polynomial. In this section we show how to do this; the full multiplication protocol using this verification step appears in [1].

We carry out this verification step by first having the dealer share the polynomial $C'(x) = C(x)$ using F_{VSS}, and then having each party P_j verify that $C'(\alpha_j) = C(\alpha_j)$ (where $C'(\alpha_j)$ is its output from F_{VSS} and $C(\alpha_j)$ is the result of its computation based on its shares of $A(x), B(x)$ and $D_1(x), \ldots, D_t(x)$). If equality does not hold then the party complains. As we have explained, we cannot have the dealer change the share of a complaining party, but rather the party needs to "prove" that its share does not lie on the polynomial. This is achieved by having the parties run a subprotocol to check whether or not the complaint is legitimate. Note that the parties all hold shares of $C'(x)$ and $C(x)$ so in principle there is enough information to verify a complaint. However, care must be taken not to reveal more information than allowed, in case the complaint is false. If there is a legitimate complaint against the dealer then the computation halts. Otherwise, we are guaranteed that the degree-t polynomial $C'(x)$ shared using F_{VSS} agrees with the computed polynomial $C(x)$ on at least $2t + 1$ honest parties' shares. Since $C(x)$ is of degree at most $2t$ (recall that $C(x) = A(x) \cdot B(x) - \sum_{\ell=1}^{t} x^\ell \cdot D_\ell(x)$ where every $D_\ell(x)$ is guaranteed to be of degree at most t since it was shared using F_{VSS}), this implies that $C(x) = C'(x)$ and so it is actually of degree-t, with the desired free coefficient.

3.1 The Verification Protocol

The verification procedure is defined formally in Functionality 1.

FUNCTIONALITY 1 (The F_{vrfy} functionality)

1. F_{vrfy} receives the shares $\{\beta_j^A, \beta_j^B, \beta_j^{D_1}, \ldots, \beta_j^{D_t}, \beta_j^{C'}\}_{j \notin I}$ of honest parties.
2. Let $A, B, D_1, \ldots, D_t$ and C' be the polynomials that are defined from the shares $\beta_j^A, \beta_j^B, \beta_j^{D_1}, \ldots, \beta_j^{D_t}$, and $\beta_j^{C'}$, respectively; this assumes that the polynomials $A, B, D_1, \ldots, D_t, C'$ are all of degree-t (see Footnote 4). Functionality F_{vrfy} sends the corrupted parties' shares $\{A(\alpha_i), B(\alpha_i), D_1(\alpha_i), \ldots, D_t(\alpha_i), C'(\alpha_i)\}_{i \in I}$ to the adversary.
3. If $C'(x) = A(x) \cdot B(x) - \sum_{\ell=1}^{t} x^\ell \cdot D_\ell(x)$ then F_{vrfy} sends 1 to every party for output, otherwise it sends 0 to every party for output and sends $A(x), B(x), D_1(x), \ldots, D_t(x), C'(x)$ to the adversary.

We stress that $C'(x)$ was already shared using F_{VSS} and so is guaranteed to be of degree-t. Thus, the aim of the parties is to verify that the shared $C'(x)$

equals $A(x) \cdot B(x) - \sum_{\ell=1}^{t} x^{\ell} \cdot D_{\ell}(x)$. We also remark that the adversary always receives the corrupted parties' shares as part of the output, and receives all of the shares in the case that the honest parties' output is 0. The fact that the adversary always receives the corrupted parties' shares makes no difference since it already knows these shares in any setting where this functionality is used. However, this is needed for technical reasons in order to prove the security of our protocol (according to simulation-based definitions). Furthermore, the fact that it learns everything if the output is 0 makes no difference because when the shares of $A, B, D_1, \ldots, D_t, C'$ are all dealt by an honest party this case never happens, and when they are dealt by a corrupted party then the adversary already knows all the shares anyway.

The protocol. The protocol is very simple. Each party P_j locally computes $\beta_j^A \cdot \beta_j^B - \sum_{\ell=1}^{t} (\alpha_j)^{\ell} \cdot \beta_j^{D_\ell}$ and complains if the result does *not* equal $\beta_j^{C'}$. In such a case, all parties use F_{eval}^j described in Section 3.2 below to publicly reconstruct all the input shares of the complainant, *without* exposing the shares of the other participating parties. This enables all parties to determine whether or not the complaint was legitimate. We stress that public reconstruction does not reveal anything since if the complaint is legitimate and so the output is 0, everything is anyway allowed to be revealed to the adversary. Furthermore, if the complaint is not legitimate then the complainant is corrupt and all that is revealed are a corrupted party's shares. See Protocol 2 for full details.

PROTOCOL 2 (Securely computing F_{vrfy} in the F_{eval}^j-hybrid model)

- **Inputs:** Each party P_i holds shares $\beta_i^A, \beta_i^B, \beta_i^{D_1}, \ldots, \beta_i^{D_t}, \beta_i^{C'}$, all on the degree-$t$ polynomials $A, B, D_1, \ldots, D_t, C'$ (resp).
- **The protocol:**
  1. Each party P_i computes $\beta' = \beta_i^A \cdot \beta_i^B - \sum_{\ell=1}^{t} (\alpha_i)^{\ell} \cdot \beta_i^{D_\ell}$.
     If $\beta' \neq \beta_i^{C'}$, then P_i broadcasts $(\mathsf{complaint}, i)$.
  2. For every party P_j that broadcast $(\mathsf{complaint}, j)$ do the following:
     (a) Run $t+3$ invocations of F_{eval}^j: Each party P_i inputs $\beta_i^A, \beta_i^B, \beta_i^{D_1}, \ldots, \beta_i^{D_t}, \beta_i^{C'}$, respectively, in each of the invocations.
     (b) Let $\tilde{\beta}_j^A, \tilde{\beta}_j^B, \tilde{\beta}_j^{D_1}, \ldots, \tilde{\beta}_j^{D_t}$ and $\tilde{\beta}_j^{C'}$ be the respective outputs from the invocations.
     (c) If $\tilde{\beta}_j^{C'} \neq \tilde{\beta}_j^A \cdot \tilde{\beta}_j^B - \sum_{\ell=1}^{t} (\alpha_j)^{\ell} \cdot \tilde{\beta}_j^{D_\ell}$, then output 0 and halt.
  3. If the output was not already determined to be 0 then output 1.

Theorem 3 *Let $t < n/3$. Then, Protocol 2 t-securely computes the F_{vrfy} functionality in the F_{eval}^j-hybrid model, in the presence of a static malicious adversary.*

The proof of this theorem is implicit in [1] (in the proof of security of the F_{VSS}^{mult} functionality).

3.2 Complaint Verification – The F^j_{eval} Functionality

When an honest party P_j complains, this implies that $C'(\alpha_j) \neq A(\alpha_j) \cdot B(\alpha_j) - \sum_{\ell=1}^{t} (\alpha_j)^\ell \cdot D_\ell(\alpha_j)$. In order to verify whether this is a legitimate complaint we need to reconstruct all the input shares of P_j, i.e. $\beta_j^A, \beta_j^B, \beta_j^{D_1}, \ldots, \beta_j^{D_t}, \beta_j^{C'}$, without revealing anything else. Note, that each such value can be calculated from the values of the honest parties as they all define a polynomial of degree t. Thus, using this information we can "extract" the values of the complaining party. However, this must be done without revealing anything but the complainants shares. We begin by formally defining the functionality; the functionality is parameterized by an index j that determines which party's share is to be revealed; equivalently, at which point the shared polynomial is to be evaluated.

FUNCTIONALITY 4 (The F^j_{eval} functionality)

1. The F^j_{eval} functionality receives the inputs of the honest parties $\{\beta_i\}_{i \notin I}$. Let $f(x)$ be the unique degree-t polynomial determined by the points $\{(\alpha_i, \beta_i)\}_{i \notin I}$. (If not all the points lie on a single degree-t polynomial, then no security guarantees are obtained. See Footnote 4.)
2. The functionality F^j_{eval} sends the output pair $(f(\alpha_i), f(\alpha_j))$ to every party P_i $(i = 1, \ldots, n)$.

We remark that although each party P_i already holds $f(\alpha_i)$ as part of its input, we need the output to include it in order to simulate in the case that a corrupted party has incorrect input. This will not make a difference in its use, since $f(\alpha_i)$ is supposed to be known to P_i.

In this paper we provide two methods for computing F^j_{eval} that are dependent on the specific implementation that we use for the secret sharing. In the following we describe the first implementation that uses univariate polynomials. The second solution uses bivariate sharing and will be given in Section 4.

Background. The parties' inputs are a (row) vector $\vec{\beta} \stackrel{\text{def}}{=} (\beta_1, \ldots, \beta_n)$ where for every i it holds that $\beta_i = f(\alpha_i)$. Thus, the parties' inputs are computed by $\vec{\beta} = V_\alpha \cdot \vec{f}$, where V_α is the $n \times (t+1)$ Vandermonde matrix with $\alpha_1, \ldots, \alpha_n$ and $\vec{f}$ is the length $t+1$ (column) vector of coefficients of the polynomial $f(x)$. Let $\vec{\alpha}_j = (1, \alpha_j, (\alpha_j)^2, \ldots, (\alpha_j)^t)$ be the jth row of V_α. Then the output of the functionality is $f(\alpha_j) = \vec{\alpha}_j \cdot \vec{f}$. We have:

$$\vec{\alpha}_j \cdot \vec{f} = \vec{\alpha}_j \cdot \left(V_\alpha^{-1} \cdot V_\alpha\right) \cdot \vec{f} = \left(\vec{\alpha}_j \cdot V_\alpha^{-1}\right) \cdot \left(V_\alpha \cdot \vec{f}\right) = \left(\vec{\alpha}_j \cdot V_\alpha^{-1}\right) \cdot \vec{\beta}$$

where V_α^{-1} is the left inverse of V_α, of degree $(t+1) \times n$. Thus, there exists a vector of *constants* $(\vec{\alpha}_j \cdot V_\alpha^{-1})$ so that the inner product of this vector and the inputs yields the desired result. In other words, F^j_{eval} is just a linear function of the parties' inputs.

The protocol. Since F^j_{eval} is simply a linear function of the parties' inputs, it can be computed by each party sharing its share and then locally computing the function on the subshares. The result is that each party P_i holds a share

δ_i of a polynomial whose free coefficient is the result $f(\alpha_j)$. Thus, the parties can now simply send their δ_i shares to each other and reconstruct the resulting polynomial. This suffices for the semi-honest case. However, malicious parties may send incorrect shares and try to cheat. In order to prevent them from doing this, the $F_{VSS}^{subshare}$ functionality is used in order to share the shares; see Section 2. Then, the reconstruction in the last stage is carried out using Reed-Solomon decoding to correct any bad values sent by the malicious parties. This ensures that $t < n/3$ malicious parties cannot affect the result. See Protocol 5 for the full description.

PROTOCOL 5 (Computing F_{eval}^{j} in the $F_{VSS}^{subshare}$-hybrid model)

- **Inputs:** Each party P_i holds a value β_i; we assume that the points (α_i, β_i) for every honest P_i all lie on a single degree-t polynomial f (see the definition of F_{eval}^{j} above and Footnote 4)
- **The protocol:**
  1. The parties invoke the $F_{VSS}^{subshare}$ functionality with each party P_i using $\beta_i = f(\alpha_i)$ as its private input.
  2. 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, $g_i(0) = \beta_i$.
  3. Each party P_i locally computes: $H(\alpha_i) = \sum_{\ell=1}^{n} \gamma_\ell \cdot g_\ell(\alpha_i)$, where $(\gamma_1, \ldots, \gamma_n) = \vec{\alpha}_j \cdot V_\alpha^{-1}$. Each party P_i sends $H(\alpha_i)$ to every other party.
  4. Upon receiving $(\hat{H}(\alpha_1), \ldots, \hat{H}(\alpha_n))$, each party runs the Reed-Solomon decoding procedure and receives $(H(\alpha_1), \ldots, H(\alpha_n))$. It then reconstructs $H(x)$ and computes $H(0)$.
  5. Each party P_i outputs $(\beta_i, H(0))$.

Theorem 6 *Let $t < n/3$. Then, Protocol 5 t-securely computes the F_{eval}^{j} functionality in the $F_{VSS}^{subshare}$-hybrid model, in the presence of a static malicious adversary.*

The motivation behind the security of the protocol appears above, and a full proof of Theorem 6 appears in [1].

4 Efficient Multiplication using Bivariate VSS

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. [15] also observed that the use of the bivariate polynomial can offer efficiency improvements; however they do not utilize this to the fullest extent possible. In this section 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. As we have discussed in Section 2, $F_{VSS}^{subshare}$ is expensive and so this is a significant improvement.

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 $(\alpha_i, f(\alpha_i))$. Multiplication then works as follows:

1. *Subsharing* – $F_{VSS}^{subshare}$*:* Given shares $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ of values a and b, each party *shares its shares* to all other parties. This step is carried out using $F_{VSS}^{subshare}$, as described above.
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 section.
3. *Linear combination:* As described in [14], once the parties all hold shares of $a_1 \cdot b_1, \ldots, a_n \cdot b_n$, 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 $a \cdot b$.

In our proposed protocol, we have an analogous invariant (as in [15]): each wire in the circuit is hidden by a (random) *bivariate polynomial* $F(x, y)$ of degree-t in both variables. As a result, the share of each party is the pair of degree-t polynomials $(F(x, \alpha_i), F(\alpha_i, y))$. 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 $F(\alpha_i, 0)$. In contrast, we will preserve the shares of the bivariate but at times will also use univariate polynomials.

In the full version of this paper [1] we define the functionalities required, including a redefinition of the VSS protocol of BGW where the output includes the bivariate shares. In addition, we show how to reconstruct and add shared secrets in this format, and provide the full details of the multiplication protocol and its proof. In what follows we focus on our new protocols and techniques. First, we will explain why the first step of computing $F_{VSS}^{subshare}$ is not needed when bivariate shares are maintained. Next, we describe a functionality that enables the conversion (or extension) of a univariate polynomial secret sharing into a bivariate secret sharing. We then use a combination of the above to securely compute the bivariate analogue of the complaint verification functionality F_{eval}^{j} (see Section 3.2). Finally, we use all of the above to construct a simpler and more efficient version of F_{VSS}^{mult}.

From here on, we assume that there are two secrets a and b that were shared amongst the parties, and we denote by $F_A(x, y)$ and $F_B(x, y)$ the bivariate polynomials that hide a and b, respectively. The shares of party P_i are defined to be the pairs of univariate polynomials $F_A(x, \alpha_i), F_A(\alpha_i, y)$ and $F_B(x, \alpha_i), F_B(\alpha_i, y)$, respectively.

4.1 $F_{VSS}^{subshare}$ 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.

Thus, in the univariate case, the parties first run the $F_{VSS}^{subshare}$ protocol at the cost of n executions of VSS plus the transmission of $O(n^3)$ field elements. 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, recall that each party P_i holds shares $F_A(x, \alpha_i), F_A(\alpha_i, y)$. Based on this, we can define the univariate "Shamir" sharing of a via the polynomial $f_a(x) \stackrel{\text{def}}{=} F_A(x, 0)$ as in the original BGW protocol; due to the properties of the bivariate sharing, $f_a(x)$ is a univariate polynomial of degree-t that hides a. Furthermore, since each party P_i holds the polynomial $F_A(\alpha_i, y)$, it can locally compute its share $a_i = F_A(\alpha_i, 0) = f_a(\alpha_i)$ on the univariate polynomial $f_a(x)$.

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 $g_{a_i}(y) = F_A(\alpha_i, y)$. This is due to the fact that each P_j holds the polynomial $F_A(x, \alpha_j)$ and so can compute $a_i^j = F_A(\alpha_i, \alpha_j) = g_{a_i}(\alpha_j)$. Observe that by the definition of the bivariate polynomial $F_A(x, y)$, it holds that $g_{a_i}(y)$ is a degree-t univariate polynomial. Furthermore, $g_{a_i}(0) = F_A(\alpha_i, 0) = a_i$ and each $a_i^j = g_{a_i}(\alpha_j)$. In other words, the values a_i^j that are locally computed by each party P_j are valid univariate shares of a_i, which is the univariate share of P_i in the polynomial $f_a(x)$ that hides a. We conclude that 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 that they already obtained in the VSS stage. (Of course, these bivariate shares need to be maintained throughout the circuit computation phase; we show how this is achieved below.)

4.2 Transformation from Univariate to Bivariate – $\widetilde{F}_{extend}$

As we will show below (in Section 4.3) 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[5] $\widetilde{F}_{extend}$ 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 7 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 $S(x, 0) = q(x)$).

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 (securely computing the bivariate VSS func-

[5] By convention, we denote bivariate-sharing based functionalities with a tilde.

FUNCTIONALITY 7 (The Reactive Functionality $\widetilde{F}_{extend}$)

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; see Footnote 4.)
2. In case that the dealer is corrupted, $\widetilde{F}_{extend}$ sends $q(x)$ to the 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 x, y, *and* $S(x,0)=q(x)$.
4. If both conditions hold, $\widetilde{F}_{extend}$ accepts the bivariate polynomial $S(x,y)$, and sends to each party P_j the pair of polynomials $(f_j(x), g_j(y))$ (which are $(S(x,\alpha_j), S(\alpha_j,y))$).
5. If either of the conditions do not hold, $\widetilde{F}_{extend}$ rejects the bivariate polynomial $S(x,y)$ and sends to each party P_j the value $\perp$.

tionality $\widetilde{F}_{VSS}$,[6] described in the full version [1]). Each party receives shares $S(x,\alpha_i), S(\alpha_i,y)$, and checks that $S(\alpha_i,0)=q(\alpha_i)$. If not, it broadcasts a complaint. The parties accept the shares of $S(x,y)$ if and only if there are at most t complaints. A formal description of the protocol appears in the full version. Before proceeding to describe why this protocol securely computes $\widetilde{F}_{extend}$, we remark that its cost is just a single VSS invocation and at most $O(n)$ broadcasts.

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 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 [1], 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,

[6] This functionality receives a bivariate polynomial $S(x,y)$ and hands each party P_i shares $S(x,\alpha_i), S(\alpha_i,y)$ if and only if $S(x,y)$ is of degree t in both variables.

then *all* of the bivariate shares must be consistent with *all* of the honest parties' points.

4.3 Bivariate Complaint Verification – The $\widetilde{F}^{k}_{eval}$ Functionality

In order to deal with complaint verification as discussed in the beginning of Section 3, we define an analogous functionality to F^{j}_{eval} in the bivariate setting. That is, given a sharing of a bivariate polynomial $S(x,y)$ of degree-t in both variables, the parties wish to evaluate the bivariate polynomial on some point α_k, or equivalently to learn the pair of polynomials $S(x,\alpha_k)$, $S(\alpha_k,y)$. Here, however, the implementation of this functionality is much easier than the implementation of F^{j}_{eval} in the univariate setting (we use the index k here instead of j since the bivariate setting requires additional indices). The $\widetilde{F}^{k}_{eval}$ functionality is defined as follows:

FUNCTIONALITY 8 (The Functionality $\widetilde{F}^{k}_{eval}$)

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 4).
2. $\widetilde{F}^{k}_{eval}$ sends every party P_i the polynomials $(S(x,\alpha_k), S(\alpha_k,y))$.

The protocol computing $\widetilde{F}^{k}_{eval}$ is straightforward and very efficient. Given input $(f_i(x), g_i(y))$ (which under the assumption on the inputs as in Footnote 4 equals $S(x,\alpha_i)$, $S(\alpha_i,y)$), each party P_i 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. Once a party holds all the points $\{S(\alpha_j,\alpha_k)\}_{j\in[n]}$, it can reconstruct the polynomial $f_k(x) = S(x,\alpha_k)$, and likewise $g_k(y) = S(\alpha_k,y)$ from $\{S(\alpha_k,\alpha_j)\}_{j\in[n]}$. Since $S(x,y)$ is of degree-t in both variables, the polynomials $f_k(x) = S(x,\alpha_k)$ and $g_k(y) = S(\alpha_k,y)$ are both of degree-t, and thus each party can reconstruct the polynomials even if the corrupted parties sent incorrect values, 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.

4.4 The $\widetilde{F}^{mult}_{VSS}$ Functionality for Sharing a Product of Shares

As we have described in the Introduction and in the beginning of Section 4, the main step for achieving secure multiplication is a method for a party P_i to share the product of its shares $a_i \cdot b_i$, while preventing a corrupted P_i from sharing an incorrect value. In the univariate case, the parties use $F^{subshare}_{VSS}$ to first share their shares, and then use F^{mult}_{VSS} 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_i, b_i, 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_i and b_i using $\widetilde{F}_{extend}$. Our goal with $\widetilde{F}_{VSS}^{mult}$ is for the parties to obtain a sharing of $a_i \cdot b_i$, by holding shares of a bivariate polynomial $C(x, y)$ whose constant term is the desired product.

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. We also write the functionality and protocol with P_1 as the dealer (i.e., the party who has shares a and b and wishes to share $a \cdot b$); in the full multiplication, each party plays the dealer in turn. See Functionality 9 for a specification of this step.

FUNCTIONALITY 9 (The reactive $\widetilde{F}_{VSS}^{mult}$ functionality)

1. The $\widetilde{F}_{VSS}^{mult}$ functionality receives input shares (a_j, b_j) from every honest party P_j ($j \notin I$).
2. $\widetilde{F}_{VSS}^{mult}$ computes the unique degree-t polynomials $A'(x)$ and $B'(x)$ 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, then see Footnote 4).
3. $\widetilde{F}_{VSS}^{mult}$ sends $(A'(x), B'(x))$ to the dealer P_1.
4. $\widetilde{F}_{VSS}^{mult}$ receives a bivariate polynomial $C(x, y)$ from P_1, and chooses $C^*(x, y)$ as follows:
   (a) If the input is the special symbol $*$, then $\widetilde{F}_{VSS}^{mult}$ chooses a random bivariate polynomial $C^*(x, y)$ of degree-t in both variables under the constraint that $C^*(0, 0) = A'(0) \cdot B'(0)$.
   (b) Else, if the input is a bivariate polynomial C such that $\deg(C) = t$ in both variables and $C(0, 0) = A'(0) \cdot B'(0)$, then $\widetilde{F}_{VSS}^{mult}$ sets $C^* = C$.
   (c) Otherwise, if either $\deg(C) > t$ or $C(0, 0) \neq A'(0) \cdot B'(0)$, then $\widetilde{F}_{VSS}^{mult}$ sets $C^*(x, y) = A'(0) \cdot B'(0)$ to be the constant polynomial equalling $A'(0) \cdot B'(0)$ everywhere.
5. $\widetilde{F}_{VSS}^{mult}$ sends $C^*(x, y)$ to the dealer P_1, and sends $(A'(\alpha_i), B'(\alpha_i), C^*(x, \alpha_i), C^*(\alpha_i, y))$ to P_i for every $i = 1, \ldots, n$.

The special input symbol $*$ is an instruction for the trusted party computing $\widetilde{F}_{VSS}^{mult}$ to choose the polynomial $C^*(x, y)$ determining the output shares uniformly at random.

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 in the case that the corrupted parties use incorrect inputs.

The protocol. As in the univariate case, the protocol for implementing this functionality is based on the BGW method "(II) Verifying that $c = a \cdot b$", with

the addition of complaint verification. In addition, here the parties will output bivariate and not univariate shares.

As described in the Introduction, the dealer chooses the univariate polynomials $D_1(x), \ldots, D_t(x)$ as instructed in BGW; see the full version for a detailed description of this. It then distributes them using bivariate polynomials that hide them. That is, in order to distribute a polynomial $D_i(x)$, the dealer selects a bivariate polynomial $D_i(x,y)$ uniformly at random under the constraint that $D_i(x,0) = D_i(x)$, and then shares it using the bivariate VSS 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 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 (see the brief explanation in the introduction), 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 free coefficient $a \cdot b$, 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 $D_i(x,y)$ 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 $C'(x)$.

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

$$C(x,0) = C'(x) = A(x) \cdot B(x) - \sum_{k=1}^{t} x^k \cdot D_k(x,0)$$

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, as in the univariate case, 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 similar to the univariate case, described in Section 3.1. That is, the parties evaluate each one of the polynomials $D_1, \ldots, D_t$, A, B, and C on the point of the complaining party. However, this time we use the bivariate evaluation functionality $\widetilde{F}^k_{eval}$ (see Section 4.3) instead of the univariate one F^j_{eval}. Observe that all of the

polynomials $D_1, \ldots, D_t, C$ are bivariate and of degree-t, and so the bivariate $\widetilde{F}_{eval}^{k}$ can be used. In contrast, $A(x)$ and $B(x)$ are only univariate polynomials and so $\widetilde{F}_{extend}$ (see Section 4.2) 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, $\widetilde{F}_{eval}^{k}$ 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 11 for a full specification.

We have the following theorem, that is proven in the full version.

Theorem 10 *Let $t < n/3$. Then, Protocol 11 t-securely computes the $\widetilde{F}_{VSS}^{mult}$ functionality in the $(\widetilde{F}_{VSS}, \widetilde{F}_{eval}^{k}, \widetilde{F}_{extend})$-hybrid model, in the presence of a static malicious adversary.*

PROTOCOL 11 (Securely computing $\widetilde{F}_{VSS}^{mult}$)

- **Inputs:** The dealer P_1 holds degree-t polynomials $A(x)$ and $B(x)$. Each party P_i holds a pair of shares a_i and b_i such that $a_i = A(\alpha_i)$ and $b_i = B(\alpha_i)$.
- **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 $\{r_{k,j}\}$ uniformly and independently at random from $\mathbb{F}$, where $k = 1, \ldots, t$, and $j = 0, \ldots, t-1$. For every $k = 1, \ldots, t$, the dealer defines the polynomial $D_k(x)$:
     $$D_k(x) = \sum_{\ell=0}^{t-1} r_{k,\ell} \cdot x^{\ell} + \left(d_{k+t} - \sum_{j=k+1}^{t} r_{j,t+k-j} \right) \cdot x^t.$$
     (c) P_1 computes the polynomial:
     $$C'(x) = D(x) - \sum_{k=1}^{t} x^k \cdot D_k(x).$$
     (d) P_1 chooses t random degree-t bivariate polynomials $D_1(x,y)$, $\ldots, D_t(x,y)$ under the constraint that $D_k(x,0) = D_k(x)$ for every $k = 1, \ldots, t$. 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 $D_k(x,y)$ for every $k = 1, \ldots, t$.

Protocol for securely computing $\widetilde{F}_{VSS}^{mult}$ (continued):

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 $c'(i) \stackrel{\text{def}}{=} a_i \cdot b_i - \sum_{k=1}^{t} (\alpha_i)^k \cdot D_k(\alpha_i, 0)$. 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_j broadcast $(\mathsf{complaint}, j)$ 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 $(A(x), B(x), C(x, y))$.

4.5 Wrapping Things Up – Perfectly-Secure Multiplication

Given bivariate shares of the input wires to a multiplication gate, the parties each in turn play the dealer in $\widetilde{F}_{VSS}^{mult}$. At the end of these executions, all parties hold bivariate shares of the product of all other parties shares (recall that a bivariate share is a pair of univariate polynomials). As in [14], the parties can obtain bivariate shares of the product of the input-wire values by just carrying out a local computation on their shares. This therefore concludes the multiplication protocol. A full description and proof appears in the full version.

Efficiency analysis. A detailed efficiency analysis of the protocols appears in [1]. 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.

References

1. G. Asharov, Y. Lindell and T. Rabin. A Full Proof of the Perfectly-Secure BGW Protocol and Improvements. *Cryptology ePrint Archive*, 2011/136, 2011.
2. Z. Beerliová-Trubíniová and M. Hirt. Efficient Multiparty Computation With Dispute Control. In the 3*rd TCC*, Springer (LNCS 3876), pages 305-328, 2006.
3. Z. Beerliová-Trubíniová and M. Hirt. Perfectly-Secure MPC with Linear Communication Complexity. In 5*th TCC*, Springer (LNCS 4948), pages 213–230, 2008.
4. M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In 20*th STOC,* pages 1–10, 1988.
5. R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
6. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In 42*nd FOCS*, pages 136–145, 2001.
7. R. Canetti, I. Damgård, S. Dziembowski, Y. Ishai and T. Malkin: Adaptive versus Non-Adaptive Security of Multi-Party Protocols. In the *Journal of Cryptology* 17(3):153–207, 2004.
8. Ronald Cramer, Ivan Damgård and Ueli M. Maurer. General Secure Multi-party Computation from any Linear Secret-Sharing Scheme. In *EUROCRYPT 2000,* Springer (LNCS 1807), pages 316–334, 2000. A more detailed version appears in the BRICS Report Series RS-97-28, November 1997.
9. I. Damgård and J.B. Nielsen. Scalable and Unconditionally Secure Multiparty Computation. In *CRYPTO 2007*, Springer (LNCS 4622), pages 572–590, 2007.
10. Y. Dodis and S. Micali. Parallel Reducibility for Information-Theoretically Secure Computation. In *CRYPTO 2000*, Springer (LNCS 1880), pages 74–92, 2000.
11. O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications*. Cambridge University Press, 2004.
12. P. Feldman. Optimal Algorithms for Byzantine Agreement. *PhD thesis, Massachusetts Institute of Technology,* 1988.
13. P. Feldman and S. Micali. An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement. In *SIAM - Journal on Computing*, 26(4):873-933, 1997.
14. R. Gennaro, M.O. Rabin and T. Rabin. Simplified VSS and Fact-Track Multiparty Computations with Applications to Threshold Cryptography. In the 17*th PODC*, pages 101–111, 1998.
15. M. Hirt, U.M. Maurer, B. Przydatek. Efficient Secure Multi-party Computation. In *ASIACRYPT 2000*, Springer (LNCS 1976), pages 143–161, 2000.
16. M. Hirt and U. Maurer. Robustness for Free in Unconditional Multi-Party Computation. In *CRYPTO 2001*, Springer (LNCS 2139), pages 101-118, 2001.
17. M. Hirt and J.B. Nielsen. Robust Multiparty Computation with Linear Communication Complexity. In *CRYPTO 2006*, Springer (LNCS 4117), pages 463-482, 2006.
18. E. Kushilevitz, Y. Lindell and T. Rabin. Information-Theoretically Secure Protocols and Security Under Composition. In the *SIAM Journal on Computing*, 39(5):2090–2112, 2010.
19. A. Shamir. How to Share a Secret. In *Communications of the ACM*, 22(11):612–613, 1979.