Covert Security with Public Verifiability: Faster, Leaner and Simpler

Cheng Hong
Alibaba Group

Jonathan Katz
UMD

Vlad Kolesnikov
Georgia Tech

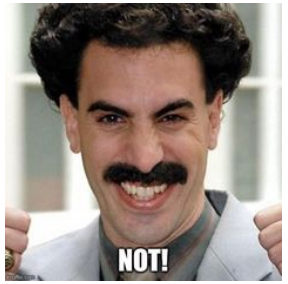
Wen-jie Lu
U. Tsukuba

Xiao Wang
MIT/BU

Outline

2PC Covert secure computation
PVC: Publicly Verifiable Covert secure computation

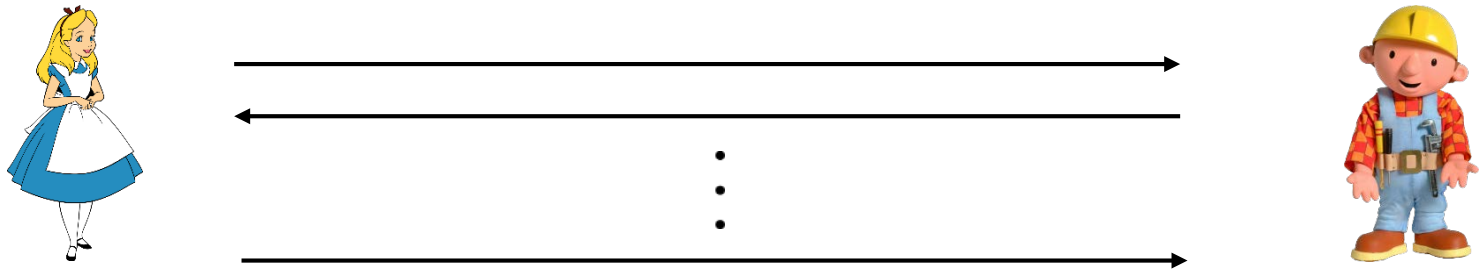
Long line of work: (NOT!)
[AsharovOrlandi12]
[KolesnikovMalozemoff15]



This work

Secure computation

$f(\cdot, \cdot)$



x y

Output: $f(x, y)$

 and  learn $f(x, y)$ while revealing *nothing else* about their inputs

Secure computation using garbled circuits (GCs)

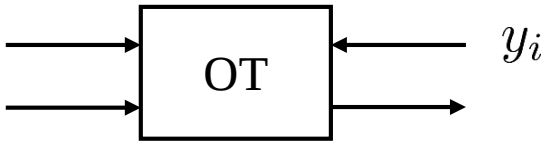
x

$f(\cdot,\cdot)$

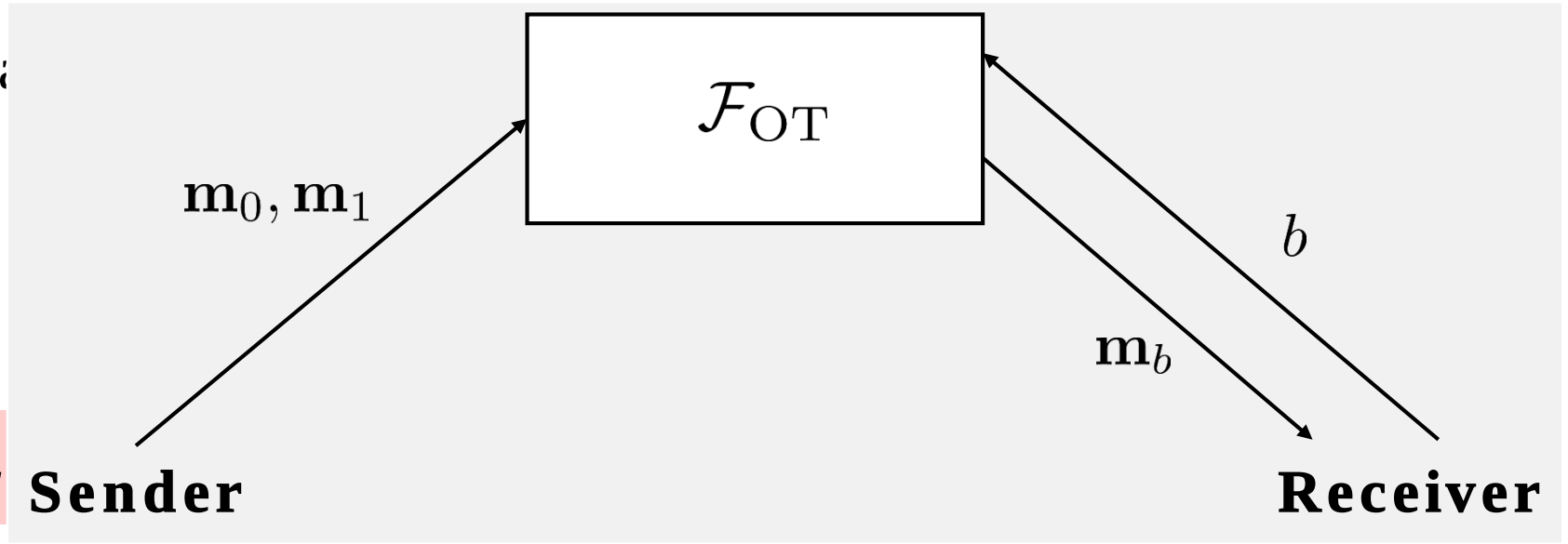
y

Step 1: Construct GC of f

Step 2:



Step 3: Send GC, wire la



Secure against ***semi-***

Adversary models

Semi-honest: Parties follow protocol

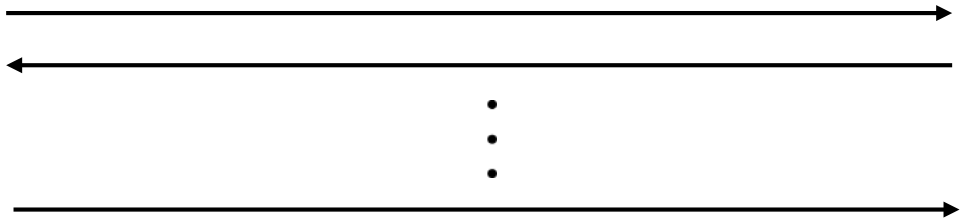
Covert: Malicious party can cheat, but gets caught with fixed probability

Malicious: No cheating possible

Covert security [AumannLindell07]

$f(\cdot, \cdot)$

x



y

Output: $f(x, y)$

can successfully **cheat** with probability $1 - \epsilon$

Covert security [AumannLindell07]

ϵ : deterrence factor

Probability ϵ :  gets **caught**

Probability $1-\epsilon$:  **cheats**

$\Rightarrow$ *very* efficient constructions (vs. malicious)

e.g. for $\epsilon = 1/2$, only *one* GC sent

Covert security using GCs [AumannLindell07]

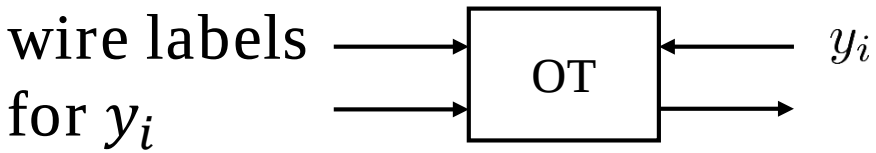
x

$f(\cdot,\cdot)$

y

Step 1: Construct λ GCs of f

Step 2:



Step 3: Send λ GCs

Step 4: Open $\lambda - 1$ GCs

Step 5: Evaluate leftover GC

Covert security: wish list

Put it on Blockchain

Covert security: wish list

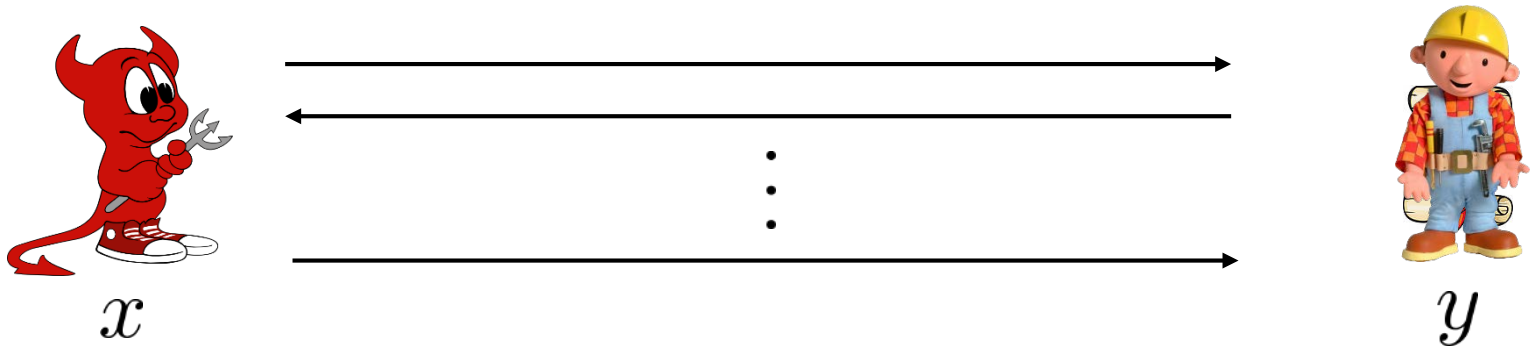
If  detects cheating, cannot *prove* to any 3rd party that cheating took place!

Would like some ***public verifiability*** condition

Publicly verifiable covert (PVC) security [AsharovOrlandi12]

$f(\cdot,\cdot)$

cheating detected!



x y

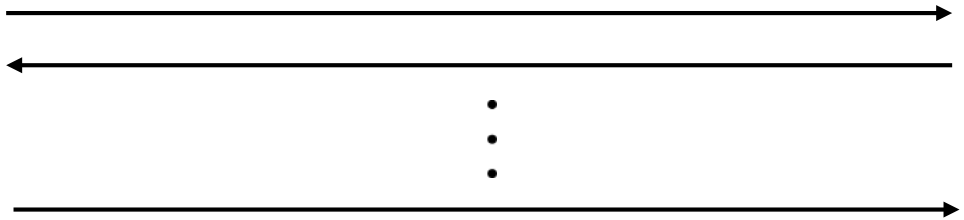
Output: $f(x,y)$

Publicly verifiable covert (PVC) security [AsharovOrlandi12]

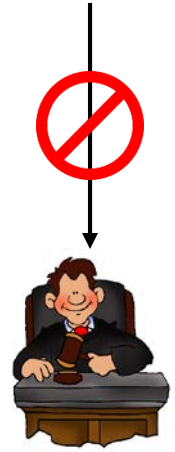
$f(\cdot,\cdot)$

let's frame Alice!

x

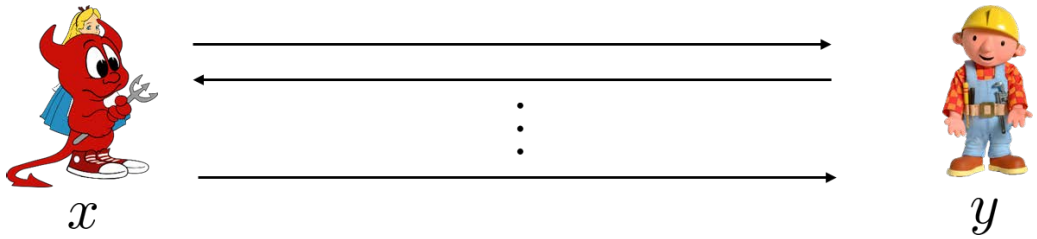


y

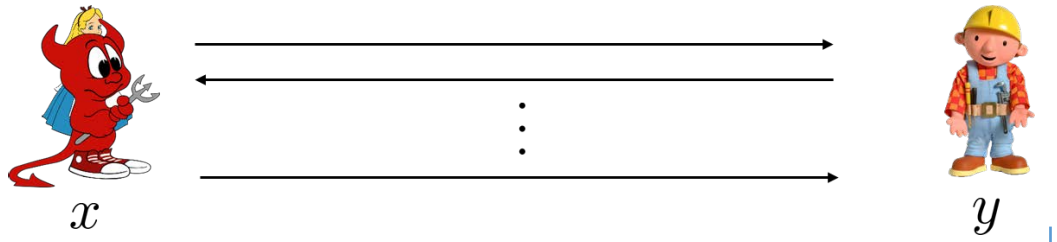


Output: $f(x,y)$

PVC



Malicious



PVC

Contract: pay $1000 to whoever can provide certificate of PVC cheating

Blockchain

PVC how-to

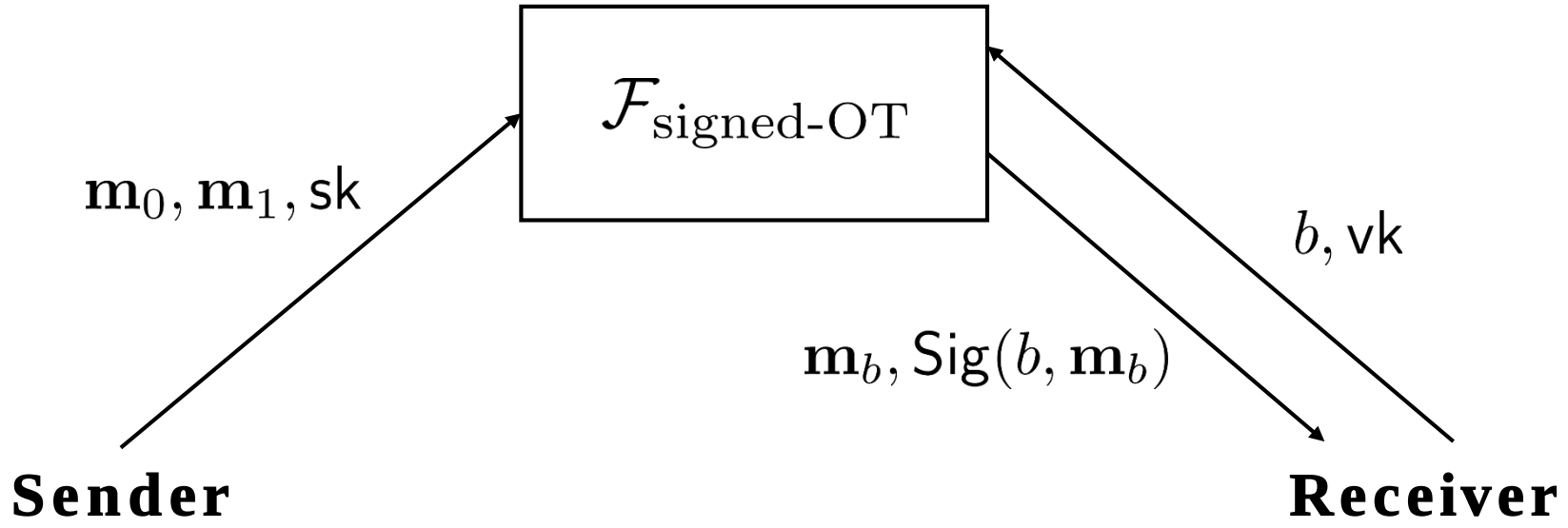
Just ask Alice to sign everything!

Issue:

1. Alice can abort (e.g. by sending a wrong sig) if asked to verify a malicious circuit she constructed.
2. Can do this even if verification is via OT.

PVC security using GCs [AsharovOrlandi12]

Same* as covert protocol, except need to use ***signed***-OT



Realized using existing (maliciously-secure) OT protocol [PeikertVaikuntanathanWaters08]

PVC protocol sketch [AsharovOrlandi12]

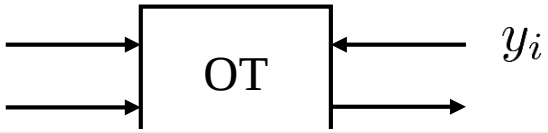
x

$f(\cdot,\cdot)$

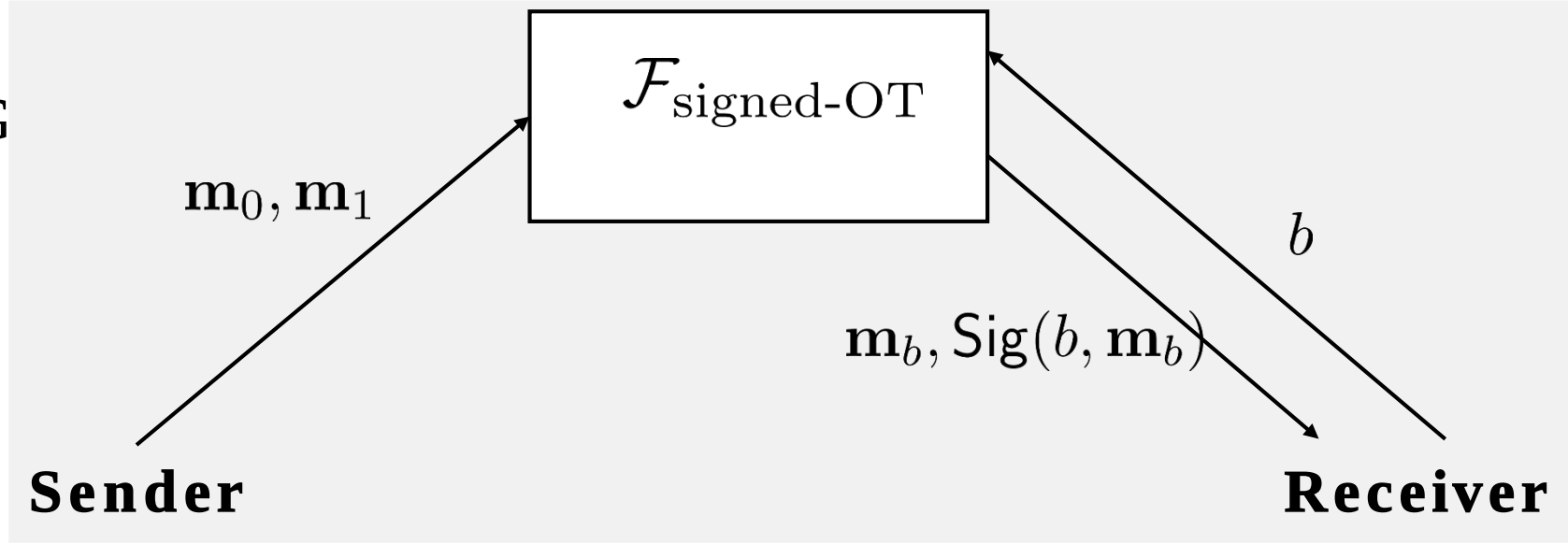
y

Step 1: Construct λ GCs of f

Step 2: Signed wire labels for y_i



Step 3: Send λ signed G



PVC security using GCs [AsharovOrlandi12]

Wish list:

1. Signed-OT not compatible with OT extension

2. Existing PVC protocol less efficient than best known covert protocol [GoyalMohasselSmith08]

⇒ Cost of PVC *much more* than that of covert

Kolesnikov-Malozemoff KM15

PVC security (almost) *at the cost of* covert

1. Signed-OT extension protocol

2. Various optimizations

KM15 wish list

Signed-OT extension:
Tailored construction, highly detailed,
Intertwined with the base OT extension
    Needs to be updated each time OT ext is improved
No implementation
Large cheating witness (think Blockchain!)

Our new idea

Get rid of signed-OT
Build from off-the-shelf standard primitives

Idea:
Derandomize + random seed is the witness for everything
Somehow make Alice sign the seed

Our new idea

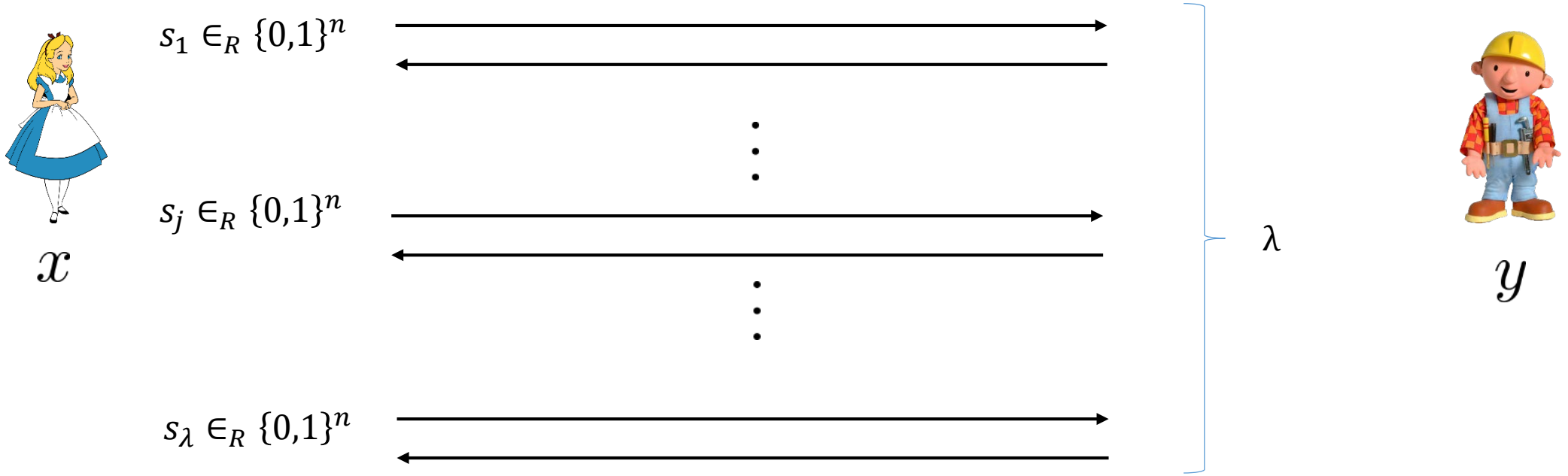
Random seed is the witness for everything:

Alice chooses seed $s_i \in_R \{0,1\}^n, i \in [1,..,\lambda]$
Alice derives all randomness from s_i *including of running OT*
for each of the λ instances
$\Rightarrow s_i$ is a witness.

This is simplification/kicking the can down the road
Let's make sure Alice always signs it...

Presentation: 1) How to catch Alice 2) How to prove cheating

Catch cheating Alice: derandomize Π_{GC}



Cut-and-choose: λ instances, random seeds s_i.
Bob *obliviously* (via OT) learns all $s_i \neq s_j$.
From this point, for $s_i \neq s_j$, Alice's execution is *deterministic!*

In each instance $i \in [1..\lambda]$, Alice garbles, sends hashes, performs OT etc based on seeds s_i.
Cheating=deviation from deterministic protocol based on s_i is easily detected by Bob.

In pictures… derandomize Π_{GC}

x

Instance i of λ

$s_i \in_R \{0,1\}^n$

PRG

OT

s_i if $i \neq j$

y

x

Π^i_{GC}

λ

Check consistency of messages in Π^i_{GC} wrt s_i

Blame cheating Alice: start signing

x

$s_i \in_R \{0,1\}^n$

PRG

OT

s_i if $i = j$

y

λ

Instance i of λ

x

$\sigma - \Pi^{i}_{GC}$

$\sigma - \Pi^{i}_{GC}$ is Π^{i}_{GC}, where Alice signs each message she sends

Inconsistent messages in $\sigma - \Pi^{i}_{GC}$ wrt s_i is now Cert

Still need to link the proof to the original seeds s_i. Alice may refuse to sign s_i for a cheating instance. How can we force her?

Idea: OT transcript (where each msg is signed by Alice) is a signature on the OT output!

Blame cheating Alice: sign everything

x

$s_i \in_R \{0,1\}^n$

PRG

Instance i of λ

x

σ −OT

$\sigma - \Pi^{i}_{GC}$

s_i if $i = j$

λ

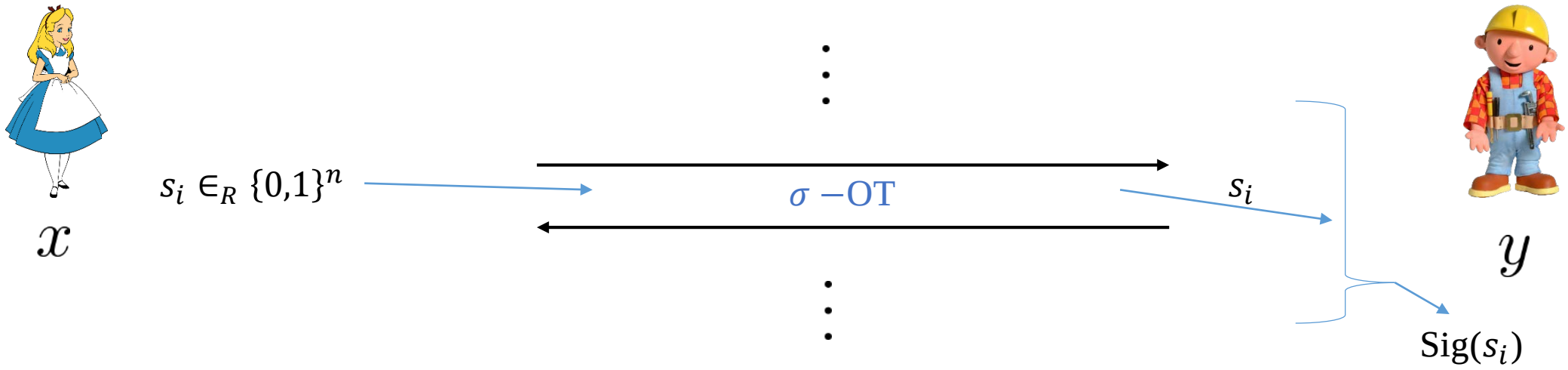
y

σ −OT transcript is a signature on s_i

$\sigma - OT$ is OT where Alice signs each message she sends

Idea: OT transcript (where each msg is signed by Alice) is a signature on the OT output!

Blame Alice: Honest Bob



Idea: σ −OT transcript is a signature on the OT output! (given honest Bob's R, y)

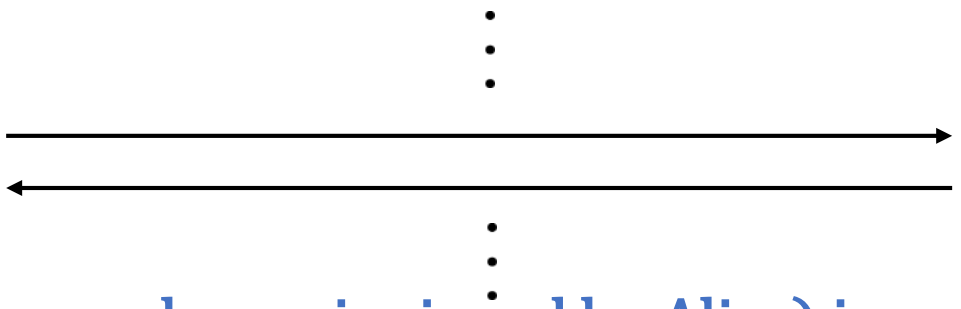
Why?

Alice's sequence of messages defines s_i (given R,y). So signed transcript is the signature

Blame Alice: arbitrary Bob

x

$s_j \in_R \{0,1\}^n$



y

Idea: OT transcript (where each msg is signed by Alice) is a signature on the OT output!

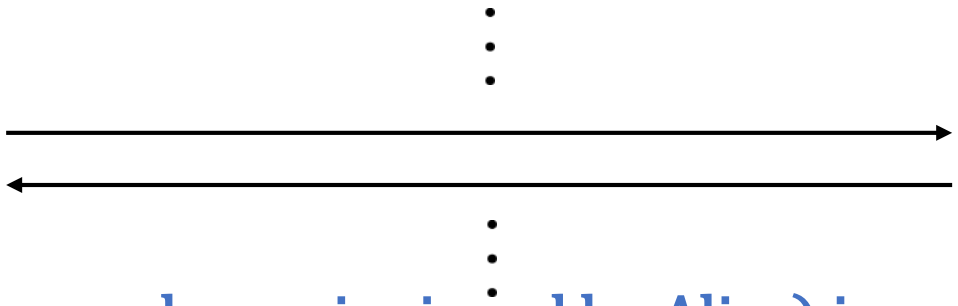
Well, this only makes sense if Bob is honest (malicious Bob can influence the protocol output he receives and then perhaps blame Alice).

Idea 2: derandomize Bob as well

Blame Alice: sign everything (even sign Bob)

x

$s_j \in_R \{0,1\}^n$



y

Idea: OT transcript (where each msg is signed by Alice) is a signature on the OT output!

Idea 2: derandomize Bob as well

Make Bob's randomness derived from seed s^B.

Bob commits $c^B = Com\ (s^B)$.

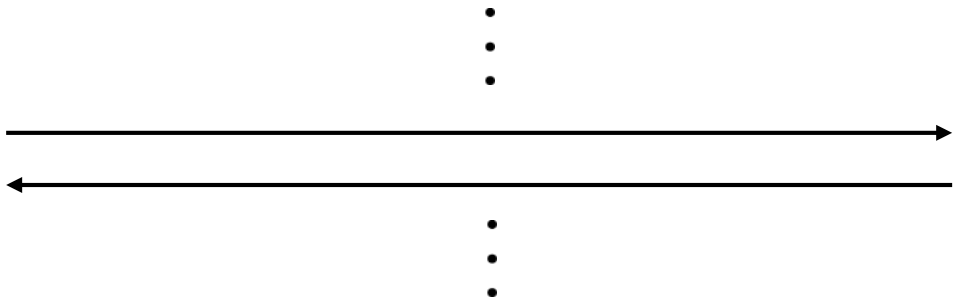
Alice signs c^B and each message she sends.

Now signed transcript and Bob's decommitment of c^B uniquely define *signed* seed s_i (s_i is defined to be the output of Bob's computation, c^B and transcript are signed by Alice)

Reveal Bob's input?

x

$s_j \in_R \{0,1\}^n$



y

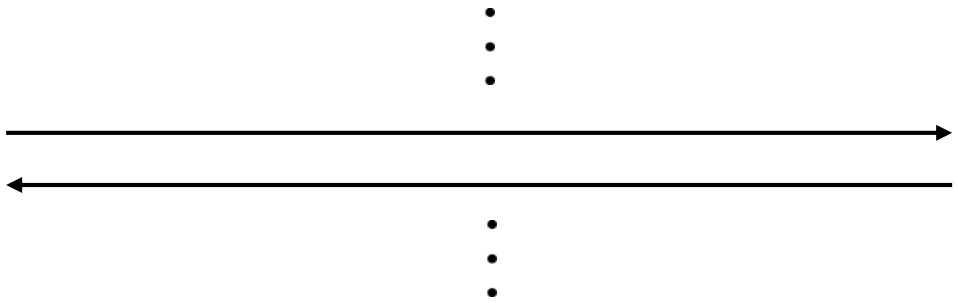
Problem: revealing s^B reveals Bob's input

Bob chooses different s_i^B in each instance and uses all-zero input in check instances.

Frame Alice?

x

$s_j \in_R \{0,1\}^n$

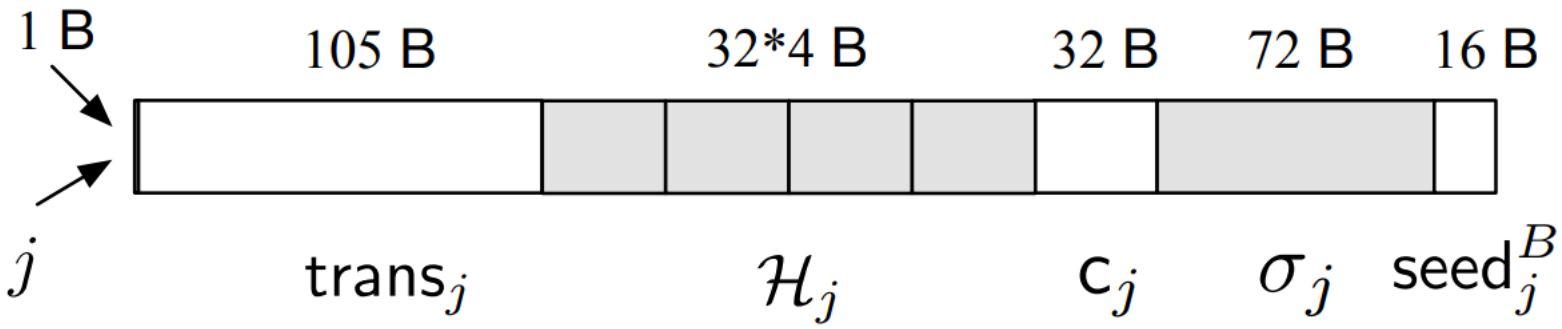


y

Defamation free?

Yes, because Bob's cert is only valid if Bob follows his (deterministic) strategy based on the seed he committed to in the beginning.

Performance



Certificate is 354 bytes long

trans_j is the transcript of the jth execution of first Π_{OT} (seed OT)
H_j is hash of the transcript of the jth execution of the second Π_{OT} (GC input labels)

Performance

Circuit	LAN setting			WAN setting		
	Our PVC	Semi-honest	Slowdown	Our PVC	Semi-honest	Slowdown
AES-128	24.53 ms	15.31 ms	1.60×	960.4 ms	820.8 ms	1.17×
SHA-128	33.67 ms	24.69 ms	1.36×	1146 ms	976.8 ms	1.17×
SHA-256	48.43 ms	38.04 ms	1.27×	1252 ms	1080 ms	1.16×
Sort.	3468 ms	2715 ms	1.28×	13130 ms	12270 ms	1.07×
Mult.	1285 ms	1110 ms	1.16×	5707 ms	5462 ms	1.04×
Hamming	2585 ms	1550 ms	1.67×	11850 ms	6317 ms	1.69×

Table 2: Comparing the running times of our protocol and a semi-honest protocol in the LAN and WAN settings.

Performance

Circuit	LAN setting			WAN setting		
	Our PVC	Malicious [20]	Speedup	Our PVC	Malicious [20]	Speedup
AES-128	24.53 ms	157.3 ms	6.41×	960.4 ms	11170 ms	11.6×
SHA-128	33.67 ms	318.8 ms	9.47×	1146 ms	13860 ms	12.1×
SHA-256	48.43 ms	611.7 ms	12.6×	1252 ms	17300 ms	13.8×
Sort.	3468 ms	45130 ms	13.0×	13130 ms	197900 ms	15.1×
Mult.	1285 ms	17860 ms	13.9×	5707 ms	99930 ms	17.5×
Hamming	2586ms	11380 ms	4.40×	11850 ms	76280 ms	6.44×

Table 3: Comparing the running times of our protocol and a malicious protocol in the LAN and WAN settings.

[20] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. CCS 2017.

Performance

	AES-128	SHA-128	SHA-256	Sort.	Mult.	Hamming
Semi-honest	0.2218 MB	1.165 MB	2.800 MB	313.1 MB	128.0 MB	96.01 MB
Malicious [12]	3.545 MB	17.69 MB	42.95 MB	2953 MB	1228 MB	662.7 MB
Our PVC	0.2427 MB	1.205 MB	2.844 MB	325.1 MB	128.2 MB	144.2 MB

Table 4: Communication complexity of our protocol and other protocols.

Conclusion

Optimized PVC setting is nearly as efficient as semi-honest
(10-60% overhead, depending on input/circ size relationship)

Code is available
https://github.com/emp-toolkit/emp-pvc