Two-party computation

By Shuoyao Zhao

2018.1.4

Problem Abstraction

Alice

Holds $x \in \{0,1\}^s$

Public function f

Bob

Holds $y \in \{0,1\}^t$

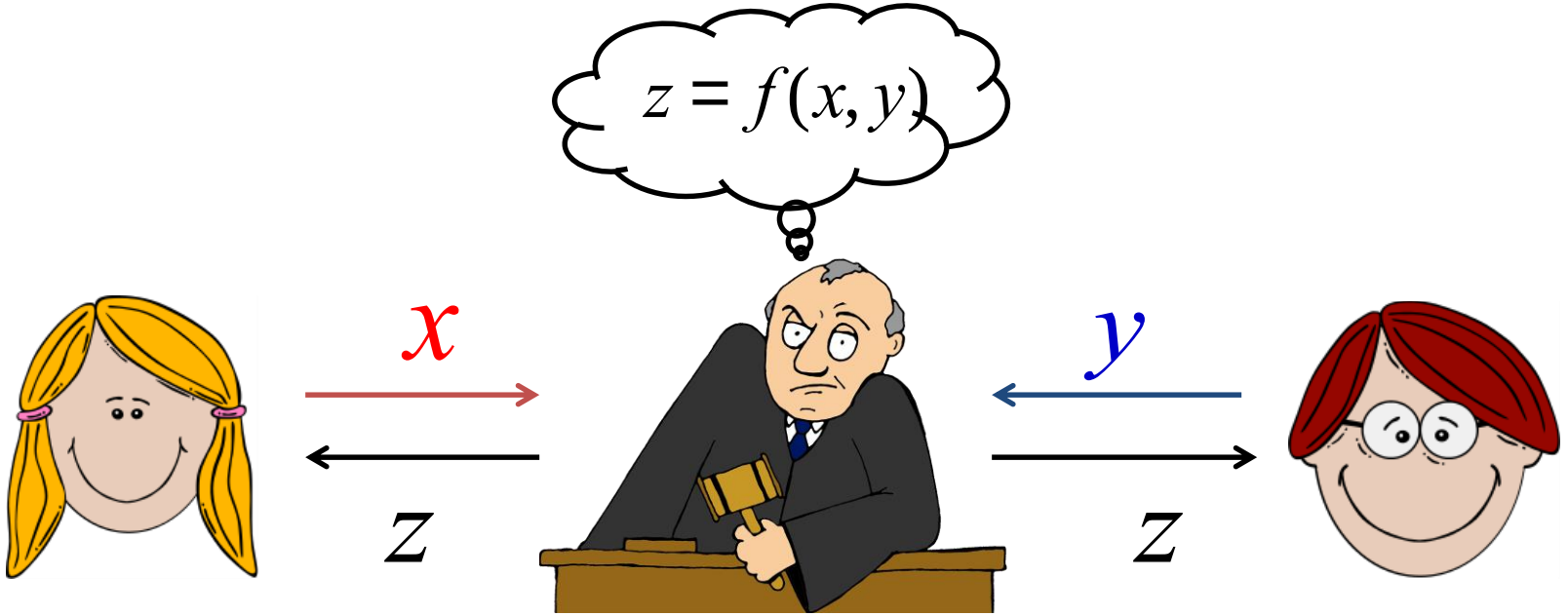
$$z = f(x, y)$$

Reveal z

but nothing ***more!***

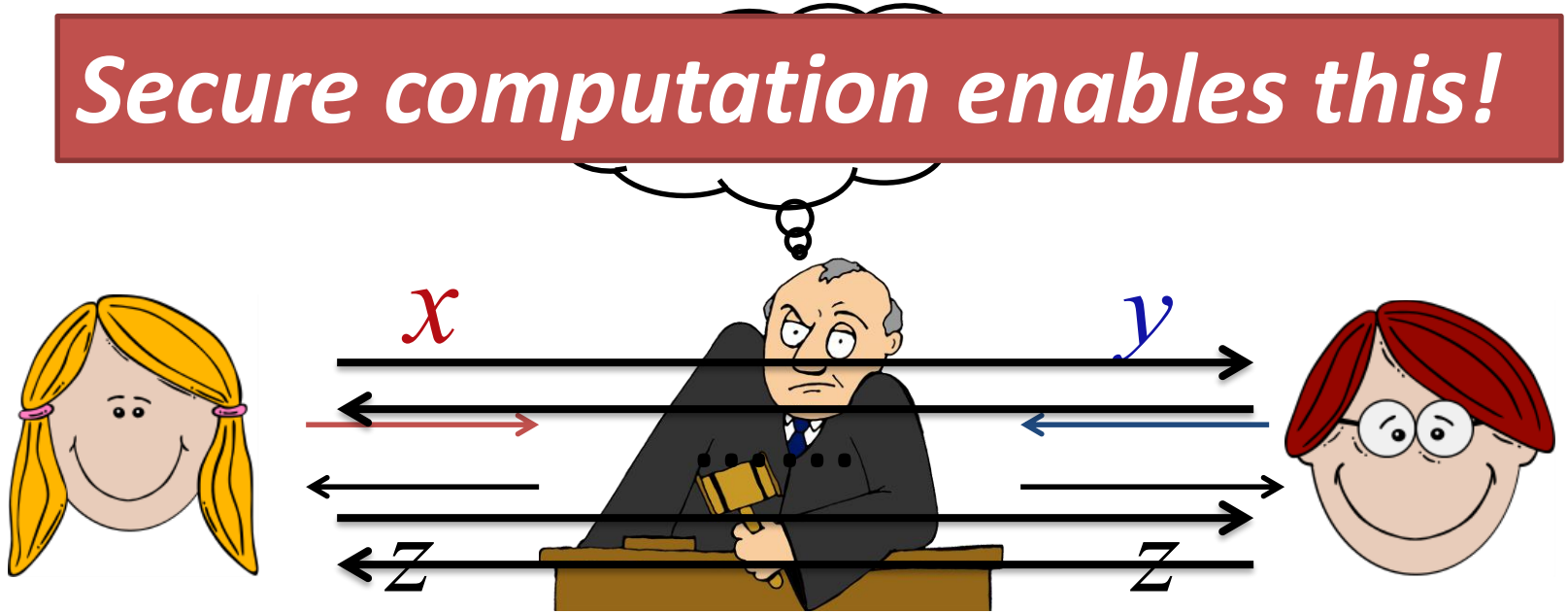
Security requirement:

Ideally, with a Trusted Party



In the Real World

Secure computation enables this!



$f(x,y)$

but nothing ***more****!*

$f(x,y)$

but nothing ***more****!*

A Binary Gate

Alice

x=0

Bob

y=0

0 **NAND** 0

[Yao, *FOCS'86*]

A Binary Gate

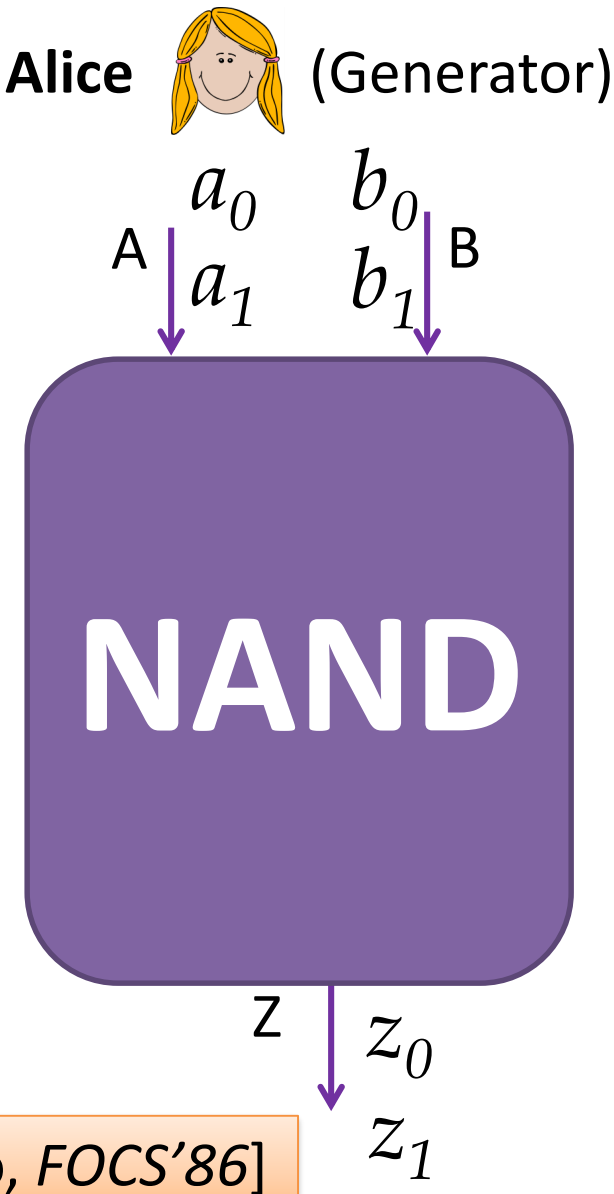
Alice  (Generator)

Bob  (Evaluator)

a_0, a_1 are *random* ***bit strings***

[Yao, *FOCS'86*]

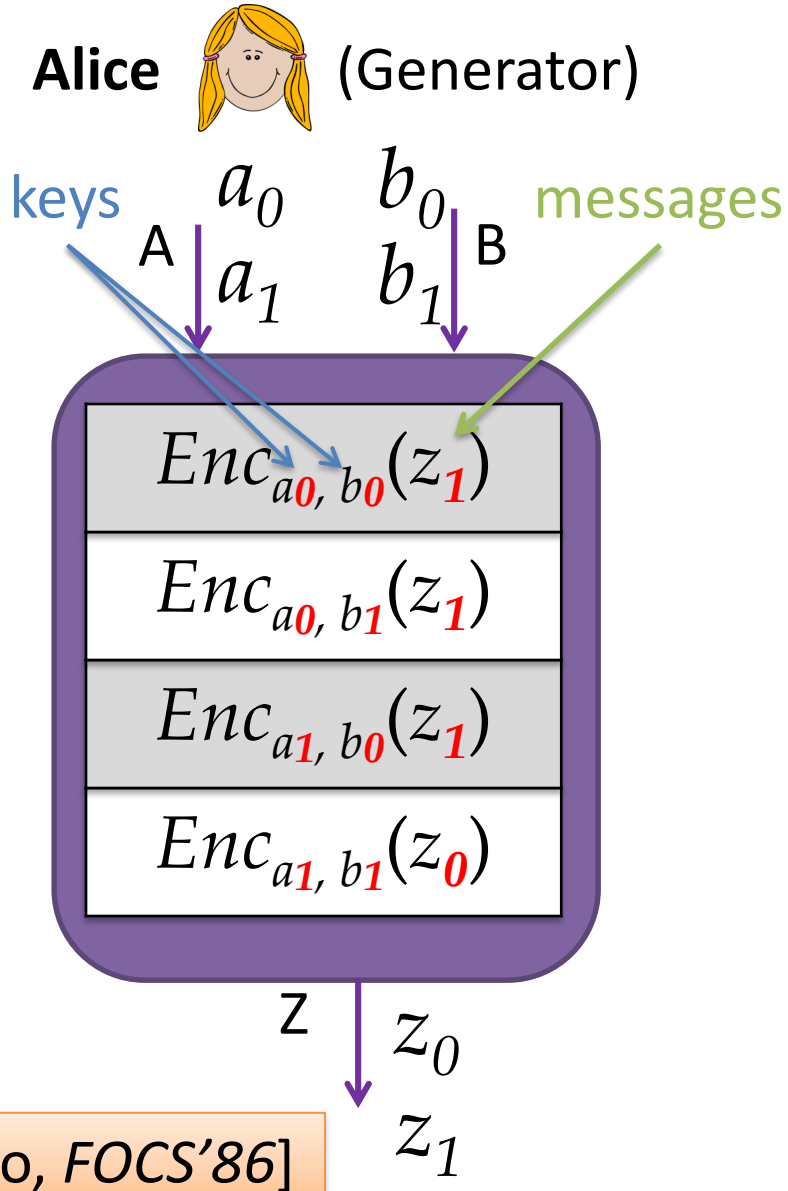
A Binary Gate



$a_0, a_1, b_0, b_1, z_0, z_1$
are *independent*
random ***bit strings***

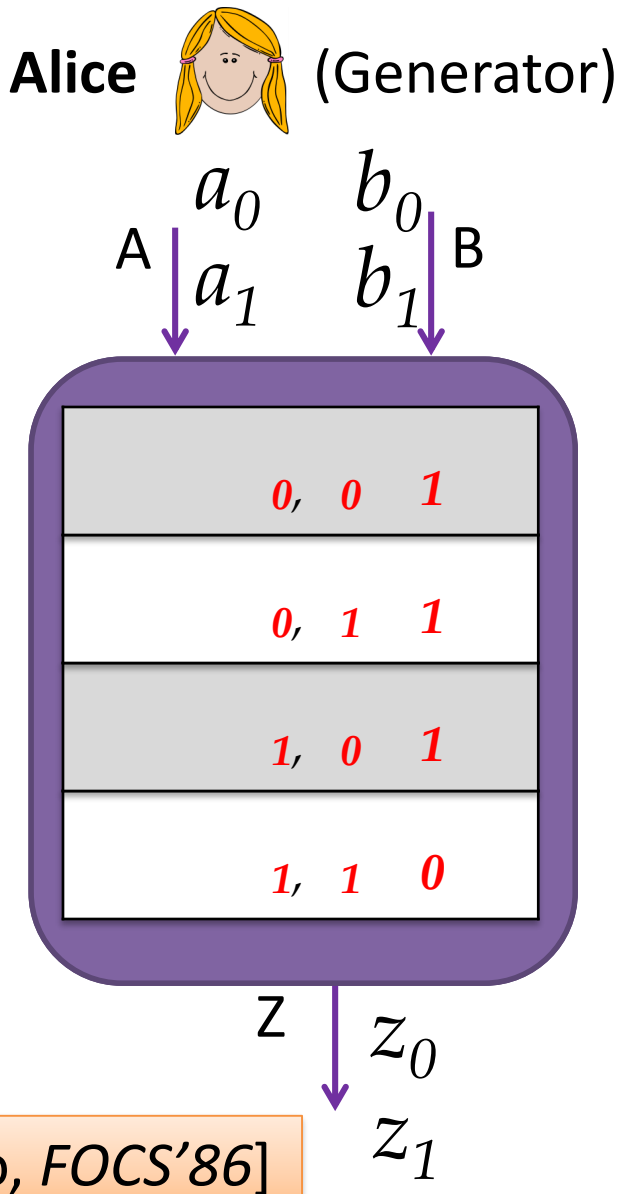
[Yao, *FOCS'86*]

A Binary Gate



[Yao, *FOCS'86*]

A Binary Gate

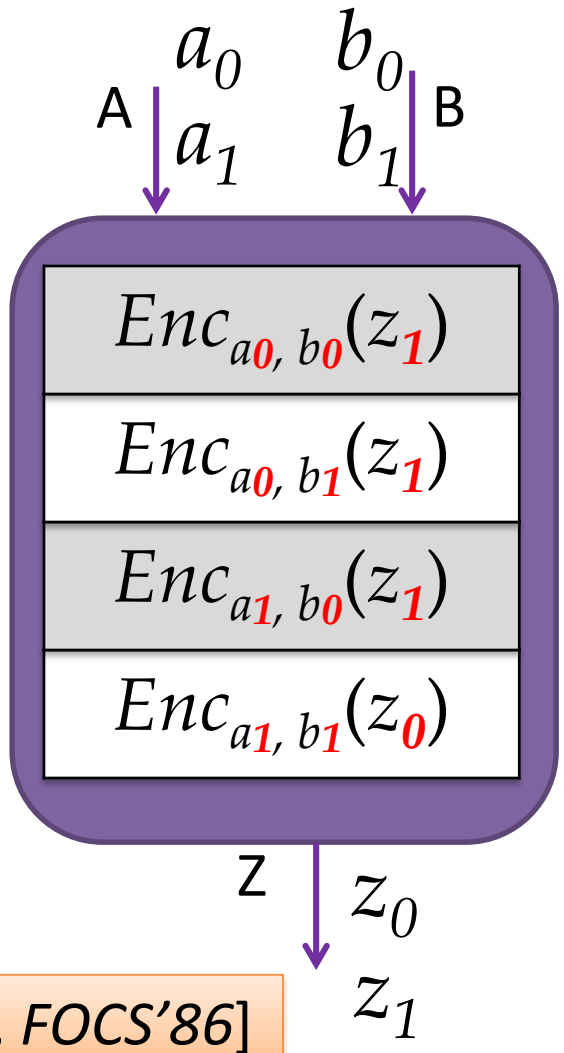


[Yao, *FOCS'86*]

A Binary Gate

Alice  (Generator)

Bob  (Evaluator)



[Yao, *FOCS'86*]

Prevent the Leak

Alice  (Generator)

Bob  (Evaluator)

$Enc_{a1,\, b1}(z_0)$
$Enc_{a1,\, b0}(z_1)$
$Enc_{a0,\, b1}(z_1)$
$Enc_{a0,\, b0}(z_1)$

a_0 b_0

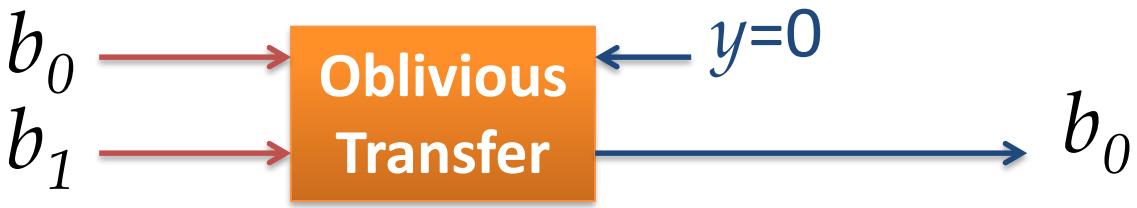
✗ $Enc_{a1,\, b1}(z_0)$
✗ $Enc_{a1,\, b0}(z_1)$
✗ $Enc_{a0,\, b1}(z_1)$
✓ $Enc_{a0,\, b0}(z_1)$

[Yao, *FOCS'86*]

Transferring b_0 obliviously

Alice  (Generator)

Bob  (Evaluator)



Transferring b_0 obliviously

Alice (Generator)

Bob (Evaluator)

b_0

b_1

Oblivious Transfer

$(\mathcal{G}, g)$ is a DDH-group. H models a random oracle.
Bob doesn't know the discrete log of $C \in \mathcal{G}$.

y

b_y

$h := (1-y)g^k + Cyg^{-k}$

$k \leftarrow \mathcal{G}$

$r \leftarrow [ord(\mathcal{G})]$

g^r

$m_0 := b_0 \oplus H(h^r, 0)$

$m_1 := b_1 \oplus H((C/h)^r, 1)$

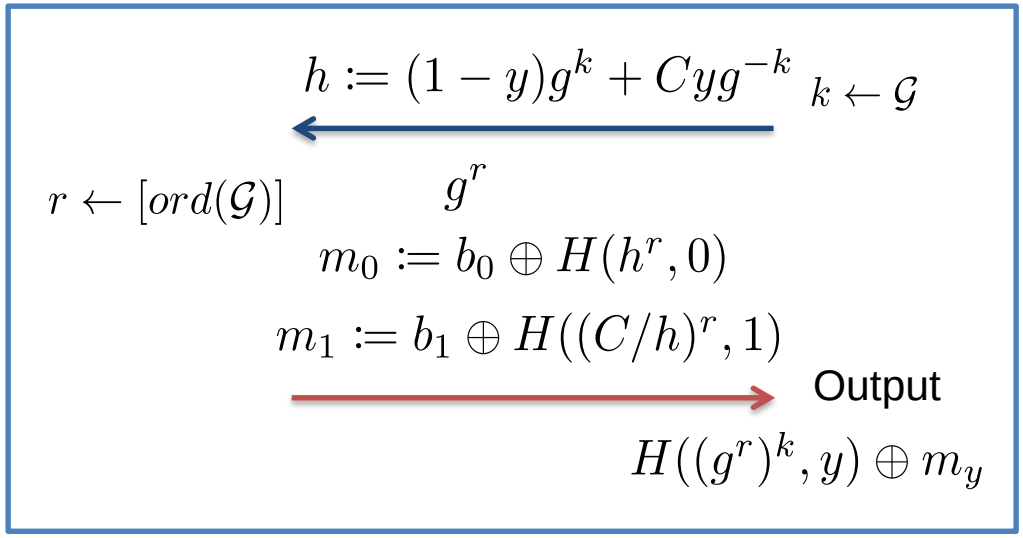
Output

$H((g^r)^k, y) \oplus m_y$

[Naor-Pinkas, *SODA'00*]

Security of NPOT

- Receiver's Privacy
  - h is uniformly random, independent of y
- Sender's Privacy
  - Receiver cannot learn b_y as it doesn't know $\log_g C$



Paper

- **A Proof of Security of Yao's Protocol for Two-Party Computation**
  Author: Yehuda Lindell , Benny Pinkas

The differences

- *f (x,y) = (f1(x,y),f2(x,y))*
- Description of Garbled gate

Table 1. Garbled OR gate

Input wire w_1	Input wire w_2	Output wire w_3	Garbled computation table
k_1^0	k_2^0	k_3^0	$E_{k_1^0}(E_{k_2^0}(k_3^0))$
k_1^0	k_2^1	k_3^1	$E_{k_1^0}(E_{k_2^1}(k_3^1))$
k_1^1	k_2^0	k_3^1	$E_{k_1^1}(E_{k_2^0}(k_3^1))$
k_1^1	k_2^1	k_3^1	$E_{k_1^1}(E_{k_2^1}(k_3^1))$

Parameter table

Symbol	Meaning
g(α,β)	Circuit-output gate
w_i , ex: w_1	Circuit-output wire
0,1	Corresponding real values
k_w^0, k_w^1	Random keys
(0, k_w^0)	Output decryption tables
$E_{k_1^0}(E_{k_2^0}(k_3^0))$	Garbled computation box
E_1, E_2, E_3, E_4	Garbled computation table

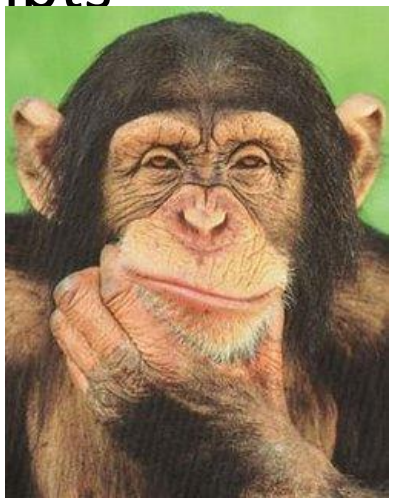
Each pair of keys open only one box for each gate!!!

Modeling Adversaries

Semi-Honest

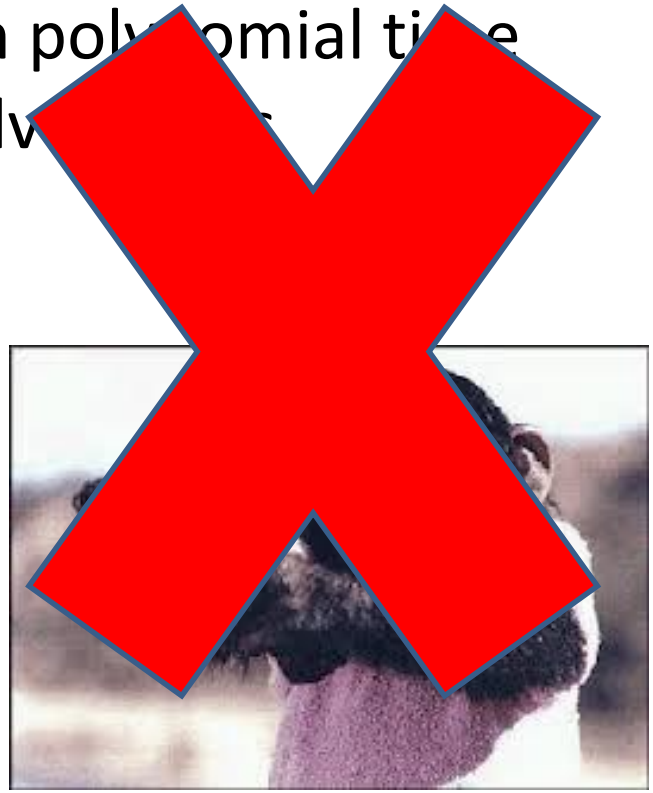
(Honest-but-curious)

Always follow the protocol but tries to learn extra from the execution transcripts



Malicious/Active

Absolutely no restriction on polynomial time adversaries



Definition(1)

- Let $f = (f_1, f_2)$ be a probabilistic polynomial-time functionality, and let π be a two-party protocol for computing f .
- The view of the i_th party (i $\in$ {1, 2}) during an execution of π on (x,y) is denoted:

$$view_{\mathrm{i}}^{\pi}(x, y) = (x, r^i, m_1^i, \ldots, m_t^i)$$

where r^i equals the contents of the i_th party's internal random tape, and m_j^i represents the j_th message that it received.

Definition(2)

- *The output of the i_th party during an execution of π on (x,y) is denoted* $output_i^{\pi}(x,y)$ *and can be computed from its own view of the execution. Denote:*

$$output_{\pi}(x,y) = \left(output_1^{\pi}(x,y), output_2^{\pi}(x,y)\right)$$

Differ from f(x,y)

Definition(3)

- *Definition 1：* *Let $f = (f_1, f_2)$ be a functionality. We say that π securely computes f in the presence of static semi-honest adversaries if there exist probabilistic polynomial-time algorithms S_1 and S_2 such that:*

$$\{(S_1(x, f_1(x,y)), f(x,y))\}_{x,y\in\{0,1\}^*} \overset{C}{\Leftrightarrow} \{(view_1^{\pi}(x,y), output_{\pi}(x,y))\}_{x,y\in\{0,1\}^*}$$

And:

$$\{(S_2(y, f_2(x,y)), f(x,y))\}_{x,y\in\{0,1\}^*} \overset{C}{\Leftrightarrow} \{(view_2^{\pi}(x,y), output_{\pi}(x,y))\}_{x,y\in\{0,1\}^*}$$

Definition(4)

- *A Simpler Formulation for Deterministic Functionalities: In the case that the functionality f is deterministic, a simpler definition can be used. Specifically, we do not need to consider the joint distribution of the simulator's output with the protocol output. Rather, we separately require that:*

$$output_{\pi}(x, y) = f(x, y)$$

And in addition, that there exist S1 and S2 such that:

$$\{S_1(x, f_1(x, y))\}_{x,y\in\{0,1\}^*} \overset{c}{\Leftrightarrow} \{view_1^{\pi}(x, y)\}_{x,y\in\{0,1\}^*}$$

$$\{S_2(y, f_2(x, y))\}_{x,y\in\{0,1\}^*} \overset{c}{\Leftrightarrow} \{view_2^{\pi}(x, y)\}_{x,y\in\{0,1\}^*}$$

Definition(5)

- *Deterministic Same-Output Functionalities We say that a functionality f = (f1,f2) is same-output if f1 = f2.*
- *In our presentation, we will show how to securely compute deterministic same output functionalities only. This suffices for obtaining secure protocols for arbitrary probabilistic functionalities.*

Definition(6)

- *Proof of the last slide:*

From deterministic Functionalities to probabilistic polynomial-time:

$$f'((x,r),(y,s)) = f(x, y, r \oplus s)$$

Deterministic Same-Output Functionalities :

$$f'((x,r),(y,s)) = f1(x,y) \oplus r \,||\, f2(x,y) \oplus s$$

Tools—private-key encryption (1)

- Let *(G,E,D)* be a private-key encryption scheme and denote the range of a key in the scheme by:

$$Rangen(k) = \{E_k(x)\}, x \in \{0,1\}^n$$

Tools—private-key encryption (2)

- We say that (G,E,D) has an elusive range if for every probabilistic polynomial time machine A, every polynomial p(·), and all sufficiently large n,

$$Pr_{k \leftarrow G(1^n)}[A(1^n) \in Rangen(k)] < \frac{1}{p(n)}$$

Tools—private-key encryption (3)

- We say that (G,E,D) has an efficiently verifiable range if there exists a probabilistic polynomial-time machine M such that :

  M(k,c) = 1 if and only if

  $$c \in \text{Rangen}(k)$$

Tools—private-key encryption (4)

- Construction:
- Let $F = \{f_k\}$ be a family of pseudorandom functions, where $f_k: \{0,1\}^n \rightarrow \{0,1\}^{2n}$ for k $\in \{0,1\}^n$. Then, define:

$$E_k(x) = \{r, f_k(r) \oplus (x || 0^n)\}$$

This E_k has an efficiently verifiable range.

Proof: $f_k(x)$ and $f_{rand}(x)$ is indistinguishable.

Tools—private-key encryption (5)

- Other properties needed for (*G,E,D):*
- For every two (known) vectors of messages x and y, no polynomial-time adversary can distinguish an encryption of the vector x from an encryption of the vector y.
- an encryption under one key will fall in the range of an encryption under another key with negligible probability.

Easy to fulfill.

Proof of correctness(1)

- *If* $E_k(x)$ has an efficiently verfiable range , then the Yao's Two-Party Protocol constructed by $E_k(x)$ is correct.
- *All we need is to prove: if* $k_1^0, k_1^1, k_2^0, k_2^1, k_3$ are uniformly independently chosen, then:

$$\Pr\left\{E_{k_1^i}\left(E_{k_2^j}(k_3)\right) \in Rangen(k_1^0, k_2^0)\right\} < \frac{1}{p(n)}$$

For each (i,j)=(0,1),(1,0),(1,1)

Proof of correctness(2)

(1) i=0, j=1:

$$\Pr\left\{E_{k_1^0}\left(E_{k_2^1}(k_3)\right) \in Rangen(k_1^0, k_2^0)\right\} =$$

$$\Pr\left\{E_{k_2^1}(k_3) \in Rangen(k_2^0)\right\} < \frac{1}{p(n)}$$

(2)i=1:

$$\Pr\left\{E_{k_1^1}\left(E_{k_2^j}(k_3)\right) \in Rangen(k_1^0, k_2^0)\right\} \leq$$

$$\Pr\left\{E_{k_1^1}(k') \in Rangen(k_1^0)\right\} < \frac{1}{p(n)}$$

Transferring b_0 obliviously

Alice (Generator)

Bob (Evaluator)

b_0

b_1

Oblivious Transfer

(f,t) is a permutation-trapdoor pair in a family of enhanced trapdoor permutation and B() is a hard-core of f

y

b_y

w_0, w_1

$v_y \leftarrow D(f), w_y = f(v_y)$

$w_{1-y} \leftarrow V(f)$

$v_0 = f^{-1}(w_0)$

$v_1 = f^{-1}(w_1)$

$m_0 = B(v_0) \oplus b_0$

$m_1 = B(v_1) \oplus b_1$

$m_0,\ m_1$

$b_y = B(v_y) \oplus b_y$

Bob have no information of t (the trapdoor), means (f,t) should be sampled by Alice and then be sent to Bob.

Tools—OT

- About: $w_{1-y} \leftarrow V(f)$
- *An enhanced trapdoor permutation has the property that it is possible to sample from the range, so that given the coins used for sampling.*
- *The* comparison of two Ots:

$$v_y \leftarrow D(f), w_y = f(v_y), w_{1-y} \leftarrow V(f)$$

VS

$$h_y \leftarrow g^k, h_{1-y} \leftarrow Cg^{-k}$$

Transferring b_0 obliviously

Alice (Generator)

Bob (Evaluator)

b_0

b_1

Oblivious Transfer

$(\mathcal{G}, g)$ is a DDH-group. H models a random oracle.
Bob doesn't know the discrete log of $C \in \mathcal{G}$.

y

b_y

$h := (1-y)g^k + Cyg^{-k}$

$k \leftarrow \mathcal{G}$

$r \leftarrow [ord(\mathcal{G})]$

g^r

$m_0 := b_0 \oplus H(h^r, 0)$

$m_1 := b_1 \oplus H((C/h)^r, 1)$

Output

$H((g^r)^k, y) \oplus m_y$

[Naor-Pinkas, *SODA'00*]

Transferring b_0 obliviously

Alice (Generator)

Bob (Evaluator)

b_0

b_1

Oblivious Transfer

(f,t) is a permutation-trapdoor pair in a family of enhanced trapdoor permutation and B() is a hard-core of f

y

b_y

w_0, w_1

$v_y \leftarrow D(f), w_y = f(v_y)$

$w_{1-y} \leftarrow V(f)$

$v_0 = f^{-1}(w_0)$

$v_1 = f^{-1}(w_1)$

$m_0 = B(v_0) \oplus b_0$

$m_1 = B(v_1) \oplus b_1$

$m_0,\ m_1$

$b_y = B(v_y) \oplus b_y$

Bob have no information of t (the trapdoor), means (f,t) should be sampled by Alice and then be sent to Bob.

Security of the OT

- *Showed in the reference of the paper :O. Goldreich, Foundations of Cryptography; vol. 2: Basic Applications (Cambridge University Press,Cambridge, 2004) Sec. 7.3.2*
- Let S_1^{OT} and S_2^{OT} *be the simulator of P1 and P2 in the* oblivious transfer.

Security of double encryption(1)

Def: $E_{k1}(E_{k2}(x))$ *is CPA-security iff:*

For every PPT adversary A ,which can query the encrypt function $E_{k1}(.)$ *and* $E_{k2}(.)$ *and* $E_{k1}\left(E_{k2}(.)\right)$ *:*

$$\Pr\left\{A_1(1^n) \to (m_0, m_1, s), c \xleftarrow{\$} (c_1, c_2), A_2(c, s) \to b'\right\} < \frac{1}{p(n)}$$

Where $c_i = E_{k1}(E_{k2}(m_i))$ *computed by the challenger.*

Security of double encryption(2)

If $E_k(m)$*is CPA-security, then* $E_{k1}(E_{k2}(x))$ *is also CPA-security.*

Review of the Definition

- The view of the i_th party (i ∈ {1, 2}) during an execution of π on (x,y) is denoted:

$$view_{\mathrm{i}}^{\pi}(x,y) = (x, r^i, m_1^i, \ldots, m_t^i)$$

where r^i equals the contents of the i_th party's internal random tape, and m_j^i represents the j_th message that it received.

Security of Yao's Two-Party Protocol (1)

$view_1^{\pi}(x,y) = (x, r_C, R_1^{OT}(k_{n+1}^0, k_{n+1}^1), \dots, R_1^{OT}(k_{2n}^0, k_{2n}^1), f(x,y))$

Construction of S_1:

$$S_1 = (x, r_C, S_1^{OT}(k_{n+1}^0, k_{n+1}^1), \dots, S_1^{OT}(k_{2n}^0, k_{2n}^1), f(x,y))$$

Proof of S_1:

$$\{S_1\big(x, f_1(x,y)\big)\}_{x,y\in\{0,1\}^*} \overset{C}{\Leftrightarrow} \{view_1^{\pi}(x,y)\}_{x,y\in\{0,1\}^*}$$

Using hybrid argument!!

Security of Yao's Two-Party Protocol (2)

$$H_i = (x, r_C, S_1^{OT}(k_{n+1}^0, k_{n+1}^1), \ldots, S_1^{OT}\left(k_{n+i}^0, k_{n+i}^1\right) R_1^{OT}\left(k_{n+i+1}^0, k_{n+i+1}^1\right), \ldots, f(x,y))$$

Then we prove $\{H_0\}_{x,y\in\{0,1\}^*} \overset{C}{\Leftrightarrow} \{H_n\}_{x,y\in\{0,1\}^*}$

Otherwise

$$\Pr\{(D\left(H0(.)\right) = 1\} - \Pr\{(D(Hn(.)) = 1\} > \frac{1}{p(n)}$$

Means for some i:

$$\Pr\{(D\left(H_i(.)\right) = 1\} - \Pr\{(D(H_{i+1}(.)) = 1\} > \frac{1}{n*p(n)}$$

$$\Rightarrow \Pr\{(D\left(R_1^{OT}\left(k_{n+i+1}^0, k_{n+i+1}^1\right)\right) = 1\} - \Pr\{(D\left(S_1^{OT}\left(k_{n+i+1}^0, k_{n+i+1}^1\right)\right) = 1\} > \frac{1}{n*p(n)}$$

Which is contradicted with S_1^{OT} is the simulator of P1 in the oblivious transfer.

Security of Yao's Two-Party Protocol (3)

$$\{S_2(y, f_2(x,y))\}_{x,y\in\{0,1\}^*} \overset{C}{\Leftrightarrow} \{view_2^{\pi}(x,y)\}_{x,y\in\{0,1\}^*}$$

$$view_2^{\pi}(x,y) = (x, \mathrm{G}(C), k_1^{x_1}, \ldots, k_n^{x_n}, R_2^{OT}(k_1^0, k_1^1), \ldots, R_2^{OT}(k_n^0, k_n^1), f(x,y))$$

Step1: Simulate one gate g(α,β) :

$Enc_{a_0, b_0}(z_1)$	$Enc_{a_0, b_0}(z_1)$
$Enc_{a_0, b_1}(z_1)$	$Enc_{a_0, b_1}(z_1)$
$Enc_{a_1, b_0}(z_1)$	$Enc_{a_1, b_0}(z_1)$
$Enc_{a_1, b_1}(z_0)$	$Enc_{a_1, b_1}(z_0)$

Security of Yao's Two-Party Protocol (4)

Step2: Simulate the output decryption table：

$$\left\{\left(z_i, k_{out_i}^1\right), \left(1 - z_i, k_{out_i}^0\right)\right\}, i = 1,2, \ldots\ldots, n;\ Z = f(x, y)$$

let the "fake" circuit be: $\mathrm{G}'(C)$

Step3:Simulate the oblivious transfer like S_1.

Security of Yao's Two-Party Protocol (5)

Construction of S_2:

$$S_2 = (x, \mathrm{G}'(C), k_1^{x_1}, \dots, k_n^{x_n}, S_2^{OT}(k_1^0, k_1^1), \dots, S_2^{OT}(k_n^0, k_n^1))$$

Proof of S_2:

$$\{S_2\big(x, f_2(x,y)\big)\}_{x,y\in\{0,1\}^*} \overset{C}{\Leftrightarrow} \{view_2^{\pi}(x,y)\}_{x,y\in\{0,1\}^*}$$

Using hybrid argument as well!!

Security of Yao's Two-Party Protocol (6)

Let $H_{OT} = (x, \mathrm{G}(C), k_1^{x_1}, \ldots, k_n^{x_n}, S_2^{OT}(k_1^0, k_1^1), \ldots, S_2^{OT}(k_n^0, k_n^1))$

From the proof of S_1, we know :

$$\{H_{OT}\}_{x,y\in\{0,1\}^*} \overset{C}{\Leftrightarrow} \{view_2^{\pi}(x,y)\}_{x,y\in\{0,1\}^*}$$

The hybrid experiment H_i means the experiment use the first i gates in $\mathrm{G}(C)$ and others in $\mathrm{G}'(C)$.

At last we just need to prove :

$$\{H_{|C|}\}_{x,y\in\{0,1\}^*} \overset{C}{\Leftrightarrow} \{H_0\}_{x,y\in\{0,1\}^*}$$

Security of Yao's Two-Party Protocol (7)

Assume that there exists a nonuniform probabilistic polynomial-time distinguisher D,s.t.

$$\Pr\{(D\big(H_0(.)\big) = 1\} - \Pr\{(D(H_n(.)) = 1\} > \frac{1}{p(n)}$$

Means for some i:

$$\Pr\{(D\big(H_i(.)\big) = 1\} - \Pr\{(D(H_{i+1}(.)) = 1\} > \frac{1}{|C|*p(n)}$$

$$\Rightarrow \Pr\{(D\big(E_i^1, E_i^2, E_i^3, E_i^4\big) = 1\} - \Pr\{(D\big(E'^1_i, E'^2_i, E'^3_i, E'^4_i\big) = 1\} > \frac{1}{|C|*p(n)}$$

Which is impossible, which can be reduced from the security of the double encryption.

Paper

- **An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries**
- Author: Yehuda Lindell , Benny Pinkas

Modeling Adversaries

Semi-Honest

(Ho t-but rious)

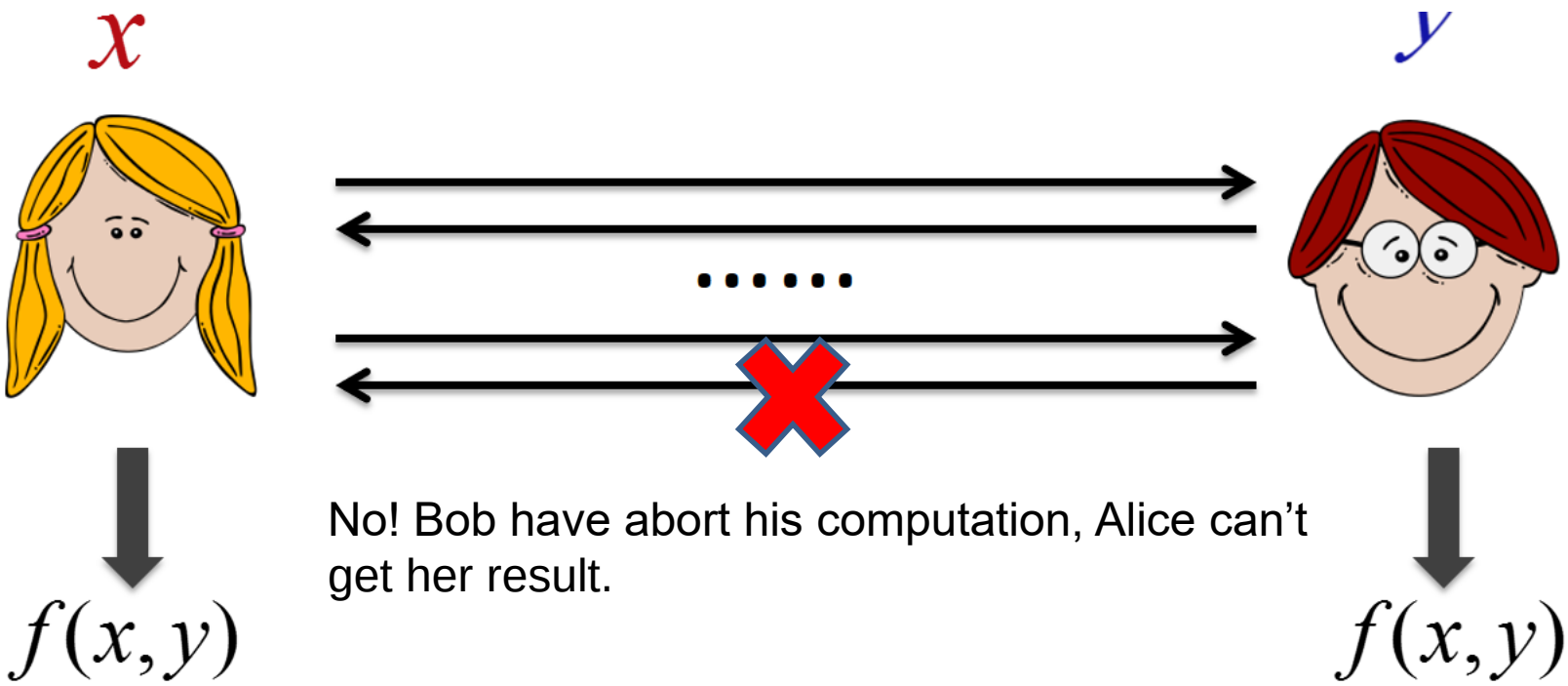
A
proto to learn
extra fr execution
trans

Malicious/Active

Absolutely no restriction on polynomial time adversaries

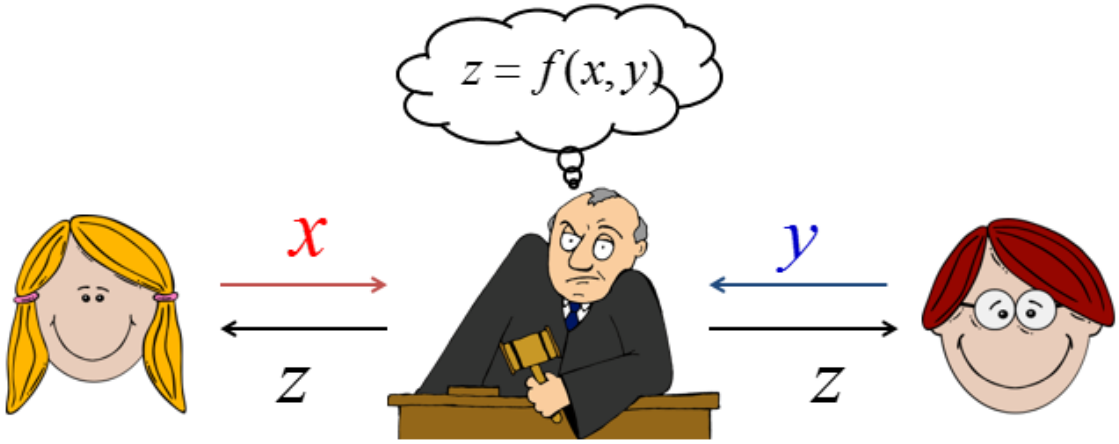
Malicious Adversary (1)

- A Malicious adversary can do anything in the processing. Such as:



Malicious Adversary (2)

- If the last message send by the malicious party in the protocol, it can abort the computation to make the other party get no result. So the ideal model can't be achieved.



New ideal model(1)

- To define the security of the protocol under the attack of the malicious adversary, we need define a new ideal model.
- Malicious attack means get the input of the other party or let the other party get the wrong result. But…
- Make the other party get no result can't be avoid. (A so-called inevitable disadvantage)

New ideal model(2)

- Inputs: Each party obtains an input, denoted w (w = x for P1, and w = y for P2).
- Send inputs to trusted party: An honest party always sends w to the trusted party. A malicious party may, depending on w, either abort or send some other w' to the trusted party.
- Trusted party answers first party: In case it has obtained an input pair (x, y), the trusted party first replies to the first party with f1(x, y). Otherwise (i.e., in case it receives only one valid input), the trusted party replies to both parties with a special symbol $\perp$.

New ideal model(3)

- Trusted party answers second party: In case the first party is malicious it may, depending on its input and the trusted party's answer, decide to stop the trusted party by sending it $\perp$. In this case the trusted party sends $\perp$ to the second party. Otherwise the trusted party sends f2(x, y) to the second party.
- Outputs: An honest party always outputs the message it has obtained from the trusted party. A malicious party may output an arbitrary (probabilistic polynomial-time computable) function of its initial input and the message obtained from the trusted party.

Definition

- Secure two-party computation in malicious model:

  Protocol π is said to securely compute f (in the malicious model) if for every pair of admissible non-uniform probabilistic polynomial-time machines A = (A1, A2) for the real model, there exists a pair of admissible non-uniform probabilistic expected

  polynomial-time machines B = (B1, B2) for the ideal model, such that:

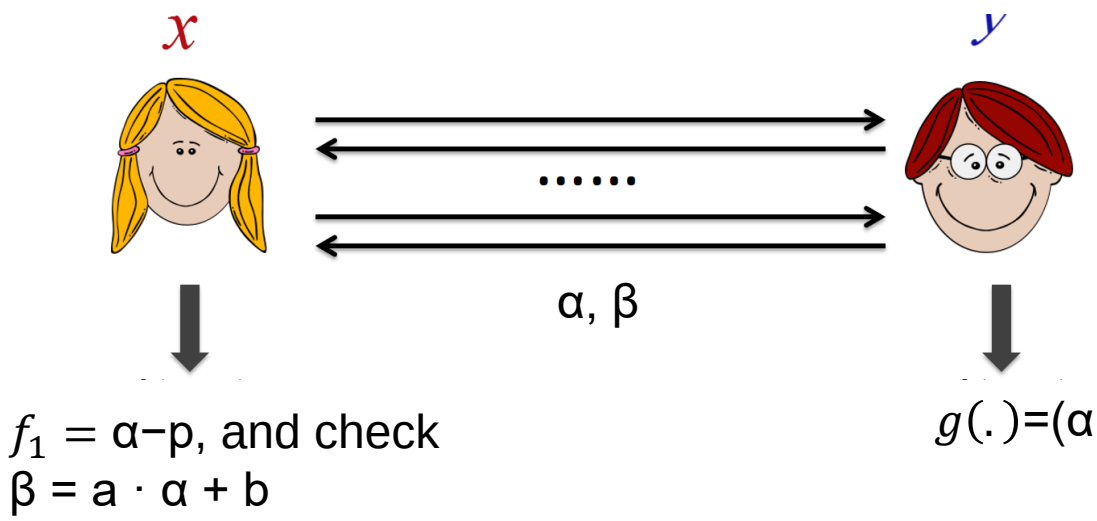
$$\{Ideal_{f,B(x,y)}\}_{x,y} \equiv \{Real_{\pi,A(x,y)}\}_{x,y}$$

Notice of Definition

- Admissible means at most one of the two parties is malicious. Which means the security of the malicious party should not be achieved.
- The ideal model means the model referred before.
- $Ideal_{f,B(x,y)}$ means the output of the two parties (malicious or not) in the ideal model. $Real_{\pi,A(x,y)}$ means the output of the two parties (malicious or not) in the real model.

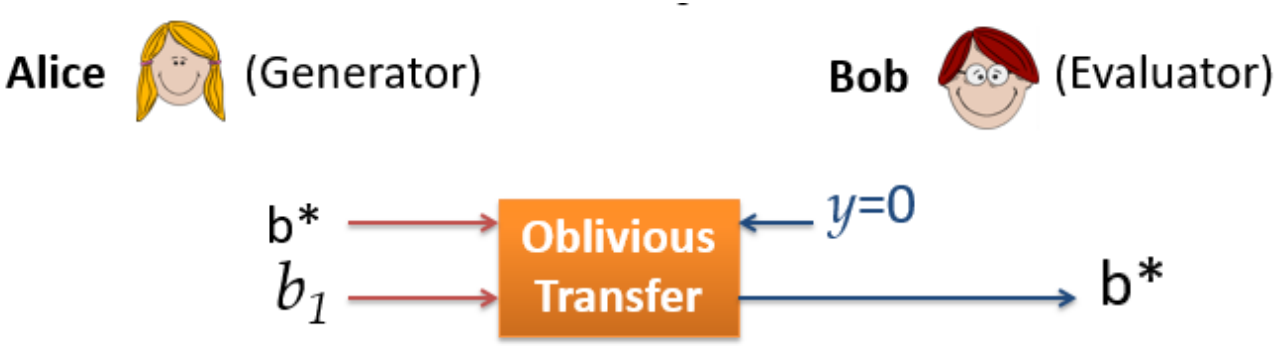
The simple case

- The presentation of our protocol is simpler for the case that only party P2 receives output. (f=f2, and f1 does not exist, but WHY?)
- g((p, a, b, x), y) =(α, β, f2(x, y)), where α = p + f1(x, y), β = a · α + b; a,b,p are chosen by P1.



Attack for Yao's protocol

- In this definition, Yao's protocol is not secure.
- Attack 1: attack in the obliviously transfer.



Attack 2: Alice make a fake circuit.

Achieve Active Security(1) (against the malicious adversary)

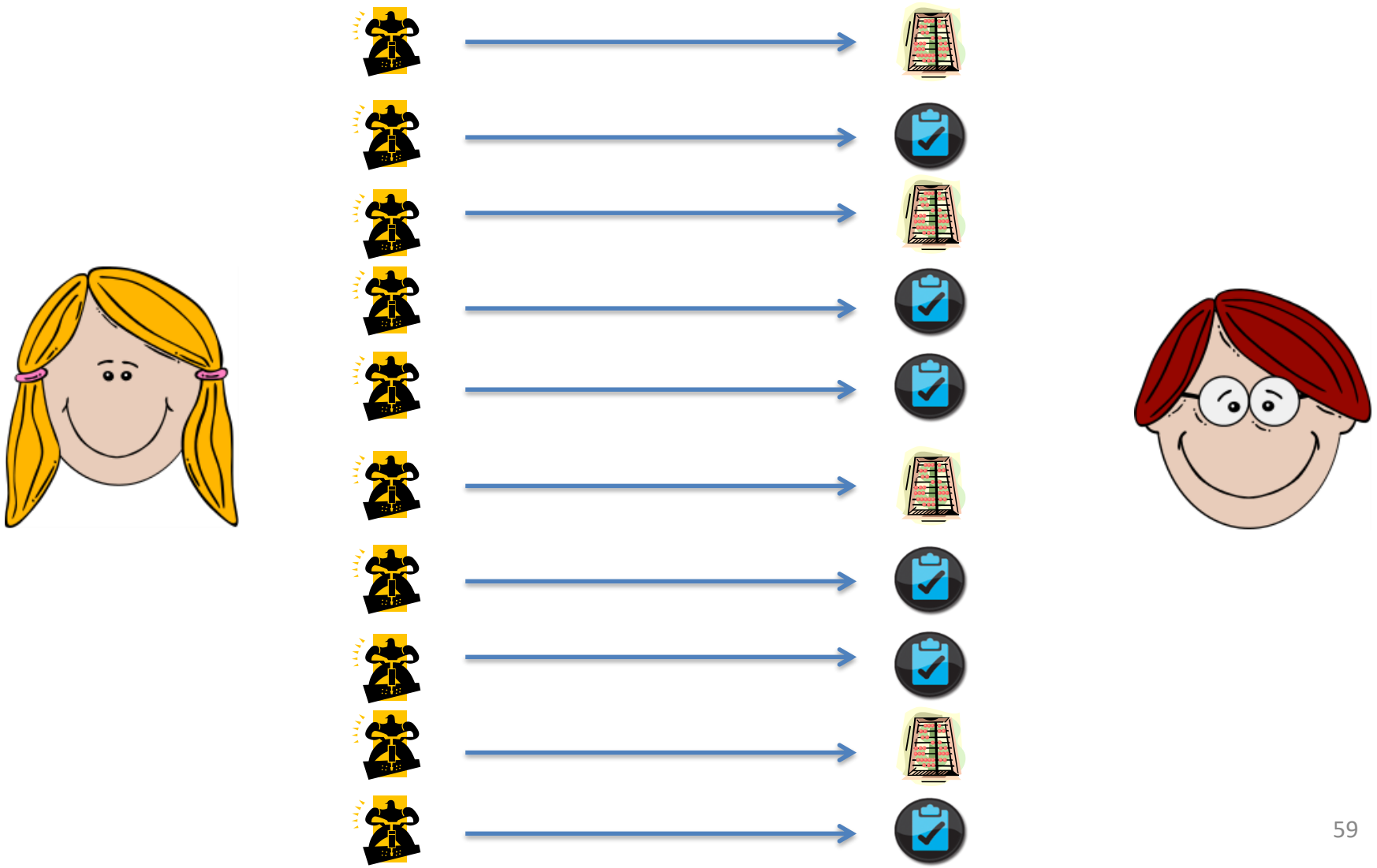
- Solution of attack1: make P2's input longer.

$$\hat{y} = \hat{y}_1, \ldots, \hat{y}_{ns}$$

$$y_i = \hat{y}_{((i-1)\cdot s+1)} \oplus \cdots \oplus \hat{y}_{is}$$

- Solution of attack2: cut-and-choose

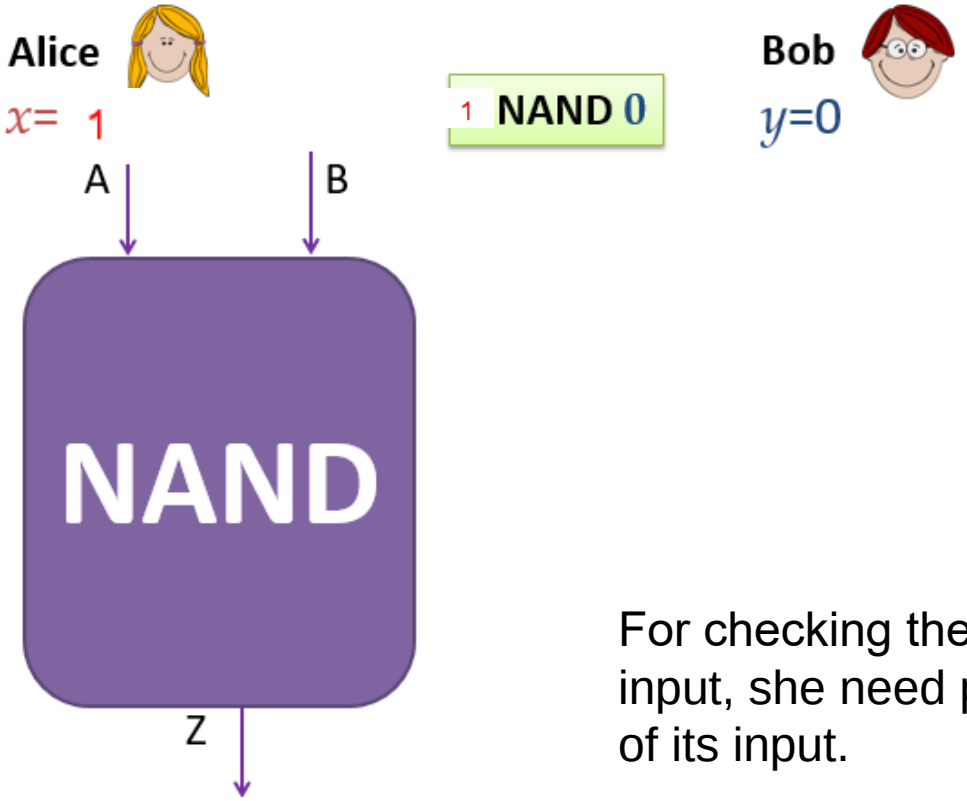
Achieve Active Security(2)



Achieve Active Security(3)

- P1 constructs s independent copies of a garbled circuit of C, denoted GC1, . . . , GCs, and P1 commits to the garbled values of the all the wires.
- P2 randomly choose some circuits for checking, P1 open the check-circuits for P2 verify. The remain circuits are used for the evaluation.
- But new questions appear: How can make P1 use the same input in the processing of evaluation.

Achieve Active Security(4)

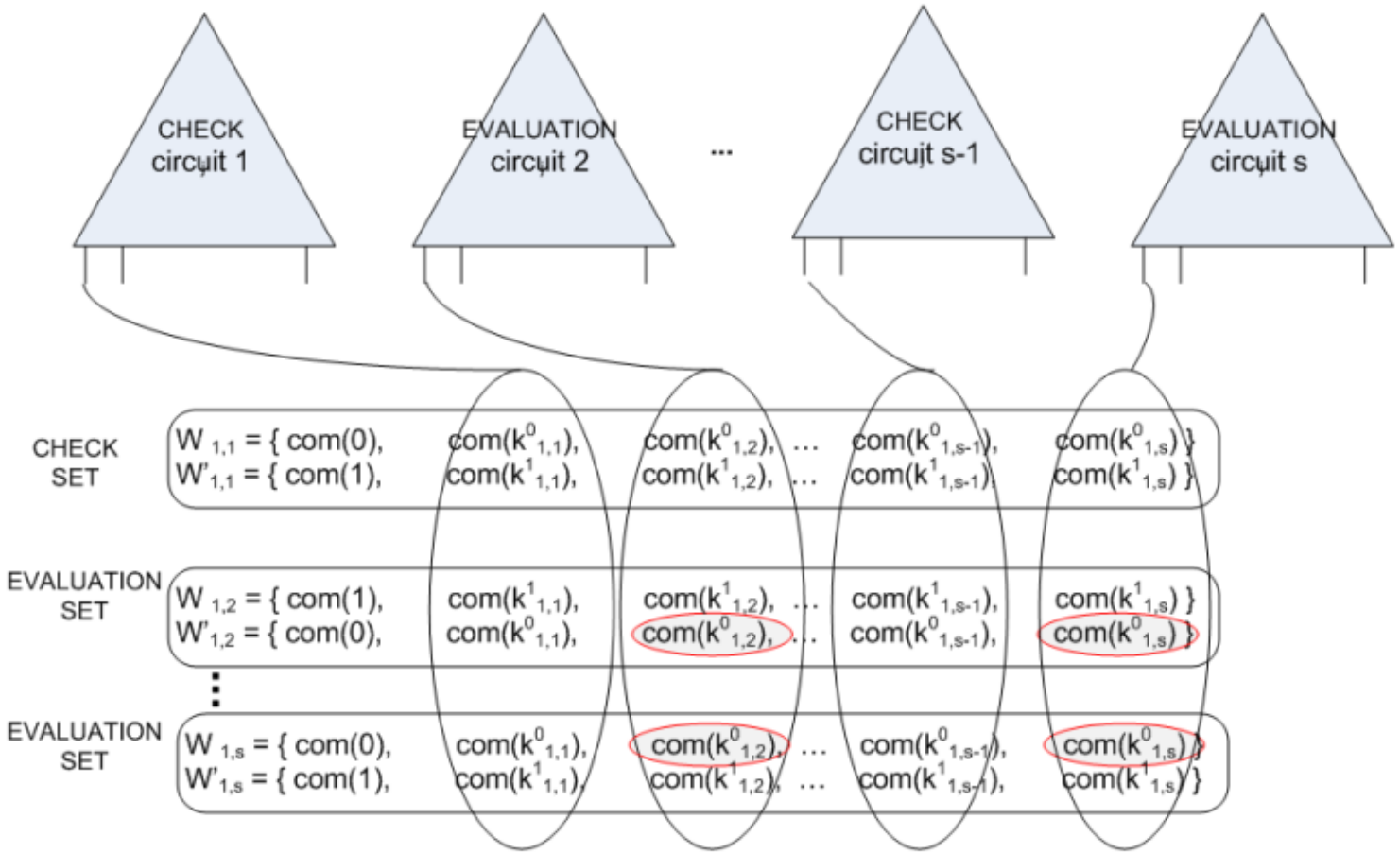


But!!

For checking the consistency of Alice's input, she need provide more commitment of its input.

No change for z!!!

Achieve Active Security(5)



Full protocol

Protocol 2 (protocol for computing $f(x,y)$):

- **Input:** P_1 *has input* $x \in \{0,1\}^n$ *and* P_2 *has input* $y \in \{0,1\}^n$.
- **Auxiliary input:** *a statistical security parameter* s *and the description of a circuit* C^0 *such that* $C^0(x,y) = f(x,y)$.
- **Specified output:** *party* P_2 *should receive* $f(x,y)$ *and party* P_1 *should receive no output. (Recall that this suffices for the general case where both parties receive possibly different outputs; see Section 2.2.)*
- **The protocol:**

  0. CIRCUIT CONSTRUCTION: *The parties replace* C^0 *with a circuit* C *which is constructed by replacing each input wire of* P_2 *by the result of an exclusive-or of* s *new input wires of* P_2. *(We show in Section 5.2 how the number of new input bits can be reduced.) The number of input wires of* P_2 *is increased from* $|y| = n$ *to* sn. *Let the bit-wise representation of* P_2*'s original input be* $y = y_1 \ldots y_n$. *Denote its new input as* $\hat{y} = \hat{y}_1, \ldots, \hat{y}_{ns}$. P_2 *chooses its new input at random subject to the constraint* $y_i = \hat{y}_{(i-1)\cdot s+1} \oplus \cdots \oplus \hat{y}_{i\cdot s}$.

  1. COMMITMENT CONSTRUCTION: P_1 constructs the circuits and commits to them, as follows:

     (a) P_1 *constructs* s *independent copies of a garbled circuit of* C, *denoted* $GC_1, \ldots, GC_s$.

     (b) P_1 *commits to the garbled values of the wires corresponding to* P_2*'s input to each circuit. That is, for every input wire* i *corresponding to an input bit of* P_2, *and for every circuit* GC_r, P_1 *computes the ordered pair* $(\mathsf{com}_b(k^0_{i,r}), \mathsf{com}_b(k^1_{i,r}))$, *where* $k^b_{i,r}$ *is the garbled value associated with* b *on input wire* i *in circuit* GC_r.

     (c) P_1 *computes commitment-sets for the garbled values that correspond to its own inputs to the circuits. That is, for every wire* i *that corresponds to an input bit of* P_1, *it generates* s *pairs of commitment sets* $\{W_{i,j}, W'_{i,j}\}_{j=1}^s$, *in the following way:*
     Denote by $k^b_{i,r}$ *the garbled value that was assigned by* P_1 *to the value* $b \in \{0,1\}$ *of wire* i *in* GC_r. *Then,* P_1 *chooses* $b \in_R \{0,1\}$ *and computes*

$$W_{i,j} = \langle \mathsf{com}_b(b), \mathsf{com}_b(k^b_{i,1}), \ldots, com_b(k^b_{i,s}) \rangle, \qquad \text{and}$$
$$W'_{i,j} = \langle \mathsf{com}_b(1-b), \mathsf{com}_b(k^{1-b}_{i,1}), \ldots, \mathsf{com}_b(k^{1-b}_{i,s}) \rangle$$

For each i, j, the sets are constructed using independent randomness, and in particular the value of b is chosen independently for every $j = 1 \ldots s$. There are a total of ns commitment-sets. We divide them into s supersets, *where superset S_j is defined as $S_j = \{(W_{1,j}, W'_{1,j}), \ldots, (W_{n,j}, W'_{n,j})\}$. Namely, S_j is the set containing the j^{th} commitment set for all wires.*

2. OBLIVIOUS TRANSFERS: *For every input bit of P_2, parties P_1 and P_2 run a 1-out-of-2 oblivious transfer protocol in which P_2 receives the garbled values for the wires that correspond to its input bit (in every circuit). That is, let $c^b_{i,r}$ denote the commitment to the garbled value $k^b_{i,r}$ and let $dc^b_{i,r}$ denote the decommitment value for $c^b_{i,r}$. Furthermore, let $i_1, \ldots, i_{ns}$ be the input wires that correspond to P_2's input.*

   Then, for every $j = 1, \ldots, ns$, parties P_1 and P_2 run a 1-out-of-2 OT protocol in which:
   (a) P_1's input is the pair of vectors $([dc^0_{i_j,1}, \ldots, dc^0_{i_j,s}], [dc^1_{i_j,1}, \ldots, dc^1_{i_j,s}])$.
   (b) P_2's input is its j^{th} input bit $\hat{y}_j$ (and its output should thus be $[dc^{\hat{y}_j}_{i_j,1}, \ldots, dc^{\hat{y}_j}_{i_j,s}]$).
   If the oblivious transfer protocol provides security for parallel execution, then these executions are run in parallel. Otherwise, they are run sequentially.

3. SEND CIRCUITS AND COMMITMENTS: *P_1 sends to P_2 the garbled circuits (i.e., the gate and output tables), as well as all of the commitments that it prepared above.*

4. PREPARE CHALLENGE STRINGS: *(1) P_2 chooses a random string $\rho_2 \in_R \{0,1\}^s$ and sends $\mathsf{com}_h(\rho_2)$ to P_1. (2) P_1 chooses a random string $\rho_1 \in \{0,1\}^s$ and sends $\mathsf{com}_b(\rho_1)$ to P_2. (3) P_2 decommits, revealing ρ_2. (4) P_1 decommits, revealing ρ_1. (5) P_1 and P_2 set $\rho = \rho_1 \oplus \rho_2$. The above steps are run a second time, defining an additional string ρ'.*[5]

5. DECOMMITMENT PHASE FOR CHECK-CIRCUITS: *From here on, we refer to the circuits for which the corresponding bit in ρ is 1 as* check-circuits, *and we refer to the other circuits as* evaluation-circuits. *Likewise, if the j^{th} bit of ρ' equals 1, then the commitments sets in $S_j = \{(W_{i,j}, W'_{i,j})\}_{i=1\ldots n}$ are referred to as* check-sets; *otherwise, they are referred to as* evaluation-sets.

   For every check-circuit GC_r, *party P_1 operates in the following way:*

   (a) *For every input wire i corresponding to an input bit of P_2, party P_1 decommits to the pair $(\mathsf{com}(k^0_{i,r}), \mathsf{com}(k^1_{i,r}))$ (namely to both of P_2's inputs).*

   (b) *For every input wire i corresponding to an input bit of P_1, party P_1 decommits to the appropriate values in the superset S_j, in the* check-sets $\{W_{i,j}, W'_{i,j}\}$. *Specifically, P_1 decommits to the $\mathsf{com}(k^0_{i,r})$ and $\mathsf{com}(k^1_{i,r})$ values in $(W_{i,j}, W'_{i,j})$, for every check-set S_j (see Figure 2).*

   For every pair of check-sets $(W_{i,j}, W'_{i,j})$, *party P_1 decommits to the first value in each set (i.e., to the value that is supposed to be a commitment to the indicator bit,* $\mathsf{com}(0)$ *or* $\mathsf{com}(1)$*).*

6. DECOMMITMENT PHASE FOR P_1'S INPUT IN EVALUATION-CIRCUITS: *P_1 decommits to the garbled values that correspond to its inputs in evaluation-circuits. Let i be the index of an input wire that corresponds to P_1's input (the following procedure is applied to all such wires). Let b be the binary value that P_1 assigns to input wire i. In every evaluation-set $(W_{i,j}, W'_{i,j})$, P_1 chooses the set (out of $(W_{i,j}, W'_{,j})$), which corresponds to the value b. It then opens in this set the commitments that correspond to evaluation-circuits. Namely, to the values $k^b_{i,r}$, where r is an index of an evaluation circuit (see Figure 3).*

7. CORRECTNESS AND CONSISTENCY CHECKS: *P_2 performs the following checks; if any of them fails it aborts.*

   (a) Checking correctness of the check-circuits: *P_2 verifies that each check-circuit GC_i is a garbled version of C. This check is carried out by P_2 first constructing the input tables that associate every garbled value of an input wire to a binary value. The input tables for P_2's inputs are constructed by checking that the decommitments to the pairs $(\mathsf{com}(k^0_{i,r}), \mathsf{com}(k^1_{i,r}))$ (where i is a wire index and r is a circuit index) are valid, and then interpreting the first value to be associated with 0 and the second value to be associated with 1.*
   Next, P_2 checks the decommitments to P_1's inputs. This check involves first checking that the decommitment values are valid. Then, P_2 verifies that in each pair of check-sets, one of $(W_{i,j}, W'_{i,j})$ begins with a commitment to 0 (henceforth the 0-tuple), and the other begins with a commitment to 1 (henceforth the 1-tuple). Then P_2 checks that for every wire, the values that are decommitted to in the 0-tuples in all check-sets are all equal, and that a similar property holds for the 1-tuples. P_2 then assigns the logical value of 0 to all of the opened commitments in the 0-tuples, and the logical value of 1 to the opened commitments in the 1-tuples.
   Finally, given all the garbled values to the input wires and their associated binary values, P_2 decrypts the circuit and compares it with the circuit C.

   (b) Verifying P_2's input in the check-circuits: *P_2 verifies that P_1's decommitments to the wires corresponding to P_2's input values in the check-circuits are correct, and agree with the logical values of these wires (the indicator bits). P_2 also checks that the inputs it learned in the oblivious transfer stage for the check-circuits correspond to its actual input. Specifically, it checks that the decommitment values that it received in the oblivious transfer stage open the committed values that correspond to the garbled values of its logical input (namely, that it received the first value in the pair if the input bit is 0 and the second value if it is 1).*[6]

(c) Checking P_1's input to evaluation-circuits: *Finally, P_2 verifies that for every input wire i of P_1 the following two properties hold:*

   i. *In every evaluation-set P_1 chose one of the two sets and decommitted to all the commitments in it which correspond to evaluation-circuits.*

   ii. *For every evaluation-circuit, all of the commitments that P_1 opened in evaluation-sets are for the same garbled value.*

8. CIRCUIT EVALUATION: *If any of the above checks fails, P_2 aborts and outputs $\perp$. Otherwise, P_2 evaluates the evaluation circuits (in the same way as for the semi-honest protocol of Yao). It might be that in certain circuits the garbled values provided for P_1's inputs, or the garbled values learned by P_2 in the OT stage, do not match the tables and so decryption of the circuit fails. In this case P_2 also aborts and outputs $\perp$. Otherwise, P_2 takes the output that appears in most circuits, and outputs it (the proof shows that this value is well defined).*

Proof of the paper

Security against a Malicious P1: The proof constructs an ideal-model adversary/simulator which has access to P1 and to the trusted party, and can simulate the view of an actual run of the protocol. It uses the fact that the strings ρ, ρ_0, which choose the circuits and commitment sets that are checked, are uniformly distributed even if P1 is malicious. The simulator runs the protocol until P1 opens the commitments of the checked circuits and checked commitment sets, and then rewinds the execution and runs it again with new random ρ, ρ_0 values. We expect that about one quarter of the circuits are checked in the first execution and evaluated in the second execution. For these circuits, in the first execution the simulator learns the translation between the garbled values of P1's input wires and the actual values of these wires, and in the second execution it learns the garbled values that are associated with P1's input (this association is learned from the garbled values that P1 sends to P2). Combining the two, it learns P1's input x, which can then be sent to the trusted party. The trusted party answers with f(x, y), which we use to define P2's output and complete the simulation.

- Security against a Malicious P2. Intuitively, the security in this case is derived from the fact that:
- (a) the oblivious transfer protocol is secure, and so P2 only learns a single set of keys (corresponding to a single input y) for decrypting the garbled circuits, and
- (b) the commitment schemes are hiding and so P2 does not know what input corresponds to the garbled values that P1 sends it for evaluating the circuit.
- Of course, in order to formally prove security we construct an ideal-model simulator B2 working with an adversary A2 that has corrupted P2.

- The simulator first extracts P2's input bits from the oblivious transfer protocol, and then sends the input y it obtained to the trusted party and receives back z = f(x, y). Given the output, the simulator constructs the garbled circuits. However, rather than constructing them all correctly, for each circuit it tosses a coin and, based on the result, either constructs the circuit correctly, or constructs it to compute the constant function outputting z (the output is received from the trusted party). In order to make sure that the simulator is not caught cheating, it biases the coin-tossing phase so that all of the correctly constructed garbled circuits are check-circuits, and all of the other circuits are evaluation-circuits (this is why the protocol uses joint coin-tossing rather than let P2 alone choose the circuits to be opened). *A*2 then checks the correctly-constructed circuits, and is satisfied with the result as if it were interacting with a legitimate *P*1. *A*2 therefore continues the execution with the circuits which always output *z*.

Reducing the Number of Oblivious Transfers(1)

- For Bob, his original input is $y = y_1 y_2 \ldots\ldots y_n$, we can encode y like:

$$y_i = w_i + w_{i+1} + \cdots + w_{i+s-1}$$

y has been encoded as $w = w_1 w_2 \ldots\ldots w_{n+s-1}$

we use w as the input of Bob. It is secure for each **single** bit of y.

BUT!!! $y_1 \oplus y_2 = w_1 \oplus w_{s+1}$

Reducing the Number of Oblivious Transfers(2)

- We should encode y to $w = w_1 w_2 \ldots \ldots w_m$ to make sure that the exclusive-or of every subset of y can' t be written as any sw_i's exclusive-or.
- If $y_i = \sum_{j=1}^{m} a_{ij} w_j$, we call the encode of y_i is $c_i = a_{i1} a_{i2} \ldots a_{im}$ then the requirement is converted to minimal hamming distance of code C $= \{c_1, c_2, \ldots c_n\}$ is s.

Reducing the Number of Oblivious Transfers(3)

- Gilbert bounds:

$$\frac{2^m}{\sum_{i=1}^{s-1}\binom{m}{i}} \leq N = 2^n$$

we can find that: $m \leq n + s \cdot (logn + logs)$

Still longer!? We can use randomization.

Reducing the Number of Oblivious Transfers(4)

- c_i are randomly chosen in $\{0,1\}^m$ and m=max{4n,8s}=O(n,s), then we assume $n > 2s$, otherwise expand the input's length.
- The encode of the exclusive-or of every subset of y is also a random code in $\{0,1\}^m$, let its hamming weight be X then:
- $\Pr\{X < s\} = \Pr\left\{\left|\frac{X}{4n} - \frac{1}{2}\right| > \frac{3}{8}\right\} < 2e^{-9n/8}$

Thank you

Q&A