

Secure Multiparty Computation (SMC or MPC)

Presenter: Yi LIU

Cryptography

- Conventional Usages

 - Confidentiality

 - E.g. Encryption

 - Integrity

 - E.g. MAC

 - Authentication

 - E.g. Signature

- Secure Computing

 - 1 party (e.g. FHE)

 - 2 parties (e.g. Yao's GC)

 - 3+ parties (e.g. secret sharing based MPC)

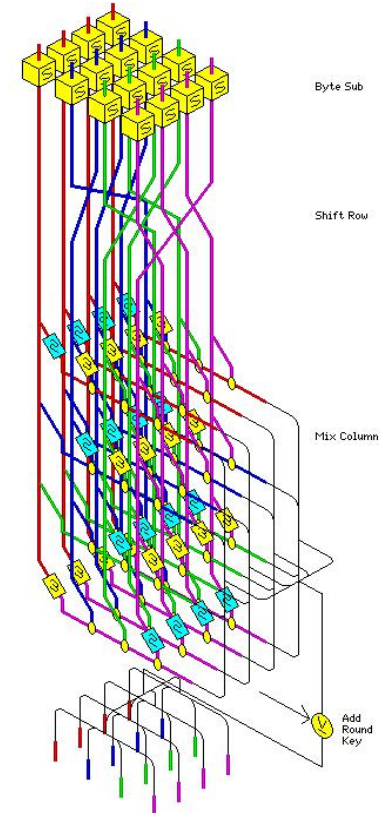
What is Security?

What is Security?

- Why your password is secure?
Hard to guess

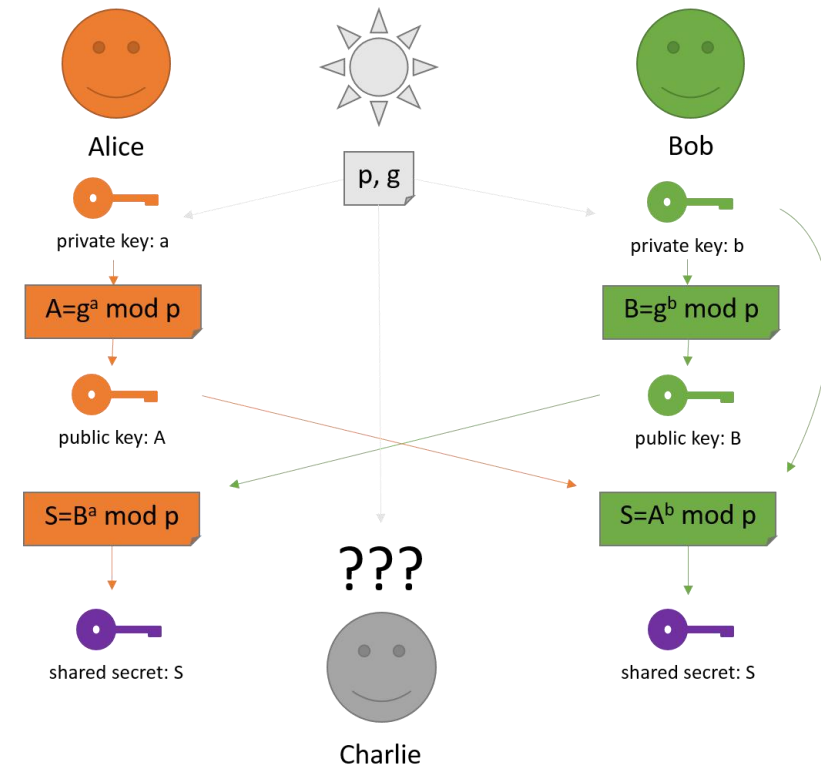
What is Security?

- Why your password is secure?
Hard to guess
- Why AES (Advanced Encryption Standard) is secure?
Assumption: tautology



What is Security?

- Why Diffie-Hellman key exchange is secure?
 - DDH (Decisional Diffie-Hellman) Assumption
 - DDH stronger than DLP (Discrete Logarithm Problem)
 - i.e., we can break DDH (specific)
WHILE DLP (general) is secure.



What is Security?

- Why Diffie-Hellman key exchange is secure?
 - DDH (Decisional Diffie-Hellman) Assumption
 - DDH stronger than DLP (Discrete Logarithm Problem)
 - i.e., we can break DDH (specific)
WHILE DLP (general) is secure.
- RSA?

What is Security?

- Why Diffie-Hellman key exchange is secure?
 - DDH (Decisional Diffie-Hellman) Assumption
 - DDH stronger than DLP (Discrete Logarithm Problem)
 - i.e., we can break DDH (specific)
WHILE DLP (general) is secure.
- RSA?
 - RSA Assumption

What is Security?

- Why Diffie-Hellman key exchange is secure?
 - DDH (Decisional Diffie-Hellman) Assumption
 - DDH stronger than DLP (Discrete Logarithm Problem)
 - i.e., we can break DDH (specific)
WHILE DLP (general) is secure.
- RSA?
 - RSA Assumption
 - RSA Assumption stronger than/or equal to integer factorization problem
 - i.e., we may break RSA
WHILE integer factorization is secure. (*Boneh, Venkatesan 98*)

What is Security?

- We say SOMETHING is (τ, ϵ) -secure if and only if no adversary with running time less than τ can break it with probability more than ϵ .

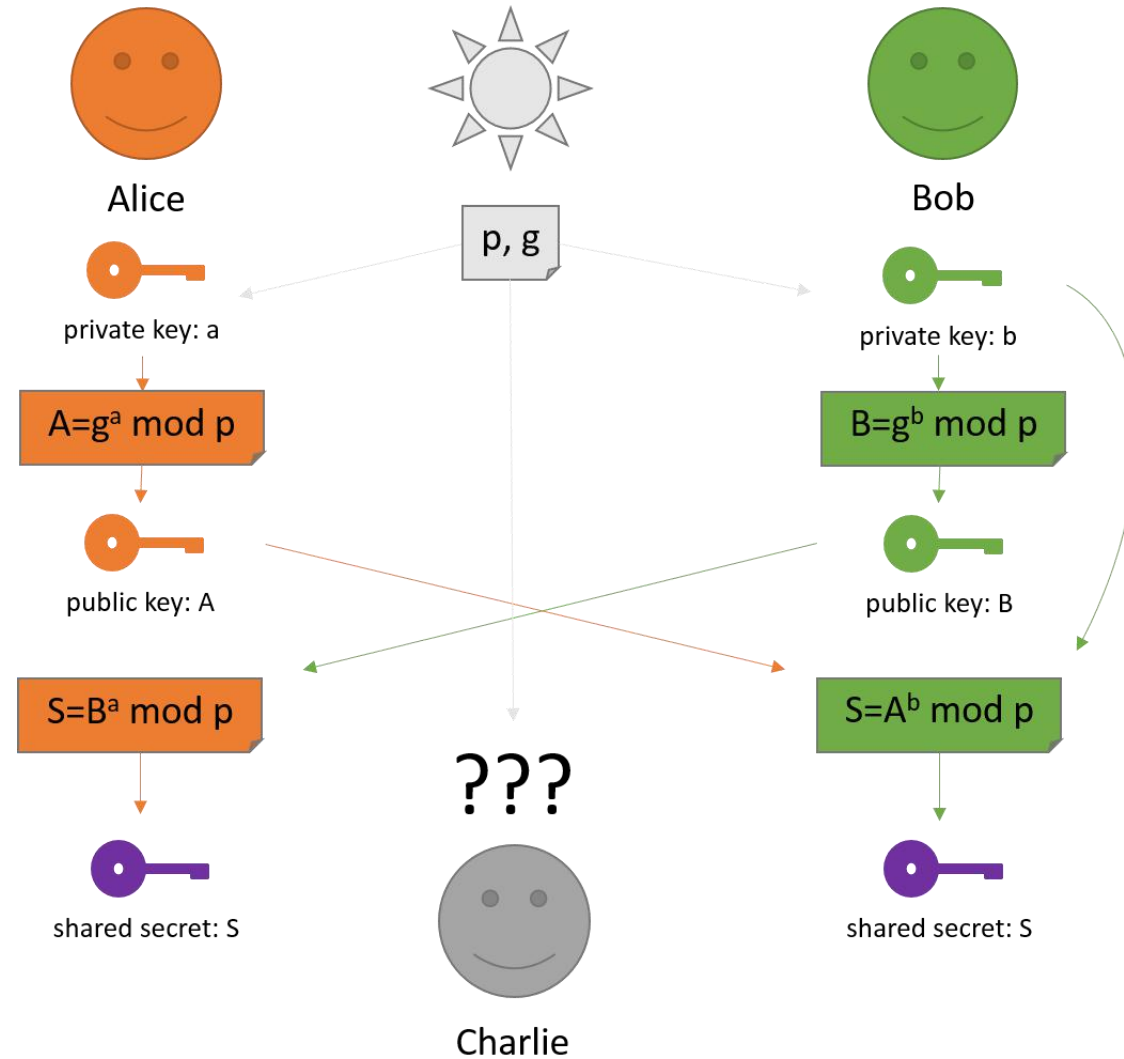
What is Security?

- We say SOMETHING is (τ, ϵ) -secure if and only if no adversary with running time less than τ can break it with probability more than ϵ .
- P protocol is WHAT SECURE under X assumption

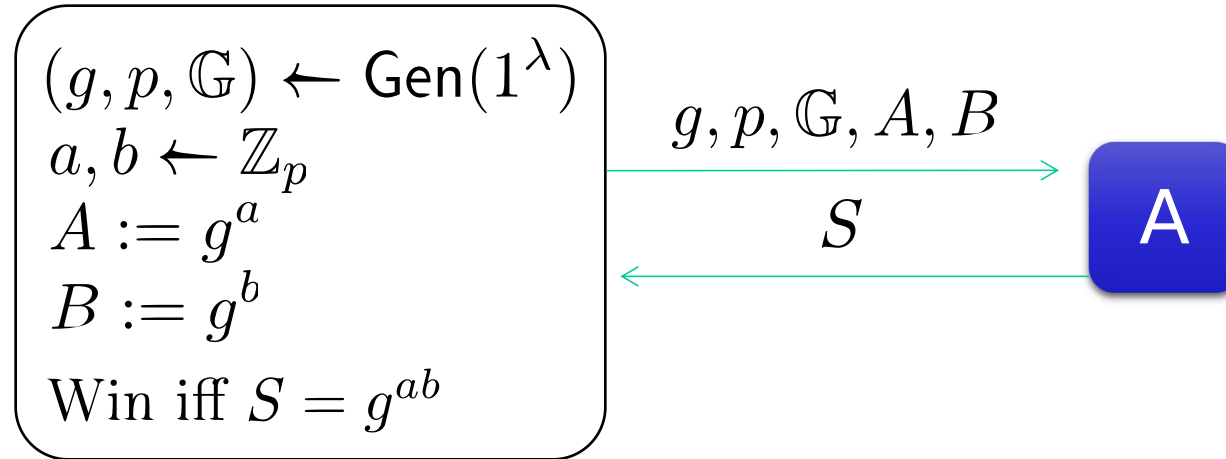
Design a Game

- What does “break it” mean?
Define adversary’s capability via a game

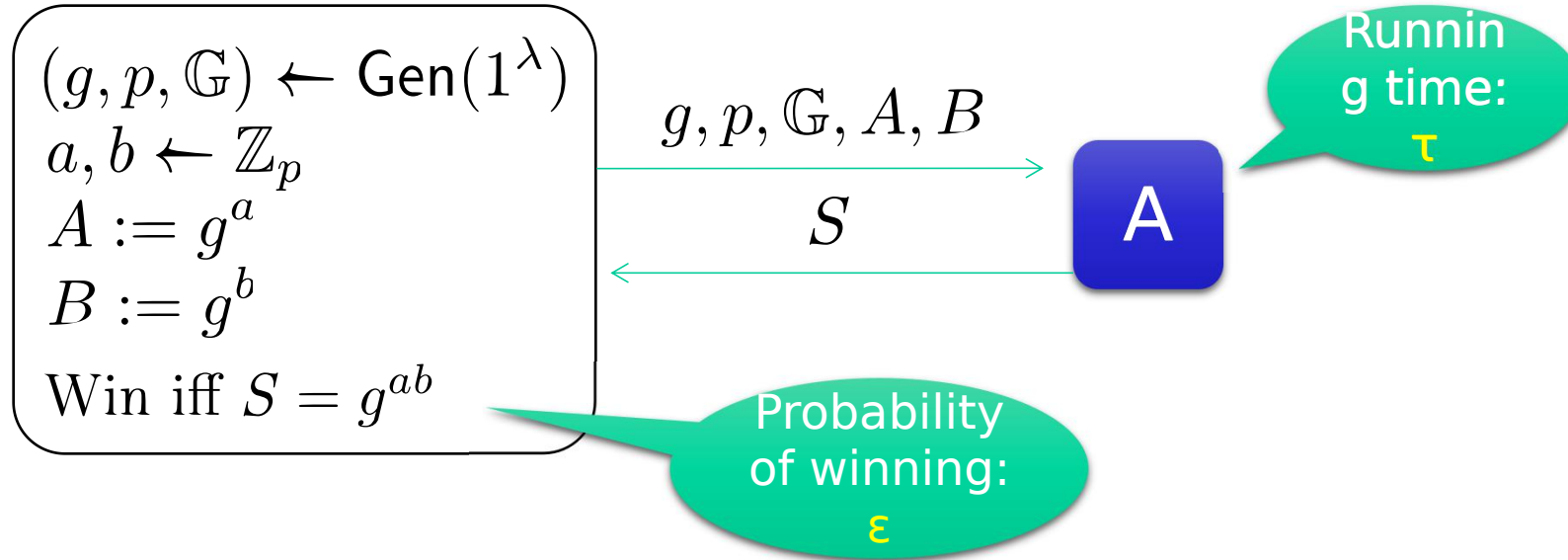
Design a Game



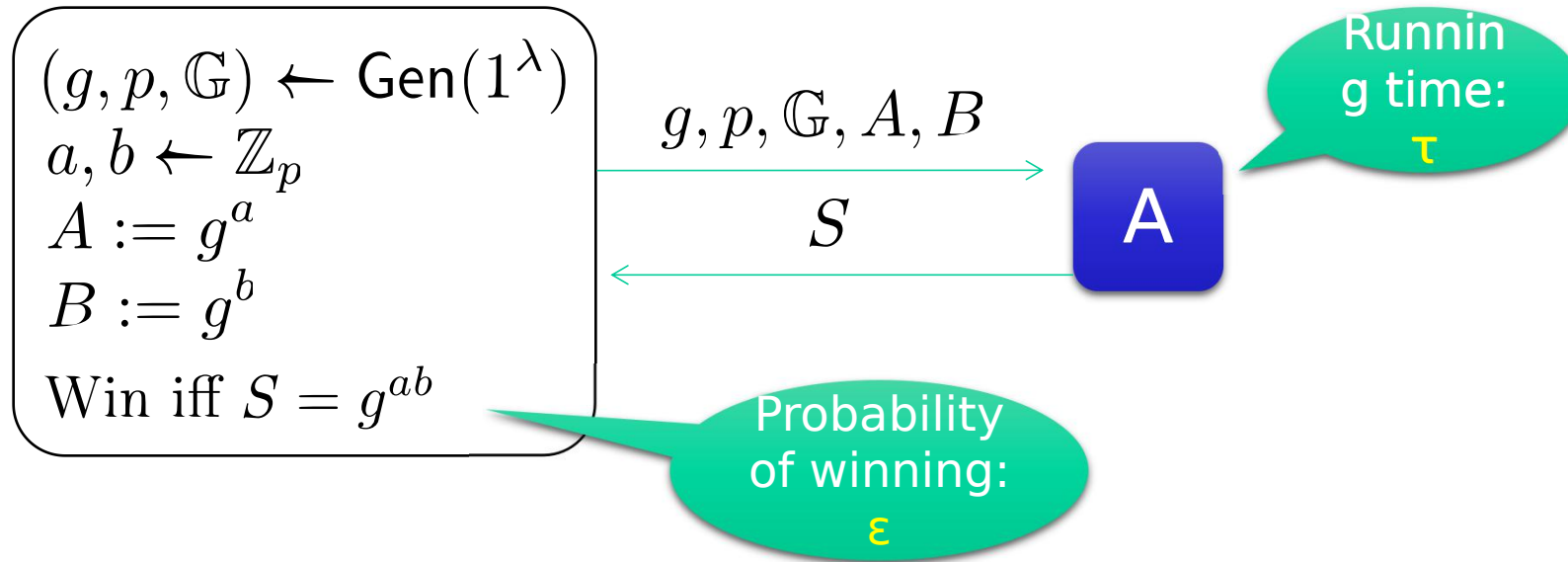
Design a Game



Design a Game

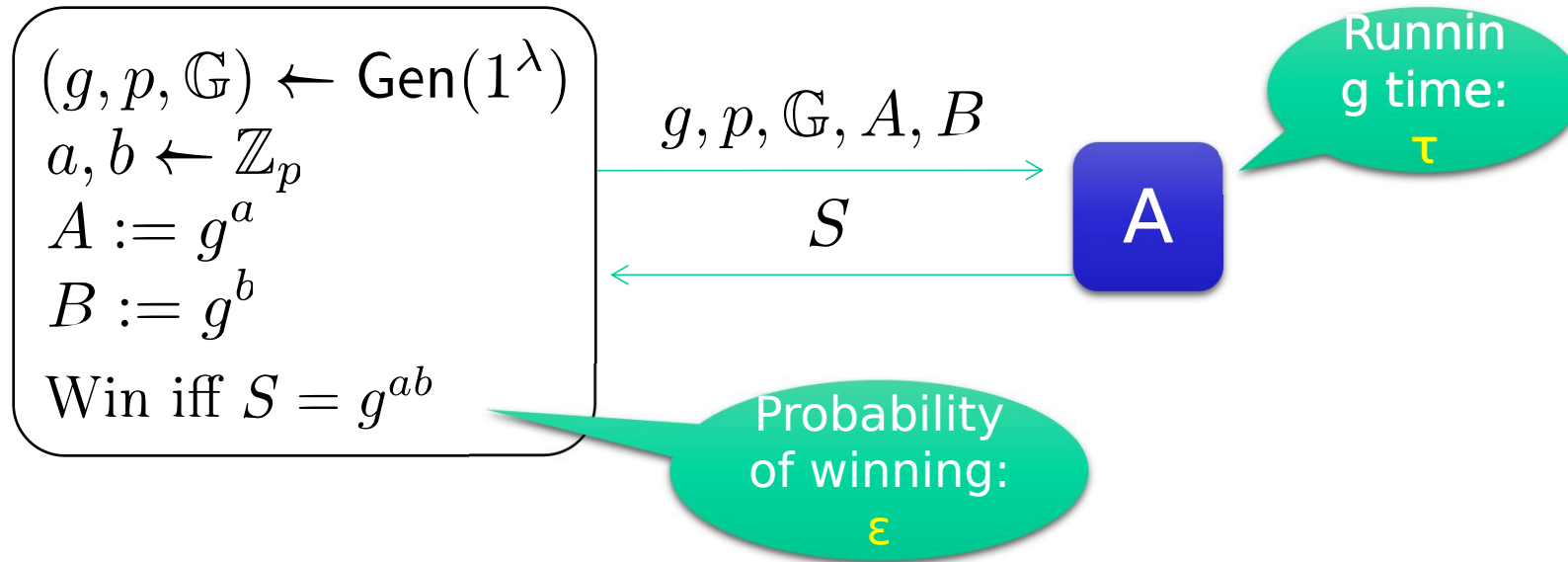


Design a Game



- What if the adversary can learn the last 10 bits of the key ?
This does not let the adversary win our security game.

Design a Game



- What if the adversary can learn the last 10 bits of the key ?
This does not let the adversary win our security game.
- We need a better security definition
No matter how the key will be used latter
Quantify the leakage of the key

Design a Game

- Idea: use indistinguishability to model “leakage”
If the adversary cannot even distinguish the real key from a fake one then she knows nothing about the key.

Design a Game

- Idea: use indistinguishability to model “leakage”
If the adversary cannot even distinguish the real key from a fake one then she knows nothing about the key.

$(g, p, \mathbb{G}) \leftarrow \text{Gen}(1^\lambda)$
 $a, b \leftarrow \mathbb{Z}_p$
 $A := g^a$
 $B := g^b$
 $K_0 := g^{ab}$
 $K_1 \leftarrow \mathbb{G}$
 $b \leftarrow \{0, 1\}$
Win iff $b = b^*$

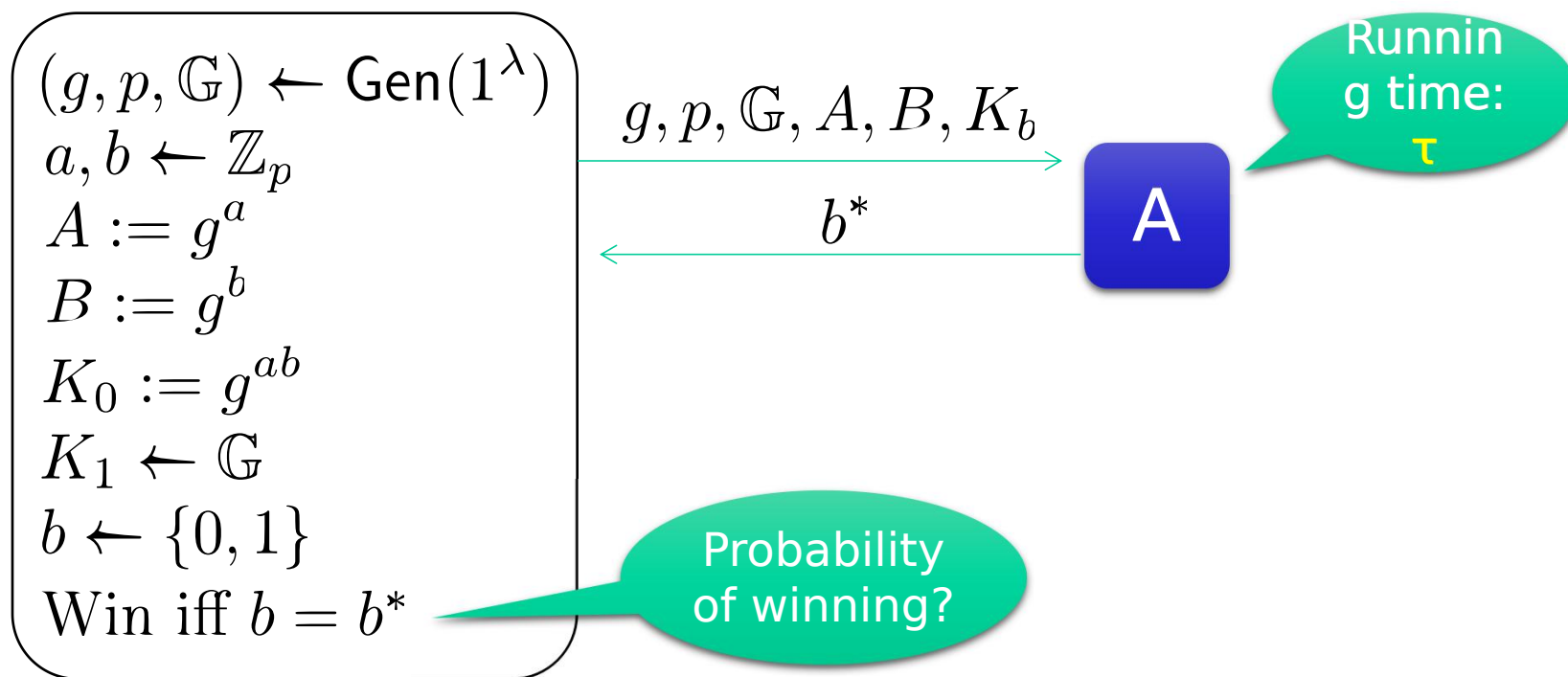
$g, p, \mathbb{G}, A, B, K_b$

b^*

A

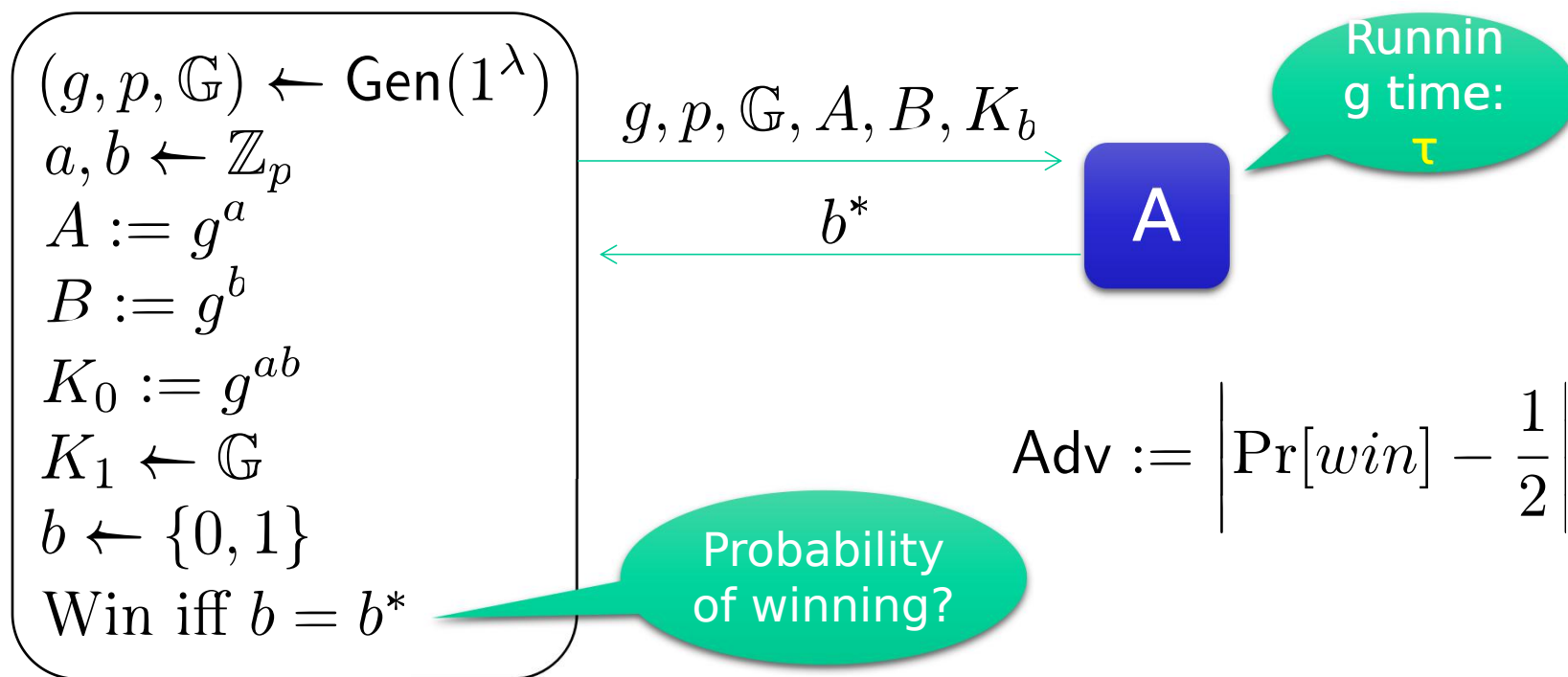
Design a Game

- Idea: use indistinguishability to model “leakage”
If the adversary cannot even distinguish the real key from a fake one then she knows nothing about the key.



Design a Game

- Idea: use indistinguishability to model “leakage”
If the adversary cannot even distinguish the real key from a fake one then she knows nothing about the key.



Provable Security

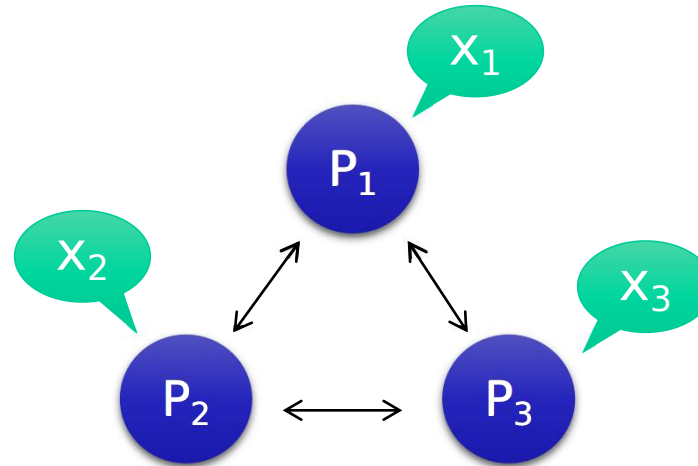
- Rigorous security definition
 - Adversarial model (a.k.a. attacker model)
 - Property based
 - Simulation based

 - Property-based VS Simulation-based
 - <https://crypto.stackexchange.com/questions/3814/simulation-based-security>

- Precise assumption
 - Hard problems

- Formal proof
 - Usually via reduction

Secure Multiparty Computation



- ✧ Input parties
- ✧ Computing parties
- ✧ Output parties

$F(x_1, x_2, x_3)$

Secure Multiparty Computation

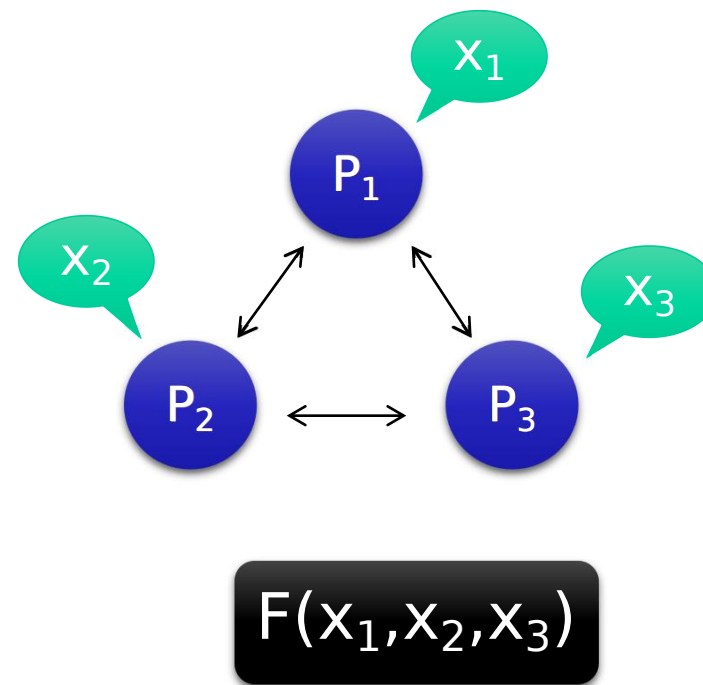
- Secure two-party computation (2PC) FOCS'82
- Yao's Millionaires' Problem



姚期智
Turing Award (2000)

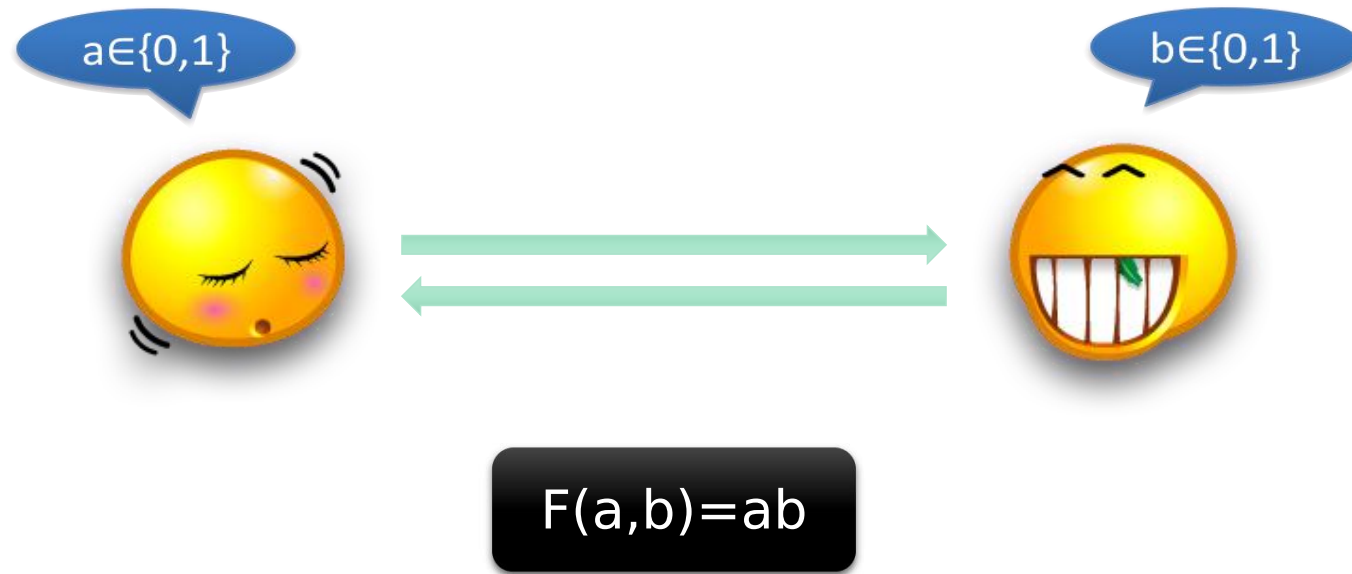
Goal of MPC

- What is the security goal?
 - o Input privacy:
 - P_i 's input is unknown
 - o Output correctness:
 - $F(x_1, x_2, x_3)$ is correct
 - o Input independency:
 - One player's input should not depends on the others'.
 - o Fairness:
 - Either everyone get the output or none of them gets the output.
 - o Guaranteed output delivery (GOD)



Special Case: 2-party Computation

oExample: Private VETO



Oblivious Transfer (OT)

- Sender has x_0, x_1 ; receiver has b
- Receiver obtains x_b only
- Sender learns nothing

Oblivious Transfer

- ▶ **Trapdoor permutation (I, D, F, F^{-1})**
 - I : samples a function f and trapdoor t in the family
 - $D(f)$: uniformly samples a value in the domain of f
 - $F(f, x)$: computes $f(x)$
 - $F^{-1}(t, y)$: computes $f^{-1}(y)$
 - Hard to invert a random y , given f (but not t)

Oblivious Transfer

- ▶ **Trapdoor permutation (I, D, F, F^{-1})**
 - I : samples a function f and trapdoor t in the family
 - $D(f)$: uniformly samples a value in the domain of f
 - $F(f, x)$: computes $f(x)$
 - $F^{-1}(t, y)$: computes $f^{-1}(y)$
 - Hard to invert a random y , given f (but not t)
- ▶ **Enhanced trapdoor permutations**
 - Hard to invert y , even given the random coins used to sample y (using D)

Oblivious Transfer

▶ Hard-core predicate B

- Given $y=f(x)$, can guess $B(x)$ with probability only negligibly greater than $\frac{1}{2}$
- Equivalently, given $y=f(x)$, the bit $B(x)$ is pseudorandom

Oblivious Transfer Protocol

- ▶ Sender's input: (z_0, z_1) ; receiver's input b

Oblivious Transfer Protocol

- ▶ Sender's input: (z_0, z_1) ; receiver's input b
- ▶ Sender's first message:
 - Sender chooses (f, t) using sampling algorithm I
 - Sender sends f to receiver

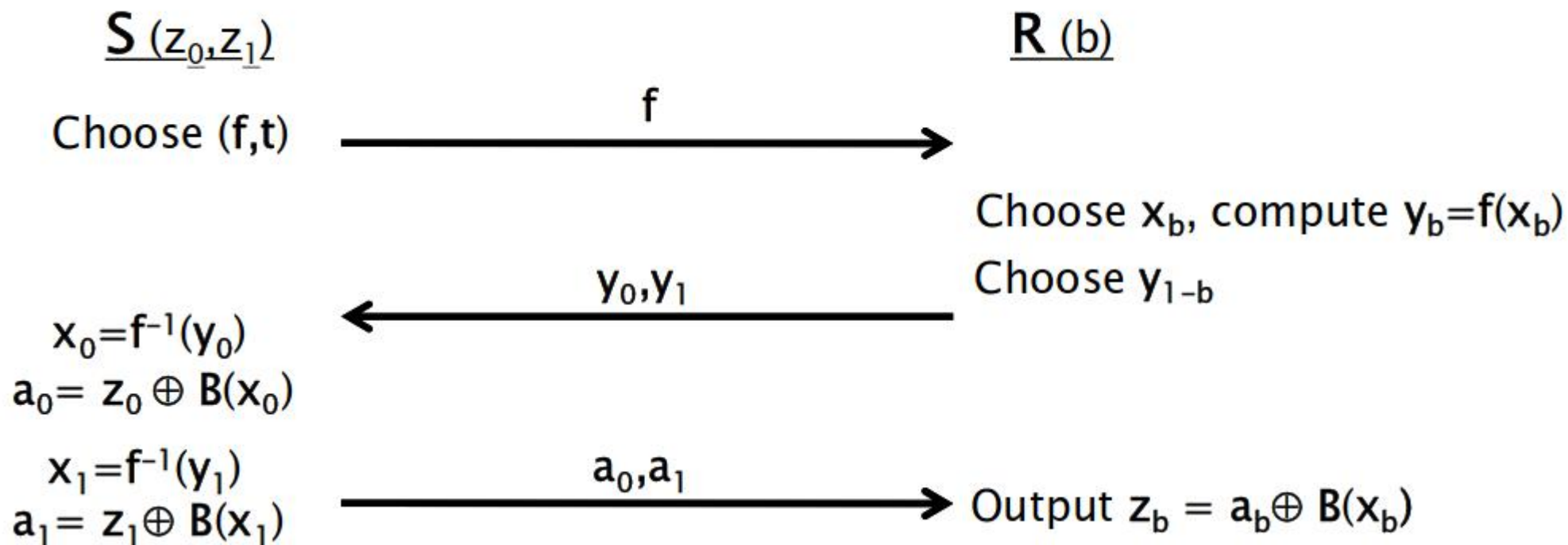
Oblivious Transfer Protocol

- ▶ **Sender's input:** (z_0, z_1) ; receiver's input b
- ▶ **Sender's first message:**
 - Sender chooses (f, t) using sampling algorithm I
 - Sender sends f to receiver
- ▶ **Receiver's first message:**
 - Receiver chooses x_b and computes $y_b = f(x_b)$
 - Receiver chooses random y_{1-b}
 - Receiver sends (y_0, y_1) to sender

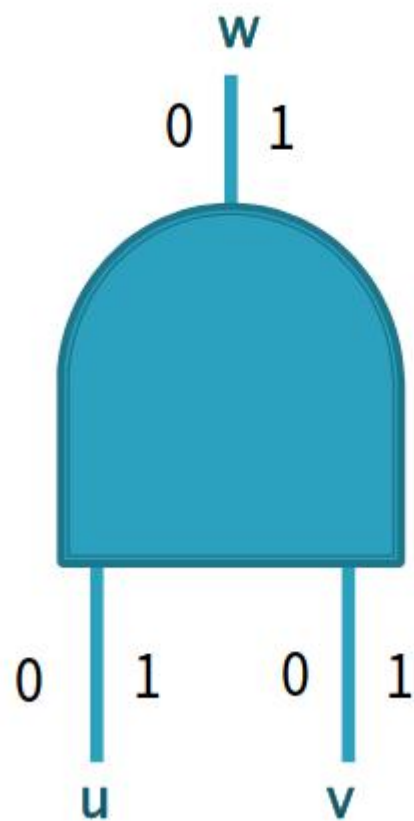
Oblivious Transfer Protocol

- ▶ **Sender's input:** (z_0, z_1) ; receiver's input b
- ▶ **Sender's first message:**
 - Sender chooses (f, t) using sampling algorithm I
 - Sender sends f to receiver
- ▶ **Receiver's first message:**
 - Receiver chooses x_b and computes $y_b = f(x_b)$
 - Receiver chooses random y_{1-b}
 - Receiver sends (y_0, y_1) to sender
- ▶ **Sender's second message:**
 - Sender computes (x_0, x_1) by inverting
 - Sender computes $a_i = z_i \oplus B(x_i)$
 - Sender sends (a_0, a_1) to receiver
- ▶ **Receiver outputs** $z_b = a_b \oplus x_b$

Oblivious Transfer Protocol

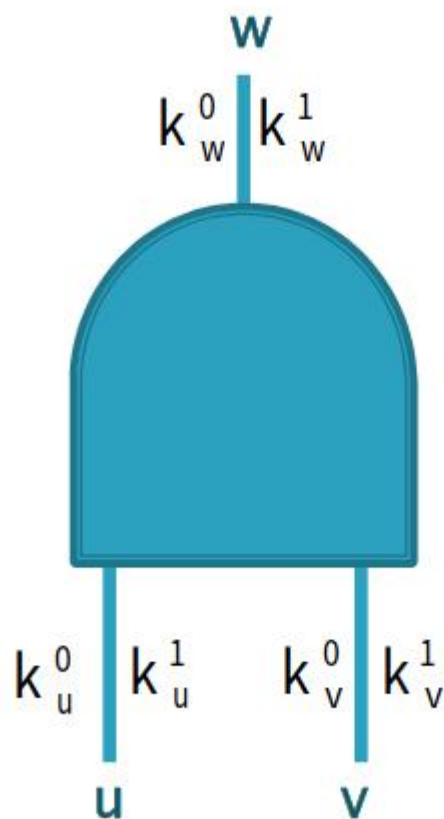


Yao's Garbled Circuit



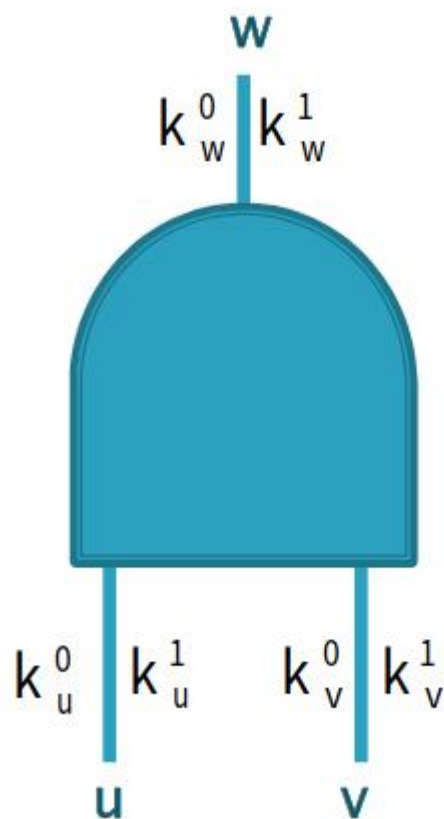
u	v	w
0	0	0
0	1	0
1	0	0
1	1	1

Yao's Garbled Circuit



u	v	w
k_u^0	k_v^0	k_w^0
k_u^0	k_v^1	k_w^0
k_u^1	k_v^0	k_w^0
k_u^1	k_v^1	k_w^1

Yao's Garbled Circuit



u	v	w
k_u^0	k_v^0	$E_{k_u^0}(E_{k_v^0}(k_w^0))$
k_u^0	k_v^1	$E_{k_u^0}(E_{k_v^1}(k_w^0))$
k_u^1	k_v^0	$E_{k_u^1}(E_{k_v^0}(k_w^0))$
k_u^1	k_v^1	$E_{k_u^1}(E_{k_v^1}(k_w^1))$

Yao's Garbled Circuit

- ▶ The actual garbled gate

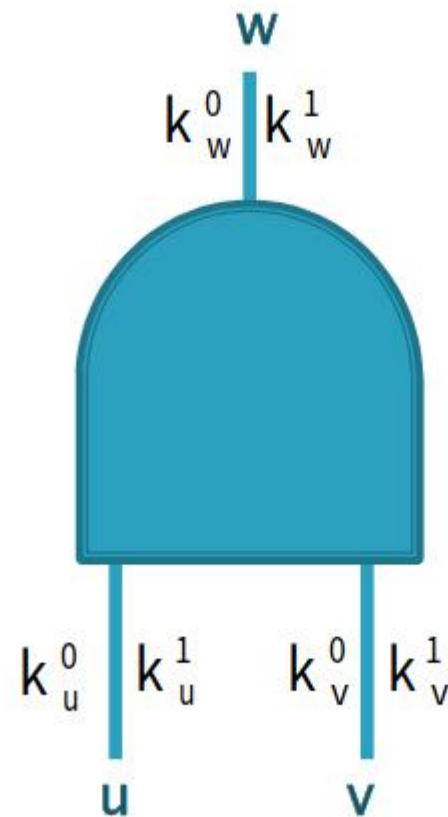
$$E_{k_u^1}(E_{k_v^0}(k_w^0))$$

$$E_{k_u^0}(E_{k_v^1}(k_w^0))$$

$$E_{k_u^1}(E_{k_v^1}(k_w^1))$$

$$E_{k_u^0}(E_{k_v^0}(k_w^1))$$

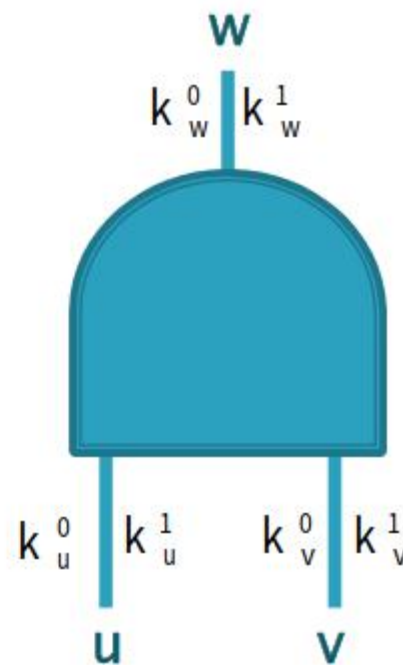
- ▶ Given k_u^0 and k_v^1 can obtain k_w^0 only
- ▶ Furthermore, since the table is permuted, the party has no idea if it obtained the 0 or 1 key



Yao's Garbled Circuit

- ▶ If the gate is an output gate, need to provide the “decryption” of the output wire
- ▶ Output translation table

$$[(0, k_w^0), (1, k_w^1)]$$



Yao's Protocol

- ▶ Input: x and y of length n

Yao's Protocol

- ▶ **Input:** x and y of length n
- ▶ P_1 generates a garbled circuit $G(C)$
 - k_L^0, k_L^1 are the keys on wire w_L
 - Let w_1, \dots, w_n be the input wires of P_1 and w_{n+1}, \dots, w_{2n} be the input wires of P_2

Yao's Protocol

- ▶ Input: x and y of length n
- ▶ P_1 generates a garbled circuit $G(C)$
 - k_L^0, k_L^1 are the keys on wire w_L
 - Let w_1, \dots, w_n be the input wires of P_1 and w_{n+1}, \dots, w_{2n} be the input wires of P_2
- ▶ P_1 sends P_2 the strings $k_1^{x_1}, \dots, k_n^{x_n}$

Yao's Protocol

- ▶ **Input:** x and y of length n
- ▶ P_1 generates a garbled circuit $G(C)$
 - k_L^0, k_L^1 are the keys on wire w_L
 - Let w_1, \dots, w_n be the input wires of P_1 and w_{n+1}, \dots, w_{2n} be the input wires of P_2
- ▶ P_1 sends P_2 the strings $k_1^{x_1}, \dots, k_n^{x_n}$
- ▶ P_1 and P_2 run n OTs in parallel
 - P_1 inputs k_{n+i}^0, k_{n+i}^1
 - P_2 inputs y_i

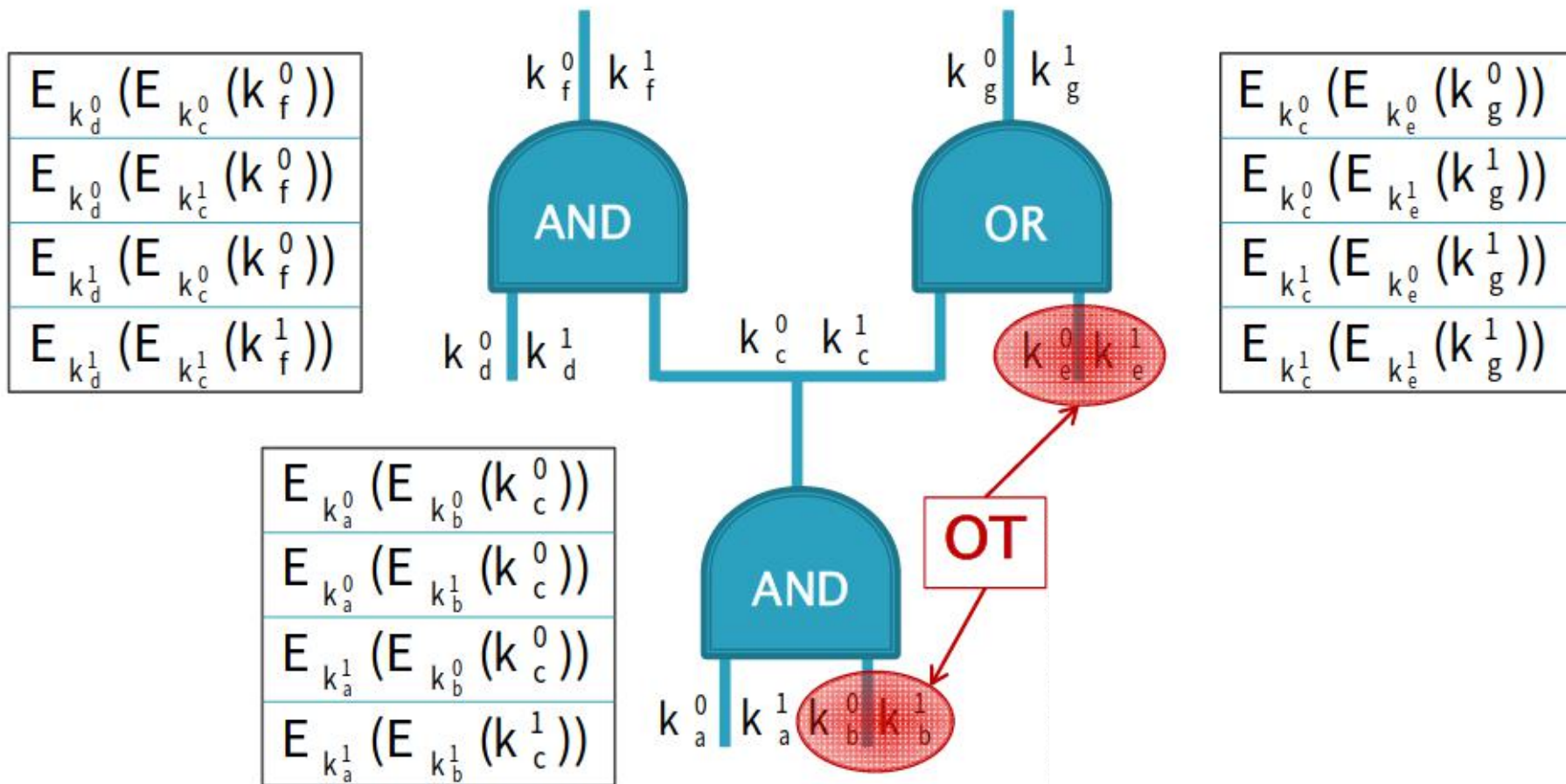
Yao's Protocol

- ▶ **Input:** x and y of length n
- ▶ P_1 generates a garbled circuit $G(C)$
 - k_L^0, k_L^1 are the keys on wire w_L
 - Let w_1, \dots, w_n be the input wires of P_1 and w_{n+1}, \dots, w_{2n} be the input wires of P_2
- ▶ P_1 sends P_2 the strings $k_1^{x_1}, \dots, k_n^{x_n}$
- ▶ P_1 and P_2 run n OTs in parallel
 - P_1 inputs k_{n+i}^0, k_{n+i}^1
 - P_2 inputs y_i
- ▶ Given all keys, P_2 computes $G(C)$ and obtains $C(x, y)$
 - P_2 sends result to P_1

The Example Circuit

(input wires $P_1 = d, a$; $P_2 = b, e$)

$$[(0, k_f^0), (1, k_f^1)] \quad [(0, k_g^0), (1, k_g^1)]$$



How to gambling over Wechat?

How to Gambling over Wechat?

- What is a commitment?

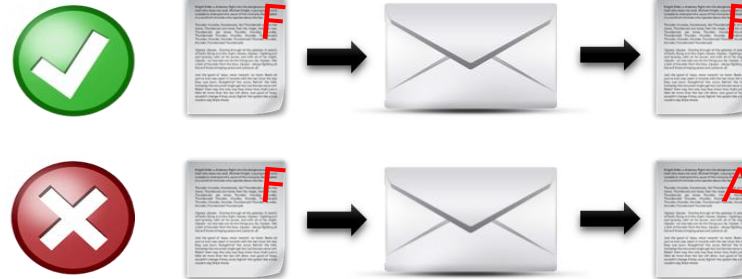


How to gambling over Wechat?

- What is a commitment?



- Binding Property:



How to gambling over Wechat?

- What is a commitment?



- Binding Property:



- Hiding Property:



Design a Commitment

- First attempt
 - $H(m)$ or g^m

Design a Commitment

- First attempt
 - $H(m)$ or g^m
- Second attempt
 - $H(m||r)$ or $g^{m||r}$

Design a Commitment

➤ First attempt

- $H(m)$ or g^m

➤ Second attempt

- $H(m||r)$ or $g^{m||r}$

➤ Pederson commitment

- Commitment key: $ck := (g, h)$
- Committing: $\text{Com}_{ck}(m; r) := g^m h^r$
- Opening: reveal (m, r) and checking

Pederson Commitment

➤ Perfect Hiding

Pederson Commitment

➤ Perfect Hiding

$$\begin{aligned}\Pr[C = c] &= \sum \Pr[C = c | M = m] \cdot \Pr[M = m] \\ &= \sum \Pr[R = r \text{ s.t. } c = \text{Com}_{\text{ck}}(m; r)] \cdot \Pr[M = m] \\ &= \sum \frac{1}{|\mathbb{G}|} \cdot \Pr[M = m] = \frac{1}{|\mathbb{G}|} \sum \Pr[M = m] = \frac{1}{|\mathbb{G}|}\end{aligned}$$

Pederson Commitment

➤ Perfect Hiding

$$\begin{aligned}\Pr[C = c] &= \sum \Pr[C = c | M = m] \cdot \Pr[M = m] \\ &= \sum \Pr[R = r \text{ s.t. } c = \text{Com}_{\text{ck}}(m; r)] \cdot \Pr[M = m] \\ &= \sum \frac{1}{|\mathbb{G}|} \cdot \Pr[M = m] = \frac{1}{|\mathbb{G}|} \sum \Pr[M = m] = \frac{1}{|\mathbb{G}|}\end{aligned}$$

$$\begin{aligned}\Pr[M = m | C = c] &= \frac{\Pr[C = c | M = m] \cdot \Pr[M = m]}{\Pr[C = c]} \\ &= \frac{\Pr[R = r \text{ s.t. } c = \text{Com}_{\text{ck}}(m; r)] \cdot \Pr[M = m]}{1/|\mathbb{G}|} \\ &= \frac{1/|\mathbb{G}| \cdot \Pr[M = m]}{1/|\mathbb{G}|} = \Pr[M = m]\end{aligned}$$

Pederson Commitment

$$\text{Com}_{\text{ck}}(m; r) := g^m h^r$$

➤ Computational Binding

If the adversary does not know the discrete logarithm of h , then she could not double open the commitment.

Assuming discrete logarithm problem is hard, the Pederson commitment is binding.

Given (m_1, r_1) and (m_2, r_2) such that

$$\text{Com}_{\text{ck}}(m_1; r_1) = c = \text{Com}_{\text{ck}}(m_2; r_2)$$

we can compute $\text{DLog}_g(h) = \frac{m_1 - m_2}{r_2 - r_1}$

Pederson Commitment

$$\text{Com}_{\text{ck}}(m; r) := g^m h^r$$

➤ Computational Binding

If the adversary does not know the discrete logarithm of h , then she could not double open the commitment.

Assuming discrete logarithm problem is hard, the Pederson commitment is binding.

Given (m_1, r_1) and (m_2, r_2) such that

$$\text{Com}_{\text{ck}}(m_1; r_1) = c = \text{Com}_{\text{ck}}(m_2; r_2)$$

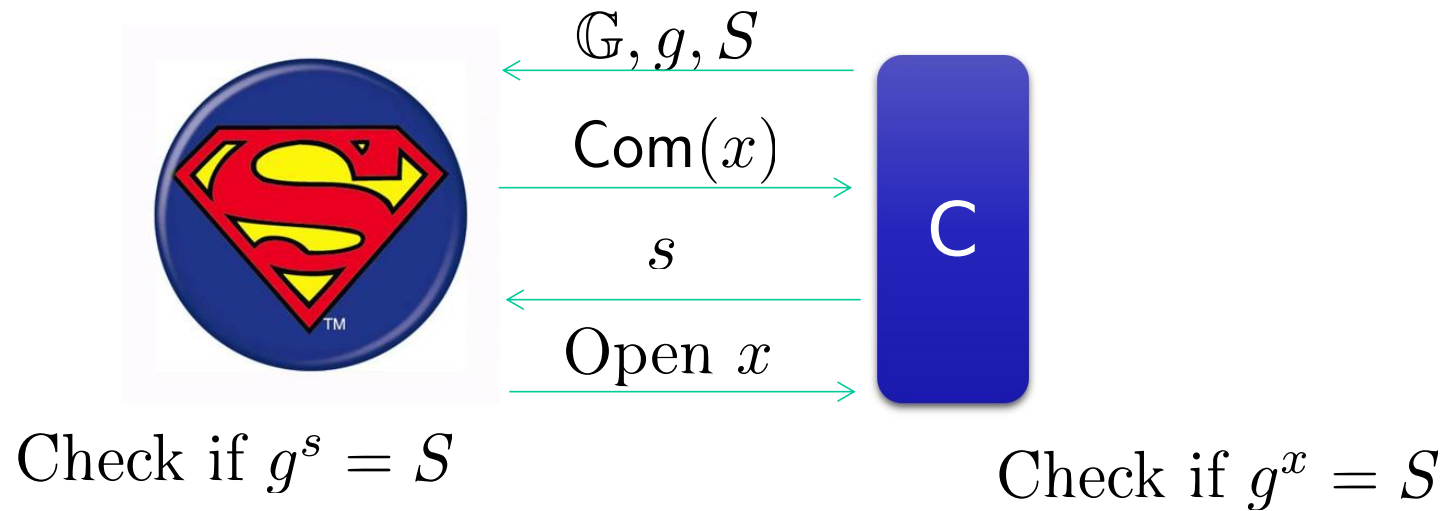
we can compute $\text{DLog}_g(h) = \frac{m_1 - m_2}{r_2 - r_1}$

Toy Protocol

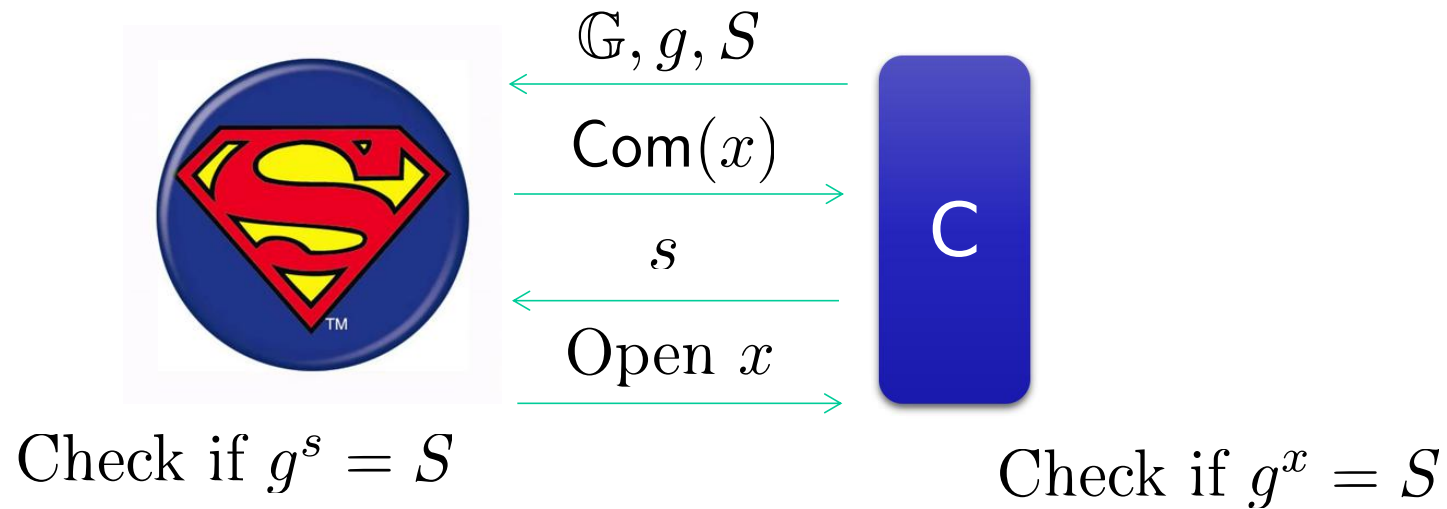
- Suppose a man claims that he has a machine (e.g. quantum computer) that can **efficiently** solve any discrete logarithm problem.

Toy Protocol

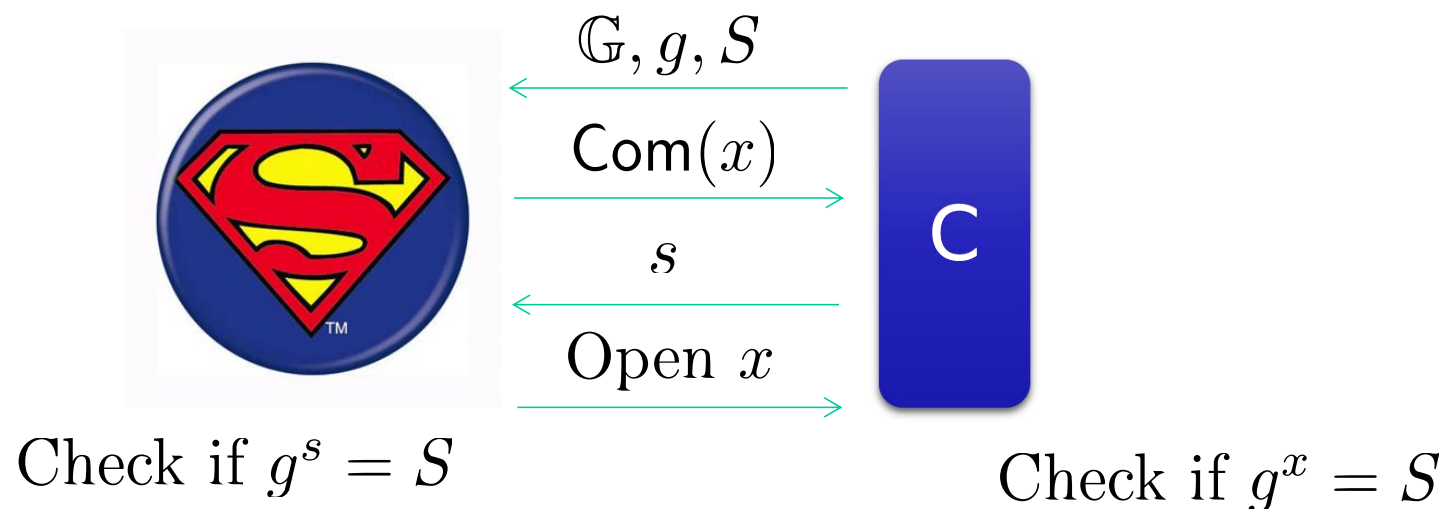
- Suppose a man claims that he has a machine (e.g. quantum computer) that can **efficiently** solve any discrete logarithm problem.
 - o The protocol to verify his claim:



Plug-in Pederson Commitment



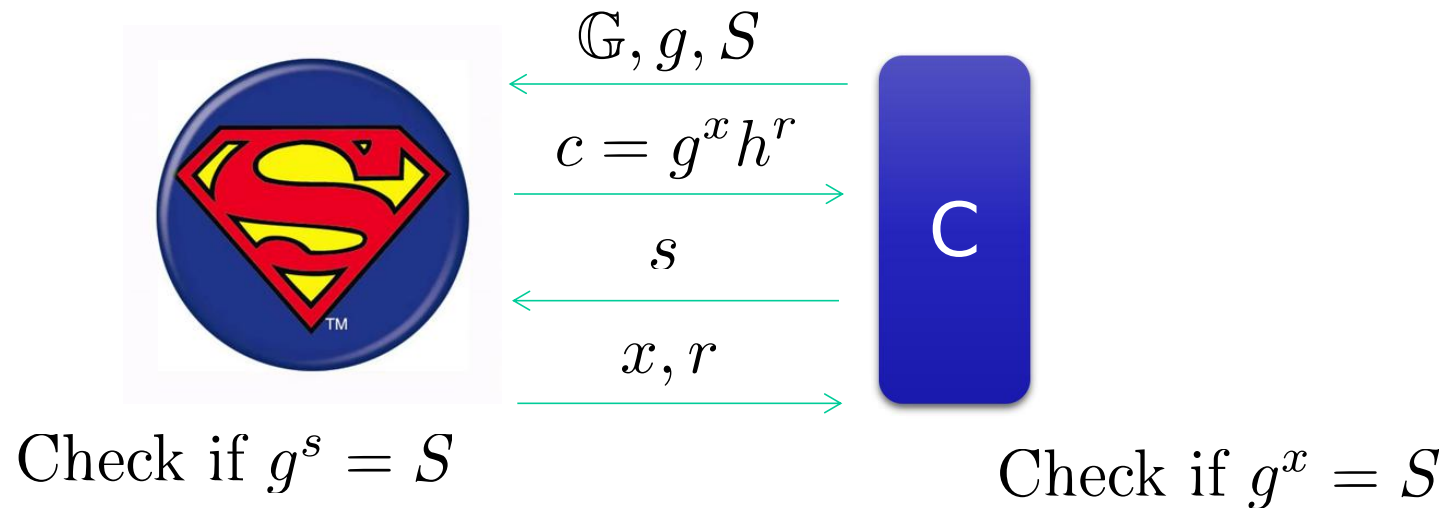
Plug-in Pederson Commitment



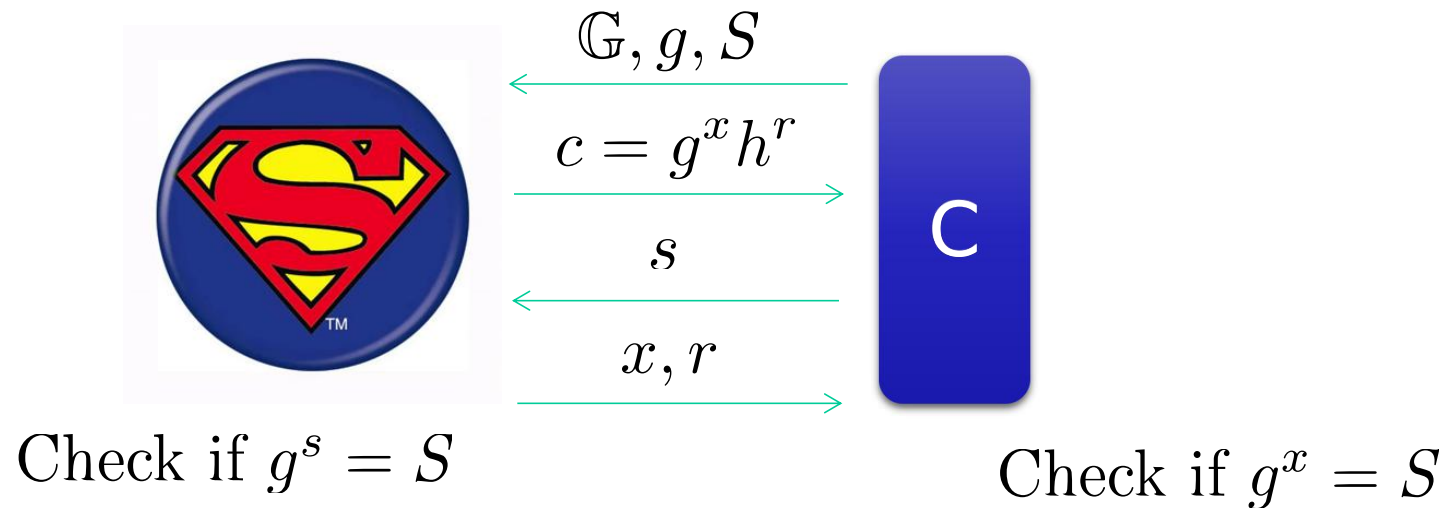
Is this protocol secure?

Compute the commitment as $c = S \cdot h^r$,
and later send $x = s$ and r to open it.

Plug-in Pederson Commitment

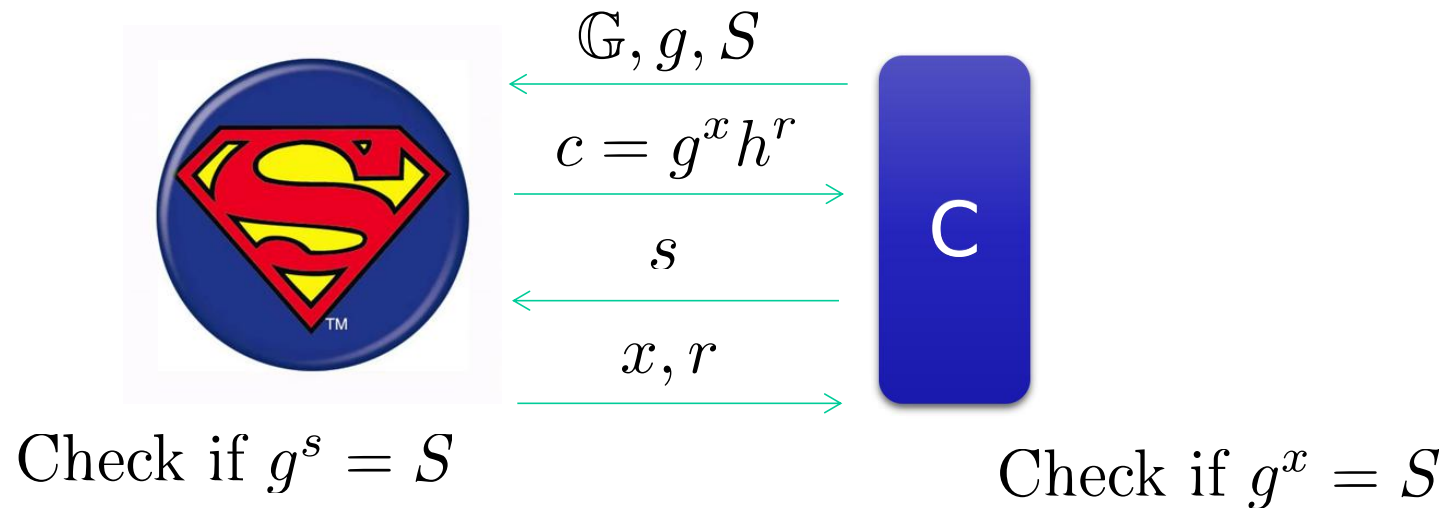


Plug-in Pederson Commitment



Is this protocol secure?

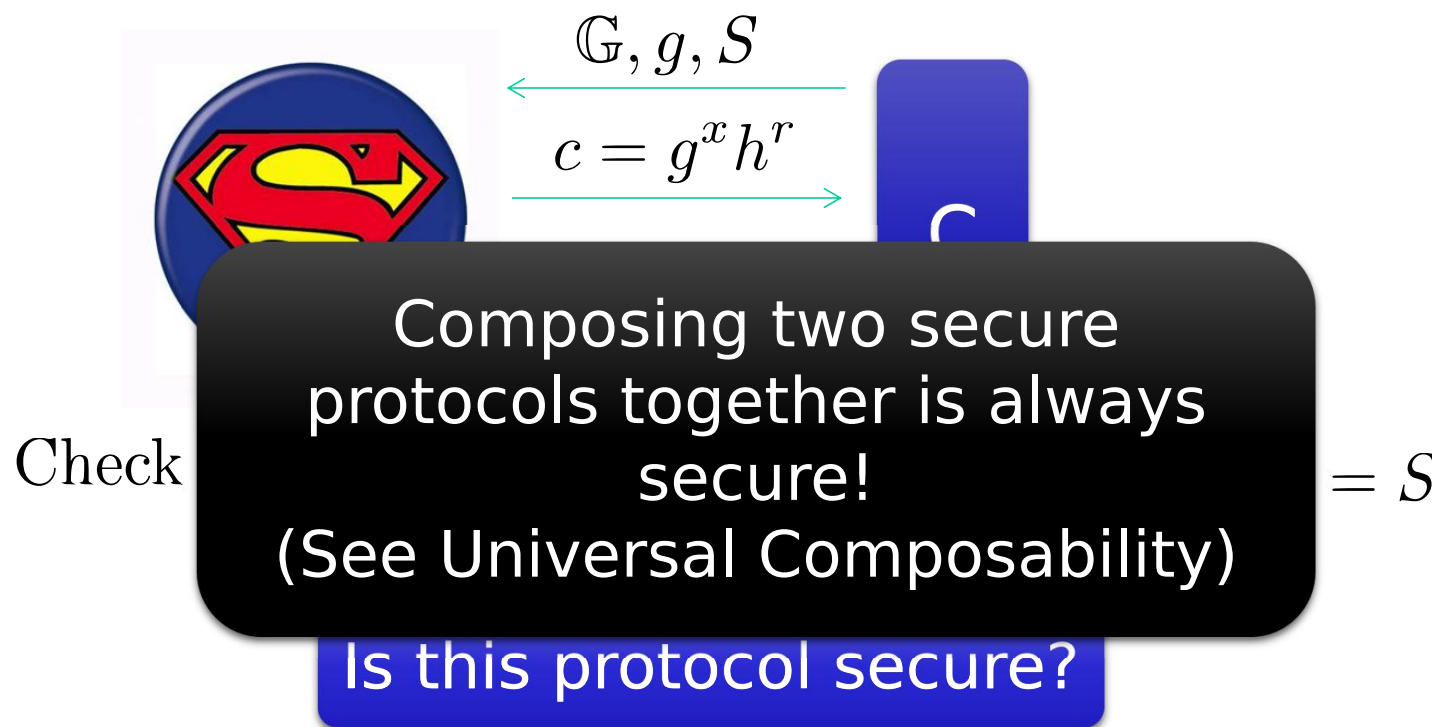
Plug-in Pederson Commitment



Is this protocol secure?

Compute the commitment as $c = S \cdot h^r$,
and later send $x = s$ and r to open it.

Plug-in Pederson Commitment



Compute the commitment as $c = S \cdot h^r$,
and later send $x = s$ and r to open it.

Recommended Reading

So You're Starting a PhD?

Mike Rosulek

<http://web.engr.oregonstate.edu/~rosulekm/advising.html>

Acknowledge

Some materials are extracted from the slides
created by Prof. Bingsheng Zhang, Yehuda Lindell.