Cryptography
CS 555

Week 14:

- Digital Signatures Continued
- Multiparty Computation
- Yao's Garbled Circuits

Readings: Katz and Lindell Chapter 10 & Chapter 11.1-11.2, 11.4

Recap: Identification Scheme

- Interactive protocol that allows one party to prove its identify (authenticate itself) to another
- Two Parties: Prover and Verifier
  - Prover has secret key sk and Verifier has public key pk

1. Prover runs P_1(sk) to obtain (I,st) ---- initial message I, state st
   - Sends I to Verifier
2. Verifier picks random message r from distribution Ω_{pk} and sends r to Prover
3. Prover runs P_2(sk,st,r) to obtain s and sends s to verifier
4. Verifier checks if V(pk,r,s)=I

Recap: Fiat-Shamir Transform

- Identification Schemes can be transformed into signatures
- $\text{Sign}_{sk}(m)$
  - First compute $(I,st)= P_1(sk)$ (as prover)
  - Next compute the challenge $\boldsymbol{r} = \boldsymbol{H}(\boldsymbol{I}, \boldsymbol{m})$ (as verifier)
  - Compute the response $s = P_2(sk,st,r)$
  - Output signature (r,s)
- $\text{Vrfy}_{pk}(m,(r,s))$
  - Compute $I := V(pk,r,s)$
  - Check that $H(I,m)=r$

Theorem 12.10: If the identification scheme is secure and H is a random oracle then the above signature scheme is secure.

Schnorr Identification Scheme

- Verifier knows $h=g^x$
- Prover knows x such that $h=g^x$

1. Prover runs $P_1(x)$ to obtain $\left(k \in \mathbb{Z}_q, I = g^k\right)$ and sends initial message I to verifier
2. Verifier picks random $r \in \mathbb{Z}_q$ (q is order of the group) and sends r to prover
3. Prover runs $P_2(x,k,r)$ to obtain $s \coloneqq [rx + k \ mod \ q]$ and sends s to Verifier
4. Verifier checks if $g^s * (h^{-1})^r = I = g^k$

Schnorr Identification Scheme

- Verifier knows $h=g^x$
- Prover knows x such that $h=g^x$

1. Prover runs $P_1(x)$ to obtain $\left(k \in \mathbb{Z}_q, I = g^k\right)$ and sends initial message I to verifier
2. Verifier picks random $r \in \mathbb{Z}_q$ (q is order of the group) and sends r to prover
3. Prover runs $P_2(x,k,r)$ to obtain $s \coloneqq [rx + k \ mod \ q]$ and sends s to Verifier
4. Verifier checks if $g^s * (h^{-1})^r = I = g^k$

$$g^s * (h^{-1})^r = g^{rx+k \ mod \ q} * g^{-xr} = g^k$$

Schnorr Identification Scheme

- Verifier knows $h=g^x$
- Prover knows x such that $h=g^x$
- Prover runs $P_1(x)$ to obtain $\left(k \in \mathbb{Z}_q, I = g^k\right)$ and sends initial message I to verifier
- Verifier picks random $r \in \mathbb{Z}_q$ (q is order of the group) and sends r to prover
- Prover runs P1(x,k,r) to obtain $s \coloneqq [rx + k \bmod q]$ and sends s to Verifier
- Verifier checks if $g^s * (h^{-1})^r = I = g^k$

Theorem 12.11: If the discrete-logarithm problem is hard (relative to group generator) then Schnorr identification scheme is secure.

Schnorr Signatures via Fiat-Shamir

- Public Key: $h=g^x$ in cyclic group $\langle g \rangle$ of order q.
- Secret Key: x
- $Sign_{sk}(m)$
  1. Select random $k \in \mathbb{Z}_q$ and set $I = g^k$.
  2. ***$r = H(I, m)$***
  3. Return $\sigma = (r, s)$ where $s := [rx + k \; mod \; q]$
- $Verify_{pk}\big(m, \sigma = (r, s)\big)$
  - Compute $g^s * (h^{-1})^r = g^{s-rx}$ and check if $r = H(g^{s-rx}, m)$

Schnorr Signatures

- $Sign_{sk}(m)$
  1. Select random $k \in \mathbb{Z}_q$ and set $I = g^k$.
  2. $\boldsymbol{r = H(I, m)}$
  3. Return $\sigma = (r, s)$ where s $:= [rx + k \ mod \ q]$
- $Verify_{pk}\big(m, \sigma = (r, s)\big)$
  - Compute $g^s * (h^{-1})^r = g^{s-rx}$ and check if r $= H(g^{s-rx}, m)$

Corollary (of Thms 12.10 + 12.11): If the discrete-logarithm problem is hard (relative to group generator) then Schnorr Signatures are secure in the random oracle model.

- Independent of size of original group (r^{th} residue subgroup).
- Independent of #bits to represent group element (Elliptic Curve Pairs)

Depends only on order of the <u>subgroup</u> q!

DLOG 128 bit security: $\lceil \log_2 q \rceil \approx 256$

$= g^k.$

$+ k \ mod \ q$

check if r =

Advantages:

- **Short Signatures** $\|\sigma\| = \|r\| + \|s\| = 2\lceil \log_2 q \rceil$ bits
- Fast and Efficient
- Patent Expired: February 2008

Digital Signature Algorithm (DSA)

DSA: $\langle \boldsymbol{g} \rangle$ is subgroup of $\mathbb{Z}_p^*$ of order q

ECDSA: $\langle \boldsymbol{g} \rangle$ is order q subgroup of elliptic curve

- Secret key is x, public key is $h=g^x$ along with generator g (of order q)
- $\text{Sign}_{sk}(m)$
  - Pick random $\left(k \in \mathbb{Z}_q\right)$ and set $r = F\left(g^k\right) \in \mathbb{Z}_q$
  - Compute $s \coloneqq [k^{-1}(xr + H(m))\ mod\ q]$
  - Output signature (r,s)
- $\text{Vrfy}_{pk}(m,(r,s))$ check to make sure that

$$r = F\left(g^{H(m)s^{-1}} h^{rs^{-1}}\right)$$

Digital Signature Algorithm (DSA)

- $\text{Sign}_{sk}(m)$
  - Pick random $(k \in \mathbb{Z}_q)$ and set $r = F(g^k) = [g^k \mod q]$
  - Compute $s \coloneqq [k^{-1}(xr + H(m)) \mod q]$
  - Output signature (r,s)
- $\text{Vrfy}_{pk}(m,(r,s))$ check to make sure that

$$
\begin{aligned}
r &= F\left(g^{H(m)s^{-1}} h^{rs^{-1}}\right) \\
&= F\left(g^{H(m)k(xr+H(m))^{-1}} g^{xrk(xr+H(m))^{-1}}\right) \\
&= F\left(g^{(H(m)+xr)k(xr+H(m))^{-1}}\right) \\
&= F\left(g^k\right) \coloneqq r
\end{aligned}
$$

Digital Signature Algorithm (DSA)

- Secret key is x, public key is $h=g^x$ along with generator g (of order q)
- $\text{Sign}_{sk}(m)$
  - Pick random $\left(k \in \mathbb{Z}_q\right)$ and set $r = F\left(g^k\right) = [g^k \; mod \; q]$
  - Compute $s \coloneqq [k^{-1}(xr + H(m)) \; mod \; q]$
  - Output signature (r,s)
- Vrfy_{pk}(m,(r,s)) check to make sure that

$$r = F\left(g^{H(m)s^{-1}} h^{rs^{-1}}\right)$$

Theorem: If H and F are modeled as random oracles then DSA is secure.

Weird Assumption for F(.)?

- **Theory:** DSA Still lack compelling proof of security from standard crypto assumptions
- **Practice:** DSA has been used/studied for decades without attacks

Digital Signature Algorithm (DSA)

- Secret key is x, public key is $h=g^x$
- $\text{Sign}_{sk}(m)$
  - Pick random $(k \in \mathbb{Z}_q)$ and set $r = F(g^k) = [g^k \; mod \; q]$
  - Compute $s \coloneqq [k^{-1}(xr + H(m)) \; mod \; q]$
  - Output signature (r,s)
- $\text{Vrfy}_{pk}(m,(r,s))$ check to make sure that

$$r = F\left(g^{H(m)s^{-1}} h^{rs^{-1}}\right)$$

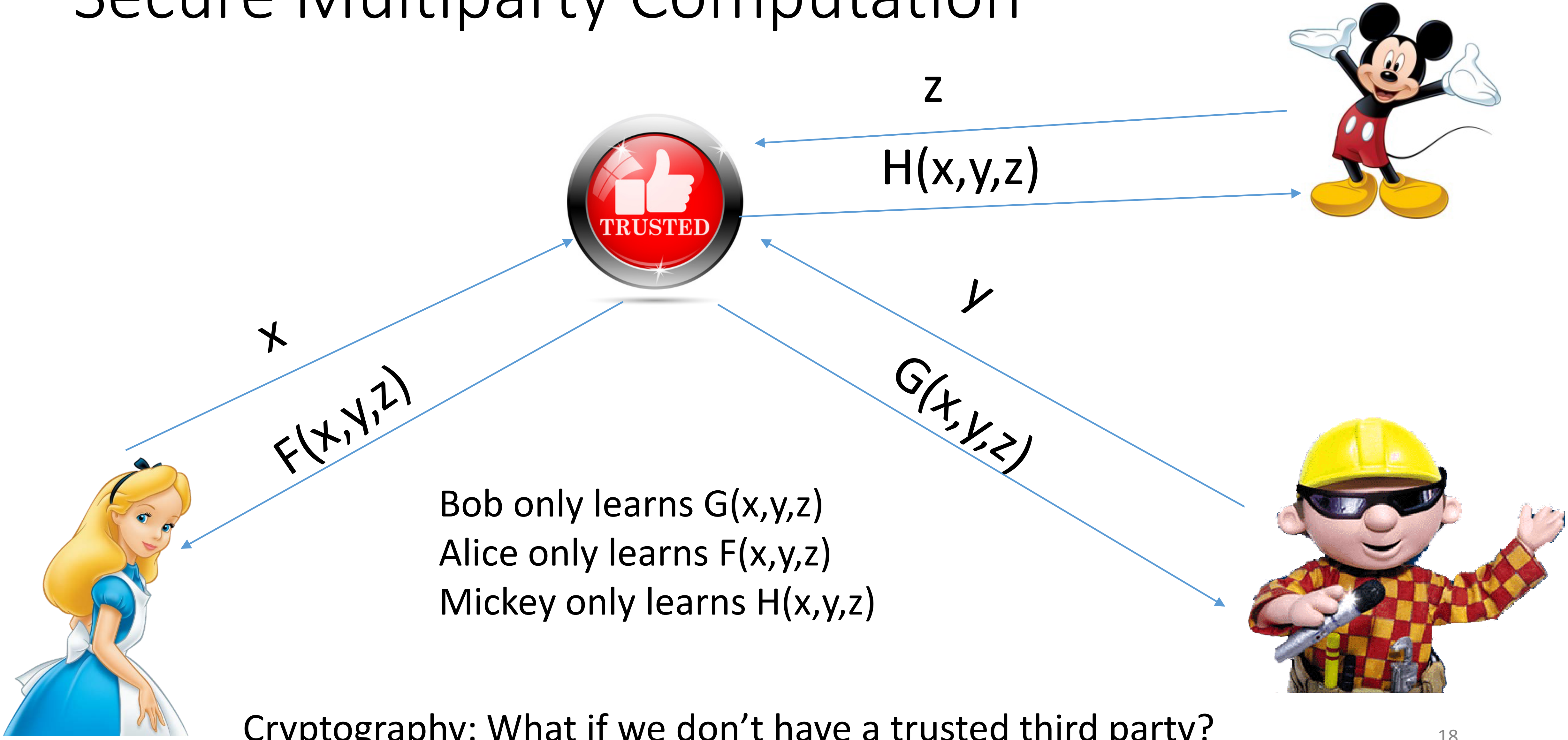
Remark: If signer signs two messages with same random $k \in \mathbb{Z}_q$ then attacker can find secret key sk!

- **Theory:** Negligible Probability this happens
- **Practice:** Will happen if a weak PRG is used
- Sony PlayStation (PS3) hack in 2010.

Certificate Authority

- Trusted Authority (CA)
  - $m_{CA\rightarrow Amazon}$="Amazon's public key is pk_{Amazon} (date,expiration,###)"
  - $cert_{CA\rightarrow Amazon} = Sign_{SK_{CA}}(m)$
- Delegate Authority to other CA_1
  - Root CA signs m= "CA_1 public key is pk_{CA1} (date,expiration,###) can issue certificates"
  - Verifier can check entire certification chain
- Revocation List Signed Daily
- Decentralized Web of Trust (PGP)

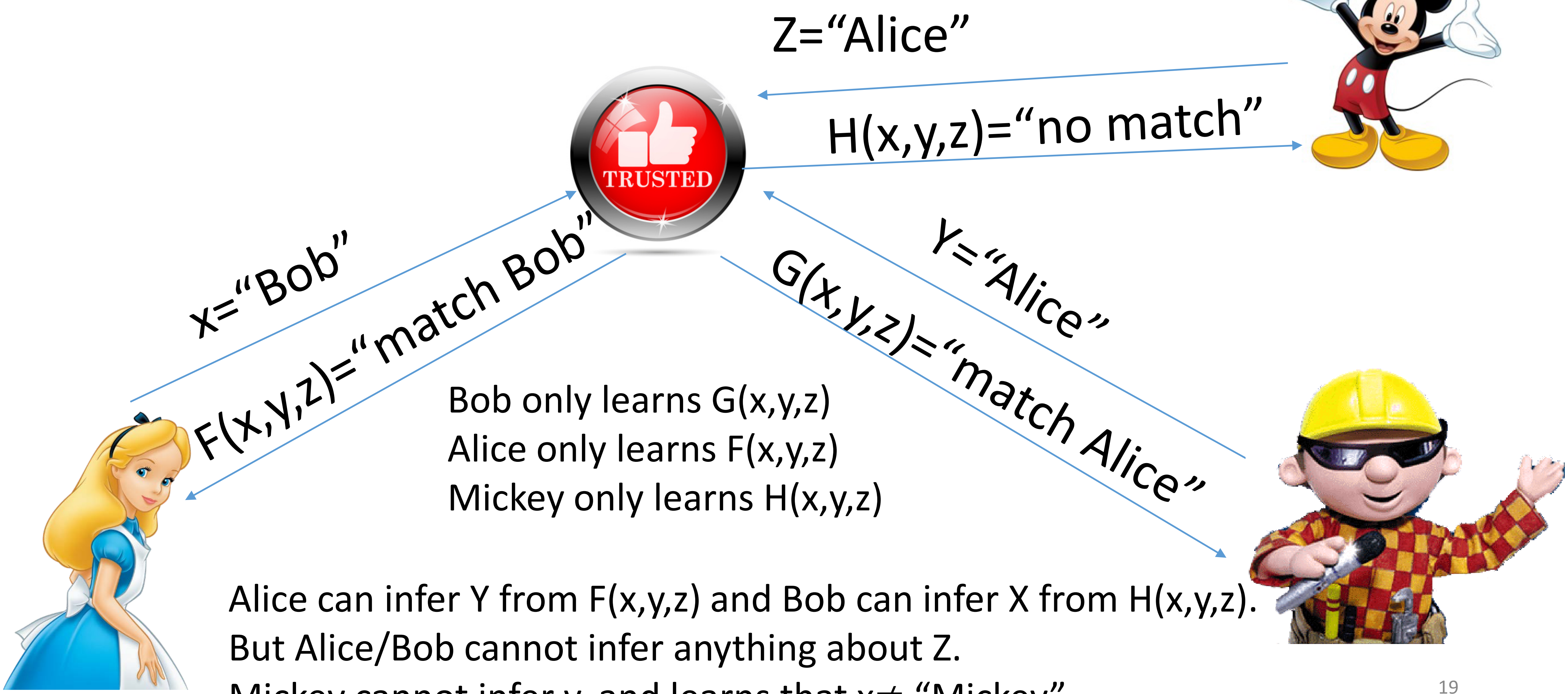
Secure Multiparty Computation



Bob only learns G(x,y,z)
Alice only learns F(x,y,z)
Mickey only learns H(x,y,z)

Cryptography: What if we don't have a trusted third party?

Secure Multiparty Computation (Crushes)



Bob only learns G(x,y,z)
Alice only learns F(x,y,z)
Mickey only learns H(x,y,z)

Alice can infer Y from F(x,y,z) and Bob can infer X from H(x,y,z).
But Alice/Bob cannot infer anything about Z.
Mickey cannot infer y, and learns that x≠ "Mickey"

Secure Multiparty Computation (Crushes)

Key Point: The output H(x,y,z) may leak info about inputs. Thus, we cannot prevent Mickey from learning anything about x,y but Mickey should not learn anything else besides H(x,y,z)!

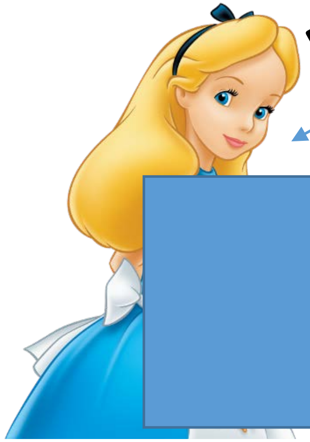
x="Bob"

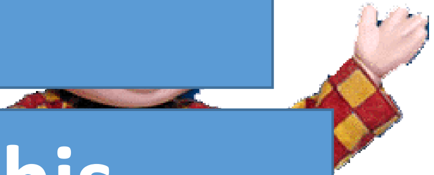
F(x,y,z)="match Bo

Bob o

Alice c

Micke





Though Question: How can we formalize this property?

Mickey cannot infer y, and learns that x≠ "Mickey"

Adversary Models

- Semi-Honest (“honest, but curious”)
  - All parties follow protocol instructions, but…
  - dishonest parties may be curious to violate privacy of others when possible
- Fully Malicious Model
  - Adversarial Parties may deviate from the protocol arbitrarily
    - Quit unexpectedly
    - Send different messages
  - It is much harder to achieve security in the fully malicious model
- Convert Secure Semi-Honest Protocol into Secure Protocol in Fully Malicious Mode?
  - Tool: Zero-Knowledge Proofs

Computational Indistinguishability

Definition: We say that an ensemble of distributions $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are computationally indistinguishable if for all PPT distinguishers D, there is a negligible function negl(n), such that we have

$$Adv_{D,n} = \left| Pr_{s\leftarrow X_\ell}[D(s)=1] - Pr_{s\leftarrow Y_\ell}[D(s)=1] \right| \leq negl(n)$$

Notation: $\{X_n\}_{n\in\mathbb{N}} \equiv_C \{Y_n\}_{n\in\mathbb{N}}$ means that the ensembles are computationally indistinguishable.

Security (Semi-Honest Model)

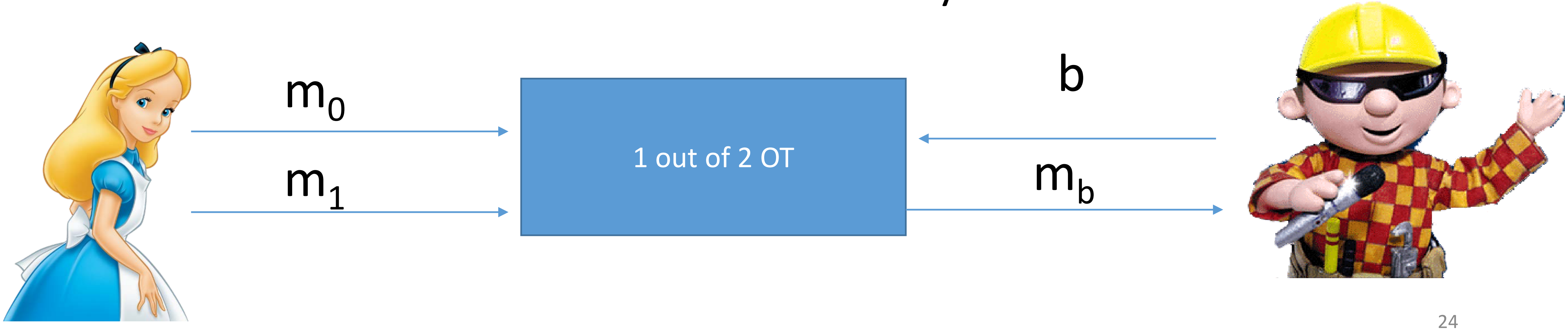
- Let $B_n = trans_B(n, x, y)$ (resp. $A_n = trans_A(n, x, y)$) be the protocol transcript from Bob's perspective (resp. Alice's perspective) when his input is y and Alice's input is x (assuming that Alice follows the protocol).
- **Security**: Assuming that Alice and Bob are both semi-honest (follow the protocol) then there exist PPT simulators S_A and S_B s.t.

$$\{A_n\}_{n\in\mathbb{N}} \equiv_C \{S_A(n, x, f_A(x, y)\ )\}_{n\in\mathbb{N}}$$
$$\{B_n\}_{n\in\mathbb{N}} \equiv_C \left\{S_B\left(n, y, f_B(x, y)\right)\right\}_{n\in\mathbb{N}}$$

- **Remark**: Simulator S_A is only shown Alice's input y and Alice's output $f_A(x, y)$ (similarly, S_B is only shown Bob's input x and Bob's output $f_B(x, y)$)

Building Block: Oblivious Transfer (OT)

- 1 out of 2 OT
  - Alice has two messages m_0 and m_1
  - At the end of the protocol
    - Bob gets exactly one of m_0 and m_1
    - Alice does not know which one, and Bob learns nothing about other message
- Oblivious Transfer with a Trusted Third Party

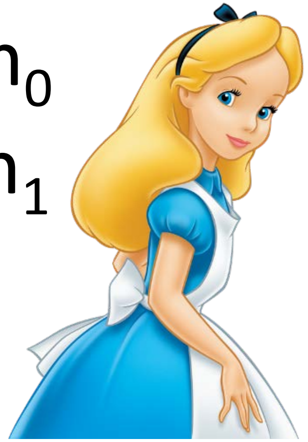


Bellare-Micali 1-out-of-2-OT protocol

- Oblivious Transfer without a Trusted Third Party
  - g is a generator for a prime order group G_q in which CDH problem is hard

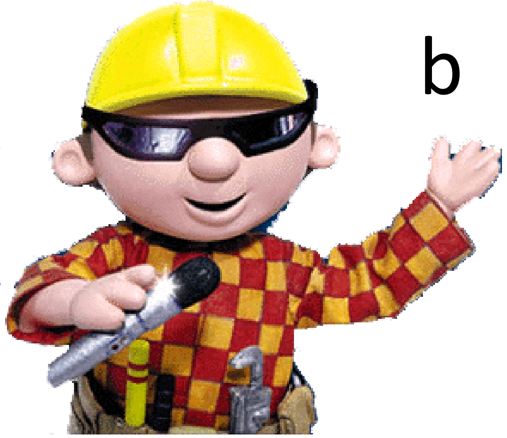
m_0

m_1



$c \leftarrow_R G_q$

b



$k \leftarrow_R Z_q$

$z_b = g^k, z_{1-b} = cg^{-k}$

$\xrightarrow{\quad c \quad}$

$\xleftarrow{\quad z_0, z_1 \quad}$

$$C_0 = [g^{r_0}, H(z_0^{r_0}) \oplus m_0]$$
$$C_1 = [g^{r_1}, H(z_1^{r_1}) \oplus m_1]$$

Bob can decrypt C_b

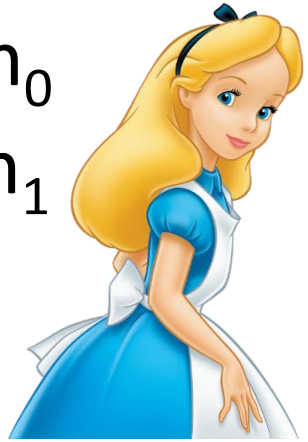
$$z_b^{r_b} = g^{kr_b}$$

Bellare-Micali 1-out-of-2-OT protocol

- Oblivious Transfer without a Trusted Third Party
  - g is a generator for a prime order group G_q in which CDH is Hard

m_0

m_1



$c \leftarrow_R G_q$

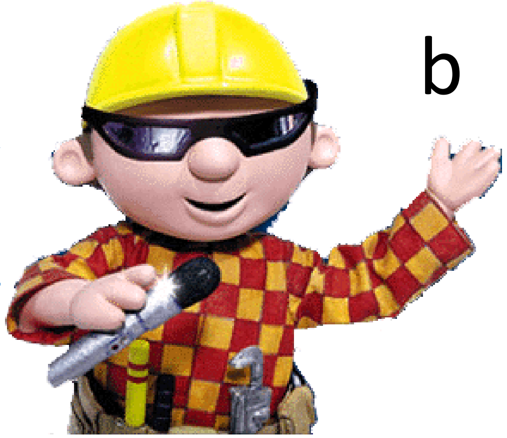
c

z_0, z_1

$C_0 = [g^{r_0}, H(z_0^{r_0}) \oplus m_0]$

$C_1 = [g^{r_1}, H(z_1^{r_1}) \oplus m_1]$

b



$k \leftarrow_R Z_q$

$z_b = g^k, z_{1-b} = cg^{-k}$

$= c(z_b)^{-1}$

Alice must check that

$z_1 = c(z_0)^{-1}$

Bob can decrypt C_b

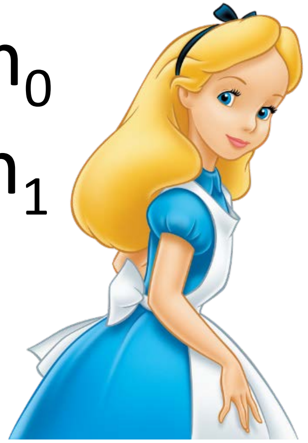
$z_b^{r_b} = g^{kr_b}$

Bellare-Micali 1-out-of-2-OT protocol

- Oblivious Transfer withou
  - g is a generator for a prime

m_0

m_1



$c \leftarrow_R G_q$

$C_0 =$

$C_1 =$

Alice does not learn b because

- $z_1 = c(z_0)^{-1}$ and
- $z_0 = c(z_1)^{-1}$ and
- z_1, z_0 are distributed uniformly at random subject to these condition.

This is an information theoretic guarantee!

$z_b = g^k, z_{1-b} = cg^{-k}$
$= c(z_b)^{-1}$

Alice must check that

$z_1 = c(z_0)^{-1}$

Bob can decrypt C_b

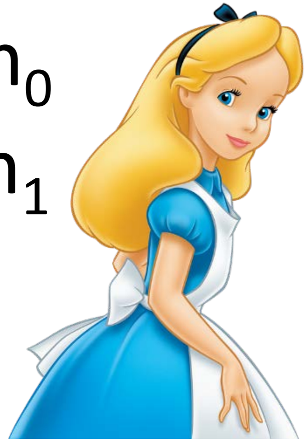
$z_b^{r_b} = g^{kr_b}$

Bellare-Micali 1-out-of-2-OT protocol

- Oblivious Transfer withou
  - g is a generator for a prime

m_0

m_1



$c \leftarrow_R G_q$

$C_0 =$

$C_1 =$

Bob cannot decrypt $C_{\mathbf{1-b}}$

Unless he queries random oracle at

- $c^{r_{1-b}} g^{-kr_{1-b}}$
- Given this value we can obtain $c^{r_{1-b}}$
- Thus, we can break CDH assumption

given random $c = g^m$ and $g^{r_{1-b}}$ it is hard to find $c^{r_{1-b}} = g^{mr_{1-b}}$

Alice must check that

$z_1 = c(z_0)^{-1}$

Bob can decrypt $C_{\mathbf{b}}$

$z_b^{r_b} = g^{kr_b}$

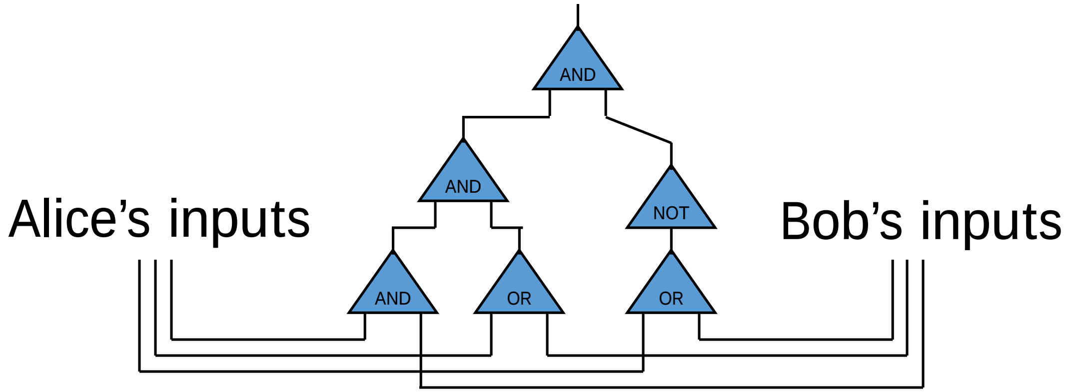
$z_b = g^k, z_{1-b} = cg^{-k}$
$= c(z_b)^{-1}$

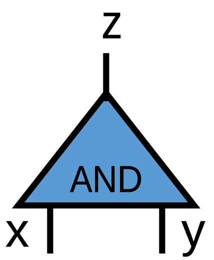
Yao’s Protocol

Vitaly Shmatikov

Yao's Protocol

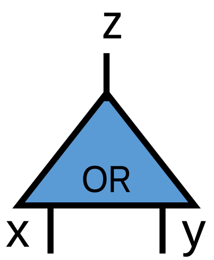
- Compute any function securely
  - … in the semi-honest model
- First, convert the function into a boolean circuit





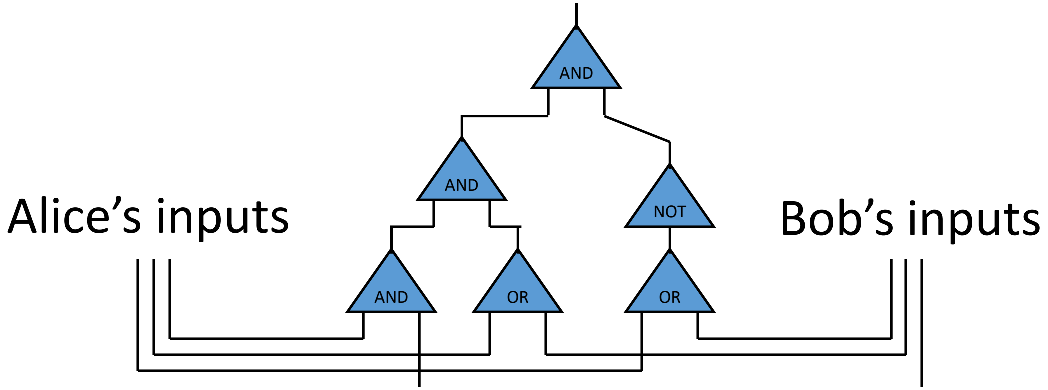
Truth table:

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1



Truth table:

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1



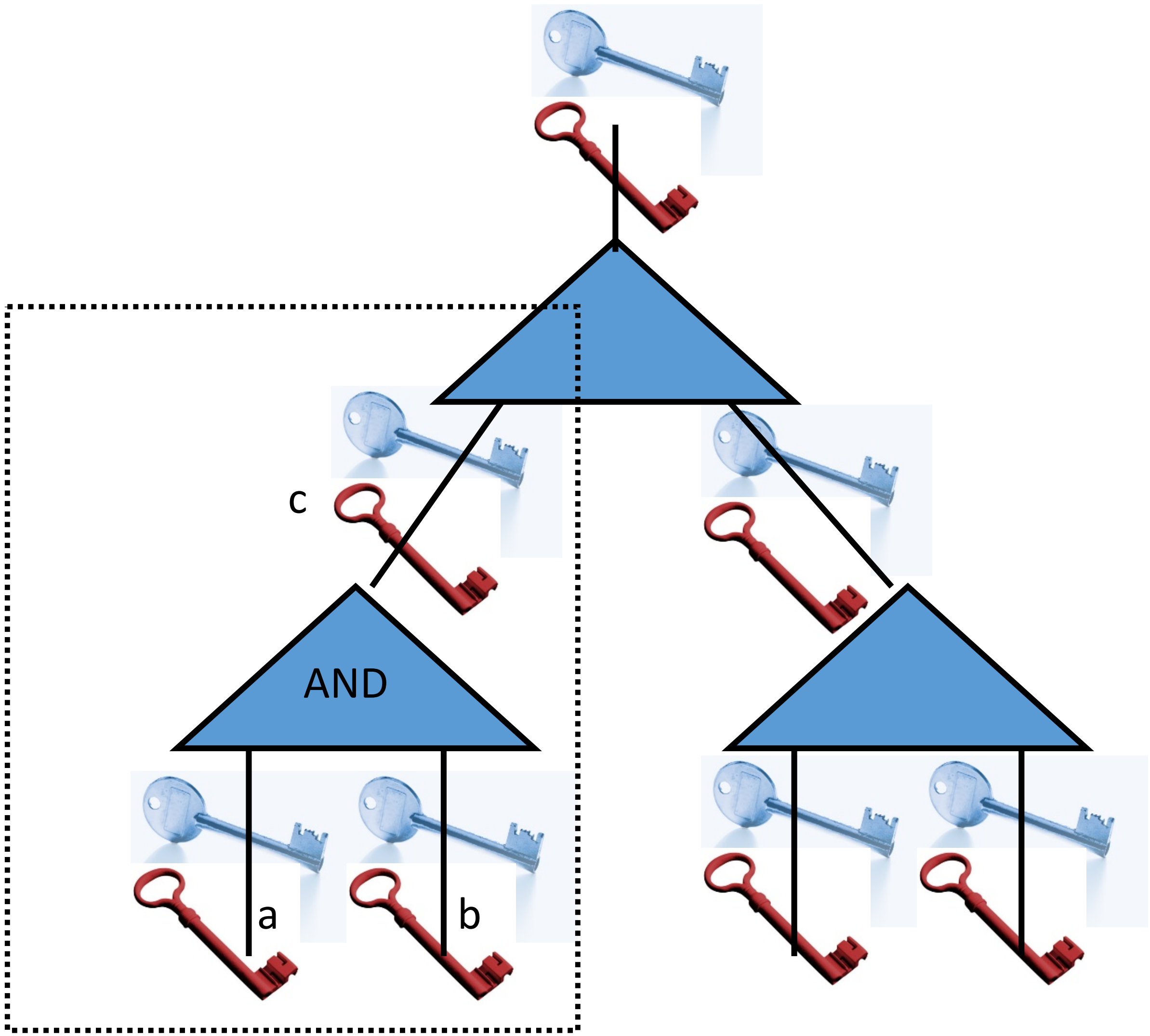
Overview:

1. Alice prepares "garbled" version **C'** of C
2. Sends "encrypted" form **x'** of her input x
3. Allows Bob to obtain "encrypted" form **y'** of his input y via OT
4. Bob can compute from **C'**,**x'**,**y'** the "encryption" **z'** of z=C(x,y)
5. Bob sends **z'** to Alice and she decrypts and reveals to him z

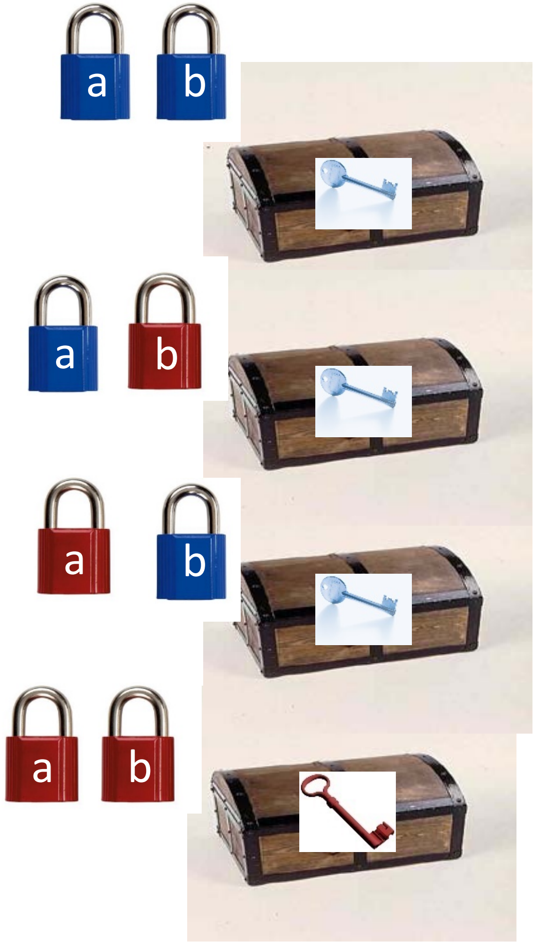
Crucial properties:

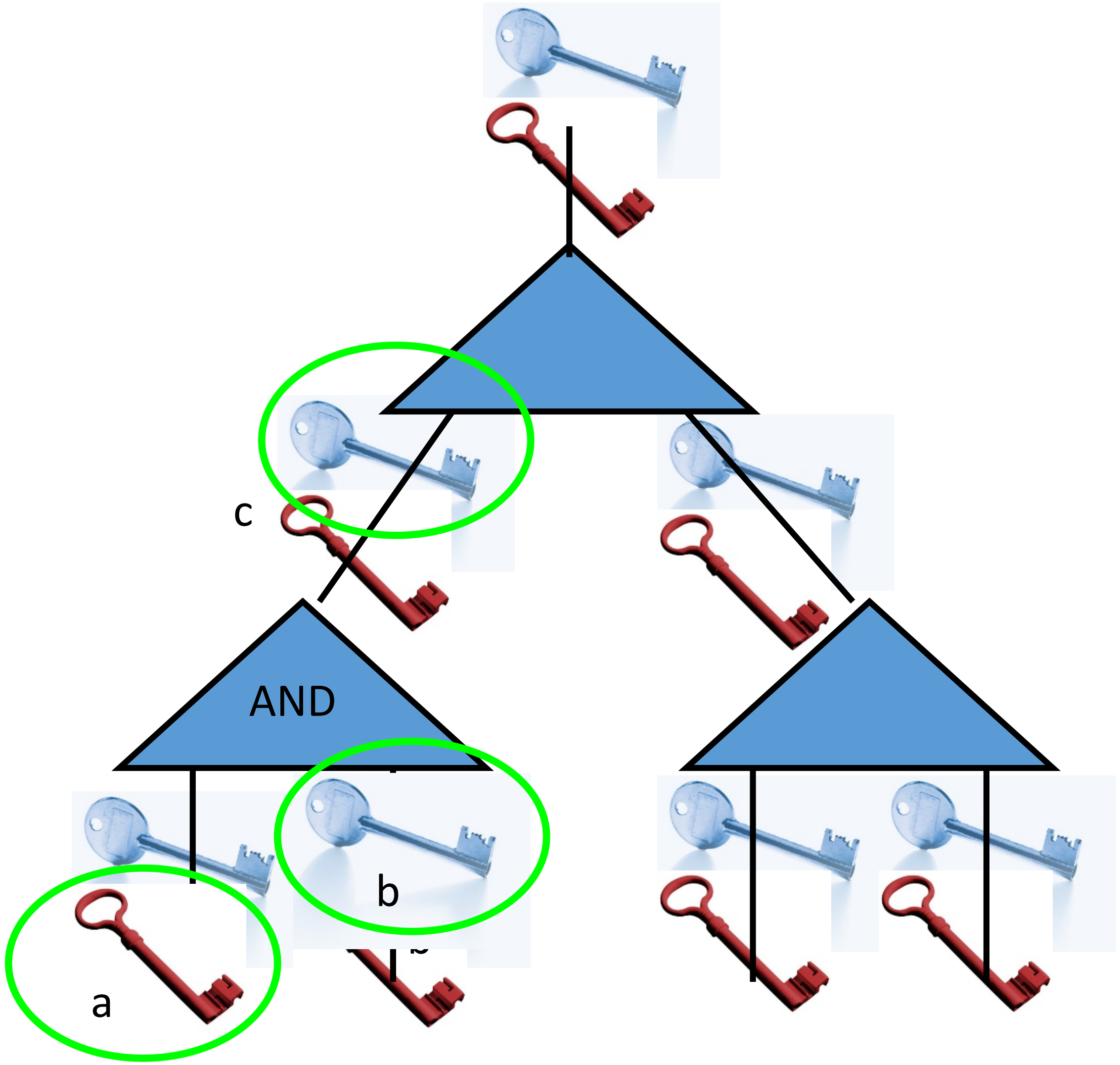
1. Bob never sees Alice's input x in unencrypted form.
2. Bob can obtain encryption of y without Alice learning y.
3. Neither party learns intermediate values.
4. Remains secure even if parties try to cheat.

Intuition



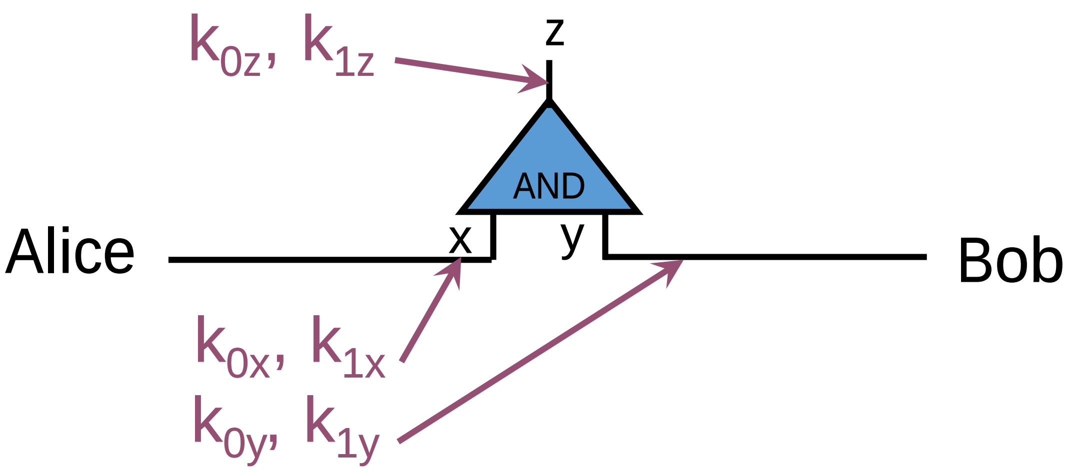
Intuition





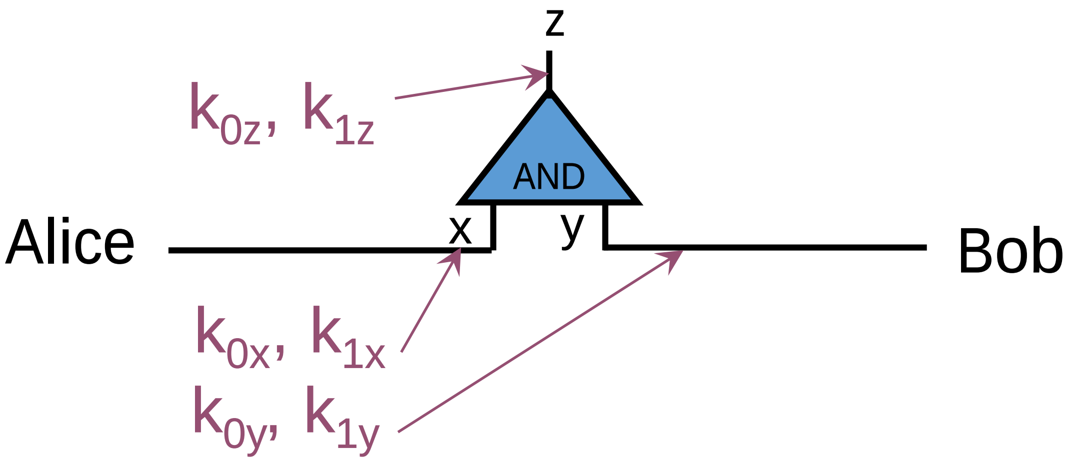
1: Pick Random Keys For Each Wire

- Next, evaluate one gate securely
  - Later, generalize to the entire circuit
- Alice picks two random keys for each wire
  - One key corresponds to “0”, the other to “1”
  - 6 keys in total for a gate with 2 input wires



2: Encrypt Truth Table

- Alice encrypts each row of the truth table by encrypting the output-wire key with the corresponding pair of input-wire keys



Original truth table:

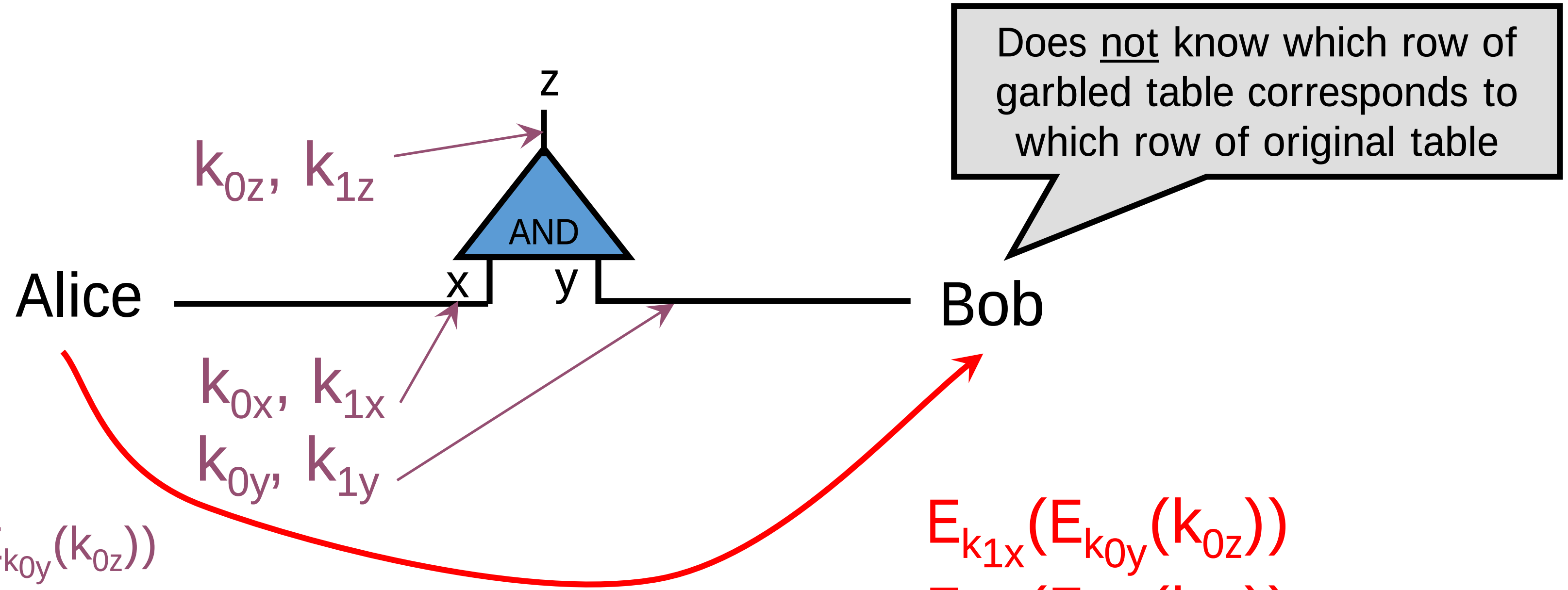
x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

Encrypted truth table:

$$E_{k_{0x}}(E_{k_{0y}}(k_{0z}))$$
$$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$$
$$E_{k_{1x}}(E_{k_{0y}}(k_{0z}))$$
$$E_{k_{1x}}(E_{k_{1y}}(k_{1z}))$$

3: Send Garbled Truth Table

- Alice randomly permutes (“garbles”) encrypted truth table and sends it to Bob



$E_{k_{0x}}(E_{k_{0y}}(k_{0z}))$

$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$

$E_{k_{1x}}(E_{k_{0y}}(k_{0z}))$

$E_{k_{1x}}(E_{k_{1y}}(k_{1z}))$

Garbled truth table:

$E_{k_{1x}}(E_{k_{0y}}(k_{0z}))$

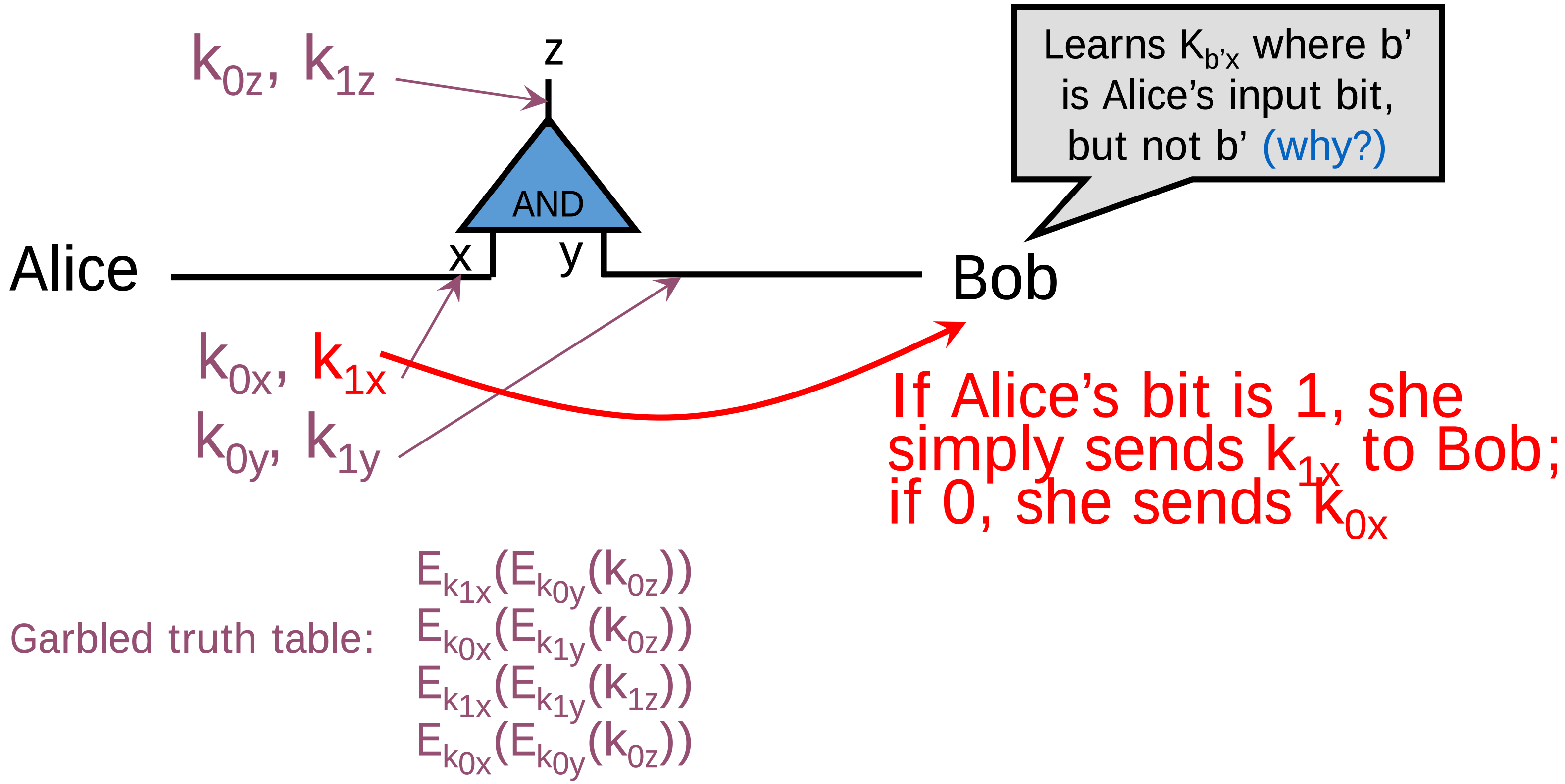
$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$

$E_{k_{1x}}(E_{k_{1y}}(k_{1z}))$

$E_{k_{0x}}(E_{k_{0y}}(k_{0z}))$

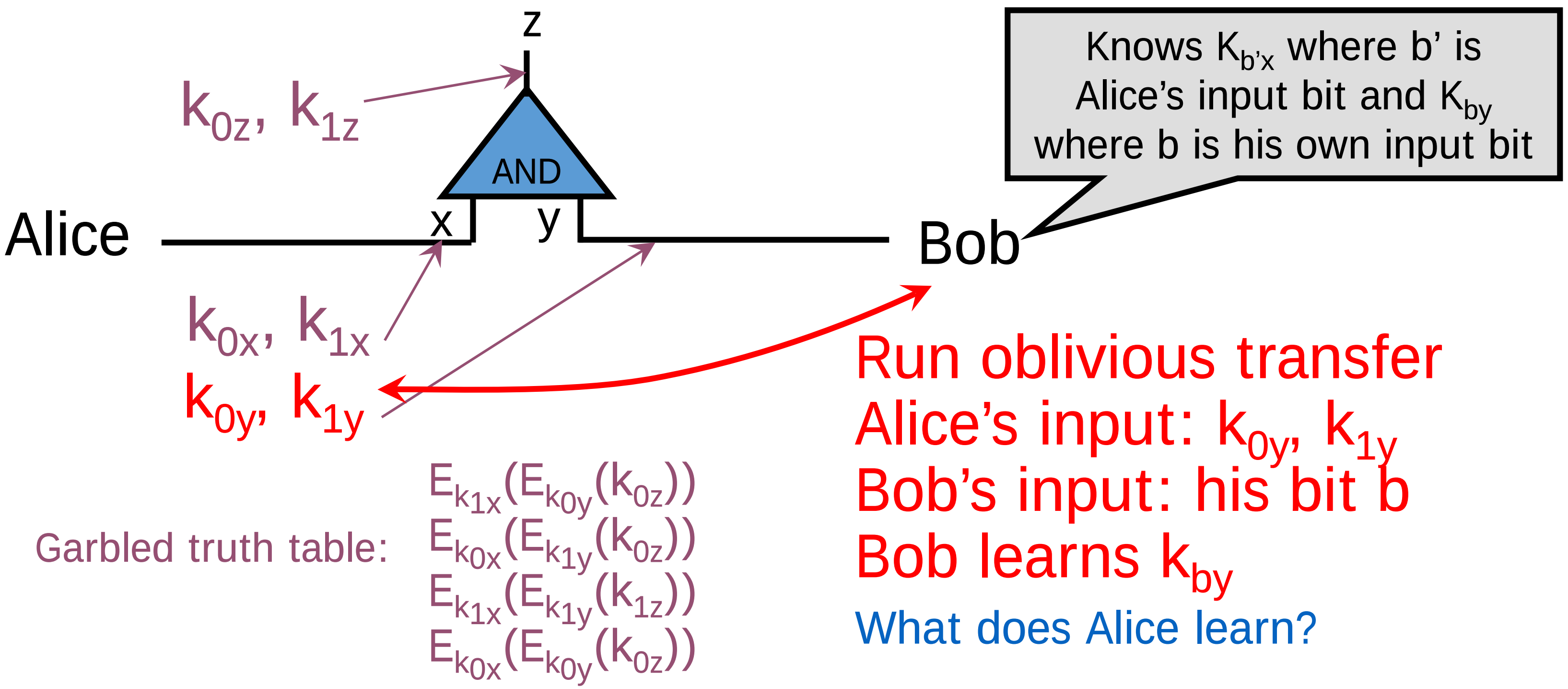
4: Send Keys For Alice's Inputs

- Alice sends the key corresponding to her input bit
  - Keys are random, so Bob does not learn what this bit is



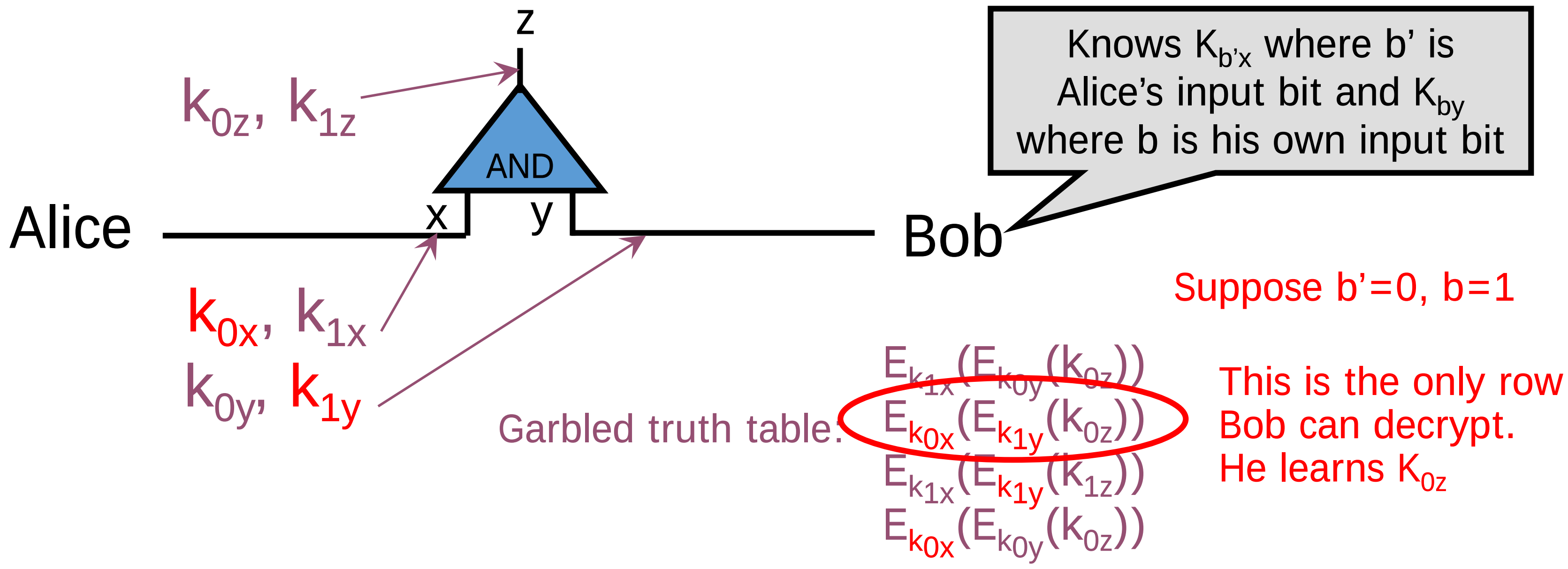
5: Use OT on Keys for Bob’s Input

- Alice and Bob run oblivious transfer protocol
  - Alice’s input is the two keys corresponding to Bob’s wire
  - Bob’s input into OT is simply his 1-bit input on that wire



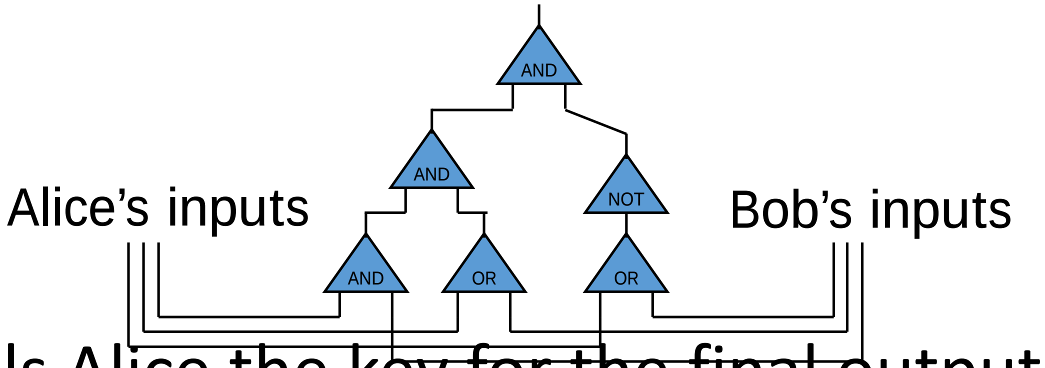
6: Evaluate Garbled Gate

- Using the two keys that he learned, Bob decrypts exactly one of the output-wire keys
  - Bob does not learn if this key corresponds to 0 or 1
    - Why is this important?



7: Evaluate Entire Circuit

- In this way, Bob evaluates entire garbled circuit
  - For each wire in the circuit, Bob learns only one key
  - It corresponds to 0 or 1 (Bob does not know which)
    - Therefore, Bob does not learn intermediate values (why?)



- Bob tells Alice the key for the final output wire and she tells him if it corresponds to 0 or 1
  - Bob does not tell her intermediate wire keys (why?)

Security (Semi-Honest Model)

- **Security**: Assuming that Alice and Bob are both semi-honest (follow the protocol) then there exist PPT simulators S_A and S_B s.t.

$$\{A_n\}_{n\in\mathbb{N}} \equiv_C \{S_A(n, x, f_A(x, y)\ )\}_{n\in\mathbb{N}}$$
$$\{B_n\}_{n\in\mathbb{N}} \equiv_C \left\{S_B\left(n, y, f_B(x, y)\right)\right\}_{n\in\mathbb{N}}$$

- **Remark**: Simulator S_A is only shown Alice's output $f_A(x, y)$ (similarly, S_B is only shown Bob's output $f_B(x, y)$)

Theorem (informal): If the oblivious transfer protocol is secure, and the underlying encryption scheme is CPA-secure then Yao's protocol is secure in the semi-honest adversary model.

Brief Discussion of Yao's Protocol

- Function must be converted into a circuit
  - For many functions, circuit will be huge
- If m gates in the circuit and n inputs from Bob, then need 4m encryptions and n oblivious transfers
  - Oblivious transfers for all inputs can be done in parallel
- Yao's construction gives a constant-round protocol for secure computation of any function in the semi-honest model
  - Number of rounds does not depend on the number of inputs or the size of the circuit!

Fully Malicious Security?

1. Alice could initially garble the wrong circuit C(x,y)=y.
2. Given output of C(x,y) Alice can still send Bob the output f(x,y).
3. Can Bob detect/prevent this?

Fix: Assume Alice and Bob have both committed to their input: c_A=com(x,r_A) and c_B=com(y,r_B).

- Alice and Bob can use zero-knowledge proofs to convince other party that they are behaving honestly.
- **Example**: After sending a message A Alice proves that the message she just sent is the same message an honest party would have sent with input x s.t. c_A=com(x,r_A)
- Here we assume that Alice and Bob have both committed to correct inputs (Bob might use y which does not represent his real vote etc… but this is not a problem we can address with cryptography)

Fully Malicious Security

- Assume Alice and Bob have both committed to their input: c_A=com(x,r_A) and c_B=com(y,r_B).
  - Here we assume that Alice and Bob have both committed to correct inputs (Bob might use y which does not represent his real vote etc… but this is not a problem we can address with cryptography)
  - Alice has c_B and can unlock c_A
  - Bob has c_A and can unlock c_B

1. Alice sets C_f = GarbleCircuit(f,r).
   1. Alice sends to Bob.
   2. Alice convinces Bob that C_f = GarbleCircuit(f,r) for some r (using a zero-knowledge proof)
2. For each original oblivious transfer if Alice's inputs were originally x_0,x_1
   1. Alice and Bob run OT with y_0,y_1 where y_i=Enc_K(x_i)
   2. Bob uses a zero-knowledge proof to convince Alice that he received the correct y_i (e.g. matching his previous commitment c_B)
   3. Alice sends K to Bob who decrypts y_i to obtain x_i