Course Feedback

Course Summary					
Course Code	Course Title	Survey Start Date	Survey End Date	Report Access Start	Response Rate
wl.202120.CS.55500.FNY.18101	Cryptography	4/19/2021 9:00 AM	5/2/2021 11:59 PM	5/12/2021 12:00 AM	0.00% (0/8)

- Your feedback is valuable to me! What did you like about the course? What could be improved? Let me know! I carefully read through any comments after the semester is over.
- Your feedback is anonymous and will not impact your grade (I cannot view your feedback until after grades are entered).

Cryptography
CS 555

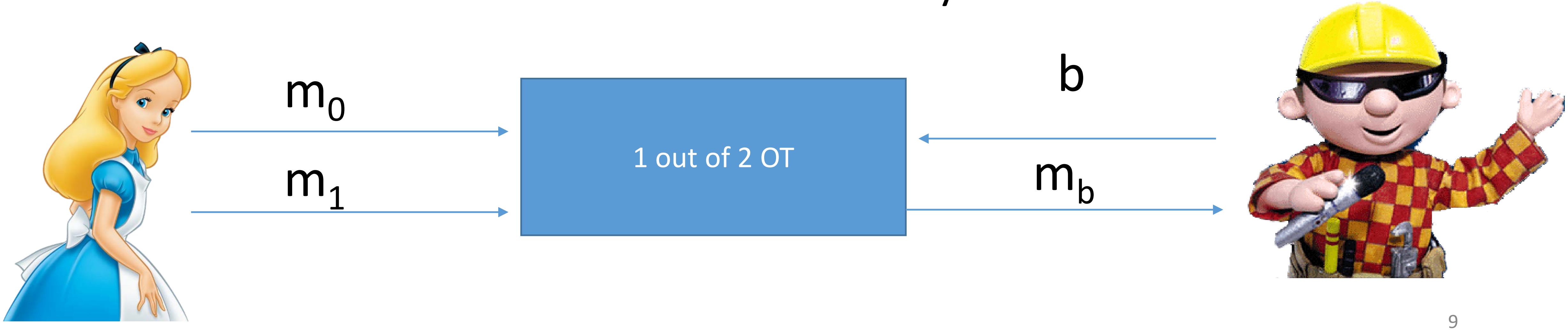
Week 14:

- Multiparty Computation
- Yao's Garbled Circuits
- Zero-Knowledge Proofs
- Shamir Secret Sharing

Homework 5 due April 29th at 11:59 PM on Gradescope

Recap: 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



Yao's Garbled Circuits

- Alice and Bob want to compute a function $f(a_1, \dots, a_m, b_1, \dots, b_n)$
- Alice does not want to reveal her secret inputs $a_1, \dots, a_m$
- Bob does not want to reveal his secret inputs $b_1, \dots, b_n$
- Assume that Alice/Bob are semi-honest (honest, but curious)
  - They will faithfully follow the Garbled Circuit Protocol, but afterwards they are curious to learn about the other's secret inputs
  - Alice/Bob should learn nothing additional except for $f(a_1, \dots, a_m, b_1, \dots, b_n)$
- MPC Security formalized by simulator
  - Alice's Transcript: All of the messages she sends/receives as part of the protocol
  - Simulator Inputs: $a_1, \dots, a_m$ and $f(a_1, \dots, a_m, b_1, \dots, b_n)$
  - Simulator S_A is not given Bob's input
  - Outputs transcript which is computationally indistinguishable from Alice's real transcript
  - Conclusion: Alice learns nothing aside from $a_1, \dots, a_m$ and $f(a_1, \dots, a_m, b_1, \dots, b_n)$

Yao's Garbled Circuits

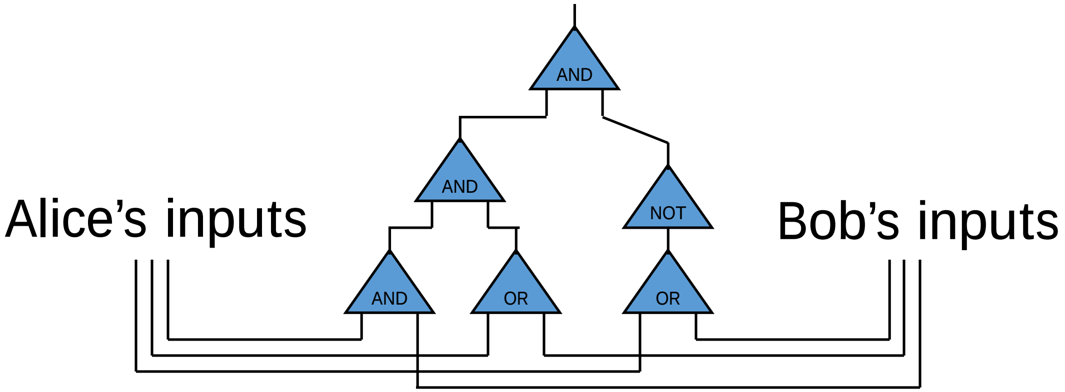
- Alice and Bob want to compute a function $f(a_1, \dots, a_m, b_1, \dots, b_n)$
- Alice does not want to reveal her secret inputs $a_1, \dots, a_m$
- Bob does not want to reveal his secret inputs $b_1, \dots, b_n$
- Assume that Alice/Bob are semi-honest (honest, but curious)
  - They will faithfully follow the Garbled Circuit Protocol, but afterwards they are curious to learn about the other's secret inputs
  - Alice/Bob should learn nothing additional except for $f(a_1, \dots, a_m, b_1, \dots, b_n)$
- MPC Security formalized by simulator
  - Bob's Transcript: All of the messages she sends/receives as part of the protocol
  - Simulator Inputs: $b_1, \dots, b_n$ and $f(a_1, \dots, a_m, b_1, \dots, b_n)$
  - Simulator S_B is not given Alice's input
  - Outputs transcript which is computationally indistinguishable from Bob's real transcript
  - Conclusion: Bob learns nothing aside from $b_1, \dots, b_n$ and $f(a_1, \dots, a_m, b_1, \dots, b_n)$

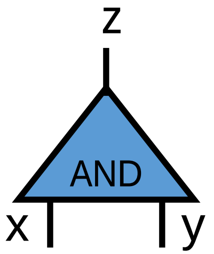
Yao's Protocol

Vitaly Shmatikov

Yao's Protocol

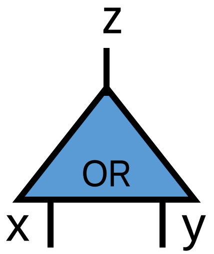
- Compute any function securely
  - … in the semi-honest model
- First, convert the function into a boolean circuit





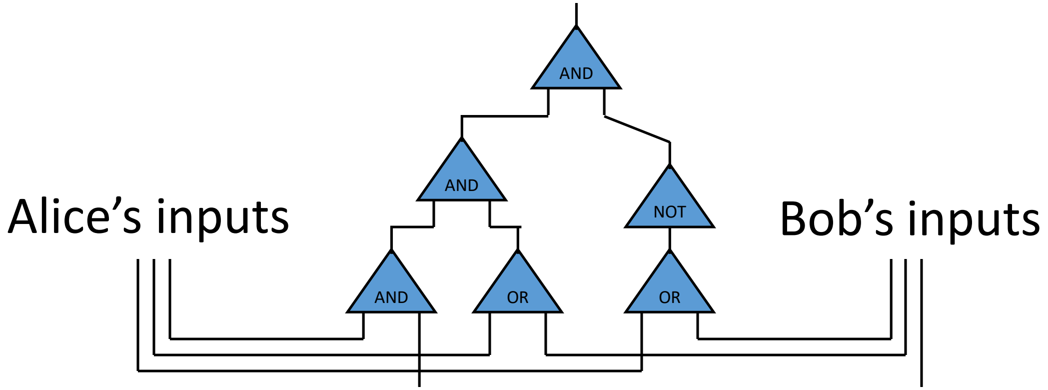
Truth table:

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



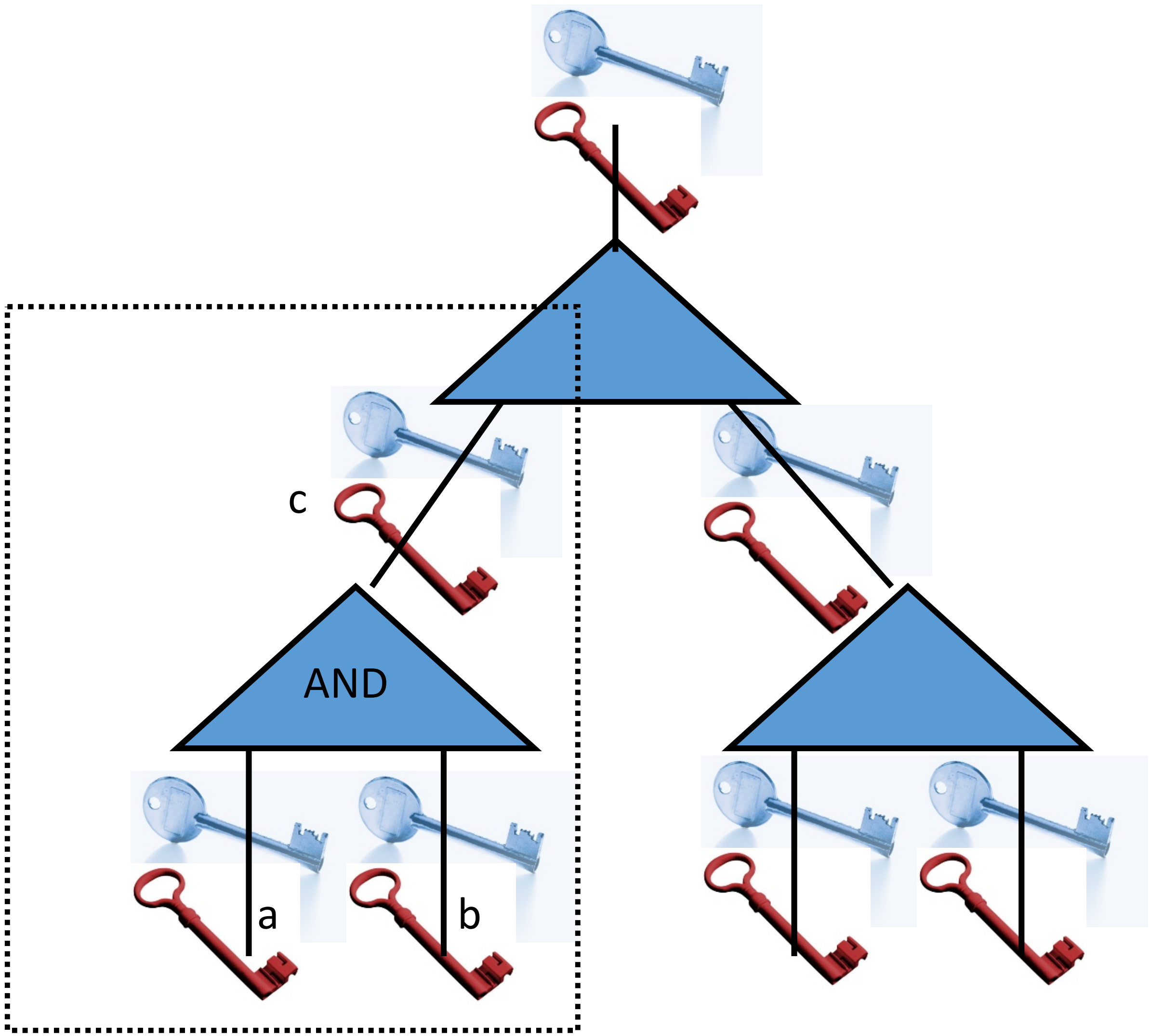
Overview:

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

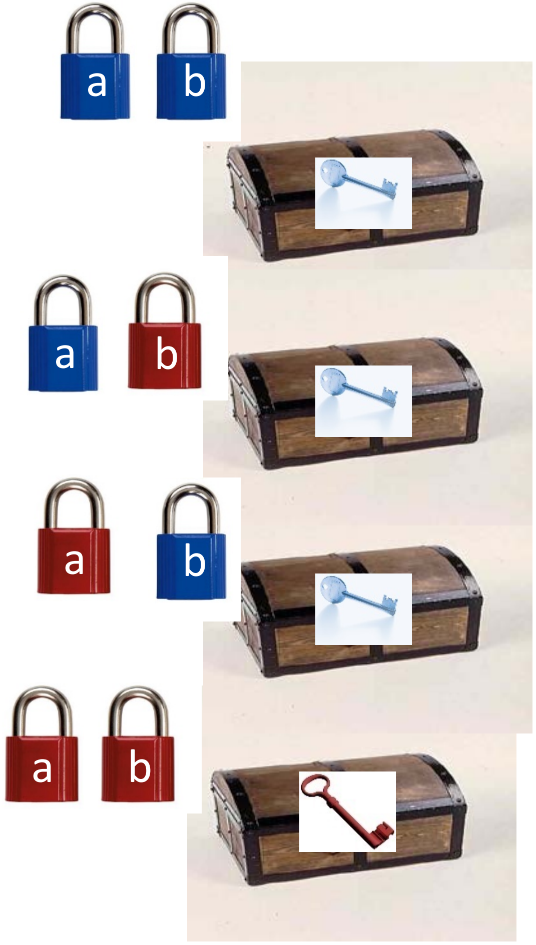
Crucial properties:

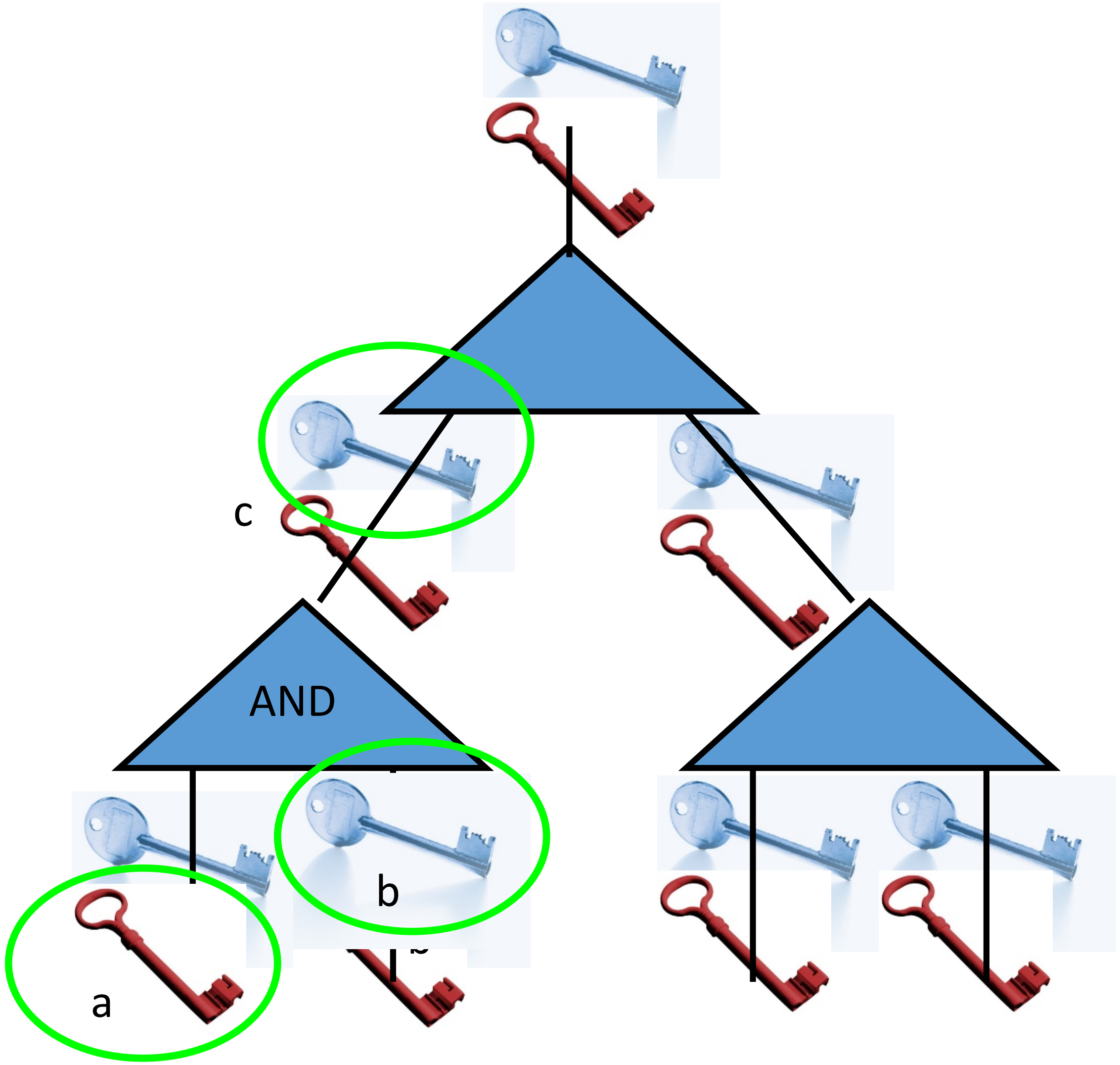
1. Bob never sees Alice's input x in unencrypted form.
2. Bob can obtain encryption of y without Alice learning y.
3. Neither party learns intermediate values.
4. Remains secure even if parties try to cheat.

Intuition



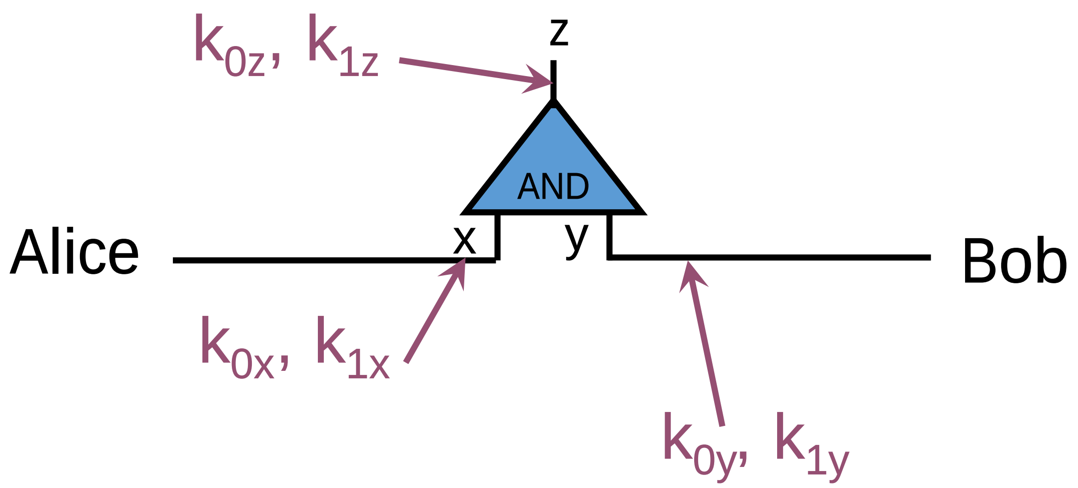
Intuition





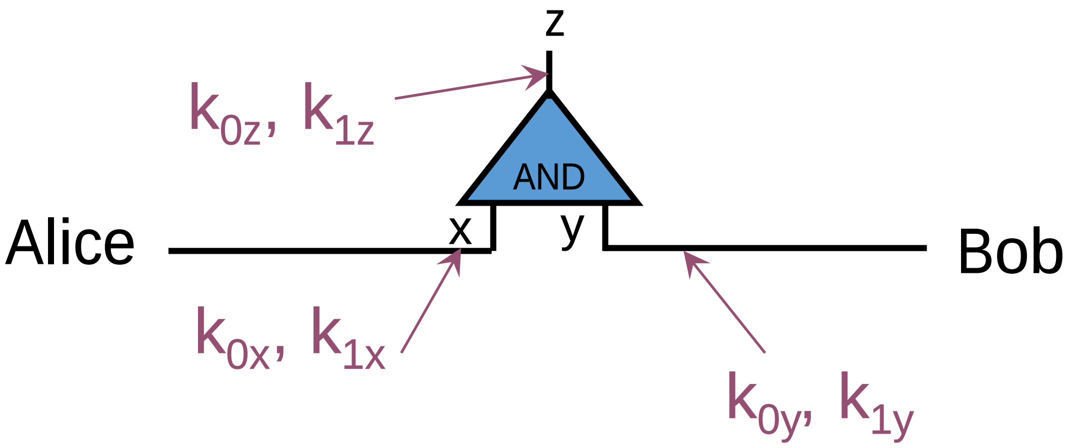
1: Pick Random Keys For Each Wire

- Next, evaluate <u>one gate</u> 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:

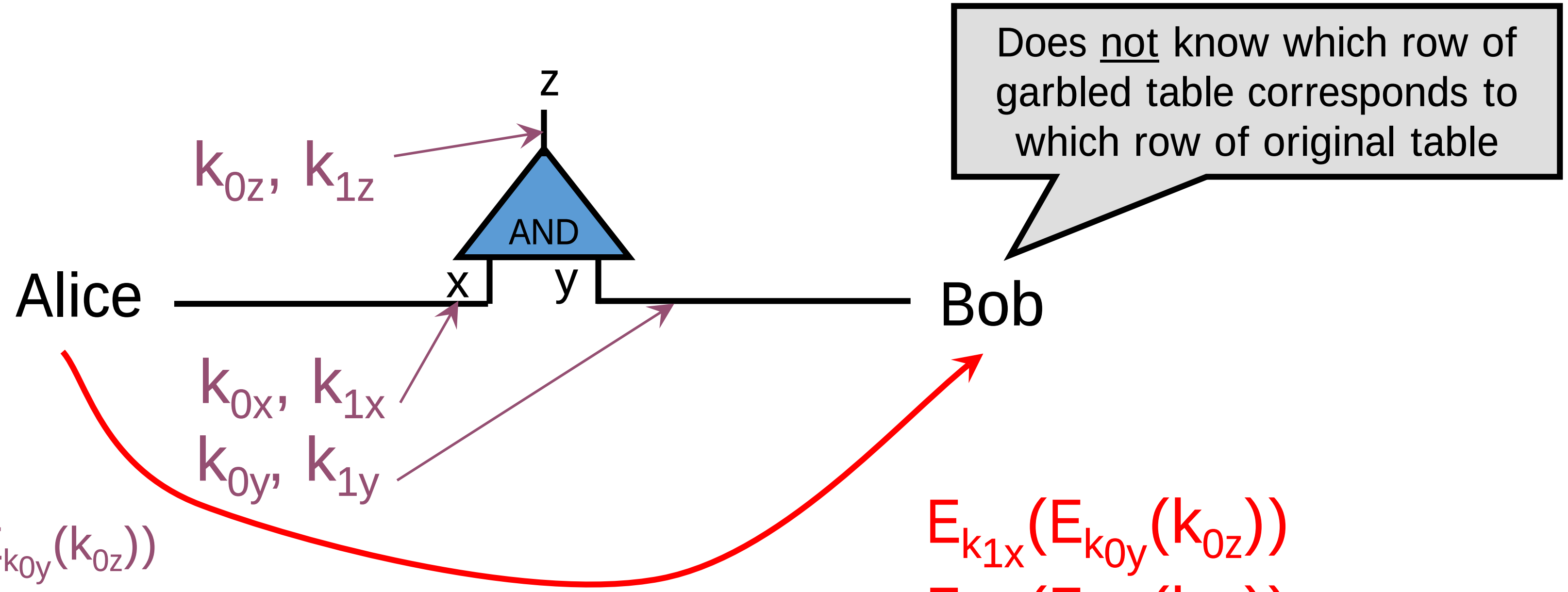
x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

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:

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

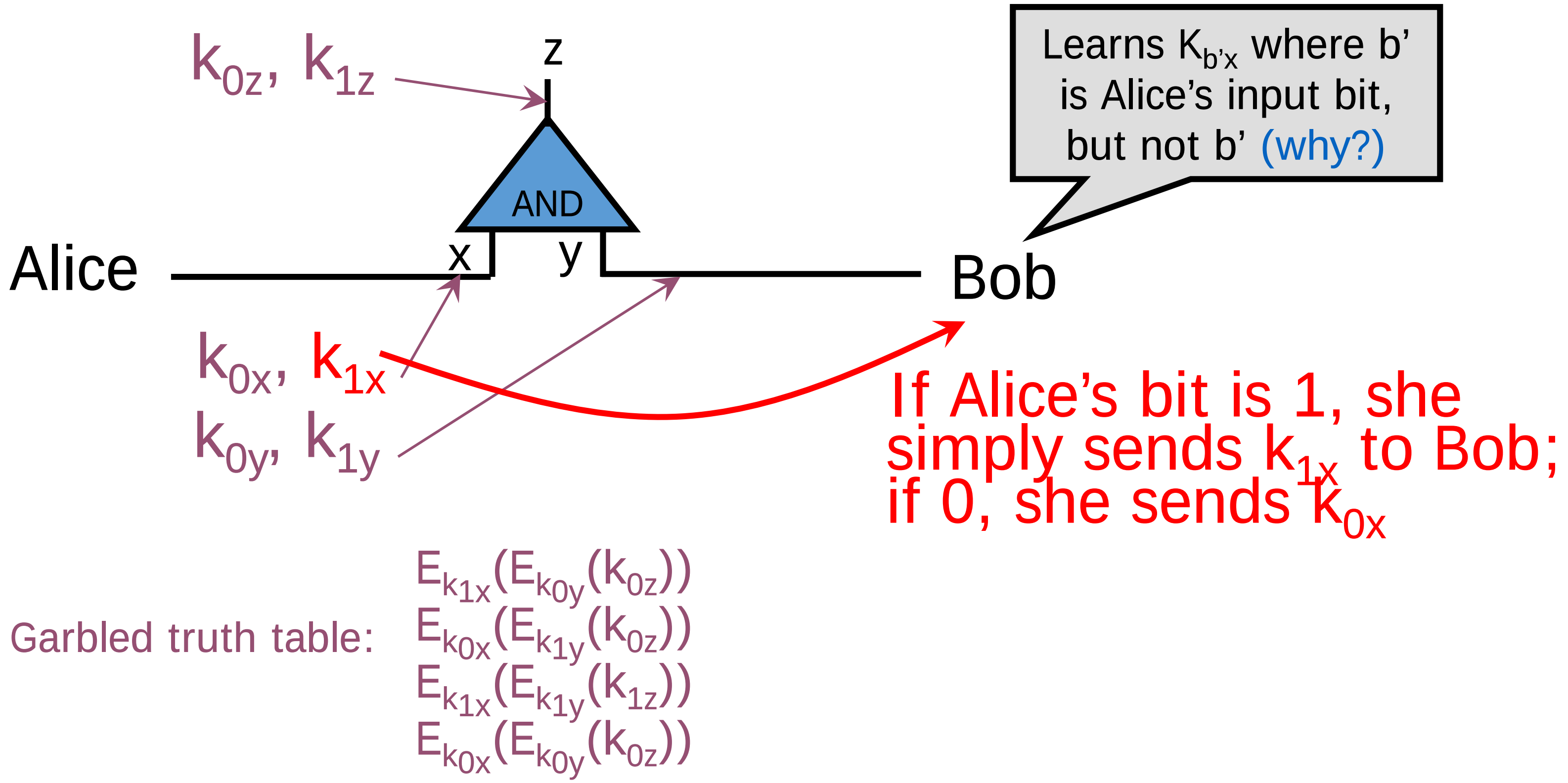
$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$

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

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

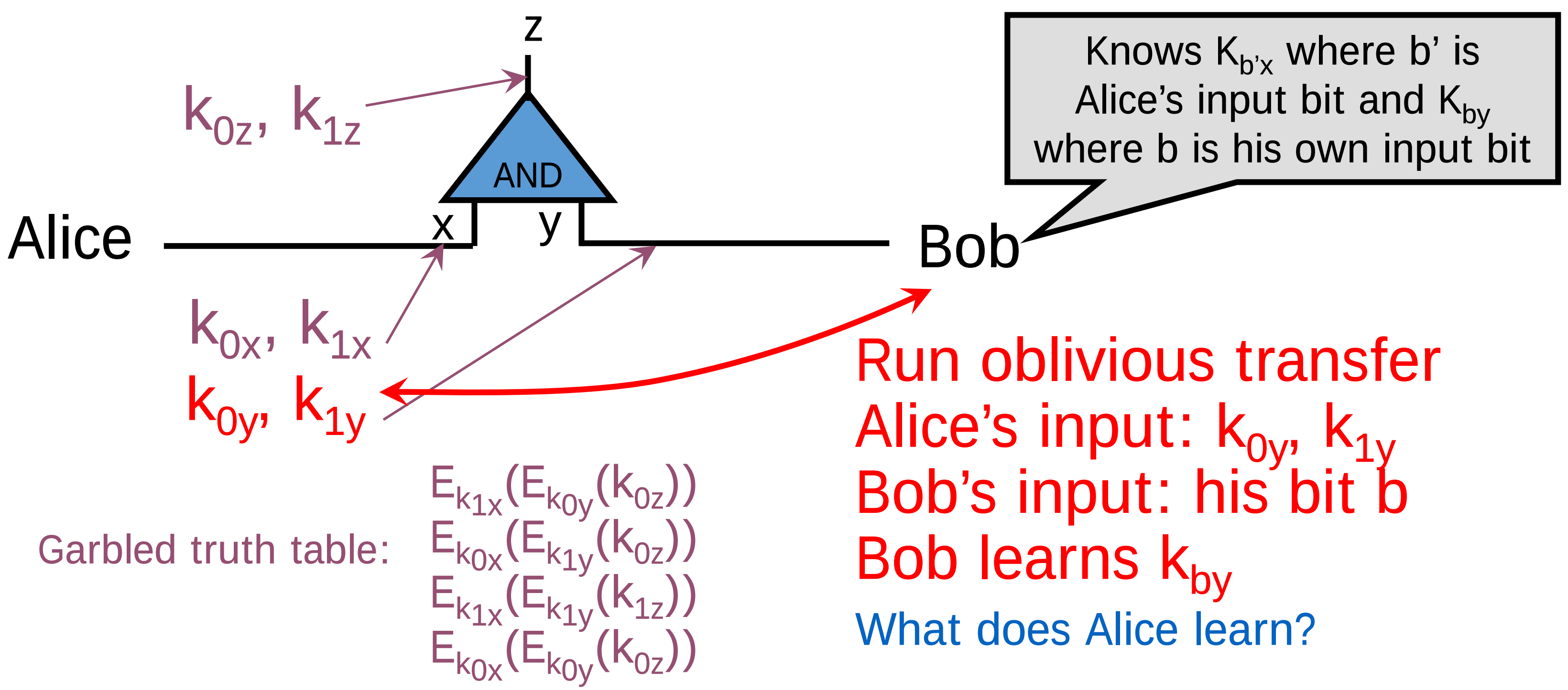
4: Send Keys For Alice's Inputs

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



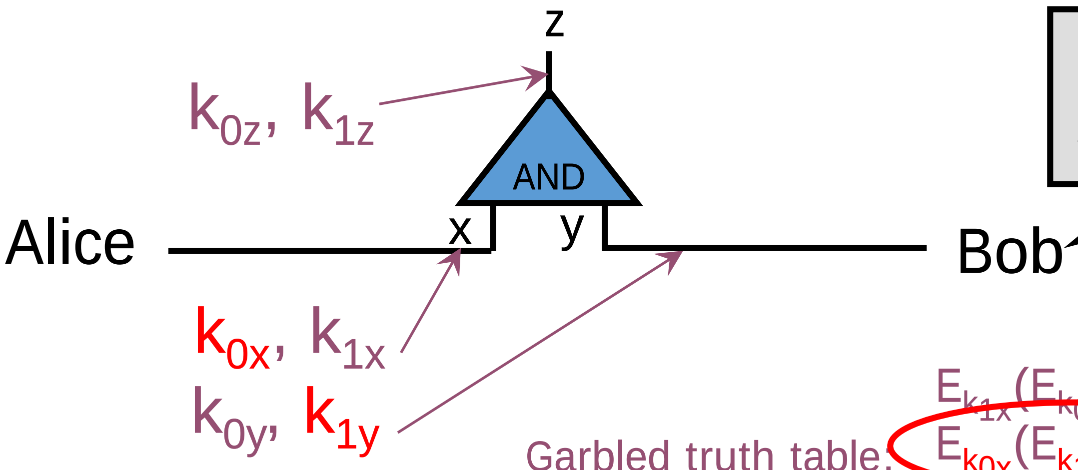
5: Use OT on Keys for Bob's Input

- 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?



Knows $K_{b'x}$ where b' is Alice's input bit and K_{by} where b is his own input bit

Suppose b'=0, b=1

Garbled truth table:

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

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

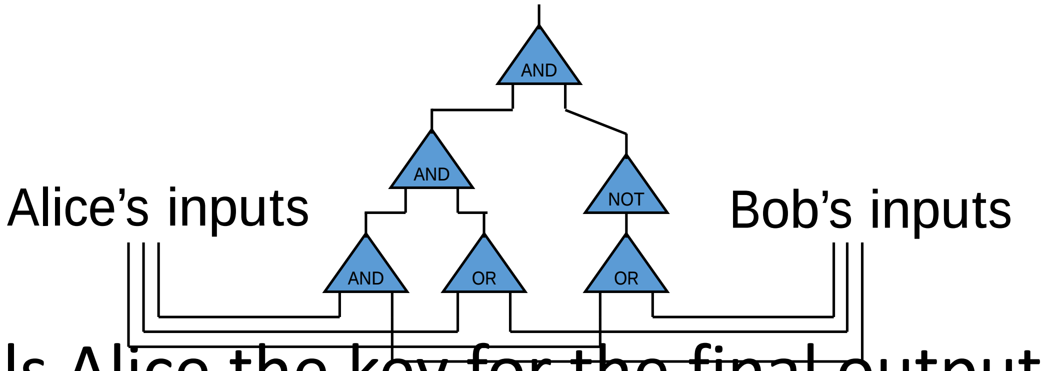
$E_{k_{1x}}(E_{k_{1y}}(k_{1z}))$

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

This is the only row Bob can decrypt. He learns K_{0z}

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?)



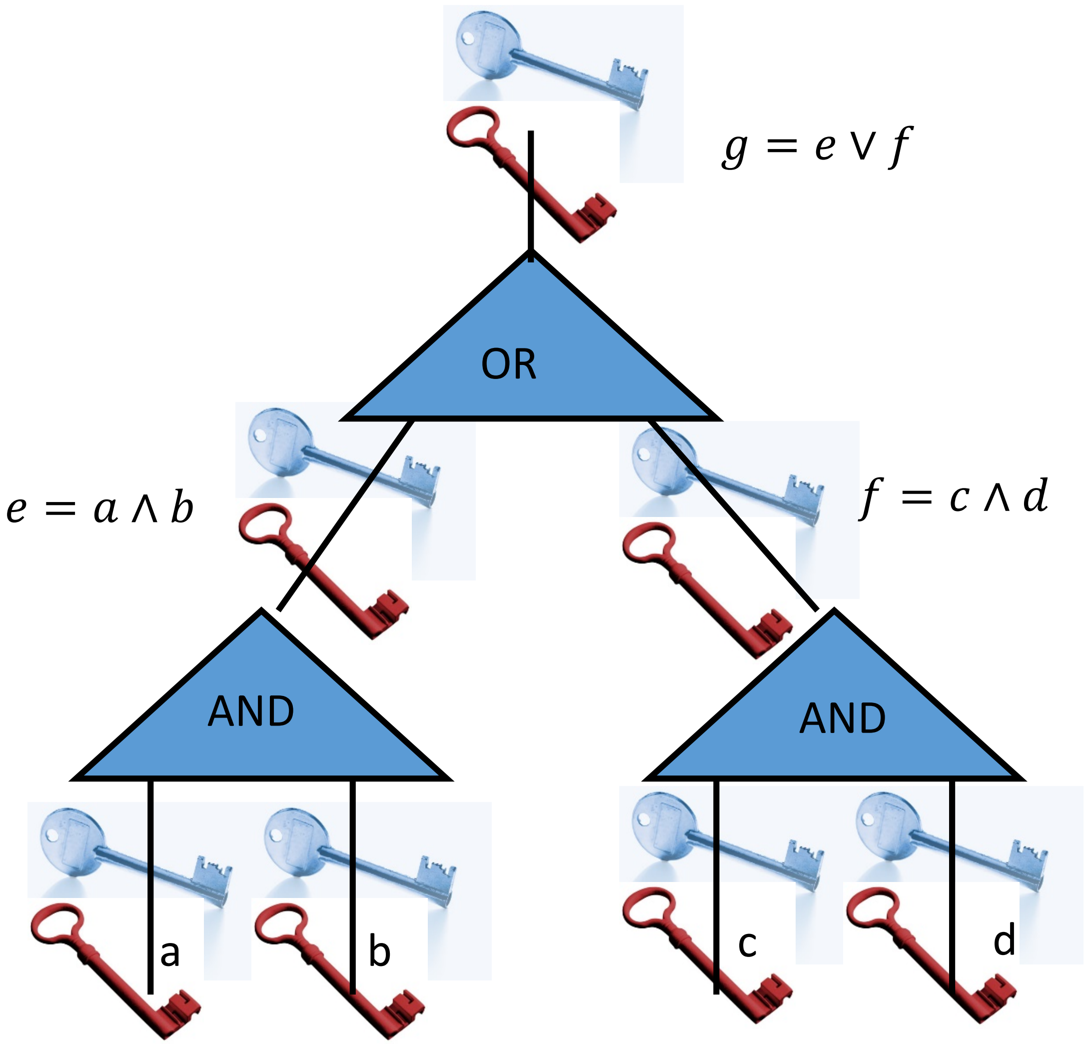
- Bob tells Alice the key for the final output wire and she tells him if it corresponds to 0 or 1
  - Bob does <u>not</u> tell her intermediate wire keys (why?)

Example

Alice's Input: a,b

Bob's Input: c,d

$$f(a, b, c, d) = (a \wedge b) \vee (c \wedge d)$$



Example

Alice's Input: a,b

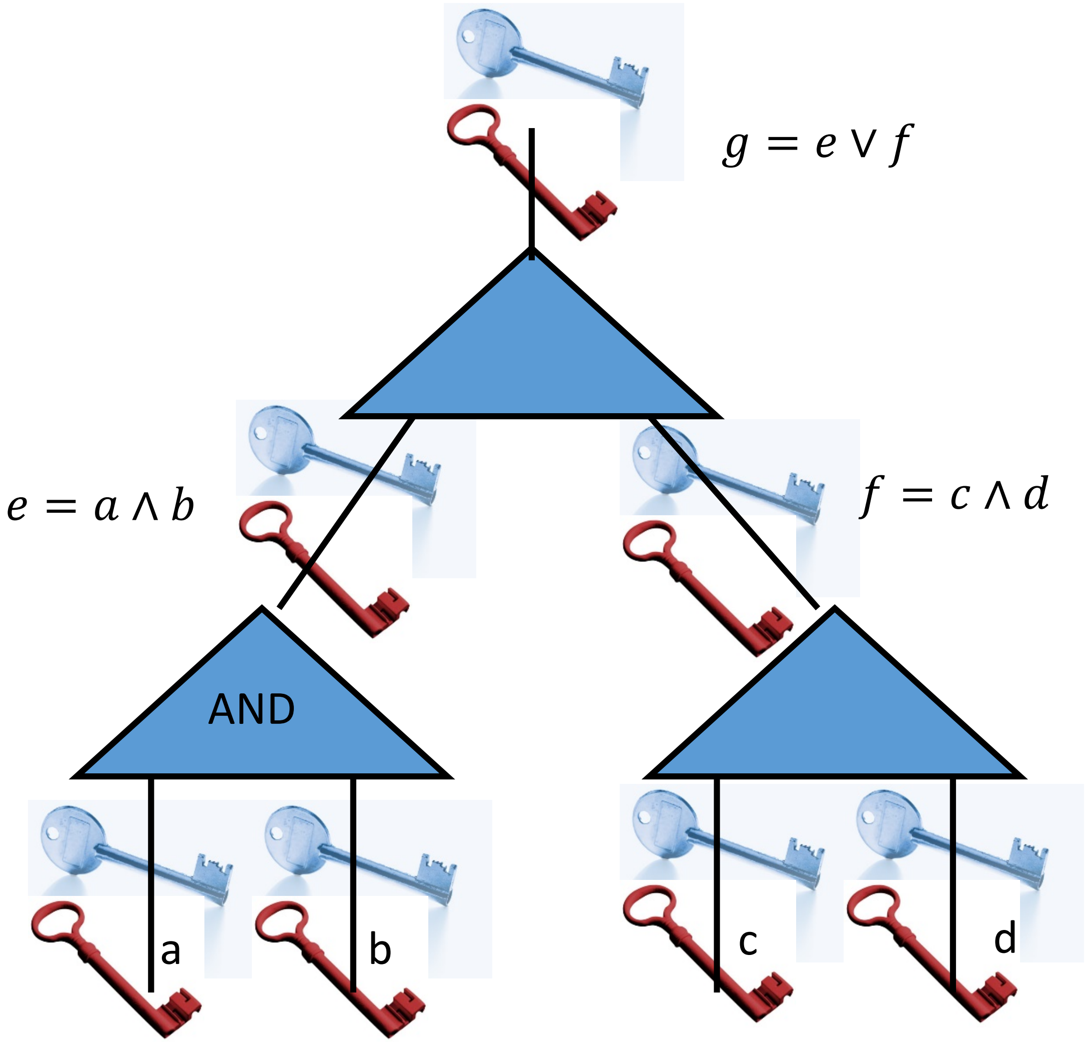
Bob's Input: c,d

$f(a, b, c, d) = (a \wedge b) \vee (c \wedge d)$

Step 1: Alice picks keys for each wire

$K_{a,0}, K_{b,0}, K_{c,0}, K_{d,0}, K_{e,0}, K_{f,0}, K_{g,0}$

$K_{a,1}, K_{b,1}, K_{c,1}, K_{d,1}, K_{e,1}, K_{f,1}, K_{g,1}$



Example

If a=0 and b=0 then e=$(a \wedge b)$=0

Alice's Input: a,b

Bob's Input: c,d

$f(a,b,c,d) = (a \wedge b) \vee (c \wedge d)$

Step 2: Alice garbles each gate (gate e)

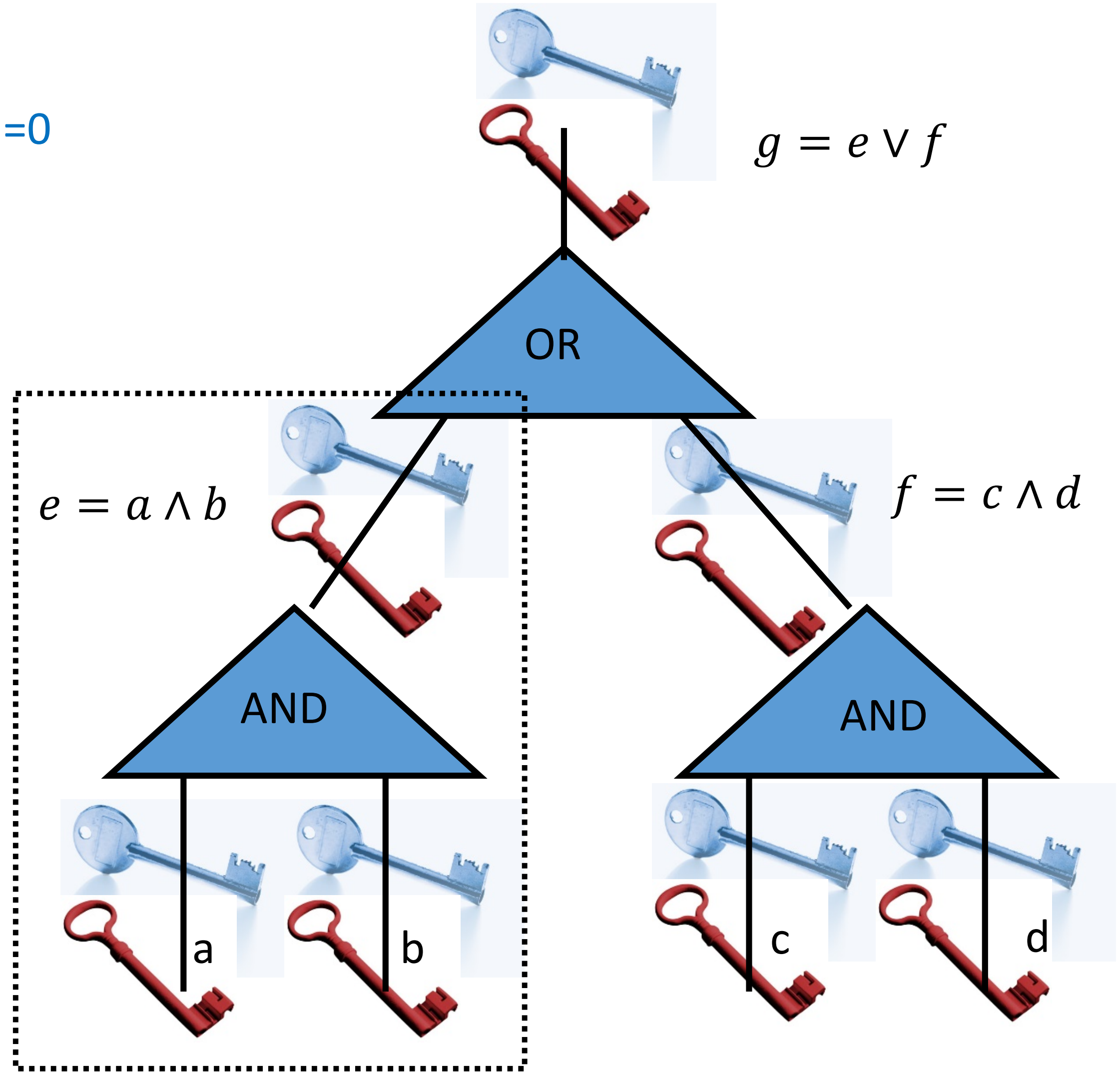
$$c_{e,0,0} = Enc_{K_{a,0}}\left(Enc_{K_{b,0}}\left(K_{e,0}\right)\right)$$

$$c_{e,0,1} = Enc_{K_{a,0}}\left(Enc_{K_{b,1}}\left(K_{e,0}\right)\right)$$

Authenticated Encryption

$$c_{e,1,0} = Enc_{K_{a,1}}\left(Enc_{K_{b,0}}\left(K_{e,0}\right)\right)$$

$$c_{e,1,1} = Enc_{K_{a,1}}\left(Enc_{K_{b,1}}\left(K_{e,1}\right)\right)$$



If a=0 and b=1 then e=$(a \wedge b)$=0

If a=1 and b=0 then e=$(a \wedge b)$=0

If a=1 and b=1 then e=$(a \wedge b)$=1

Example

Alice's Input: a,b

Bob's Input: c,d

$f(a,b,c,d) = (a \wedge b) \vee (c \wedge d)$

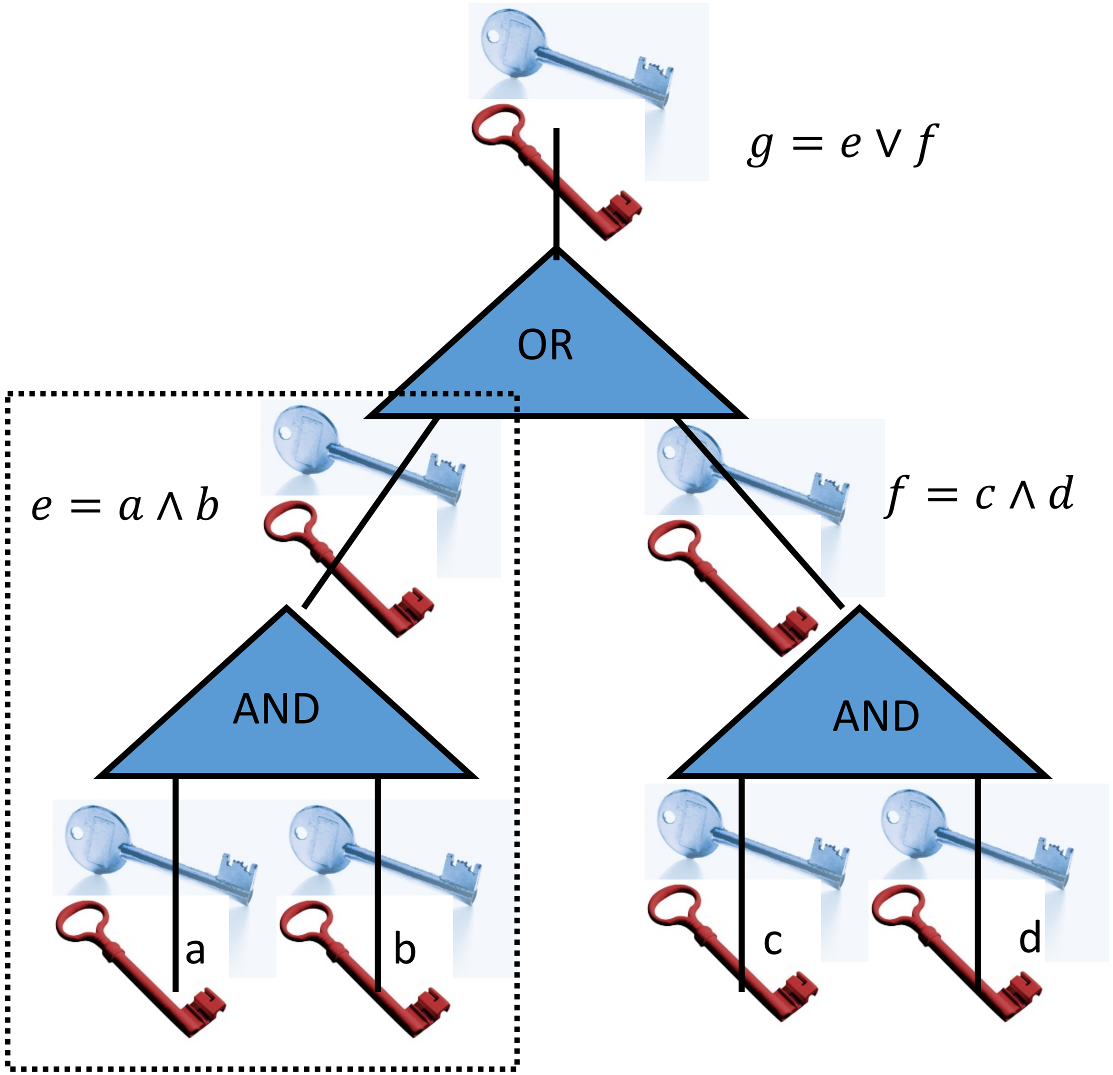
Step 2: Alice garbles each gate (+shuffle)

$$c_{e,1,1} = Enc_{K_{a,1}}\left(Enc_{K_{b,1}}\left(K_{e,1}\right)\right)$$

$$c_{e,0,1} = Enc_{K_{a,0}}\left(Enc_{K_{b,1}}\left(K_{e,0}\right)\right)$$

$$c_{e,0,0} = Enc_{K_{a,0}}\left(Enc_{K_{b,0}}\left(K_{e,0}\right)\right)$$

$$c_{e,1,0} = Enc_{K_{a,1}}\left(Enc_{K_{b,0}}\left(K_{e,0}\right)\right)$$



If Alice forgets to shuffle then Bob would notice which ciphertext decrypts successfully, identify the corresponding row in the truth table and learn the corresponding wire values!

Example

Alice's Input: a,b

Bob's Input: c,d

$f(a,b,c,d) = (a \wedge b) \vee (c \wedge d)$

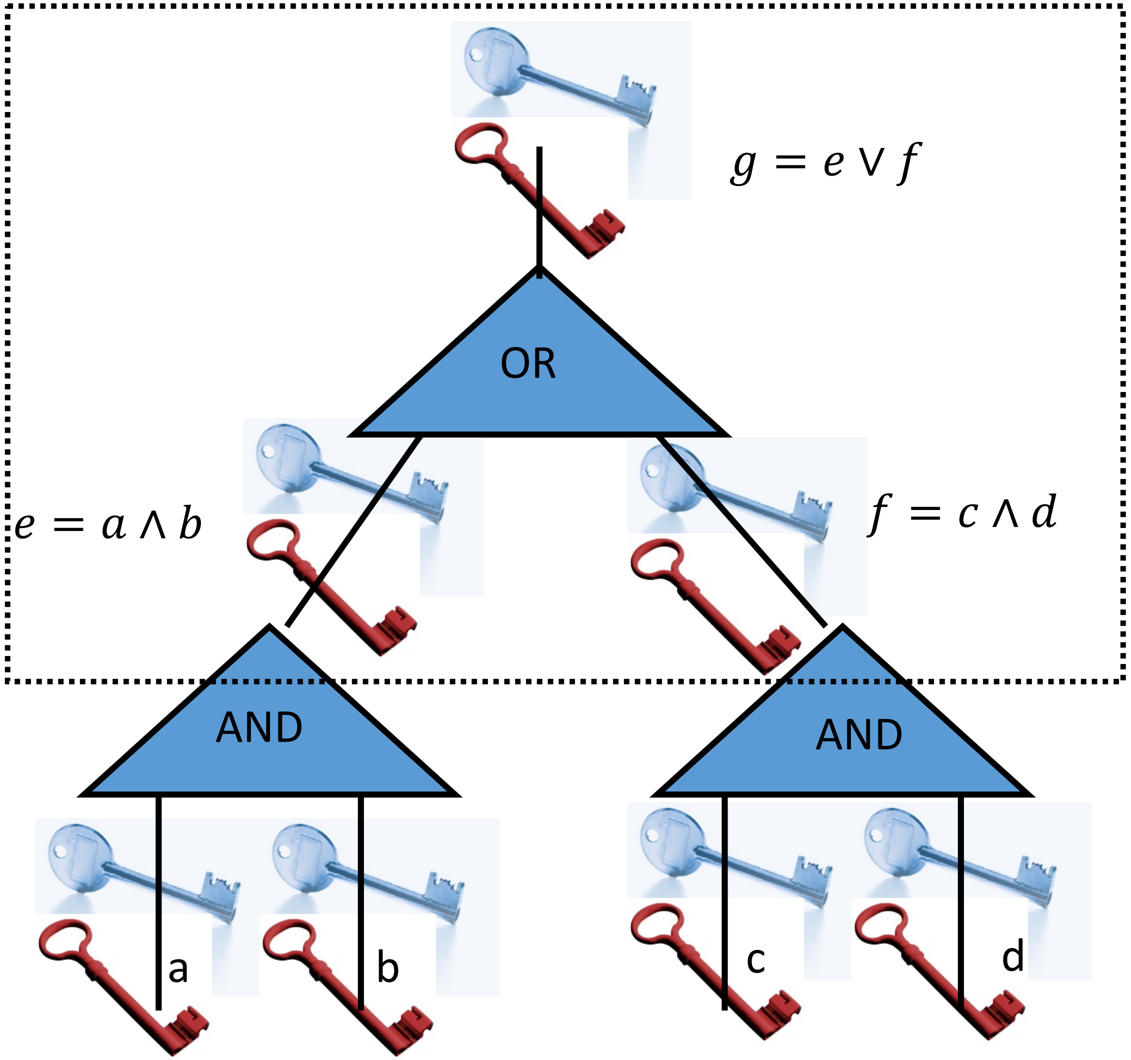
Step 2: Alice garbles each gate (gate g)

$$\boldsymbol{c_{g,0,0} = Enc_{K_{e,0}}\left(Enc_{K_{f,0}}\left(K_{g,0}\right)\right)}$$

$$\boldsymbol{c_{g,0,1} = Enc_{K_{e,0}}\left(Enc_{K_{f,1}}\left(K_{g,1}\right)\right)}$$

$$\boldsymbol{c_{g,1,0} = Enc_{K_{e,1}}\left(Enc_{K_{f,0}}\left(K_{g,1}\right)\right)}$$

$$\boldsymbol{c_{g,1,1} = Enc_{K_{e,1}}\left(Enc_{K_{f,1}}\left(K_{g,1}\right)\right)}$$



Example

Alice's Input: a,b

Bob's Input: c,d

$f(a,b,c,d) = (a \wedge b) \vee (c \wedge d)$

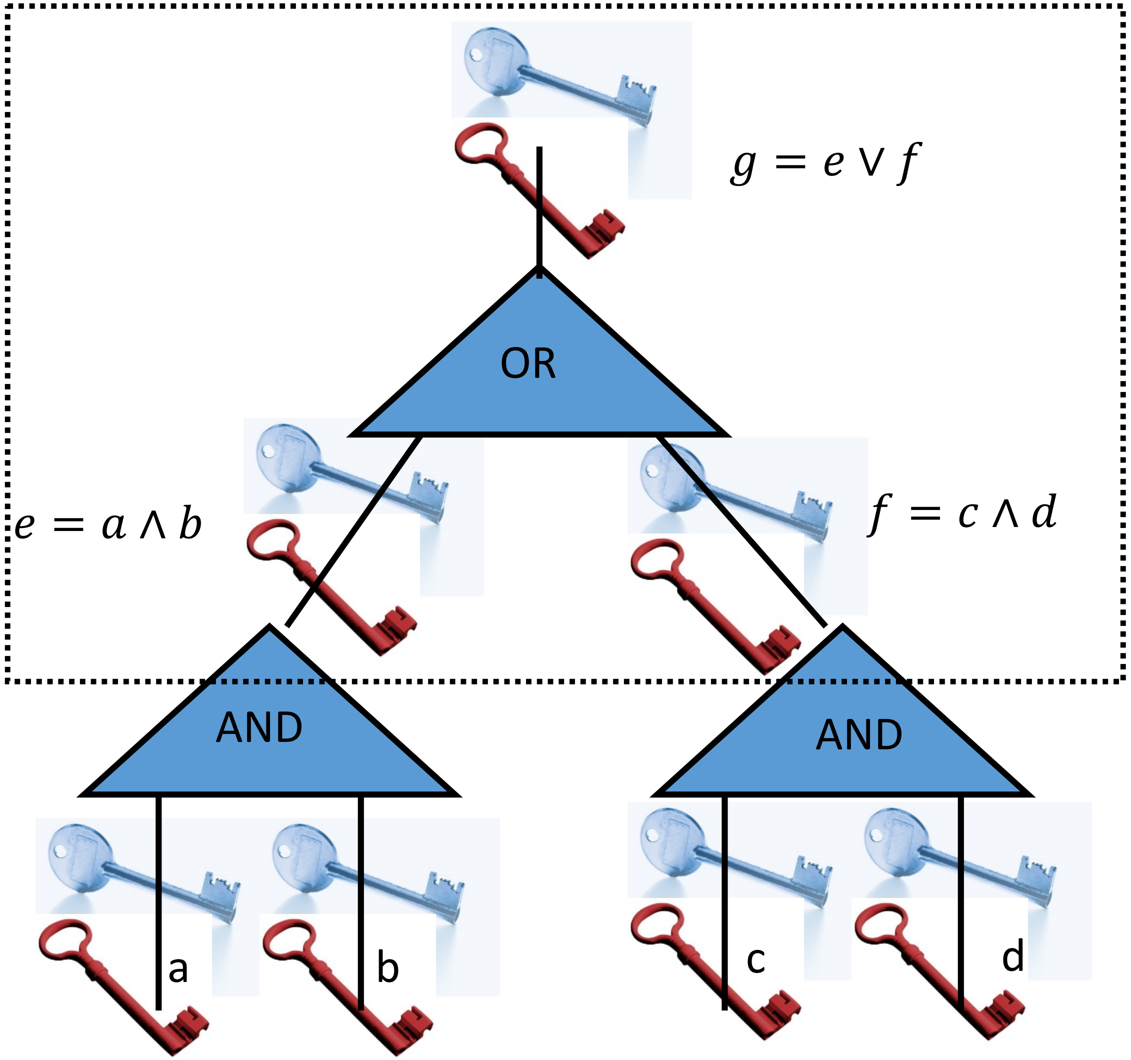
Step 2: Alice garbles each gate (+shuffle)

$$\boldsymbol{c_{g,0,0} = Enc_{K_{e,0}}\left(Enc_{K_{f,0}}\left(K_{g,0}\right)\right)}$$

$$\boldsymbol{c_{g,1,1} = Enc_{K_{e,1}}\left(Enc_{K_{f,1}}\left(K_{g,1}\right)\right)}$$

$$\boldsymbol{c_{g,1,0} = Enc_{K_{e,1}}\left(Enc_{K_{f,0}}\left(K_{g,1}\right)\right)}$$

$$\boldsymbol{c_{g,0,1} = Enc_{K_{e,0}}\left(Enc_{K_{f,1}}\left(K_{g,1}\right)\right)}$$



Example

Alice's Input: a,b

Bob's Input: c,d

$f(a, b, c, d) = (a \wedge b) \vee (c \wedge d)$

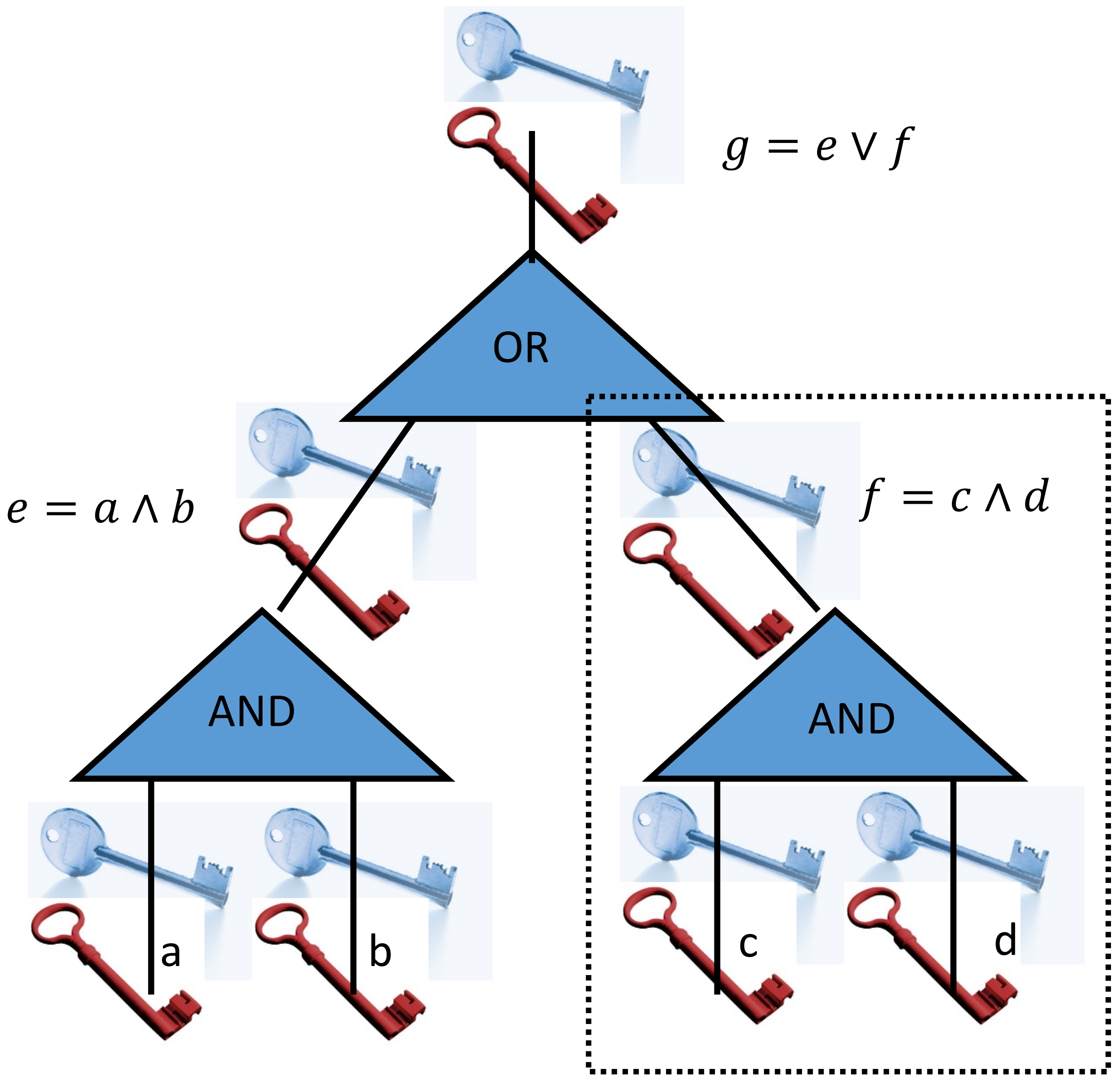
Step 2: Alice garbles each gate (gate f)

$$c_{f,0,0} = Enc_{K_{c,0}}\left(Enc_{K_{d,0}}\left(K_{f,0}\right)\right)$$

$$c_{f,0,1} = Enc_{K_{c,0}}\left(Enc_{K_{d,1}}\left(K_{f,0}\right)\right)$$

$$c_{f,1,0} = Enc_{K_{c,1}}\left(Enc_{K_{d,0}}\left(K_{f,0}\right)\right)$$

$$c_{f,1,1} = Enc_{K_{c,1}}\left(Enc_{K_{d,1}}\left(K_{f,1}\right)\right)$$



Example

Alice's Input: a,b

Bob's Input: c,d

$f(a,b,c,d) = (a \wedge b\ ) \vee (c \wedge d\ )$

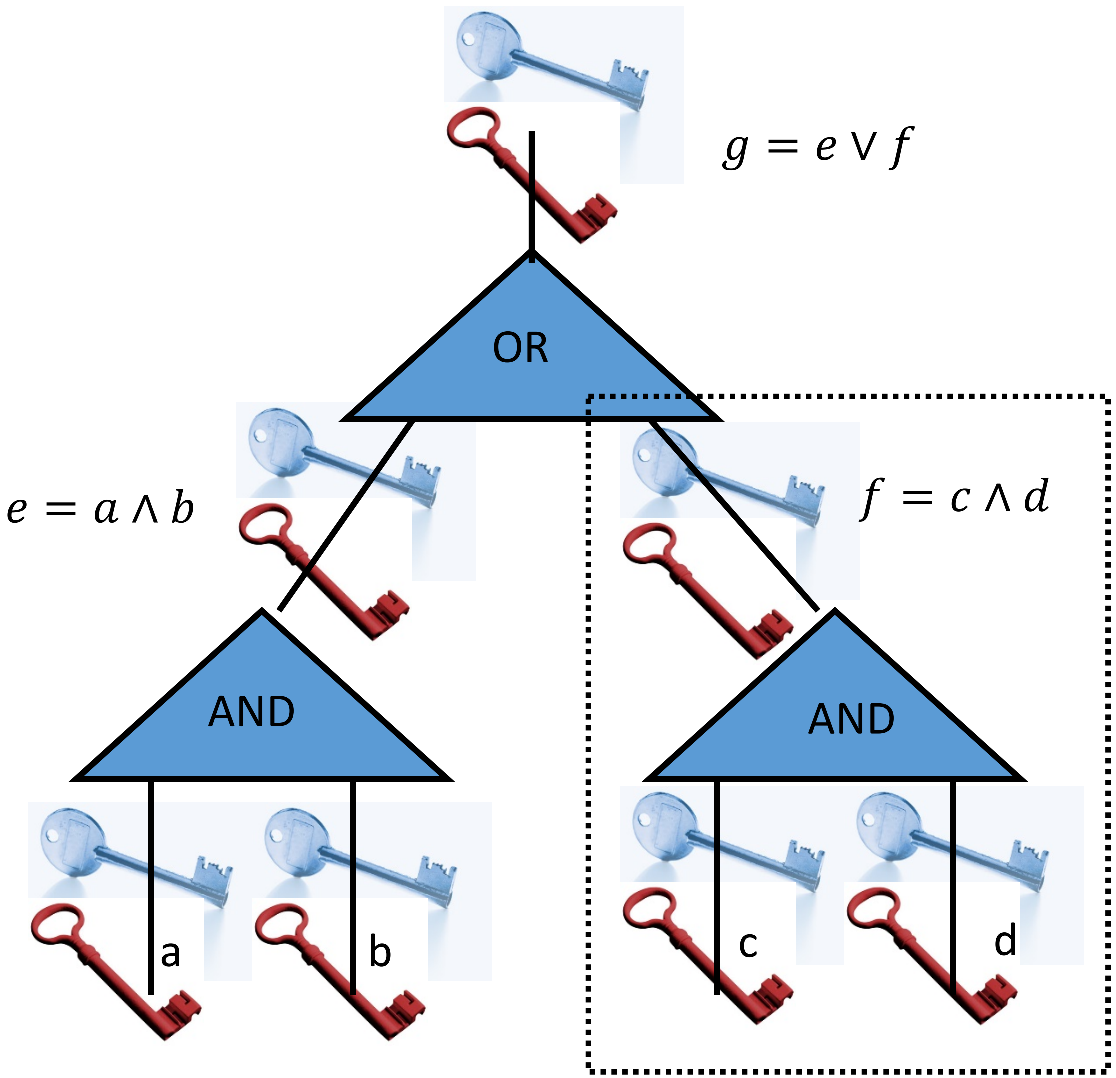
Step 2: Alice garbles each gate (+shuffle)

$$\boldsymbol{c_{f,1,0} = Enc_{K_{c,1}}\left(Enc_{K_{d,0}}\left(K_{f,0}\right)\right)}$$

$$\boldsymbol{c_{f,1,1} = Enc_{K_{c,1}}\left(Enc_{K_{d,1}}\left(K_{f,1}\right)\right)}$$

$$\boldsymbol{c_{f,0,0} = Enc_{K_{c,0}}\left(Enc_{K_{d,0}}\left(K_{f,0}\right)\right)}$$

$$\boldsymbol{c_{f,0,1} = Enc_{K_{c,0}}\left(Enc_{K_{d,1}}\left(K_{f,0}\right)\right)}$$



Example

Alice's Input: a,b

Bob's Input: c,d

$f(a, b, c, d) = (a \wedge b) \vee (c \wedge d)$

Step 3: Alice sends garbled circuit to Bob

Gate e: $c_{e,0,0}$, $c_{e,0,1}$, $c_{e,1,0}$, $c_{e,1,1}$

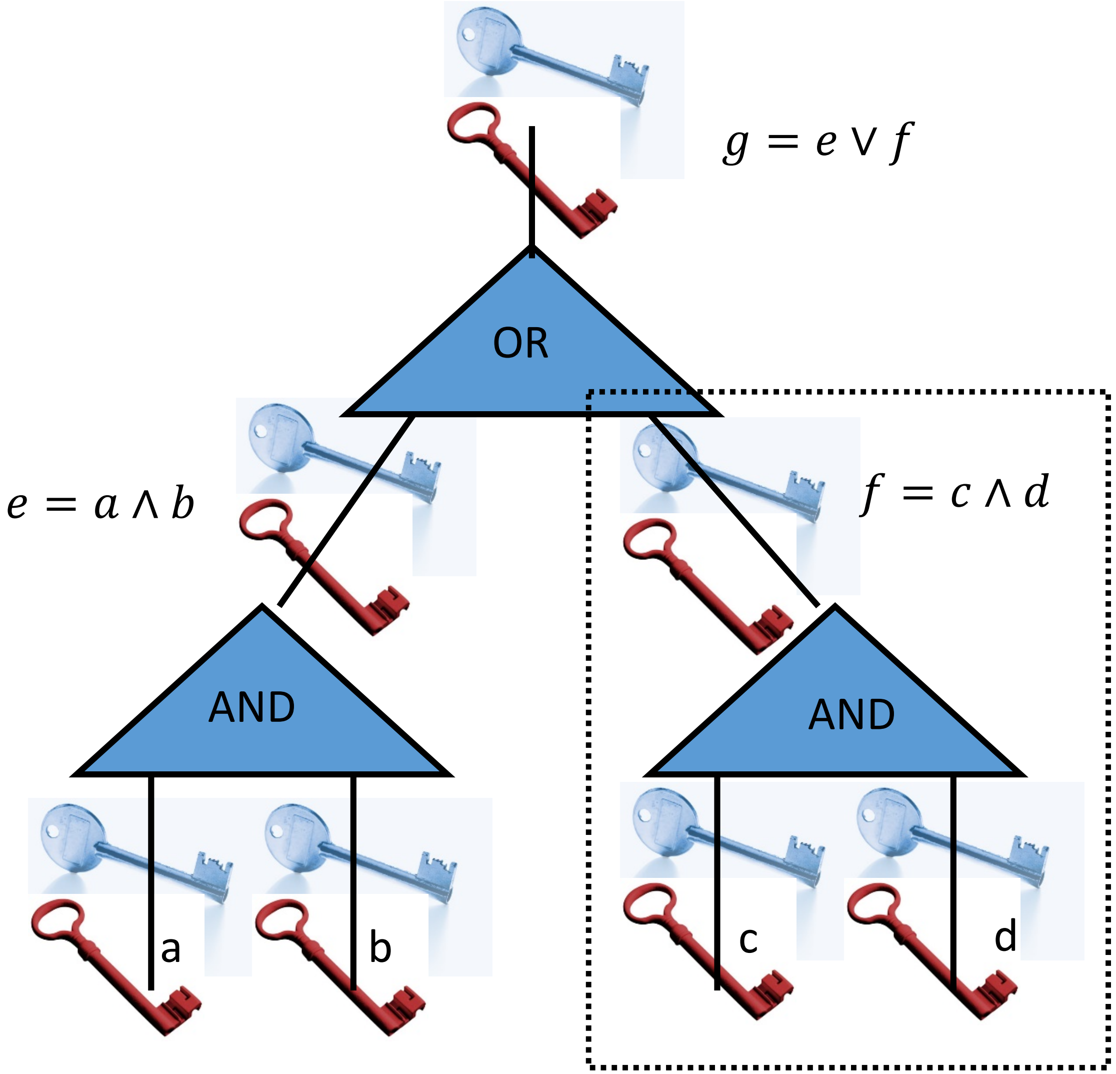
Gate f: $c_{f,1,0}$, $c_{f,1,1}$, $c_{f,0,0}$, $c_{f,0,1}$

Gate g: $c_{g,0,0}$, $c_{g,1,1}$, $c_{g,1,0}$, $c_{g,0,1}$

Step 4: Alice sends keys corresponding to her inputs

Example: a=0, b=1

Alice sends Bob $\boldsymbol{K_{a,0}}$ and $\boldsymbol{K_{b,1}}$



Example

Alice's Input: a,b

Bob's Input: c,d

$f(a, b, c, d) = (a \wedge b) \vee (c \wedge d)$

Step 5: Alice and Bob run OT for each of Bob's input wires

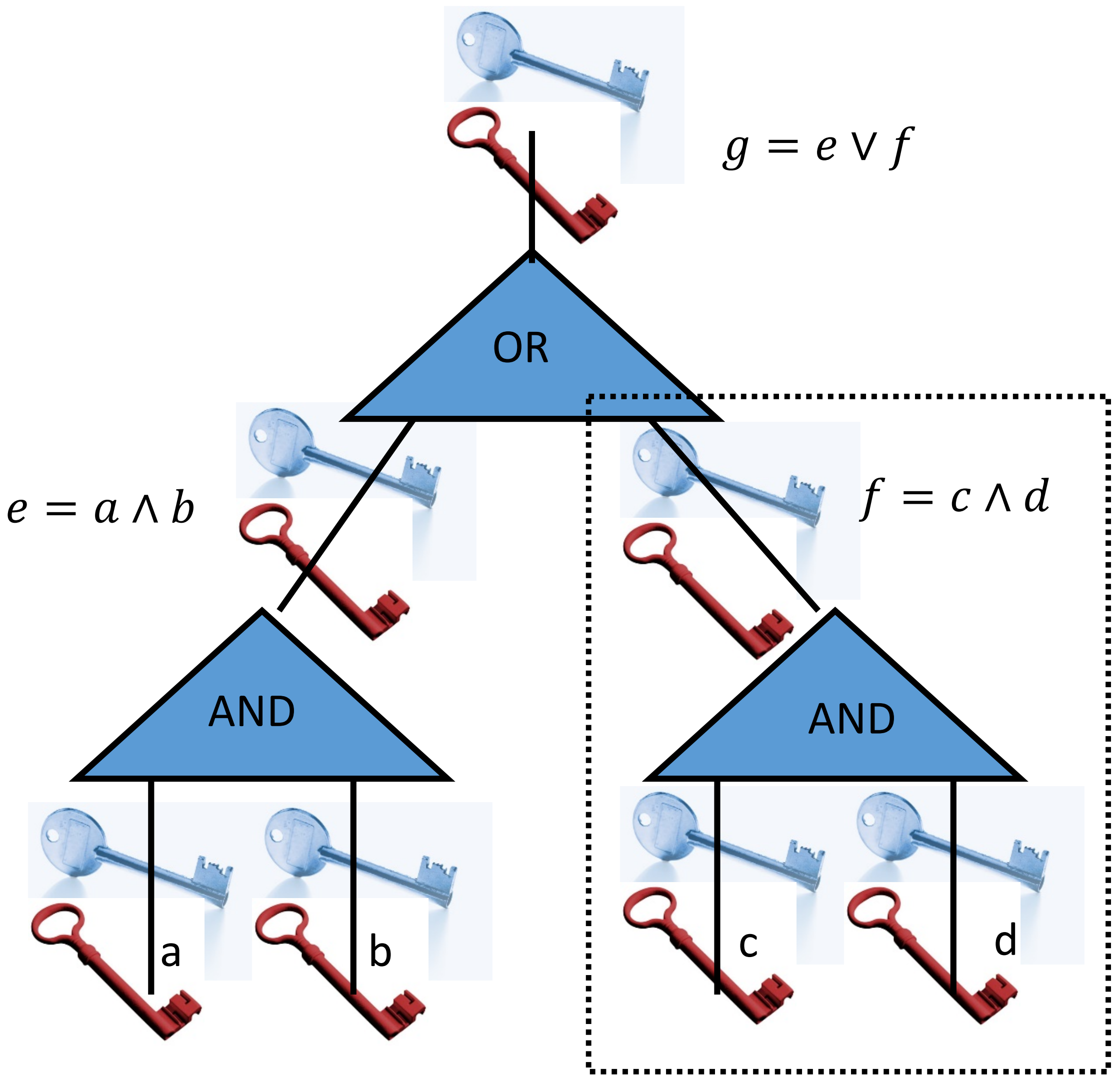
Wire C OT:

Bob's Input: 1 if c=1; 0 otherwise

Alice's Input: $\boldsymbol{K_{c,0}}$ and $\boldsymbol{K_{c,1}}$

Bob's Output: $\boldsymbol{K_{c,0}}$ if c=0; otherwise $\boldsymbol{K_{c,1}}$

Alice's Output: Nothing



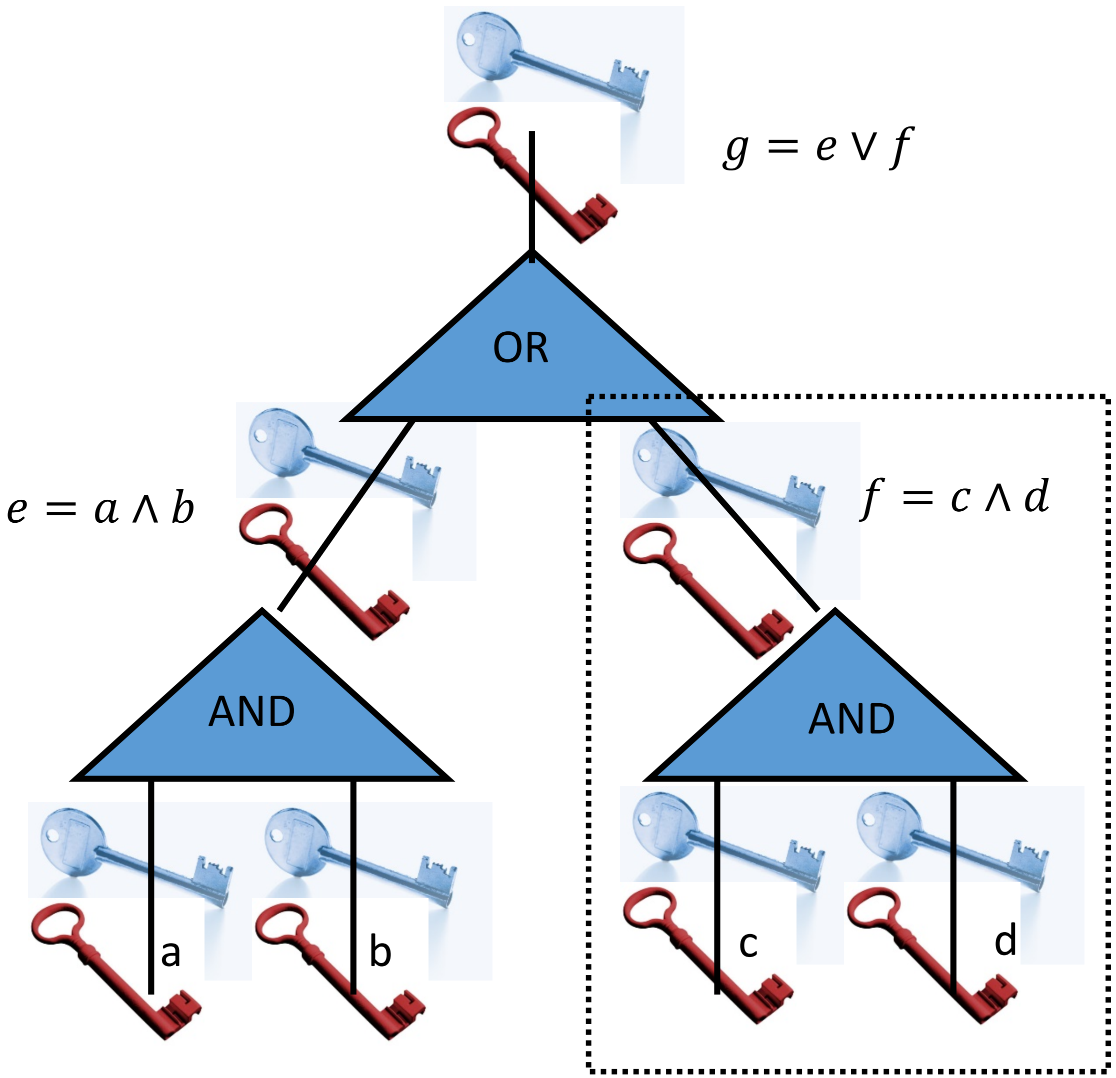
Example

Alice's Input: a,b
Bob's Input: c,d
$f(a, b, c, d) = (a \wedge b) \vee (c \wedge d)$

Step 5: Alice and Bob run OT for each of Bob's input wires

Wire D OT:

Bob's Input: 1 if d=1; 0 otherwise
Alice's Input: $\boldsymbol{K_{d,0}}$ and $\boldsymbol{K_{d,1}}$
Bob's Output: $\boldsymbol{K_{d,0}}$ if d=0; otherwise $\boldsymbol{K_{d,1}}$
Alice's Output: Nothing



Example

Alice's Input: a,b

Bob's Input: c,d

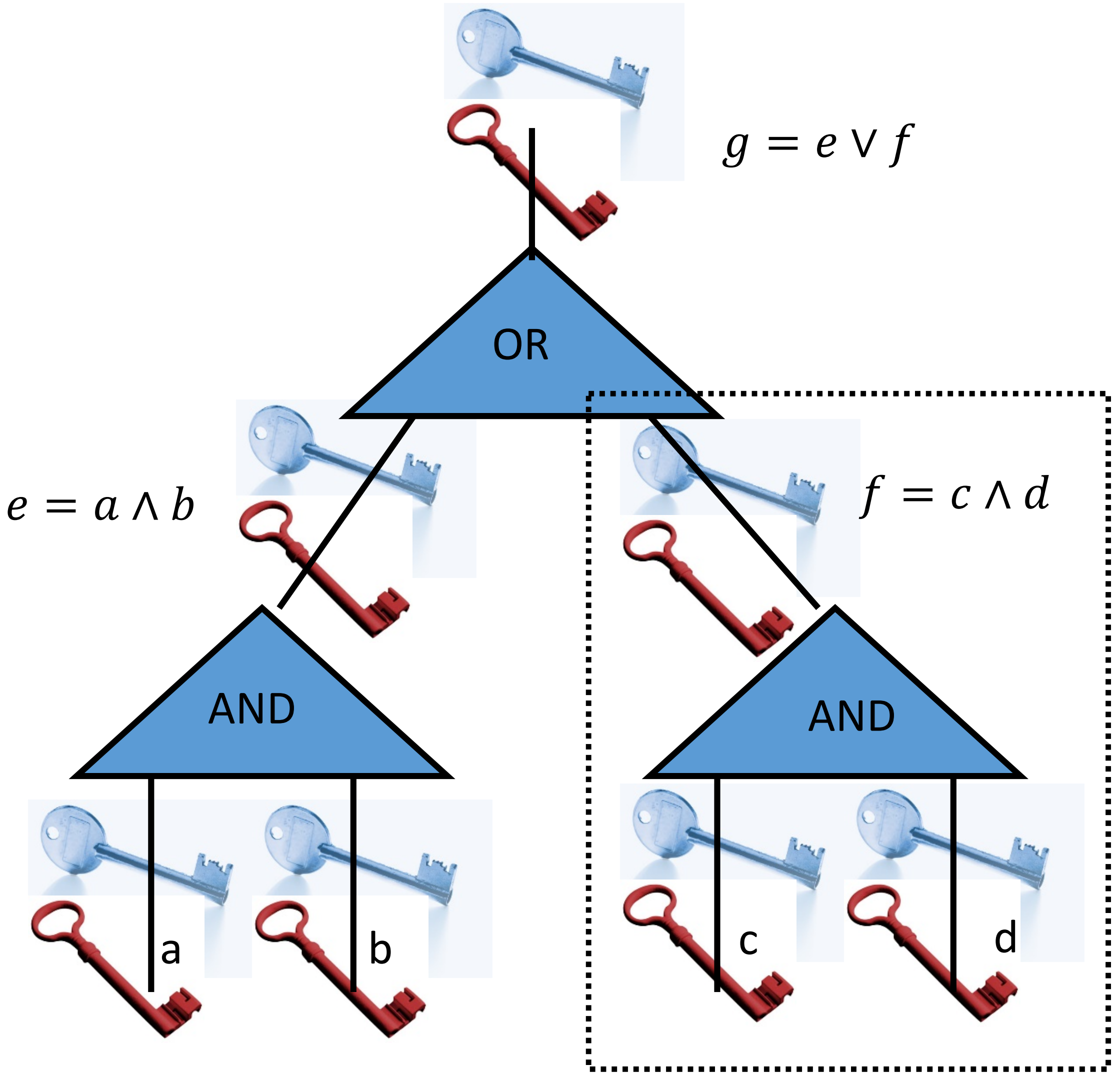
$f(a, b, c, d) = (a \wedge b) \vee (c \wedge d)$

Step 6: Bob evaluates the garbled circuit

Example: a=0, b=1, c=1,d=1

Alice sent Bob $\boldsymbol{K_{a,0}}$ and $\boldsymbol{K_{b,1}}$

Bob obtains $\boldsymbol{K_{c,1}}$ and $\boldsymbol{K_{d,1}}$ from OTs



Example

Alice's Input: a,b

Bob's Input: c,d

$f(a,b,c,d) = (a \wedge b) \vee (c \wedge d)$

Step 6: Bob evaluates the garbled circuit

Example: a=0, b=1, c=1,d=1

Alice sent Bob $\boldsymbol{K_{a,0}}$ and $\boldsymbol{K_{b,1}}$

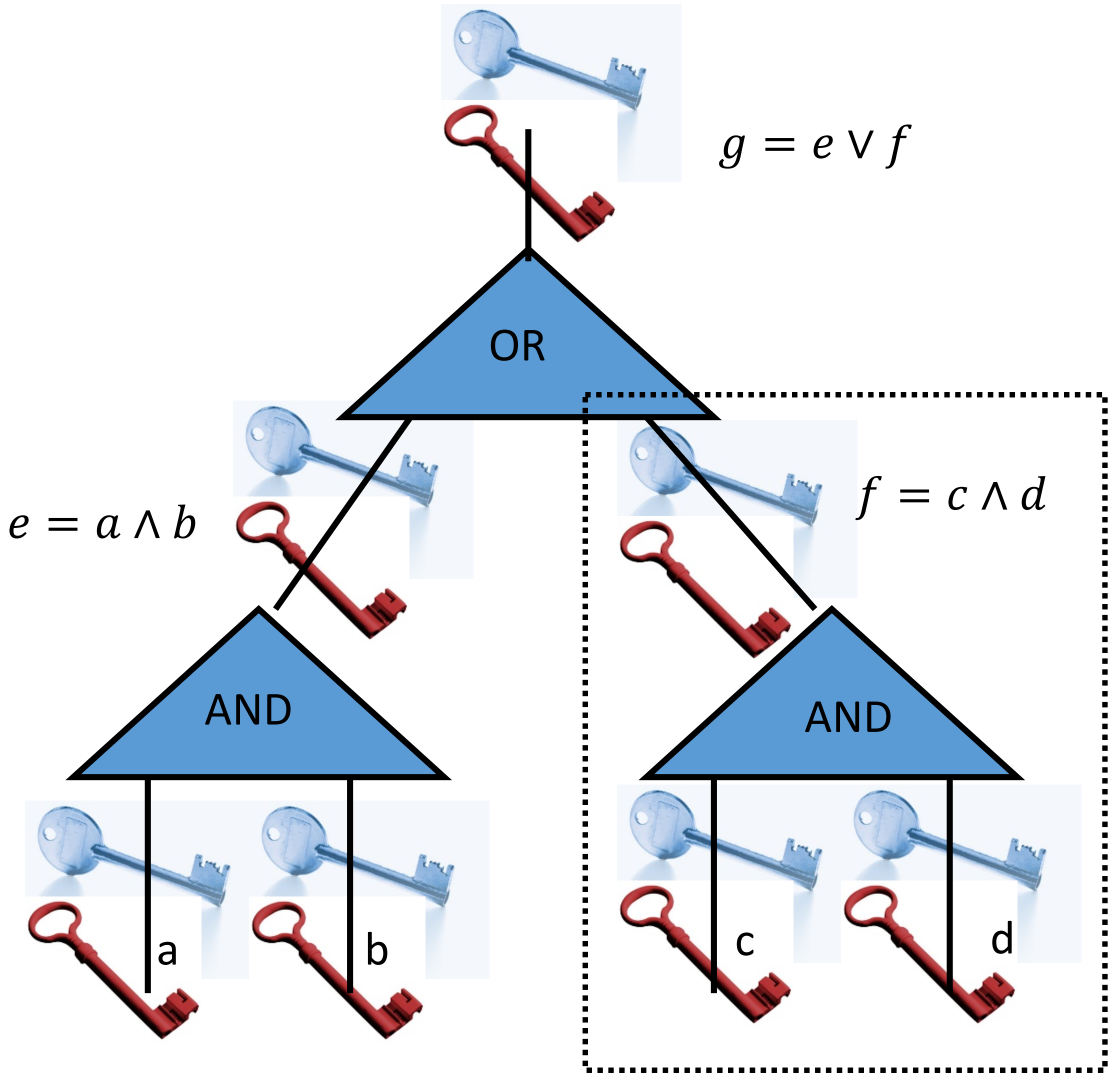
Bob obtains $\boldsymbol{K_{c,1}}$ and $\boldsymbol{K_{d,1}}$ from OTs

Bob uses $\boldsymbol{K_{a,0}}$ and $\boldsymbol{K_{b,1}}$ to obtain

$$\boldsymbol{K_{e,0} = Dec_{K_{b,1}}\left(Dec_{K_{a,0}}(c_{e,0,1})\right)}$$

Note 1: $\boldsymbol{c_{e,0,1} = Enc_{K_{a,0}}\left(Enc_{K_{b,1}}(K_{e,0})\right)}$ so

$$\boldsymbol{Dec_{K_{b,1}}\left(Dec_{K_{a,0}}(c_{e,0,1})\right) = Dec_{K_{b,1}}\left(Dec_{K_{a,0}}\left(Enc_{K_{a,0}}\left(Enc_{K_{b,1}}(K_{e,0})\right)\right)\right) = K_{e,0}}$$



Example

Alice's Input: a,b

Bob's Input: c,d

$f(a,b,c,d) = (a \wedge b) \vee (c \wedge d)$

Step 6: Bob evaluates the garbled circuit

Example: a=0, b=1, c=1,d=1

Alice sent Bob $K_{a,0}$ and $K_{b,1}$

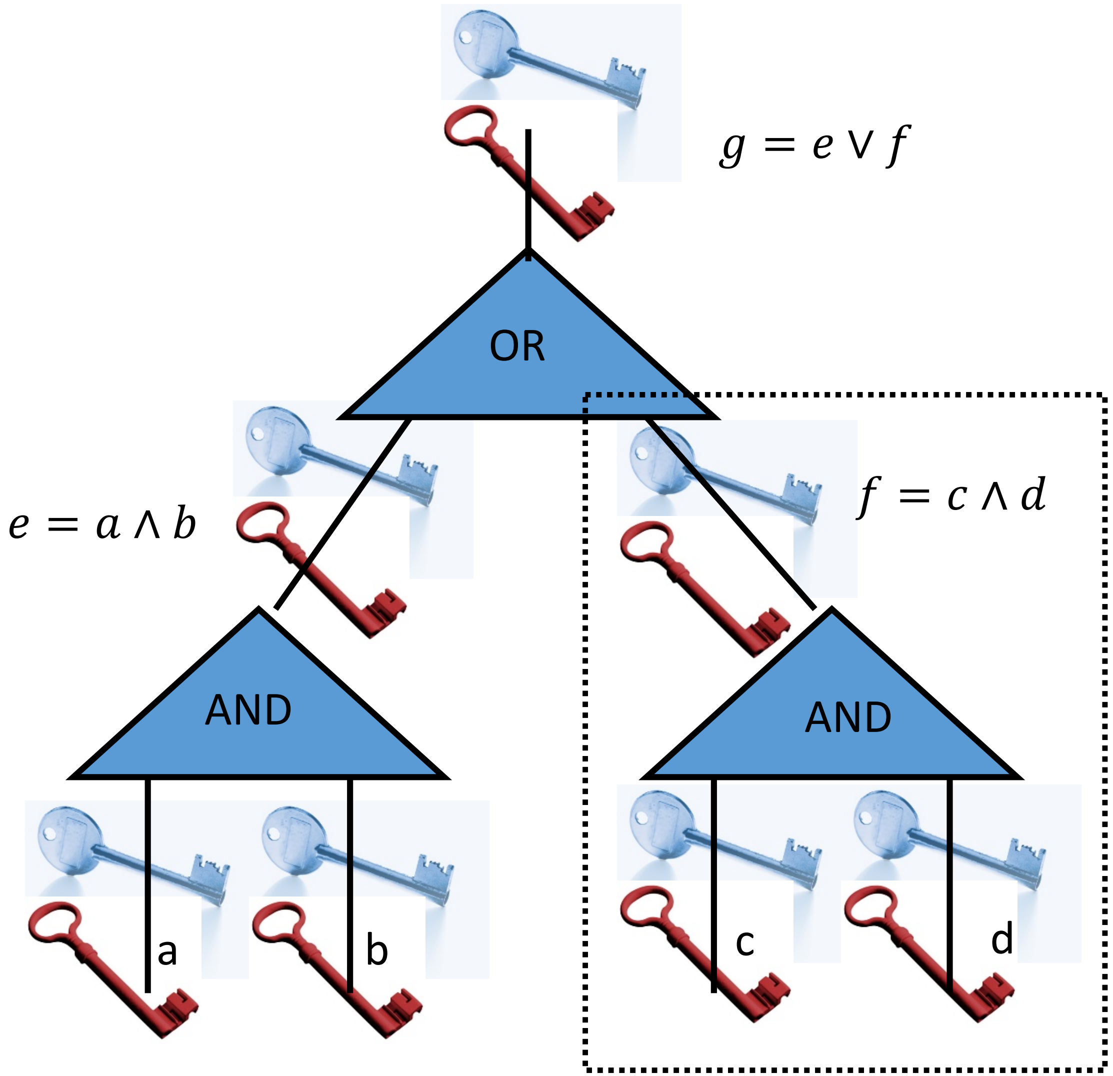
Bob obtains $K_{c,1}$ and $K_{d,1}$ from OTs

Bob uses $K_{a,0}$ and $K_{b,1}$ to obtain

$$K_{e,0} = Dec_{K_{b,1}}\left(Dec_{K_{a,0}}(c_{e,0,1})\right)$$

Note 2: $c_{e,1,1} = Enc_{K_{a,1}}\left(Enc_{K_{b,1}}(K_{e,1})\right)$ so

$$Dec_{K_{b,1}}\left(Dec_{K_{a,0}}(c_{e,1,1})\right) = Dec_{K_{b,1}}\left(Dec_{K_{a,0}}\left(Enc_{K_{a,1}}\left(Enc_{K_{b,1}}(K_{e,1})\right)\right)\right) = \perp$$



Example

Alice's Input: a,b

Bob's Input: c,d

$f(a,b,c,d) = (a \wedge b) \vee (c \wedge d)$

Step 6: Bob evaluates the garbled circuit

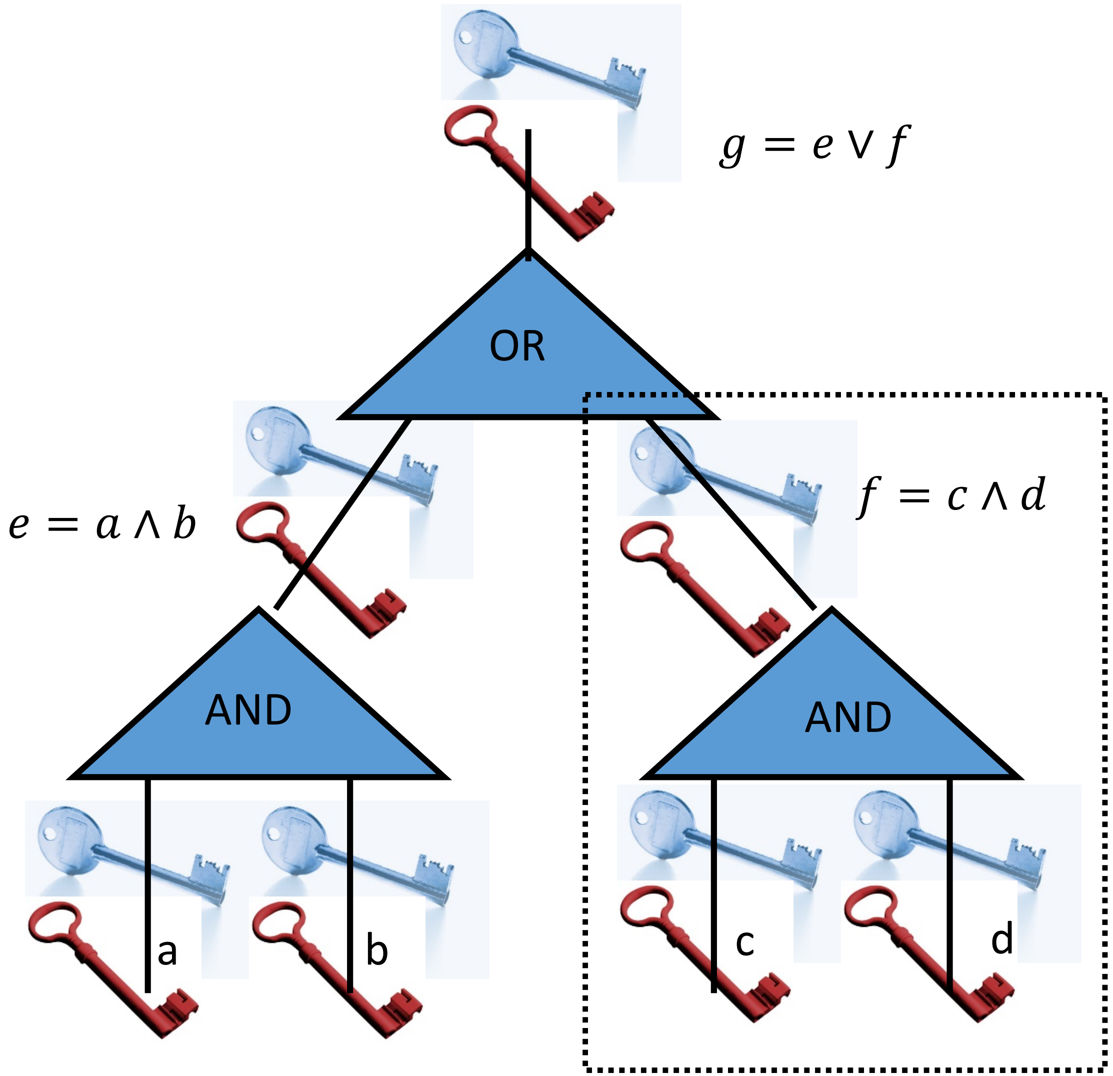
Example: a=0, b=1, c=1,d=1

Alice sent Bob $K_{a,0}$ and $K_{b,1}$

Bob obtains $K_{c,1}$ and $K_{d,1}$ from OTs

Bob uses $K_{d,1}$ and $K_{c,1}$ to obtain

$$K_{f,1} = Dec_{K_{d,1}}\left(Dec_{K_{c,1}}(c_{f,1,1})\right)$$



Example

Alice's Input: a,b

Bob's Input: c,d

$f(a,b,c,d) = (a \wedge b) \vee (c \wedge d)$

Step 6: Bob evaluates the garbled circuit

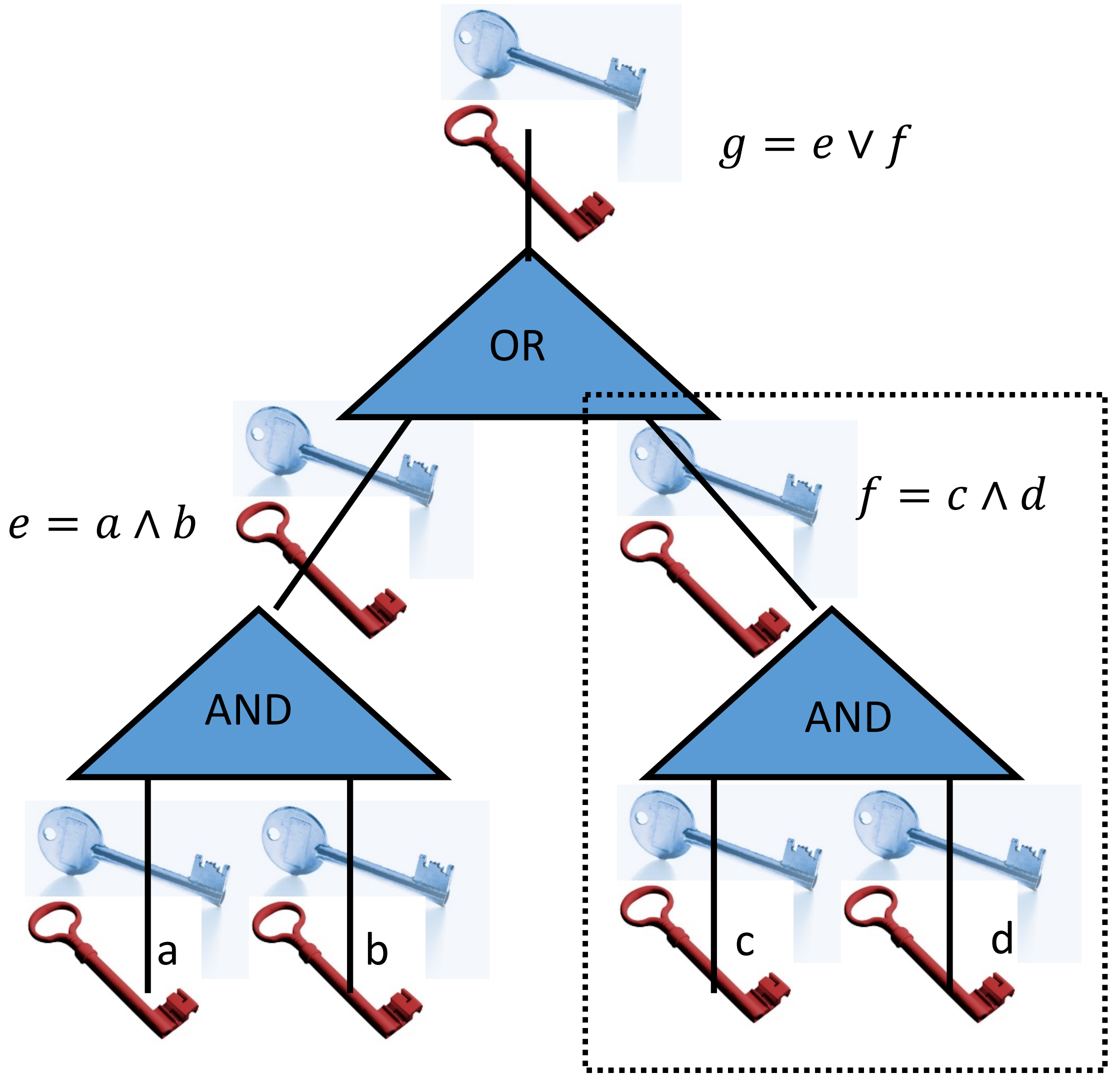
Example: a=0, b=1, c=1,d=1

Alice sent Bob $K_{a,0}$ and $K_{b,1}$

Bob obtains $K_{c,1}$ and $K_{d,1}$ from OTs

Bob uses $K_{e,0}$ and $K_{f,1}$ to obtain

$$K_{g,1} = Dec_{K_{f,1}}\left(Dec_{K_{e,0}}(c_{g,0,1})\right)$$



Example

Alice's Input: a,b

Bob's Input: c,d

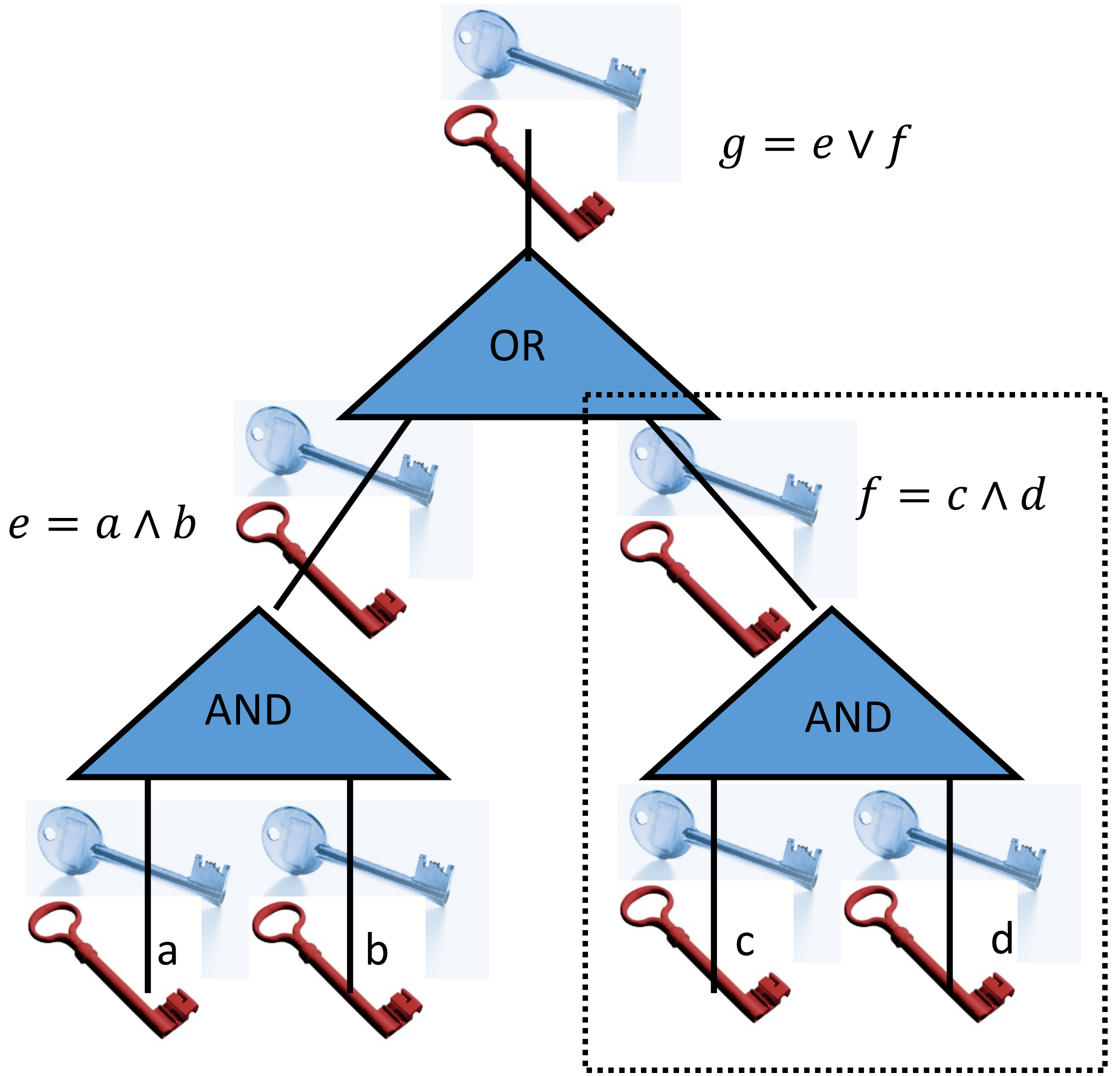
$f(a,b,c,d) = (a \wedge b) \vee (c \wedge d)$

Step 6: Bob→Alices: output key(s) $K_{g,1}$

Step 7: Alice knows the output is g=1

(since she picked $K_{g,0}$ and $K_{g,1}$)

Alice sends the output bit (g=1) back to Bob



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!

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.

Alice's Transcript $\{A_n\}_{n\in\mathbb{N}} \equiv_C \{S_A(n, x, f(x,y)\ )\}_{n\in\mathbb{N}}$

Bob's Transcript $\{B_n\}_{n\in\mathbb{N}} \equiv_C \left\{S_B\left(n, y, f(x,y)\right)\right\}_{n\in\mathbb{N}}$

- **Remark**: Simulator S_A is not given Bob's input (similarly, S_B is not given Alices'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.

Bob's Simulator

- **Simulator Inputs:** $b_1, \dots, b_m$ and $f(a_1, \dots, a_m, b_1, \dots, b_n)$
- Step 1: Simulator picks keys $K_{w,0}$ and $K_{w,1}$ for each wire in circuit C_f
- Step 2: Simulator garbles circuit and outputs (honest) garbled circuit
- Step 3: Simulator outputs keys $K_{a_1,0}$,…, $K_{a_m,0}$
  - this is what Bob would see in real protocol if Alice's input bits are 0's
  - Intuition: Distinguisher cannot tell the difference between $K_{a_1,0}$ and $K_{a_1,1}$ since both keys are picked randomly
- Step 4: Simulator runs OT protocols for each $i \leq n$
  - Sender's (Alice) Inputs: $K_{a_i,0}$ and $K_{a_i,1}$ (known to simulator)
  - Receiver's (Bob) Inputs: b_i

Bob's Simulator

- **Simulator Inputs:** $b_1, \ldots, b_m$ and $f(a_1, \ldots, a_m, b_1, \ldots, b_n)$
- …
- Step 4: Simulator runs OT protocols for each $i \leq n$
  - Sender's (Alice) Inputs: $K_{a_i,0}$ and $K_{a_i,1}$ (known to simulator)
  - Receiver's (Bob) Inputs: b_i
  - Simulator Outputs Bob's transcript from each OT protocol
- Step 5:
  - Let g_i denote value of i^{th} output wire o_i when evaluating $C_f(0, \ldots, 0, b_1, \ldots, b_n)$
  - Simulator outputs the key K_{o_i,g_i} for each output wire
  - Note: evaluating garbled circuit with given input keys yields key K_{o_i,g_i} for each output bit i
- Step 6: Simulator announces output bits $f(a_1, \ldots, a_m, b_1, \ldots, b_n)$
  - Note: These output bits are different than $C_f(0, \ldots, 0, b_1, \ldots, b_n)$
  - Distinguisher cannot tell the difference since the keys $K_{o_i,1-g_i}$ remains hidden (encrypted)
  - $K_{o_i,1-g_i}$ and K_{o_i,g_i} are just random strings

Course Feedback

Course Summary					
Course Code	Course Title	Survey Start Date	Survey End Date	Report Access Start	Response Rate
wl.202120.CS.55500.FNY.18101	Cryptography	4/19/2021 9:00 AM	5/2/2021 11:59 PM	5/12/2021 12:00 AM	0.00% (0/8)

- Your feedback is valuable to me! What did you like about the course? What could be improved? Let me know! I carefully read through any comments after the semester is over.
- Your feedback is anonymous and will not impact your grade (I cannot view your feedback until after grades are entered).

Recap: Yao's Garbled Circuits

- Alice Garbles circuit C to get C'
  - $K_{w,1}$ and $K_{w,0}$: True/False Key for Each wire w in C
  - Encrypted/Permuted Truth Table for each logical gate in C
    - **Example for AND gate:** f=c AND d
    - Given true key $\boldsymbol{K_{c,1}}$ for wire c and false key $\boldsymbol{K_{d,0}}$ for wire d should be able to recover false key $\boldsymbol{K_{f,0}}$ for wire f

$$\boldsymbol{c_{f,1,0} = Enc_{K_{c,1}}\left(Enc_{K_{d,0}}(K_{f,0})\right)}$$

Recap: Yao's Garbled Circuits

- Alice Garbles circuit C to get C'
  - $K_{w,1}$ and $K_{w,0}$: True/False Key for Each wire w in C
  - Encrypted/Permuted Truth Table for each logical gate in C
    - **Example for AND gate:** f=c AND d
    - Given true key $\boldsymbol{K_{c,1}}$ for wire c and false key $\boldsymbol{K_{d,0}}$ for wire d should be able to recover false key $\boldsymbol{K_{f,0}}$ for wire f

$$\boldsymbol{c_{f,1,0} = Enc_{K_{c,1}}\left(Enc_{K_{d,0}}\left(K_{f,0}\right)\right)}$$

- Alice directly sends Bob the relevant key for each of her input wires
- Alice/Bob use Oblivious Transfer so that Bob can learn the relevant keys for his input wires without revealing his inputs to Alice
- Bob can evaluate garbled circuit C' to obtain relevant keys for output wires and send them to Alice
- Alice can determine if each output key corresponds to true/false and send the final output back to Bob
- Protocol is secure in the semi-honest model of computation

Fully Malicious Security?

There is not much Bob can do besides following the protocol i.e., he obtains the garbled circuits + input keys and can only obtain one output key per wire.

What if Alice is malicious and does not follow the protocol?

1. Lie about the output bit(s) in the last step
2. Garble a different circuit C’

    Example: C(x,y)= x AND y while C’(x,y)= x XOR y

      Given C’(x,y) Alice learns Bob’s input (y) directly

      Alice could send back C(x,y), C’(x,y) or something entirely unrelated

Fully Malicious Security?

1. Alice could initially garble the wrong circuit C(x,y)=y.
   1. Example: Change OR gate to an XOR gate
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 inputs (x and y respectively) and the random coins (R_A and R_B respectively) they will use during the protocol:

Let c_A=com(x,R_A;r_A) be Alice's commitment to x, R_A and c_B=com(y, R_B;r_B).

- Alice and Bob can use a tool called zero-knowledge proofs to convince the other party that they are behaving honestly.
- 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?

Fix: Assume Alice and Bob have both committed to their inputs (x and y respectively) and random coins (R_A and R_B respectively):

Let c_A=com(x,R_A;r_A) be Alice's commitment to x, R_A and c_B=com(y, R_B;r_B).

- Alice and Bob can use a tool called zero-knowledge proofs to convince the other party that they are behaving honestly.
  - **Example**: After sending a her first message (A) Alice proves that the message m she just sent is the same message an honest party would have sent
  - Alice wants to convince Bob that there exists x, R_A and r_A s.t. 1) c_A=com(x,R_A;r_A) and 2) m is the message that would be produced if Alice is honest and runs with inputs x and R_A
  - Alice also does not want to reveal x or R_A to Bob!
  - Is this possible?
  - Yes! Tool = Zero-Knowledge Proofs!

Fully Malicious Security

- Assume Alice and Bob have both committed to their input: c_A=com(x,R_A;r_A) and c_B=com(y,R_B;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(C_f; R_A).
   1. Alice sends C'_f to Bob.
   2. Alice convinces Bob that C'_f = GarbleCircuit(C_f, ; R_A) (using a zero-knowledge proof)
2. Similarly, Bob/Alice can convince each-other that the OT protocols are run honestly (additional ZK proofs)
3. Alice can convince Bob that the final output bit(s) correspond to the keys that Alice sent (additional ZK proofs)

CS 555:Week 15: Zero-Knowledge Proofs

Computational Indistinguishability

- Consider two distributions X_ℓ and Y_ℓ (e.g., over strings of length ℓ).
- Let D be a distinguisher that attempts to guess whether a string s came from distribution X_ℓ or Y_ℓ.

The advantage of a distinguisher D is

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

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

$$Adv_{D,n} \leq negl(n)$$

Computational Indistinguishability

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

- Consider two d ℓ).
- Let D be a disti came from distribution X_ℓ

The advantage of a distinguisher D is

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

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

$$Adv_{D,n} \le negl(n)$$

P vs NP

- **P** decision problems that can be solved in polynomial time

- **NP** --- decision problems whose solutions can be **verified** in polynomial time
  - Examples: SHORT-PATH, COMPOSITE, 3SAT, CIRCUIT-SAT, 3COLOR,
  - DDH
    - **Input:** $A = g^{x_1}, \mathrm{B} = g^{x_2}$ and Z
    - **Goal:** Decide if $\mathrm{Z} = g^{x_1 x_2}$ or $\mathrm{Z} \neq g^{x_1 x_2}$.
  - **NP-Complete** --- hardest problems in NP (e.g., all problems can be reduced to 3SAT)
- **Witness**
  - A short (polynomial size) string which allows a verify to check for membership
  - DDH Witness: x_1,x_2.

Zero-Knowledge Proof

Two parties: Prover P (PPT) and Verifier V (PPT)

(P is given witness for claim e.g., w=(x_1,x_2) is a witness that $A = g^{x_1}$, $\mathrm{B} = g^{x_2}$ and $\mathrm{Z} = g^{x_1 x_2}$ is a DDH tuple)

- **Completeness:** If claim is true honest prover can always convince honest verifier to accept the proof.
- **Soundness:** If claim is false then Verifier should reject with probability at least ½. (Even if the prover tries to cheat)
- **Zero-Knowledge:** Verifier doesn't learn anything about prover's input from the protocol (other than that the claim is true).
- Formalizing this last statement is tricky
- **Zero-Knowledge:** should hold even if the attacker is dishonest!

Zero-Knowledge Proof

Trans(1^n,V',P,x,w,r_p,r_v) transcript produced when V' and P interact

- V' is given input X (the problem instance e.g., $X = g^x$)
- P is given input X and w (a witness for the claim e.g., w=x)
- V' and P use randomness r_p and r_v respectively
- Security parameter is n e.g., for encryption schemes, commitment schemes etc…

$\boldsymbol{X_n} =$ **Trans(1^n,V',P,x,w)** is a distribution over transcripts (over the randomness r_p,r_v)

(Blackbox Zero-Knowledge): There is a PPT simulator S such that for every V' (possibly cheating) S, with oracle access to V', can simulate X_n without a witness w. Formally,

$$\{X_n\}_{n\in\mathbb{N}} \equiv_C \left\{S^{V'(.)}(x, 1^n)\right\}_{n\in\mathbb{N}}$$

Zero-Knowledge Proof

Trans(1^n,V',P,x,w,r_p,r_v) transcript produced when V' and P interact

- V' is given input x (the problem instance e.g., $A = g^{x_1}$, $B = g^{x_2}$ and z_b)
- P ... r the claim e.g., ...
- V' ... espectively
- Se ... yption schemes, ...

X_n ... n over transcript

Simulator S is not given witness w

Oracle V'(x,trans) will output the next message V' would output given current transcript trans

(Blackbox Zero-Knowledge): There is a PPT simulator S such that for every V' (possibly cheating) S, with oracle access to V', can simulate X_n without a witness w. Formally,

$$\{X_n\}_{n\in\mathbb{N}} \equiv_C \{S^{V'(.)}(x, 1^n)\}_{n\in\mathbb{N}}$$

Zero-Knowledge Proof for Discrete Log Solution

Bob (verifier);
A

$$\boldsymbol{B = g^{y}, C = g^{x+y}}$$

$$\boldsymbol{challenge\ c \in \{0,1\}}$$

Response $\boldsymbol{r} = \begin{cases} \boldsymbol{y} & \boldsymbol{if\ c = 0} \\ \boldsymbol{y} + x & if\ c = 1 \end{cases}$

$$\boldsymbol{Decision\ d} = \begin{cases} \boldsymbol{1} & \boldsymbol{if\ c = 0\ and\ B = g^{r}\ \ and\ AB = C} \\ \boldsymbol{1} & \boldsymbol{if\ c = 1\ and\ C = g^{r}}\ \ and\ \boldsymbol{AB = C} \\ 0 & otherwise \end{cases}$$

Alice (prover);
x s.t.
$A = g^{x}$,
$B = g^{y}$,
(random y)

Claim: There is some integer x such that $A = g^{x}$

Zero-Knowledge Proof for Discrete Log Solution

$$\boldsymbol{B = g^y, C = g^{x+y}}$$

$$\boldsymbol{challenge\ c \in \{0,1\}}$$

Response $\boldsymbol{r} = \begin{cases} \boldsymbol{y} & \boldsymbol{if\ c = 0} \\ \boldsymbol{y} + x & if\ c = 1 \end{cases}$

Bob (verifier);
$A = g^x,$

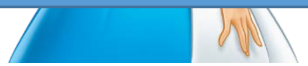
$$\boldsymbol{Decision\ d} = \begin{cases} \boldsymbol{1} & \boldsymbol{if\ c = 0\ and\ B = g^r\ \ and\ AB = C} \\ \boldsymbol{1} & \boldsymbol{if\ c = 1\ and\ C = g^r}\ \ \boldsymbol{and\ AB = C} \\ 0 & otherwise \end{cases}$$

Alice (prover);
x
$A = g^x,$
$B = g^y,$
(random y)

Correctness: If Alice and Bob are honest then Bob will always accept

Zero-Knowledge Proof for Discrete Log Solution

Case 1: Challenge (c=0)



$$\boxed{\boldsymbol{B = g^{y}}}, \boldsymbol{C = g^{x+y}}$$

$$\boldsymbol{challenge\ c} \in \{\mathbf{0}, \mathbf{1}\}$$

Response $\boldsymbol{r} = \begin{cases} \boxed{\boldsymbol{y}} & \boldsymbol{if\ c = 0} \\ \boldsymbol{y} + x & if\ c = 1 \end{cases}$

$$\boldsymbol{Decision\ d} = \begin{cases} \mathbf{1} & \boldsymbol{if\ c = 0\ and}\ \boxed{\boldsymbol{B = g^{r}\ \ and\ AB = C}} \\ \mathbf{1} & \boldsymbol{if\ c = 1\ and\ C = g^{r}}\ \ \boldsymbol{and\ AB = C} \\ 0 & otherwise \end{cases}$$

Bob (verifier);
$A = g^{x}$,

Alice (prover);
x
$A = g^{x}$,
$B = g^{y}$,
(random y)

Correctness: If Alice and Bob are honest then Bob will always accept

Zero-Knowledge Proof for Discrete Log Solution

Case 2: Challenge (c=1)

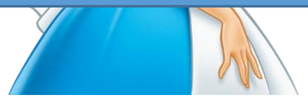
Bob (verifier);
$A = g^x,$

$$\boldsymbol{B = g^y}, \boxed{\boldsymbol{C = g^{x+y}}}$$

$$\boldsymbol{challenge\ c \in \{0, 1\}}$$

$$\textbf{Response } \boldsymbol{r} = \begin{cases} \boldsymbol{y} & \boldsymbol{if\ c = 0} \\ \boxed{\boldsymbol{y} + x} & if\ c = 1 \end{cases}$$

$$\boldsymbol{Decision\ d} = \begin{cases} \boldsymbol{1} & \boldsymbol{if\ c = 0\ and\ B = g^r\ \ and\ AB = C} \\ \boldsymbol{1} & \boldsymbol{if\ c = 1\ and}\ \boxed{\boldsymbol{C = g^r}\ \ and\ \boldsymbol{AB = C}} \\ 0 & otherwise \end{cases}$$



Alice (prover);
x
$A = g^x,$
$B = g^y,$
(random y)

Correctness: If Alice and Bob are honest then Bob will always accept

Zero-Knowledge Proof for Discrete Log Solution

$$\boldsymbol{B = g^y, C = g^{x+y}}$$

$\boldsymbol{challenge\ c} \in \{\mathbf{0, 1}\}$

Response $\boldsymbol{r} = \begin{cases} \boldsymbol{y} & \boldsymbol{if\ c = 0} \\ \boldsymbol{y} + x & if\ c = 1 \end{cases}$

$$\boldsymbol{Decision\ d} = \begin{cases} \mathbf{1} & \boldsymbol{if\ c = 0\ and\ B = g^r\ \ and\ AB = C} \\ \mathbf{1} & \boldsymbol{if\ c = 1\ and\ C = g^r}\ \ \boldsymbol{and\ AB = C} \\ 0 & otherwise \end{cases}$$

Bob (verifier);
$A = g^x,$

Alice (prover);
x
$A = g^x,$
$B = g^y,$
(random y)

Soundness: If $A \neq g^x$ for some x then (honest) Bob will reject w.p. ½ (even if Alice cheats)

Zero-Knowledge Proof fo

Assume that AB=C, now
If $B = g^y$ and $C = g^{x+y}$ for some x,y then $A = g^x$

$$B = g^y, C = g^{x+y}$$

$challenge\ c \in \{0,1\}$

Response $r = \begin{cases} y & if\ c = 0 \\ y + x & if\ c = 1 \end{cases}$

$$Decision\ d = \begin{cases} 1 & if\ c = 0\ and\ B = g^r\ and\ AB = C \\ 1 & if\ c = 1\ and\ C = g^r\ and\ AB = C \\ 0 & otherwise \end{cases}$$

Bob (verifier);
$A = g^x$,

Alice (prover);
x
$A = g^x$,
$B = g^y$,
(random y)

Soundness: If $A \neq g^x$ for some x then (honest) Bob will reject w.p. ½ (even if Alice cheats)

Case 1: for all r $B \neq g^r$

$\rightarrow Pr[reject] \geq Pr[c = 0] = \frac{1}{2}$

Assume that AB=C, now
If $B = g^y$ and $C = g^{x+y}$ for some x,y then $A = g^x$

$\boldsymbol{B = g^y, C = g^{x+y}}$

$\boldsymbol{challenge\ c \in \{0,1\}}$

Response $r = \begin{cases} y & if\ c = 0 \\ y + x & if\ c = 1 \end{cases}$

$Decision\ d = \begin{cases} 1 & if\ \boxed{c = 0\ and\ B = g^r}\ and\ AB = C \\ 1 & if\ c = 1\ and\ C = g^r\ and\ AB = C \\ 0 & otherwise \end{cases}$

Bob (verifier);
$A = g^x$,

Alice (prover);
x
$A = g^x$,
$B = g^y$,
(random y)

Soundness: If $A \neq g^x$ for some x then (honest) Bob will reject w.p. ½ (even if Alice cheats)

Case 2: for all r $C \neq g^r$

$$\rightarrow Pr[reject] \geq Pr[c = 1] = \frac{1}{2}$$

Assume that AB=C, now

If $B = g^y$ and $C = g^{x+y}$ for some x,y then $A = g^x$

$$B = g^y, C = g^{x+y}$$

$$challenge\ c \in \{0, 1\}$$

Response $r = \begin{cases} y & if\ c = 0 \\ y + x & if\ c = 1 \end{cases}$

Bob (verifier);
$A = g^x$,

$$Decision\ d = \begin{cases} 1 & if\ c = 0\ and\ B = g^r\ and\ AB = C \\ 1 & if\ c = 1\ and\ C = g^r\ and\ AB = C \\ 0 & otherwise \end{cases}$$

Alice (prover);
x
$A = g^x$,
$B = g^y$,
(random y)

Soundness: If $A \neq g^x$ for some x then (honest) Bob will reject w.p. ½ (even if Alice cheats)

Zero-Knowledge Proof for Discrete Log Solution

$$\boldsymbol{B = g^y, C = g^{x+y}}$$

$\boldsymbol{challenge\ c = V'(A,(B,C)) \in \{0,1\}}$

Response $\boldsymbol{r} = \begin{cases} \boldsymbol{y} & \boldsymbol{if\ c = 0} \\ y + x & if\ c = 1 \end{cases}$

$\boldsymbol{Decision\ d = V'(A,(B,C),c,r)}$

Dishonest (verifier);
$A = g^x,$

Alice (honest);
x
$A = g^x,$
$B = g^y,$
(random y)

Transcript: $\boldsymbol{View_{V'}} = (A,(B,C),c,r,d)$

Zero-Knowledge Proof for Discrete Log Solution

Dishonest (verifier);
$A = g^x,$

$$\boldsymbol{B = g^y, C = g^{x+y}}$$

$$\boldsymbol{challenge\ c = V'(A, (B, C)) \in \{0, 1\}}$$

Response $r = \begin{cases} \boldsymbol{y} & \boldsymbol{if\ c = 0} \\ y + x & if\ c = 1 \end{cases}$

$$\boldsymbol{Decision\ d = V'(A, (B, C), c, r)}$$

Alice (honest);
x
$A = g^x,$
$B = g^y,$
(random y)

Zero-Knowledge: For all PPT V' exists PPT Sim s.t $\boldsymbol{View_{V'}} \equiv_C \mathrm{Sim}^{V'(.)}(A)$

Zero-Knowledge Proof for Discrete Log Solution

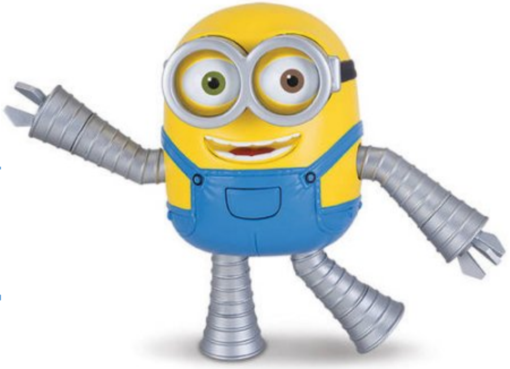
$$\begin{cases} B = g^y, C = AB & \text{if b=0} \\ B = \frac{C}{A}, C = g^y \; otherwise \end{cases}$$

$challenge\ c = V'(A,(B,C)) \in \{0,1\}$

Response $r = \begin{cases} y & if\ c{=}b \\ \perp & otherwise \end{cases}$

$Decision\ d = V'(A,(B,C),c,r)$

Dishonest (verifier);
$A = g^x,$



Simulator
Cheat bit b,
$A = g^x,$
$B = g^y,$
(random y)

Zero-Knowledge: For all PPT V' exists PPT Sim s.t $\boldsymbol{View}_{V'} \equiv_C \mathrm{Sim}^{V'(.)}(A)$

Zero-Knowledge Proof for Discrete Log Solution

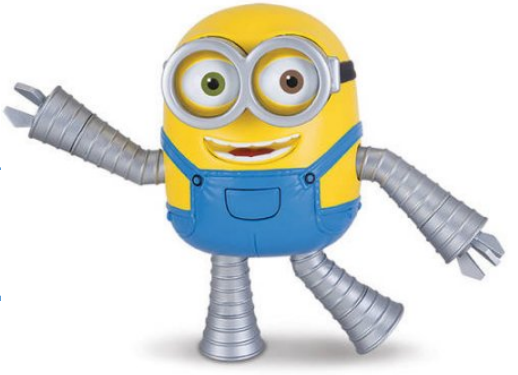
$$\begin{cases} B = g^y, C = AB & \text{if b=0} \\ B = \dfrac{C}{A}, C = g^y \; otherwise & \end{cases}$$

$challenge\ c = V'(A, (B, C)) \in \{0, 1\}$

Response $r = \begin{cases} y & if\ c{=}b \\ \perp & otherwise \end{cases}$

Dishonest (verifier);
$A = g^x,$

$Decision\ d = V'(A, (B, C), c, r)$



Simulator
$Cheat\ bit\ b,$
$A = g^x,$
$B = g^y,$
(random y)

Zero-Knowledge: Simulator can produce identical transcripts (Repeat until $r \neq \perp$)

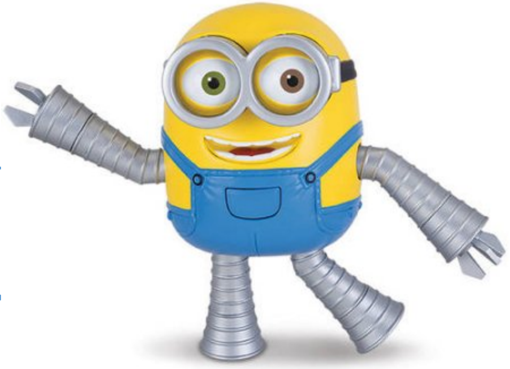
Zero-Knowledge Proof for Discrete Log Solution

$$\begin{cases} B = g^y, C = AB & \text{if b=0} \\ B = \frac{C}{A}, C = g^y \text{ otherwise} \end{cases}$$

$\textit{challenge } c = V'(A, (B, C)) \in \{0, 1\}$

Response $r = \begin{cases} y & \textit{if c=b} \\ \perp & \textit{otherwise} \end{cases}$

$\textit{Decision } d = V'(A, (B, C), c, r)$



Dishonest (verifier);
$A = g^x,$

Simulator
Cheat bit b,
$A = g^x,$
$B = g^y,$
(random y)

Zero-Knowledge: If $A = g^x$ for some x then $\boldsymbol{View}_{V'} \equiv_C \mathrm{Sim}^{V'(.)}(A)$

Zero-Knowledge Proof for Square Root mod N

$$M = zy^2 \; mod \; N$$

$$\boldsymbol{challenge\ c} \in \{\mathbf{0}, \mathbf{1}\}$$

Response $\boldsymbol{r} = \begin{cases} \boldsymbol{y} & \boldsymbol{if\ c = 0} \\ \boldsymbol{y}x & if\ c = 1 \end{cases}$

$$\boldsymbol{Decision\ d} = \begin{cases} \mathbf{1} & \boldsymbol{if\ c = 0\ and\ M = zr^2} \\ \mathbf{1} & \boldsymbol{if\ c = 1\ and\ M = r^2}\ mod\ N \\ 0 & otherwise \end{cases}$$

Bob (verifier);

z

Alice (prover);

X

$z = x^2$ mod N

(random y)

Completeness: If Alice knows x such $z = x^2$ mod N then Bob will always accept

Zero-Knowledge Proof for Square Root mod N

Bob (verifier);

z

$$M = zy^2 \; mod \; N$$

$$\boldsymbol{challenge\ c} \in \{\mathbf{0}, \mathbf{1}\}$$

Response $\boldsymbol{r} = \begin{cases} \boldsymbol{y} & \boldsymbol{if\ c = 0} \\ \boldsymbol{y}x & if\ c = 1 \end{cases}$

$$\boldsymbol{Decision\ d} = \begin{cases} \mathbf{1} & \boldsymbol{if\ c = 0\ and\ M = zr^2} \\ \mathbf{1} & \boldsymbol{if\ c = 1\ and\ M = r^2}\ mod\ N \\ 0 & otherwise \end{cases}$$

Alice (prover);

X

$z = x^2$ mod N

(random y)

Soundness: If $z \neq x^2$ for some x then (honest) Bob will reject w.p. ½ (even if Alice cheats)

Zero-Knowledge Proof for Square Root mod N

Bob (verifier);

z

$$M = zy^2 \; mod \; N$$

$$\boldsymbol{challenge \; c} \in \{\mathbf{0}, \mathbf{1}\}$$

Response $\boldsymbol{r} = \begin{cases} \boldsymbol{y} & \boldsymbol{if \; c = 0} \\ yx & if \; c = 1 \end{cases}$

$$\boldsymbol{Decision \; d} = \begin{cases} \mathbf{1} & \boldsymbol{if \; c = 0 \; and \; M = zr^2} \text{ mod N} \\ \mathbf{1} & \boldsymbol{if \; c = 1 \; and \; M = r^2} \text{ mod N} \\ 0 & otherwise \end{cases}$$

Alice (prover);

X

$z = x^2$ mod N

(random y)

Zero-Knowledge: How does the simulator work?

Zero-Knowledge Proof vs. Digital Signature

- Digital Signatures are transferrable
  - E.g., Alice signs a message m with her secret key and sends the signature σ to Bob. Bob can then send (m, σ) to Jane who is convinced that Alice signed the message m.

- Are Zero-Knowledge Proofs transferable?
  - Suppose Alice (prover) interacts with Bob (verifier) to prove a statement (e.g., z has a square root modulo N) in Zero-Knowledge.
  - Let $\boldsymbol{View_V}$ be Bob's view of the protocol.
  - Suppose Bob sends $\boldsymbol{View_V}$ to Jane.
  - Should Jane be convinced of the statement (e.g., z has a square root modulo N)>

Non-Interactive Zero-Knowledge Proof (NIZK)

$M_1, \ldots M_k$ where $M_i = y_i^2 z \; mod \; N$

$\boldsymbol{challenges\; c = (c_1, \ldots, c_k) = H}(M_1, \ldots M_k)$

Responses $r_1, \ldots, r_k$ **where** $\boldsymbol{r_i} = \begin{cases} \boldsymbol{y_i} & \boldsymbol{if\; c_i = 0} \\ \boldsymbol{y_i} x & if\; c_i = 1 \end{cases}$

Bob (verifier);

z

$$\boldsymbol{Decision\; d} = \prod_i \boldsymbol{d_i} \;\; \boldsymbol{where\; d_i} = \begin{cases} \mathbf{1} & \boldsymbol{if\; c_i = 0\; and\; M_i = r_i^2 z} \bmod \mathrm{N} \\ \mathbf{1} & \boldsymbol{if\; c_i = 1\; and\; M_i = r_i^2} \bmod \mathrm{N} \\ 0 & otherwise \end{cases}$$

Alice (prover);

X

$z = x^2 \bmod \mathrm{N}$

(random $y_1, \ldots, y_k$)

Simulator Power: Can program the random oracle

NIZK Security (Random Oracle Model)

- Simulator is given statement to proof (e.g., z has a square root modulo N)
- Simulator must output a proof π'_z and a random oracle H'

- Distinguisher D
  - World 1 (Simulated): Given z, π'_z and oracle access to H'
  - World 2 (Honest): Given z, π_z (honest proof) and oracle access to H
  - Advantage: $\mathrm{ADV_D} = |Pr[D^H(\text{z}, \pi_z) = 1] - Pr[D^{H'}(\text{z}, \pi'_z) = 1]|$
- **Zero-Knowledge:** Any PPT distinguisher D should have negligible advantage.
- NIZK proof π_z is transferrable (contrast with interactive ZK proof)

Σ-Protocols

- Prover Input: instance/claim x and witness w
- Verifier Input: Instance x
- Σ-Protocols: three-message structure
  - Prover sends first message $m=P_1(x,w;\ r_1)$
  - Verifier responds with random challenge c
  - Prover sends response $R=P_2(x,w,r_1,c;\ r_2)$
  - Verifier outputs decision $V(x,m,c,R)$
  - **Completeness:** If w is a valid witness for instance x then $\Pr[V(x,c,R)=1]=1$
  - **Soundness:** If the claim x is false then $V(x,c,R)=0$ with probability at least ½
  - **Zero-Knowledge:** Simulator can produce computationally indistinguishable transcript

Σ-Protocols and Fiat-Shamir Transform

- Convert Σ-Protocols into Non-Interactive ZK Proof
- Prover Input: instance/claim x and witness w
- Verifier Input: Instance x
- **Step 1:** Prover generates first messages for n instances of the protocol
  - $m_i = P_1(x,w;\ r_i)$ for each i=1 to n
- **Step 2:** Prover uses random oracle to extract random coins $z_j=H(x,j,\ m_1,\ldots,m_n)$ for j=1 to n
  - Prover samples challenges $c_1,\ldots,c_n$ using random strings $z_1,\ldots,z_n$ i.e., $c_i=\text{SampleChallenge}(z_i)$
- **Step 3:** Prover computes responses $R_1,\ldots,R_n$
  - $R_i \leftarrow P_2(x,w,r_i,c_i)$
- **Step 4:** Prover outputs the proof $\{(m_i, c_i, z_i)\}_{i\le n}$

Σ-Protocols and Fiat-Shamir Transform

- **Step 1:** Prover generates first messages for n instances of the protocol
  - $m_i = P_1(x,w; r_i)$ for each i=1 to n
- **Step 2:** Prover uses random oracle to extract random coins $z_i=H(x,i, m_1,\ldots,m_n)$ for i=1 to n
  - Prover samples challenges $c_1,\ldots,c_n$ using random strings $z_1,\ldots,z_n$ i.e., c_i=SampleChallenge(z_i)
- **Step 3:** Prover computes responses $R_1,\ldots,R_n$
  - $R_i \leftarrow P_2(x,w,r_i,c_i)$
- **Step 4:** Prover outputs the proof $\pi = \{(m_i, c_i, R_i)\}_{i \le n}$

Verifier: $V_{NI}(x,\pi)$ check that for all $i \le n$

1. **V(x, (m_i, c_i, R_i))=1 and**
2. c_i=SampleChallenge(z_i) **where** $z_i=H(x,i, m_1,\ldots,m_n)$

Σ-Protocols and Fiat-Shamir Transform

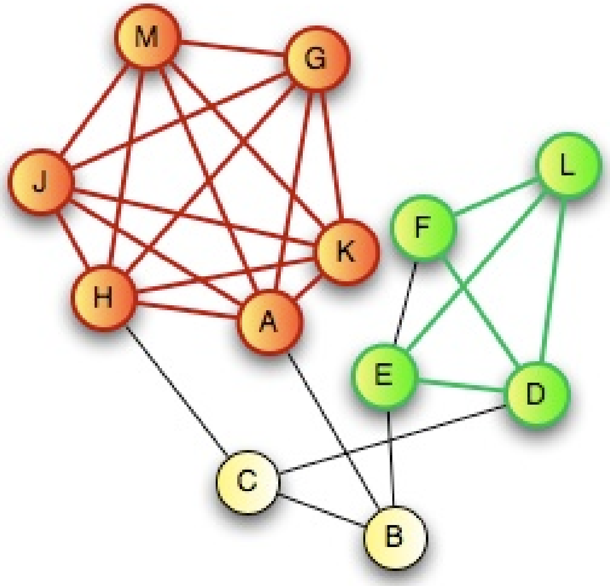
- **Step 1:** Prover generates first messages for n instances of the protocol
  - $m_i = P_1(x,w; r_i)$ for each i=1 to n
- **Step 2:** Prover uses random oracle to extract random coins $z_i=H(x,i, m_1,\ldots,m_n)$ for i=1 to n
  - Prover samples challenges $c_1,\ldots,c_n$ using random strings $z_1,\ldots,z_n$ i.e., c_i=SampleChallenge(z_i)
- **Step 3:** Prover computes responses $R_1,\ldots,R_n$
  - $R_i \leftarrow P_2(x,w,r_i,c_i)$
- **Step 4:** Prover outputs the proof $\pi = \{(m_i, c_i, R_i)\}_{i \le n}$

Zero-Knowledge (Idea):

Step 1: Run simulator for Σ n-times to obtain n transcripts (m_i, c_i, R_i) for each $i \le n$.

Step 2: Program the random oracle so that $H(x,i, m_1,\ldots,m_n)=z_i$ where c_i=SampleChallenge(z_i)

Zero-Knowledge Proof for all NP

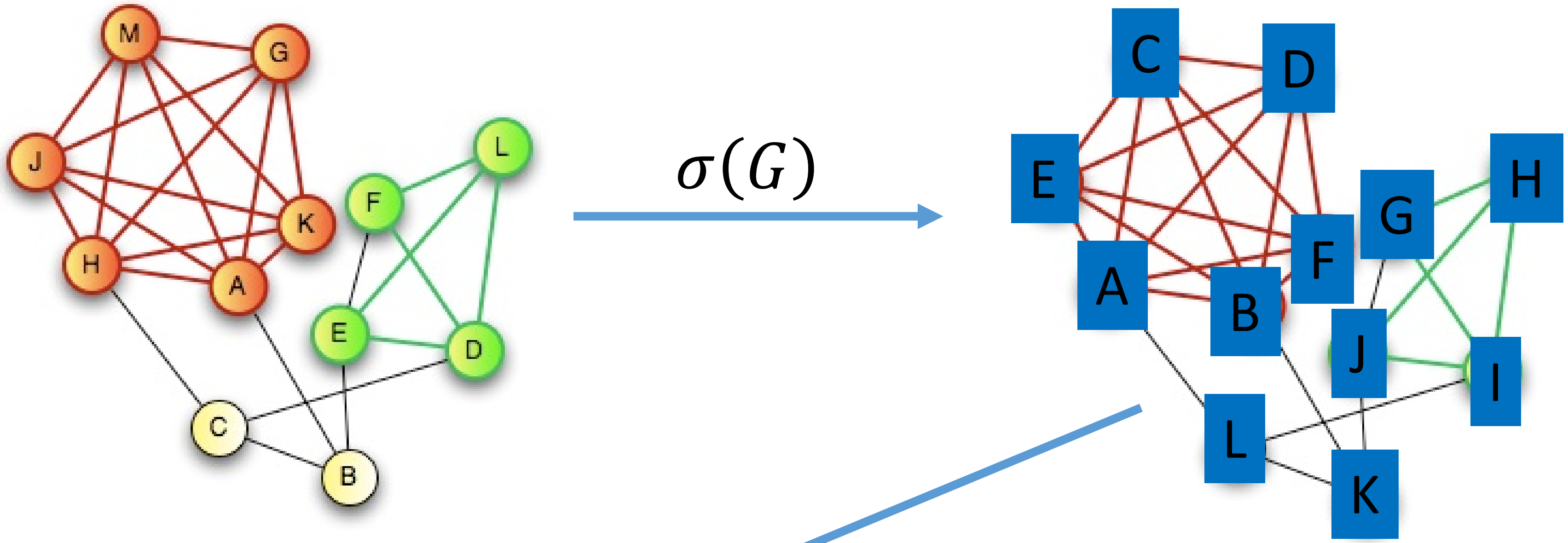


- CLIQUE
  - Input: Graph G=(V,E) and integer k>0
  - Question: Does G have a clique of size k?

- CLIQUE is NP-Complete
  - Any problem in NP reduces to CLIQUE
  - A zero-knowledge proof for CLIQUE yields proof for all of NP via reduction

- Prover:
  - Knows k vertices $v_1,\ldots,v_k$ in G=(V,E) that form a clique

Zero-Knowledge Proof for all NP



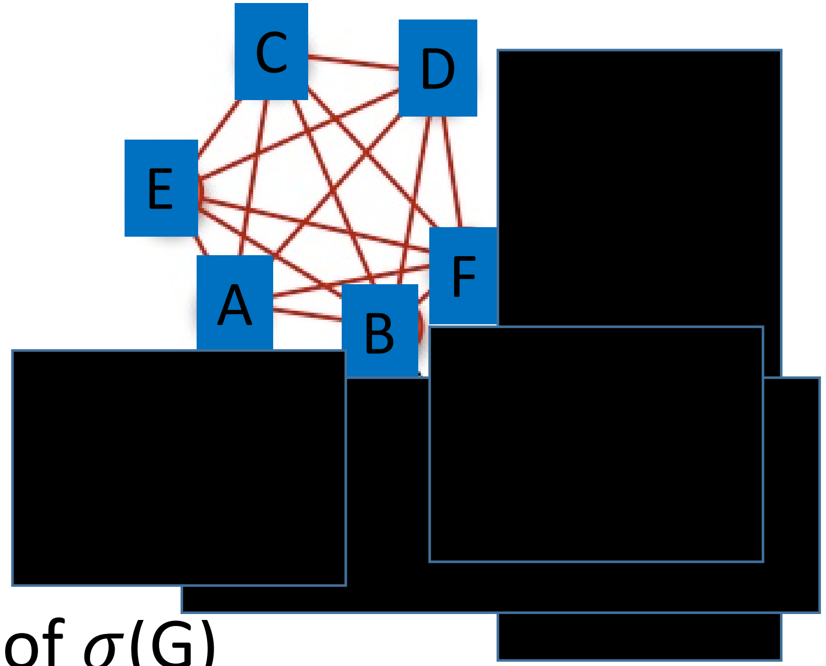
Adjacency matrix $A_{\sigma(G)}$

$$\begin{array}{c} \\ A \\ \\ L \end{array} \begin{array}{c} \begin{array}{ccc} A & & L \end{array} \\ \begin{pmatrix} 0 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 0 \end{pmatrix} \end{array}$$

Commitment to $A_{\sigma(G)}$

$$\begin{array}{c} \\ A \\ \\ L \end{array} \begin{array}{c} \begin{array}{ccc} A & & L \end{array} \\ \begin{pmatrix} Com(0, r_{A,A}) & \cdots & Com(1, r_{A,L}) \\ \vdots & \ddots & \vdots \\ Com(1, r_{L,A}) & \cdots & Com(0, r_{L,L}) \end{pmatrix} \end{array}$$

Zero-Knowledge Proof for all NP



- Prover:
  - Knows k vertices $v_1,\ldots,v_k$ in G=(V,E) that for a clique

1. Prover commits to a permutation σ over V
2. Prover commits to the adjacency matrix $A_{\sigma(G)}$ of σ(G)
3. Verifier sends challenge c (either 1 or 0)
4. If c=0 then prover reveals σ and adjacency matrix $A_{\sigma(G)}$
    1. Verifier confirms that adjacency matrix is correct for σ(G)
5. If c=1 then prover reveals the submatrix formed by first rows/columns of $A_{\sigma(G)}$ corresponding to $\sigma(v_1), \ldots, \sigma(v_k)$
    1. Verifier confirms that the submatrix forms a clique.

Zero-Knowledge Proof for all NP

- **Completeness**: Honest prover can always make honest verifier accept
- **Soundness**: If prover commits to adjacency matrix $A_{\sigma(G)}$ of σ(G) and can reveal a clique in submatrix of $A_{\sigma(G)}$ then G itself contains a k-clique. Proof invokes binding property of commitment scheme.
- **Zero-Knowledge:** Simulator cheats and either commits to wrong adjacency matrix or cannot reveal clique. Repeat until we produce a successful transcript. Indistinguishability of transcripts follows from hiding property of commitment scheme.

Secure Multiparty Computation (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
  - Prove: My behavior in the protocol is consistent with honest party