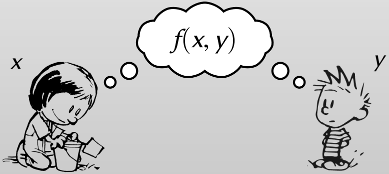
Secure Computation & Yao's Protocol

Mike Rosulek
Oregon State University OSU

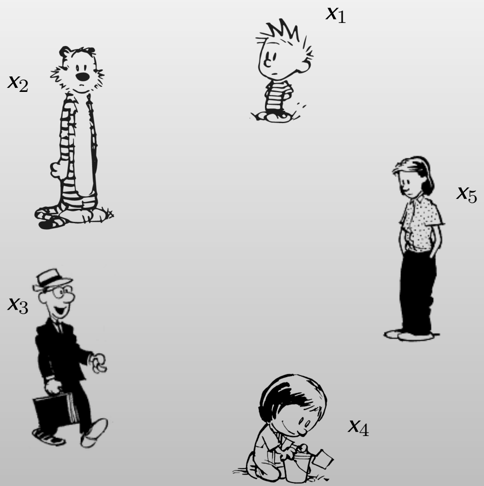
crypt@b-it 2018



Roadmap

1. **Secure computation:** Concepts & definitions
2. **Yaos' protocol:** semi-honest secure computation for boolean circuits

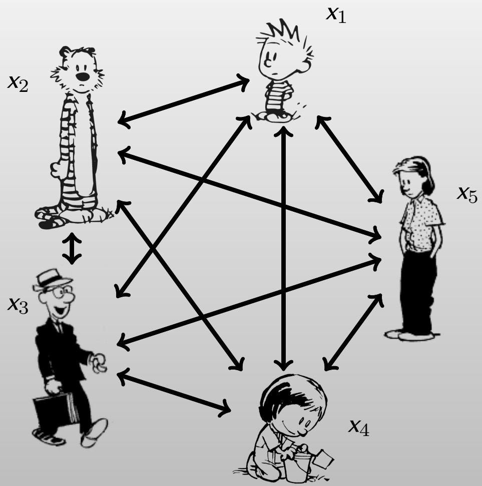
Secure computation



Premise:

- Mutually distrusting parties, each with a private input

Secure computation

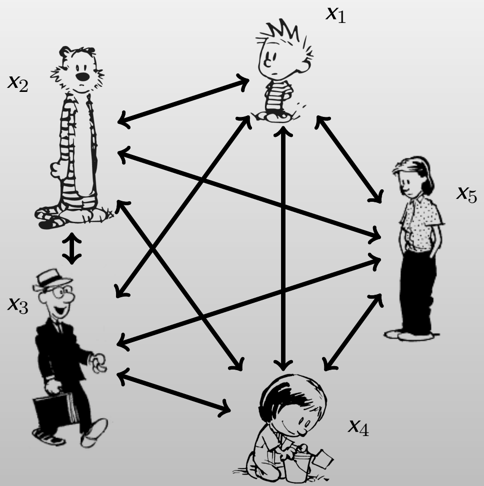


$$\therefore f(x_1, x_2, x_3, x_4, x_5)$$

Premise:

- Mutually distrusting parties, each with a private input
- Learn the result of agreed-upon computation
- *Ex:* election, auction, etc.

Secure computation



$$\therefore\ f(x_1, x_2, x_3, x_4, x_5)$$

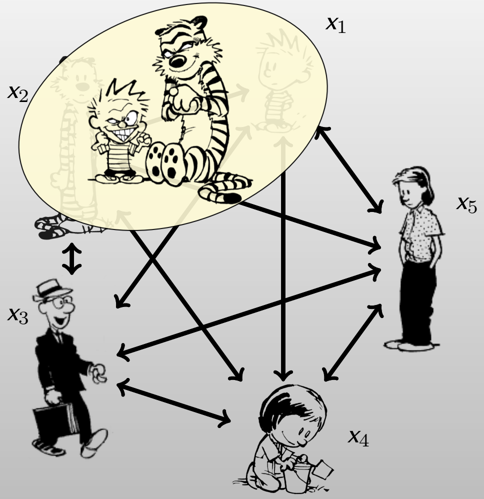
Premise:

- Mutually distrusting parties, each with a private input
- Learn the result of agreed-upon computation
- *Ex:* election, auction, etc.

Security guarantees:

- Privacy ("learn no more than" prescribed output)
- Input independence
- Output consistency, etc..

Secure computation



$$\therefore f(x_1, x_2, x_3, x_4, x_5)$$

Premise:

- Mutually distrusting parties, each with a private input
- Learn the result of agreed-upon computation
- *Ex:* election, auction, etc.

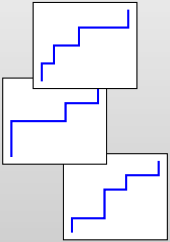
Security guarantees:

- Privacy ("learn no more than" prescribed output)
- Input independence
- Output consistency, etc..

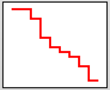
..even if some parties cheat, collude!

Examples: Sugar Beets

Beet Farmers



DANISCO

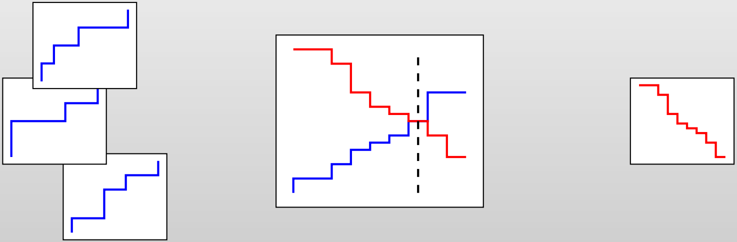


- Farmers make bids ("at price *X*, I will produce *Y* amount")
- Purchaser bids ("at price *X*, I will buy *Y* amount")

Examples: Sugar Beets

Beet Farmers

DANISCO

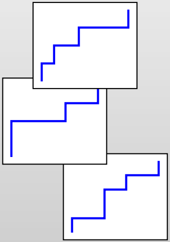


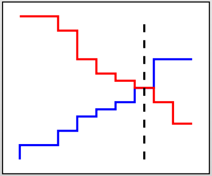
- Farmers make bids ("at price X, I will produce Y amount")
- Purchaser bids ("at price X, I will buy Y amount")
- **Market clearing price** (MCP): price at which total supply = demand

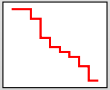
Examples: Sugar Beets

Beet Farmers

DANISCO







- Farmers make bids (“at price *X*, I will produce *Y* amount”)
- Purchaser bids (“at price *X*, I will buy *Y* amount”)
- **Market clearing price** (MCP): price at which total supply = demand
- 2009: MCP (+ bids at that price) computed via secure computation

Examples: Ad conversion

Ad impressions
alice@gmail.com
bob@gmail.com
charlie@gmail.com
dianne@gmail.com
edwin@gmail.com
frank@gmail.com
gina@gmail.com

In-store purchases	
albert@gmail.com	$80K
bob@gmail.com	**$160K**
caroline@gmail.com	$99K
edwin@gmail.com	**$99K**
felipe@gmail.com	$85K
frank@gmail.com	**$77K**
hilda@gmail.com	$113K

Examples: Ad conversion

Ad impressions
alice@gmail.com
bob@gmail.com
charlie@gmail.com
dianne@gmail.com
edwin@gmail.com
frank@gmail.com
gina@gmail.com

In-store purchases	
albert@gmail.com	$80K
bob@gmail.com	**$160K**
caroline@gmail.com	$99K
edwin@gmail.com	**$99K**
felipe@gmail.com	$85K
frank@gmail.com	**$77K**
hilda@gmail.com	$113K

```
SELECT  SUM(amount)
FROM    ads, purchases
WHERE   ads.email = purchases.email
```

- Computed with secure computation by Google and its customers

Examples: Wage Equity Study

The New York Times

How Boston Is Trying to Close the Gender Pay Gap

Through pay-negotiation workshops and partnerships with more than 100 companies, the city is trying to help female workers match the salaries of male counterparts.

BOSTON WOMEN'S WORKFORCE COUNCIL REPORT 2017

DATA SUBMISSION PROCESS:

Part of the commitment employers make when signing the Boston 100% Talent Compact is to anonymously report employee data to the BWWC biennially. The Software & Application Innovation Lab at Boston University's Rafik B. Hariri Institute of Computing and Computational Science & Engineering, the BWWC's data partner, developed a completely confidential reporting system from which anonymous data from multiple independent sources can be analyzed in the aggregate.

During the submission process, Compact signers submit their wage data in the aggregate form over a unique, web-based software program that employs encryption using a technique known as secure multi-party computation. During this process, individual compensation data never leaves each organization's server. The BWWC then receives aggregate data unconnected to any firm.

What does it mean to "securely" compute f?

Security laundry list

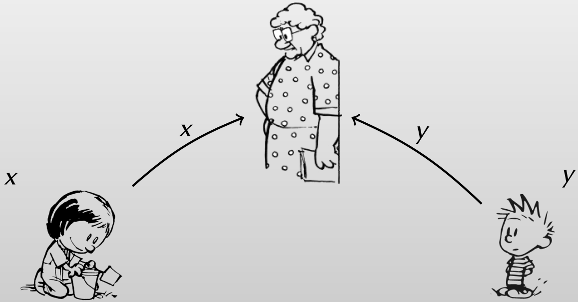
- What if adversary learns more than $f(x, y)$?
- What if adversary learns $f(x, y)$ but then prevents honest party from learning it too?
- What if adversary forces several parties to have inconsistent outputs?
- What if adversary's choice of input depends on honest party's input?
- What if . . .

Defining security: ideal world

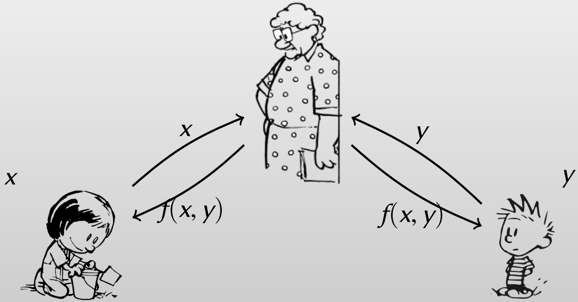
x

y

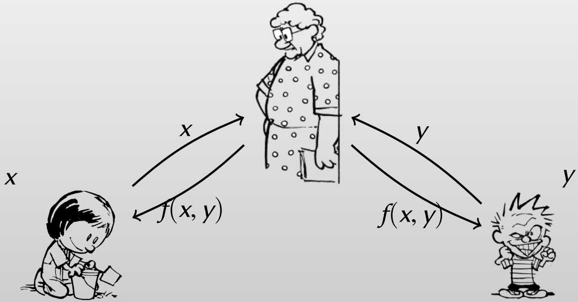
Defining security: ideal world



Defining security: ideal world

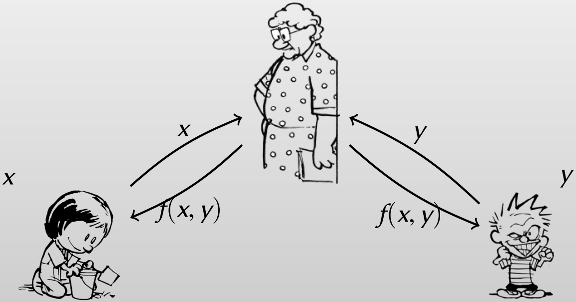


Defining security: ideal world



What can a corrupt party do in this **ideal world**?

Defining security: ideal world



What can a corrupt party do in this **ideal world**?

- Choose any input y (independent of x)
- Learn only $f(x, y)$, and nothing more
- Cause honest party to learn $f(x, y)$

Real-ideal paradigm [GoldwasserMicali84]

Security goal: real protocol interaction is ***as secure as*** *the ideal-world interaction*

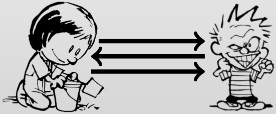
Real-ideal paradigm [GoldwasserMicali84]

Security goal: real protocol interaction is ***as secure as*** *the ideal-world interaction*

For every "attack" against real protocol, there is a way to achieve "same effect" in ideal world

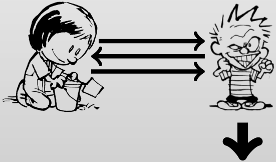
Real-ideal paradigm

What is the "effect" of a generic attack?



Real-ideal paradigm

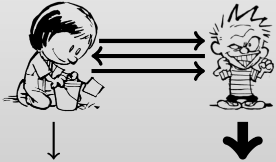
What is the "effect" of a generic attack?



- Something the adversary learns / can compute about honest party

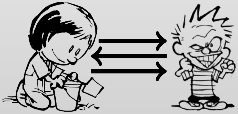
Real-ideal paradigm

What is the “effect” of a generic attack?



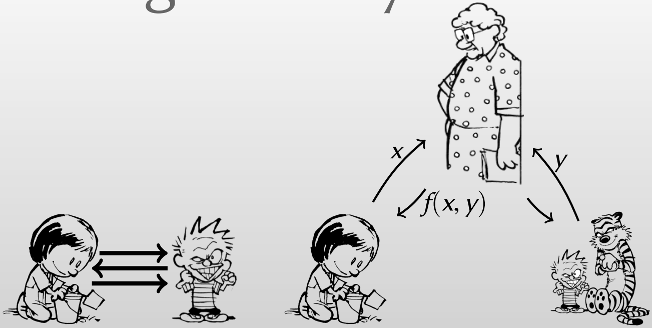
- Something the adversary learns / can compute about honest party
- Some influence on honest party’s output

Defining security



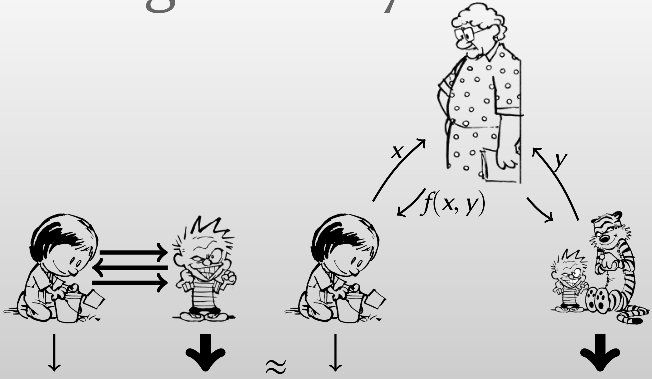
Security definition: For every real-world adversary $\mathcal{A}$

Defining security



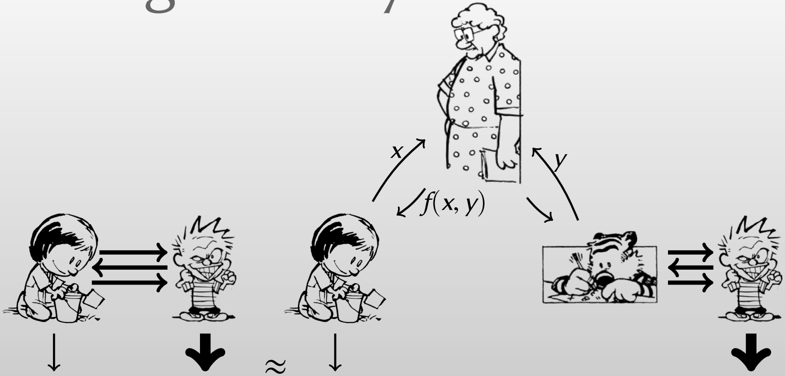
Security definition: For every real-world adversary $\mathcal{A}$, there exists an ideal adversary $\mathcal{A}'$

Defining security



Security definition: For every real-world adversary $\mathcal{A}$, there exists an ideal adversary $\mathcal{A}'$ s.t. joint distribution (HonestOutput,AdvOutput) is indistinguishable

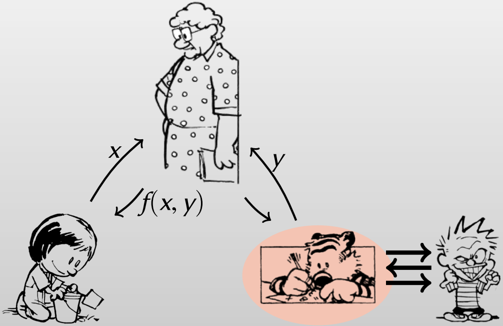
Defining security



Security definition: For every real-world adversary $\mathcal{A}$, there exists an ideal adversary $\mathcal{A}'$ s.t. joint distribution (HonestOutput,AdvOutput) is indistinguishable

WLOG: $\exists$ simulator that simulates real-world interaction in ideal world

Defining security

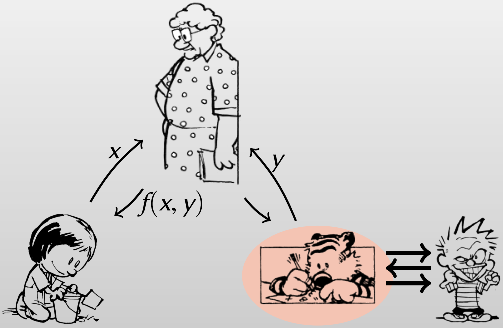


Role of simulator:

1. Send protocol messages that look like they came from honest party

2. **Extract** an f-input by examining adversary's protocol messages

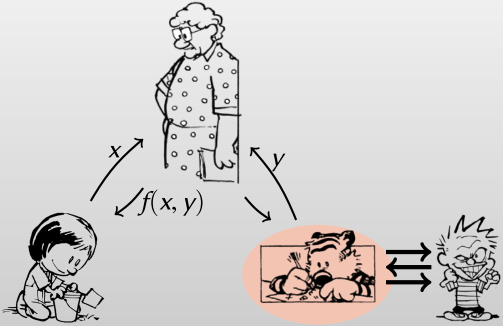
Defining security



Role of simulator:

1. Send protocol messages that look like they came from honest party
   - Demonstrates that honest party's messages leak no more than $f(x, y)$
2. **Extract** an f-input by examining adversary's protocol messages
   - "Explains" the effect on honest party's output in terms of ideal world

Semi-Honest security

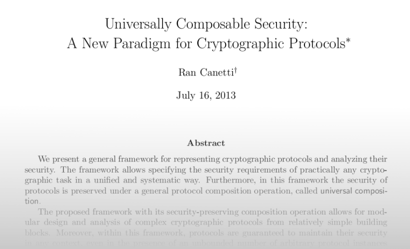


Special case: security against **semi-honest** (passive, honest-but-curious) adversary:

- Adversary assumed to *follow the protocol* on a given input
- Adversary may try to learn information based on what it sees
- No need to extract, only simulate transcript given ideal input+output

Disclaimer

Security definition here is **greatly oversimplified**



Universally Composable Security:
A New Paradigm for Cryptographic Protocols*

Ran Canetti†

July 16, 2013

Abstract

We present a general framework for representing cryptographic protocols and analyzing their security. The framework allows specifying the security requirements of practically any cryptographic task in a unified and systematic way. Furthermore, in this framework the security of protocols is preserved under a general protocol composition operation, called universal composition.

The proposed framework with its security-preserving composition operation allows for modular design and analysis of complex cryptographic protocols from relatively simple building blocks. Moreover, within this framework, protocols are guaranteed to maintain their security in any context, even in the presence of an unbounded number of arbitrary protocol instances

(87-page security definition)

Roadmap

1. **Secure computation:** Concepts & definitions
2. **Yao's protocol:** semi-honest secure computation for boolean circuits

Warm-up: garbled truth table

Alice does the following:

1. Write truth table of function f

1	1	$f(1,1)$
1	2	$f(1,2)$
1	3	$f(1,3)$
1	4	$f(1,4)$
2	1	$f(2,1)$
2	2	$f(2,2)$
2	3	$f(2,3)$
2	4	$f(2,4)$
3	1	$f(3,1)$
3	2	$f(3,2)$
3	3	$f(3,3)$
3	4	$f(3,4)$
4	1	$f(4,1)$
4	2	$f(4,2)$
4	3	$f(4,3)$
4	4	$f(4,4)$

Warm-up: garbled truth table

Alice does the following:

1. Write truth table of function f
2. For each possible input, choose random **cryptographic key**

A_1	B_1	$f(1,1)$
A_1	B_2	$f(1,2)$
A_1	B_3	$f(1,3)$
A_1	B_4	$f(1,4)$
A_2	B_1	$f(2,1)$
A_2	B_2	$f(2,2)$
A_2	B_3	$f(2,3)$
A_2	B_4	$f(2,4)$
A_3	B_1	$f(3,1)$
A_3	B_2	$f(3,2)$
A_3	B_3	$f(3,3)$
A_3	B_4	$f(3,4)$
A_4	B_1	$f(4,1)$
A_4	B_2	$f(4,2)$
A_4	B_3	$f(4,3)$
A_4	B_4	$f(4,4)$

Warm-up: garbled truth table

Alice does the following:

1. Write truth table of function f
2. For each possible input, choose random **cryptographic key**
3. Encrypt each output with corresponding keys

$\mathbb{E}_{A_1,B_1}(f(1,1))$
$\mathbb{E}_{A_1,B_2}(f(1,2))$
$\mathbb{E}_{A_1,B_3}(f(1,3))$
$\mathbb{E}_{A_1,B_4}(f(1,4))$
$\mathbb{E}_{A_2,B_1}(f(2,1))$
$\mathbb{E}_{A_2,B_2}(f(2,2))$
$\mathbb{E}_{A_2,B_3}(f(2,3))$
$\mathbb{E}_{A_2,B_4}(f(2,4))$
$\mathbb{E}_{A_3,B_1}(f(3,1))$
$\mathbb{E}_{A_3,B_2}(f(3,2))$
$\mathbb{E}_{A_3,B_3}(f(3,3))$
$\mathbb{E}_{A_3,B_4}(f(3,4))$
$\mathbb{E}_{A_4,B_1}(f(4,1))$
$\mathbb{E}_{A_4,B_2}(f(4,2))$
$\mathbb{E}_{A_4,B_3}(f(4,3))$
$\mathbb{E}_{A_4,B_4}(f(4,4))$

Warm-up: garbled truth table

Alice does the following:

1. Write truth table of function f
2. For each possible input, choose random **cryptographic key**
3. Encrypt each output with corresponding keys
4. Randomly permute ciphertexts, send to Bob

$$\begin{array}{|l|}\hline
\mathbb{E}_{A_3,B_4}(f(3,4)) \\
\mathbb{E}_{A_4,B_3}(f(4,3)) \\
\mathbb{E}_{A_3,B_3}(f(3,3)) \\
\mathbb{E}_{A_2,B_3}(f(2,3)) \\
\mathbb{E}_{A_4,B_2}(f(4,2)) \\
\mathbb{E}_{A_2,B_4}(f(2,4)) \\
\mathbb{E}_{A_4,B_4}(f(4,4)) \\
\mathbb{E}_{A_1,B_4}(f(1,4)) \\
\mathbb{E}_{A_2,B_2}(f(2,2)) \\
\mathbb{E}_{A_1,B_2}(f(1,2)) \\
\mathbb{E}_{A_2,B_1}(f(2,1)) \\
\mathbb{E}_{A_1,B_3}(f(1,3)) \\
\mathbb{E}_{A_4,B_1}(f(4,1)) \\
\mathbb{E}_{A_3,B_1}(f(3,1)) \\
\mathbb{E}_{A_1,B_1}(f(1,1)) \\
\mathbb{E}_{A_3,B_2}(f(3,2)) \\
\hline\end{array}$$

Warm-up: garbled truth table

Alice does the following:

1. Write truth table of function f
2. For each possible input, choose random **cryptographic key**
3. Encrypt each output with corresponding keys
4. Randomly permute ciphertexts, send to Bob

?? **Somehow** Bob obtains "correct" A_x, B_y ??

$\mathbb{E}_{A_3,B_4}(f(3,4))$
$\mathbb{E}_{A_4,B_3}(f(4,3))$
$\mathbb{E}_{A_3,B_3}(f(3,3))$
$\mathbb{E}_{A_2,B_3}(f(2,3))$
$\mathbb{E}_{A_4,B_2}(f(4,2))$
$\mathbb{E}_{A_2,B_4}(f(2,4))$
$\mathbb{E}_{A_4,B_4}(f(4,4))$
$\mathbb{E}_{A_1,B_4}(f(1,4))$
$\mathbb{E}_{A_2,B_2}(f(2,2))$
$\mathbb{E}_{A_1,B_2}(f(1,2))$
$\mathbb{E}_{A_2,B_1}(f(2,1))$
$\mathbb{E}_{A_1,B_3}(f(1,3))$
$\mathbb{E}_{A_4,B_1}(f(4,1))$
$\mathbb{E}_{A_3,B_1}(f(3,1))$
$\mathbb{E}_{A_1,B_1}(f(1,1))$
$\mathbb{E}_{A_3,B_2}(f(3,2))$

Warm-up: garbled truth table

Alice does the following:

1. Write truth table of function f
2. For each possible input, choose random **cryptographic key**
3. Encrypt each output with corresponding keys
4. Randomly permute ciphertexts, send to Bob

?? **Somehow** Bob obtains "correct" A_x, B_y ??

Through trial decryption, Bob learns only $f(x, y)$

$\mathbb{E}_{A_3,B_4}(f(3,4))$
$\mathbb{E}_{A_4,B_3}(f(4,3))$
$\mathbb{E}_{A_3,B_3}(f(3,3))$
$\mathbb{E}_{A_2,B_3}(f(2,3))$
$\mathbb{E}_{A_4,B_2}(f(4,2))$
$\mathbb{E}_{A_2,B_4}(f(2,4))$
$\mathbb{E}_{A_4,B_4}(f(4,4))$
$\mathbb{E}_{A_1,B_4}(f(1,4))$
$\mathbb{E}_{A_2,B_2}(f(2,2))$
$\mathbb{E}_{A_1,B_2}(f(1,2))$
$\mathbb{E}_{A_2,B_1}(f(2,1))$
$\mathbb{E}_{A_1,B_3}(f(1,3))$
$\mathbb{E}_{A_4,B_1}(f(4,1))$
$\mathbb{E}_{A_3,B_1}(f(3,1))$
$\mathbb{E}_{A_1,B_1}(f(1,1))$
$\mathbb{E}_{A_3,B_2}(f(3,2))$

Security of warm-up protocol

Suffices to show that Bob's view in the protocol can be **simulated** given just Bob's ideal input/output.

Bob's view (real):

A_4, B_2

$$\begin{array}{|c|}\hline \mathbb{E}_{A_3,B_4}(f(3,4)) \\ \mathbb{E}_{A_4,B_3}(f(4,3)) \\ \mathbb{E}_{A_3,B_3}(f(3,3)) \\ \mathbb{E}_{A_2,B_3}(f(2,3)) \\ \mathbb{E}_{A_4,B_2}(f(4,2)) \\ \mathbb{E}_{A_2,B_4}(f(2,4)) \\ \mathbb{E}_{A_4,B_4}(f(4,4)) \\ \mathbb{E}_{A_1,B_4}(f(1,4)) \\ \mathbb{E}_{A_2,B_2}(f(2,2)) \\ \mathbb{E}_{A_1,B_2}(f(1,2)) \\ \vdots \\ \hline \end{array}$$

Security of warm-up protocol

Suffices to show that Bob's view in the protocol can be **simulated** given just Bob's ideal input/output.

Bob's view (real): $\approx$ **Simulated view:**

A_4, B_2	A^*, B^*
$\mathbb{E}_{A_3,B_4}(f(3,4))$	$\mathbb{E}_{A_?,B_?}(0)$
$\mathbb{E}_{A_4,B_3}(f(4,3))$	$\mathbb{E}_{A_?,B_?}(0)$
$\mathbb{E}_{A_3,B_3}(f(3,3))$	$\mathbb{E}_{A_?,B_?}(0)$
$\mathbb{E}_{A_2,B_3}(f(2,3))$	$\mathbb{E}_{A_?,B_?}(0)$
$\mathbb{E}_{A_4,B_2}(f(4,2))$	$\mathbb{E}_{A^*,B^*}(f(x,y))$
$\mathbb{E}_{A_2,B_4}(f(2,4))$	$\mathbb{E}_{A_?,B_?}(0)$
$\mathbb{E}_{A_4,B_4}(f(4,4))$	$\mathbb{E}_{A_?,B_?}(0)$
$\mathbb{E}_{A_1,B_4}(f(1,4))$	$\mathbb{E}_{A_?,B_?}(0)$
$\mathbb{E}_{A_2,B_2}(f(2,2))$	$\mathbb{E}_{A_?,B_?}(0)$
$\mathbb{E}_{A_1,B_2}(f(1,2))$	$\mathbb{E}_{A_?,B_?}(0)$
$\vdots$	$\vdots$

Security of warm-up protocol

Suffices to show that Bob's view in the protocol can be **simulated** given just Bob's ideal input/output.

Simulation is indistinguishable, as long as $\mathbb{E}$ satisfies:

$$\mathbb{E}_{A,B}(C) \approx \mathbb{E}_{A',B'}(C')$$

if at least one of $\{A, B\}$ random and unknown to distinguisher.

Bob's view (real): $\approx$ **Simulated view:**

A_4, B_2	A^*, B^*
$\mathbb{E}_{A_3,B_4}(f(3,4))$	$\mathbb{E}_{A_?,B_?}(0)$
$\mathbb{E}_{A_4,B_3}(f(4,3))$	$\mathbb{E}_{A_?,B_?}(0)$
$\mathbb{E}_{A_3,B_3}(f(3,3))$	$\mathbb{E}_{A_?,B_?}(0)$
$\mathbb{E}_{A_2,B_3}(f(2,3))$	$\mathbb{E}_{A_?,B_?}(0)$
$\mathbb{E}_{A_4,B_2}(f(4,2))$	$\mathbb{E}_{A^*,B^*}(f(x,y))$
$\mathbb{E}_{A_2,B_4}(f(2,4))$	$\mathbb{E}_{A_?,B_?}(0)$
$\mathbb{E}_{A_4,B_4}(f(4,4))$	$\mathbb{E}_{A_?,B_?}(0)$
$\mathbb{E}_{A_1,B_4}(f(1,4))$	$\mathbb{E}_{A_?,B_?}(0)$
$\mathbb{E}_{A_2,B_2}(f(2,2))$	$\mathbb{E}_{A_?,B_?}(0)$
$\mathbb{E}_{A_1,B_2}(f(1,2))$	$\mathbb{E}_{A_?,B_?}(0)$
$\vdots$	$\vdots$

Extending warm-up protocol

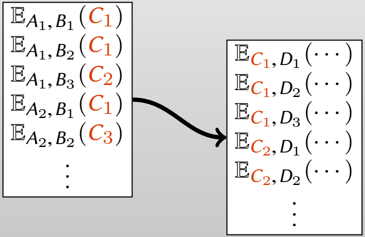
Problem: Cost scales with the truth table size of f!

Problem: How does Bob magically learn "correct" A_x, B_y?

Extending warm-up protocol

Problem: Cost scales with the truth table size of f!

- Idea: instead of encrypting outputs, encrypt **keys to yet more garbled tables**

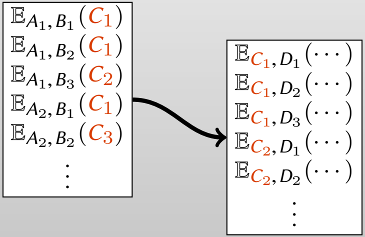


Problem: How does Bob magically learn “correct” A_x, B_y?

Extending warm-up protocol

Problem: Cost scales with the truth table size of f!

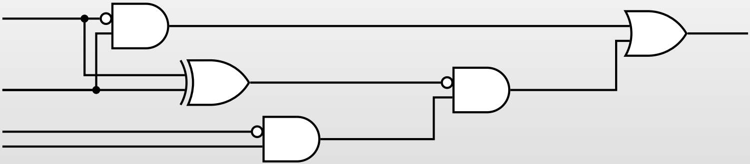
- Idea: instead of encrypting outputs, encrypt **keys to yet more garbled tables**



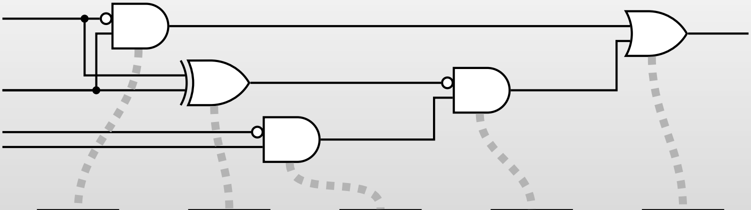
Problem: How does Bob magically learn "correct" A_x, B_y?

- Discuss later (oblivious transfer)

Garbled circuit framework [Yao86]



Garbled circuit framework [Yao86]



0	0	0
0	1	1
1	0	0
1	1	0

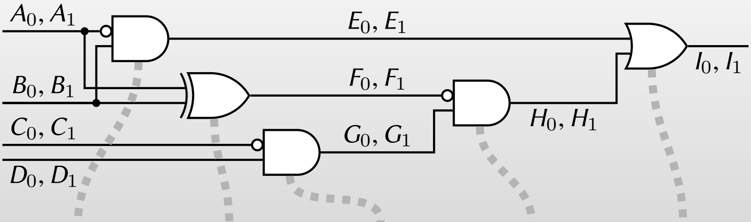
0	0	0
0	1	1
1	0	1
1	1	0

0	0	0
0	1	1
1	0	0
1	1	0

0	0	0
0	1	1
1	0	0
1	1	0

0	0	0
0	1	1
1	0	1
1	1	1

Garbled circuit framework [Yao86]



0	0	0
0	1	1
1	0	0
1	1	0

0	0	0
0	1	1
1	0	1
1	1	0

0	0	0
0	1	1
1	0	0
1	1	0

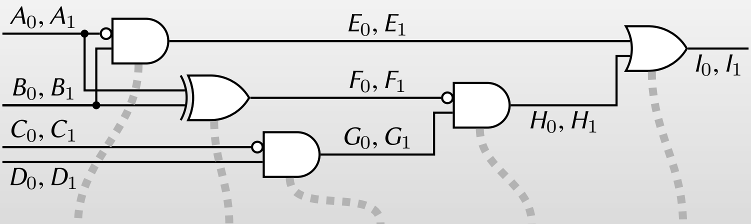
0	0	0
0	1	1
1	0	0
1	1	0

0	0	0
0	1	1
1	0	1
1	1	1

Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire

Garbled circuit framework [Yao86]



A_0	B_0	E_0
A_0	B_1	E_1
A_1	B_0	E_0
A_1	B_1	E_0

A_0	B_0	F_0
A_0	B_1	F_1
A_1	B_0	F_1
A_1	B_1	F_0

C_0	D_0	G_0
C_0	D_1	G_1
C_1	D_0	G_0
C_1	D_1	G_0

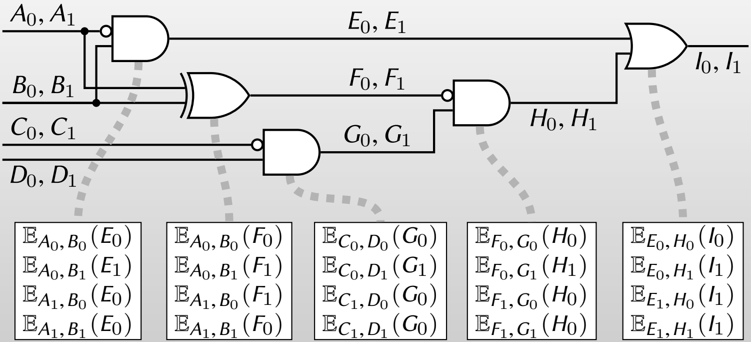
F_0	G_0	H_0
F_0	G_1	H_1
F_1	G_0	H_0
F_1	G_1	H_0

E_0	H_0	I_0
E_0	H_1	I_1
E_1	H_0	I_1
E_1	H_1	I_1

Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire

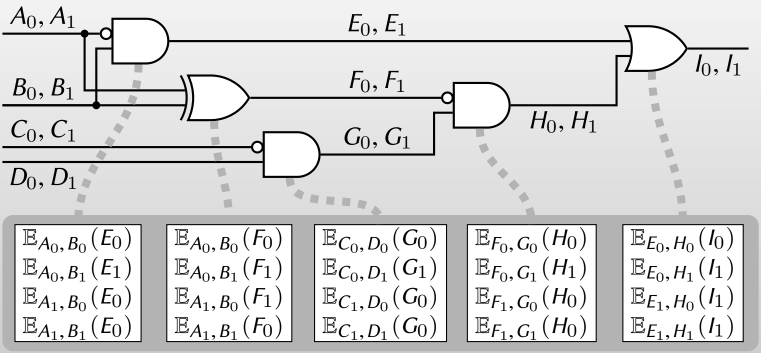
Garbled circuit framework [Yao86]



Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate

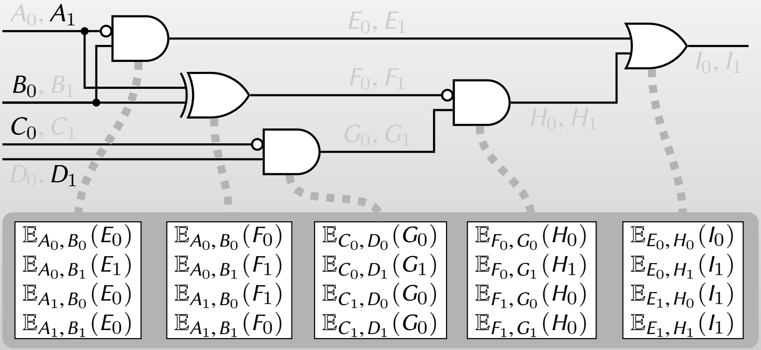
Garbled circuit framework [Yao86]



Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- “Encrypt” truth table of each gate
- **Garbled circuit** ≡ all encrypted gates

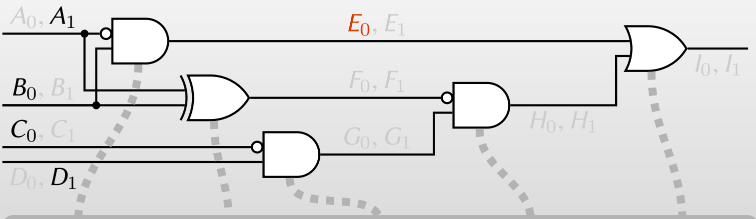
Garbled circuit framework [Yao86]



Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** ≡ all encrypted gates
- **Garbled encoding** ≡ one label per wire

Garbled circuit framework [Yao86]



$\mathbb{E}_{A_0,B_0}(E_0)$	$\mathbb{E}_{A_0,B_0}(F_0)$	$\mathbb{E}_{C_0,D_0}(G_0)$	$\mathbb{E}_{F_0,G_0}(H_0)$	$\mathbb{E}_{E_0,H_0}(I_0)$
$\mathbb{E}_{A_0,B_1}(E_1)$	$\mathbb{E}_{A_0,B_1}(F_1)$	$\mathbb{E}_{C_0,D_1}(G_1)$	$\mathbb{E}_{F_0,G_1}(H_1)$	$\mathbb{E}_{E_0,H_1}(I_1)$
$\mathbb{E}_{A_1,B_0}(E_0)$	$\mathbb{E}_{A_1,B_0}(F_1)$	$\mathbb{E}_{C_1,D_0}(G_0)$	$\mathbb{E}_{F_1,G_0}(H_0)$	$\mathbb{E}_{E_1,H_0}(I_1)$
$\mathbb{E}_{A_1,B_1}(E_0)$	$\mathbb{E}_{A_1,B_1}(F_0)$	$\mathbb{E}_{C_1,D_1}(G_0)$	$\mathbb{E}_{F_1,G_1}(H_0)$	$\mathbb{E}_{E_1,H_1}(I_1)$

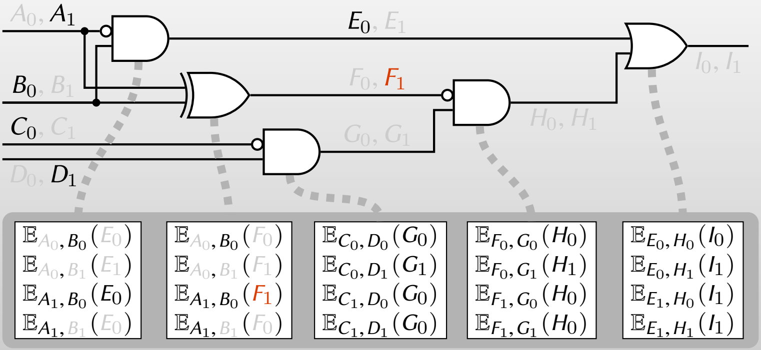
Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** ≡ all encrypted gates
- **Garbled encoding** ≡ one label per wire

Garbled evaluation:

- Only one ciphertext per gate is decryptable

Garbled circuit framework [Yao86]



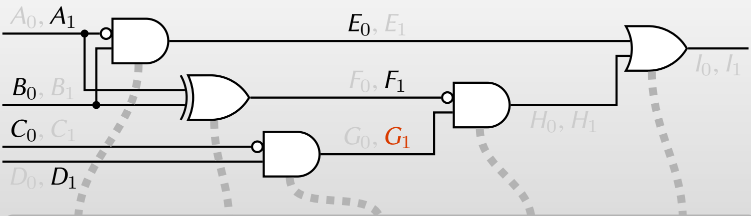
Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** ≡ all encrypted gates
- **Garbled encoding** ≡ one label per wire

Garbled evaluation:

- Only one ciphertext per gate is decryptable
- Result of decryption = value on outgoing wire

Garbled circuit framework [Yao86]



$\mathbb{E}_{A_0,B_0}(E_0)$	$\mathbb{E}_{A_0,B_0}(F_0)$	$\mathbb{E}_{C_0,D_0}(G_0)$	$\mathbb{E}_{F_0,G_0}(H_0)$	$\mathbb{E}_{E_0,H_0}(I_0)$
$\mathbb{E}_{A_0,B_1}(E_1)$	$\mathbb{E}_{A_0,B_1}(F_1)$	$\mathbb{E}_{C_0,D_1}(G_1)$	$\mathbb{E}_{F_0,G_1}(H_1)$	$\mathbb{E}_{E_0,H_1}(I_1)$
$\mathbb{E}_{A_1,B_0}(E_0)$	$\mathbb{E}_{A_1,B_0}(F_1)$	$\mathbb{E}_{C_1,D_0}(G_0)$	$\mathbb{E}_{F_1,G_0}(H_0)$	$\mathbb{E}_{E_1,H_0}(I_1)$
$\mathbb{E}_{A_1,B_1}(E_0)$	$\mathbb{E}_{A_1,B_1}(F_0)$	$\mathbb{E}_{C_1,D_1}(G_0)$	$\mathbb{E}_{F_1,G_1}(H_0)$	$\mathbb{E}_{E_1,H_1}(I_1)$

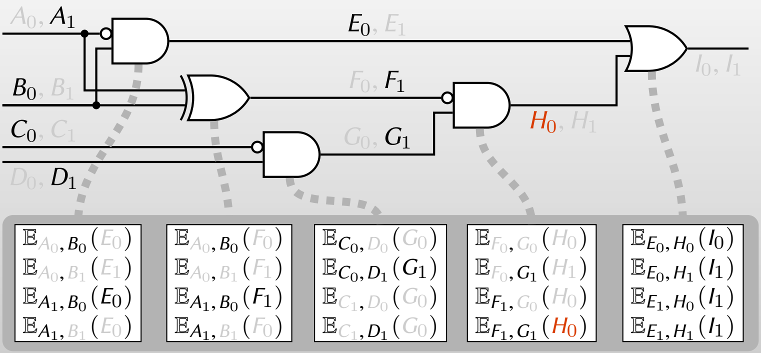
Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- “Encrypt” truth table of each gate
- **Garbled circuit** ≡ all encrypted gates
- **Garbled encoding** ≡ one label per wire

Garbled evaluation:

- Only one ciphertext per gate is decryptable
- Result of decryption = value on outgoing wire

Garbled circuit framework [Yao86]



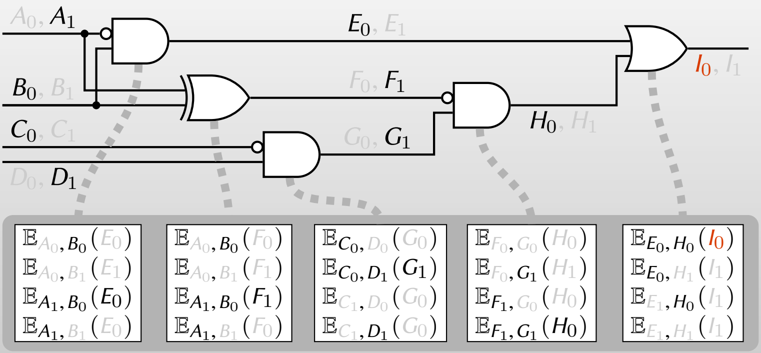
Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** ≡ all encrypted gates
- **Garbled encoding** ≡ one label per wire

Garbled evaluation:

- Only one ciphertext per gate is decryptable
- Result of decryption = value on outgoing wire

Garbled circuit framework [Yao86]



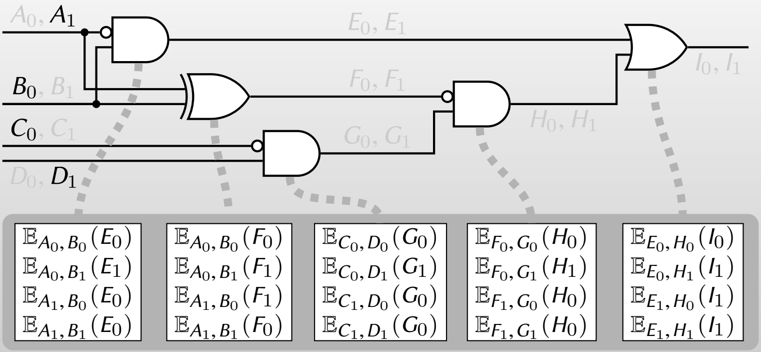
Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- “Encrypt” truth table of each gate
- **Garbled circuit** ≡ all encrypted gates
- **Garbled encoding** ≡ one label per wire

Garbled evaluation:

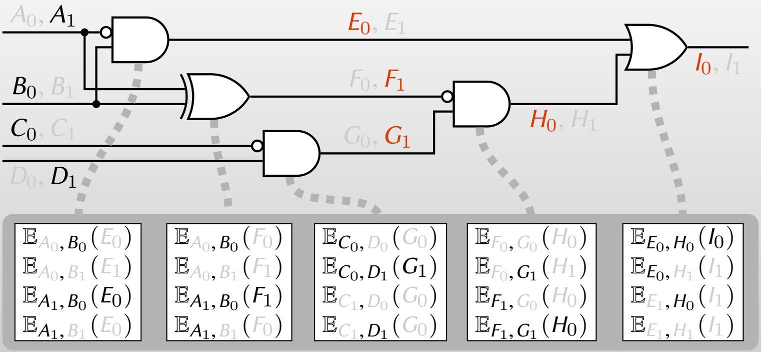
- Only one ciphertext per gate is decryptable
- Result of decryption = value on outgoing wire

Syntax & Security (informal)



Key idea: Given garbled circuit + garbled input . . .

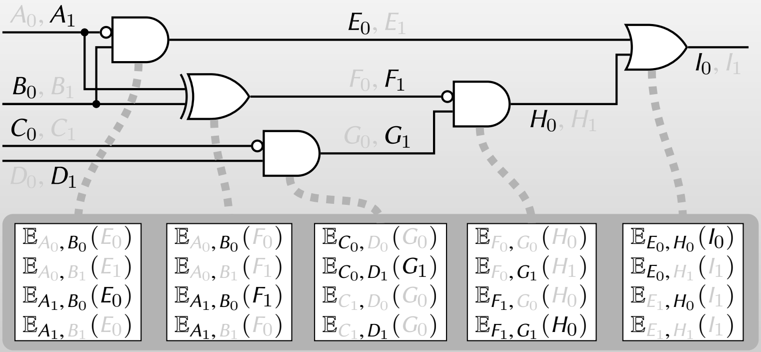
Syntax & Security (informal)



Key idea: Given garbled circuit + garbled input . . .

- . . . Only thing you can do is **(blindly) evaluate circuit** on that input

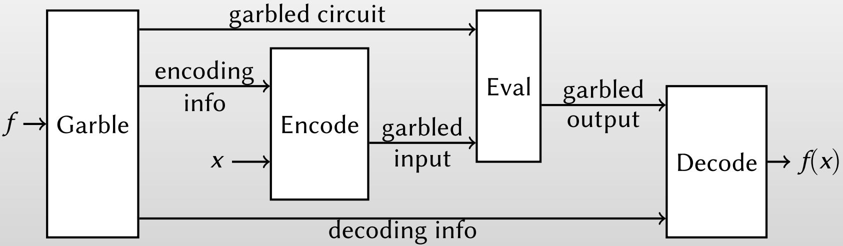
Syntax & Security (informal)



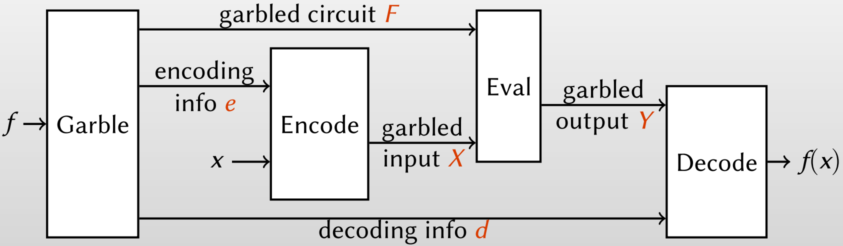
Key idea: Given garbled circuit + garbled input . . .

- . . . Only thing you can do is **(blindly) evaluate circuit** on that input
- Learn only 1 label per wire: hard to guess "complementary" label
- Seeing a single label hides logical value on wire, although . . .
- Revealing both labels on *output wires* leaks *only* circuit output

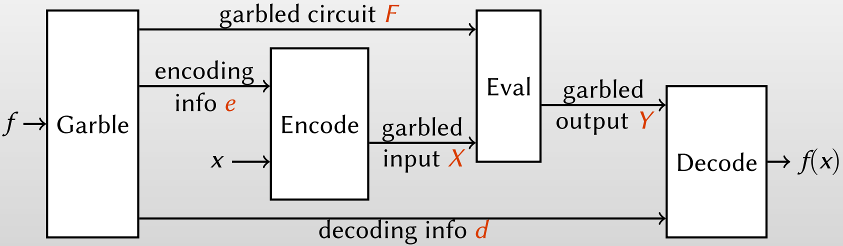
Syntax & Security [BellareHoangRogaway12]



Syntax & Security [BellareHoangRogaway12]



Syntax & Security [BellareHoangRogaway12]



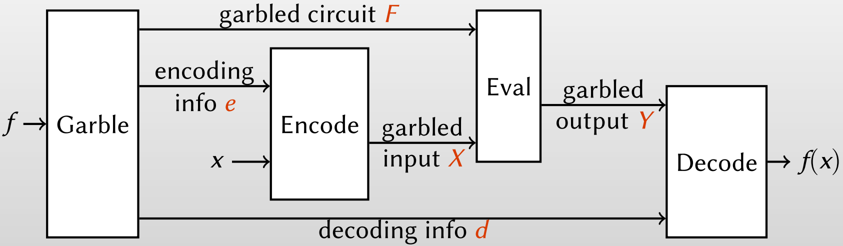
Formal security properties:

Privacy: (F, X, d) reveals nothing beyond f and $f(x)$

Obliviousness: (F, X) reveals nothing beyond f

Authenticity: given (F, X), hard to find $\widetilde{Y}$ that decodes $\notin \{f(x), \perp\}$

Syntax & Security [BellareHoangRogaway12]



Formal security properties:

Privacy: (F, X, d) reveals nothing beyond f and $f(x)$

Obliviousness: (F, X) reveals nothing beyond f

Authenticity: given (F, X), hard to find $\widetilde{Y}$ that decodes $\notin \{f(x), \perp\}$

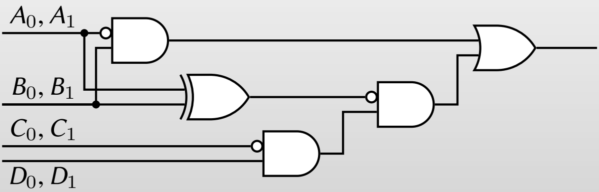
Other interesting notions we won't discuss:

Adaptive security: choice of input can depend on *garbled* circuit

Gate-hiding: (F, X, d) reveals nothing beyond *topology of f* and $f(x)$

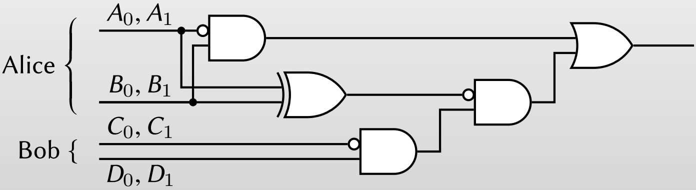
Oblivious transfer

How does evaluator (Bob) get the garbled input?



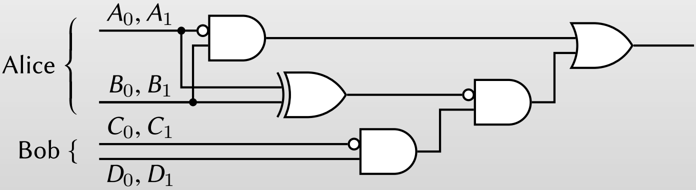
Oblivious transfer

How does evaluator (Bob) get the garbled input?



Oblivious transfer

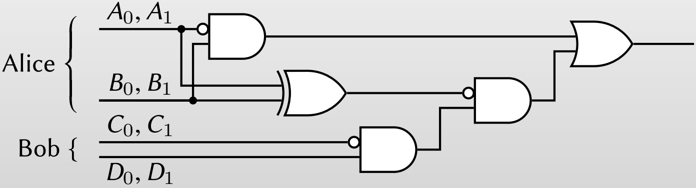
How does evaluator (Bob) get the garbled input?



Garbler's inputs: She knows both A_0, A_1, and which one is correct $\Rightarrow$ just send correct one to Bob

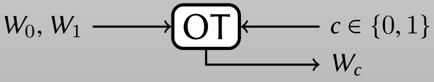
Oblivious transfer

How does evaluator (Bob) get the garbled input?



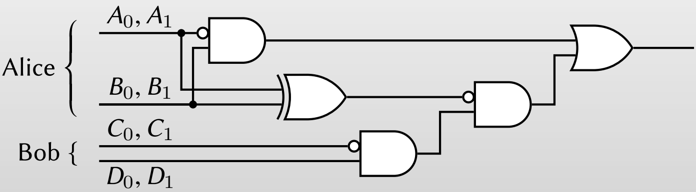
Garbler's inputs: She knows both A_0, A_1, and which one is correct $\Rightarrow$ just send correct one to Bob

Evaluator's inputs: We need the following "gadget":



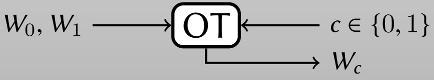
Oblivious transfer

How does evaluator (Bob) get the garbled input?

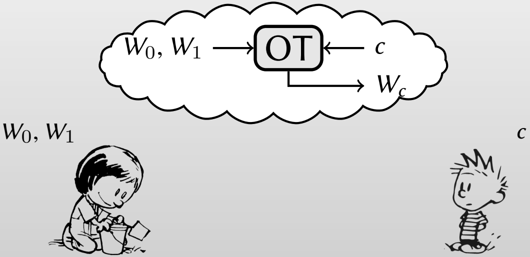


Garbler's inputs: She knows both A_0, A_1, and which one is correct $\Rightarrow$ just send correct one to Bob

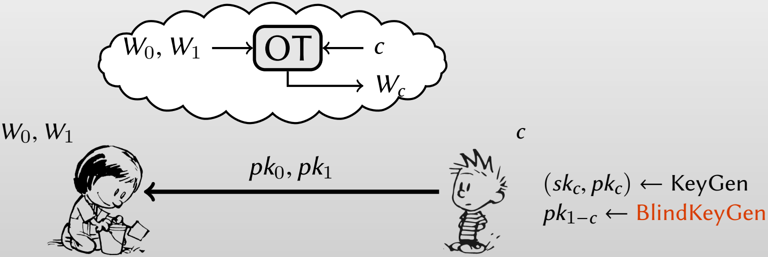
Evaluator's inputs: We need the following "gadget" (oblivious transfer):



How to construct OT?



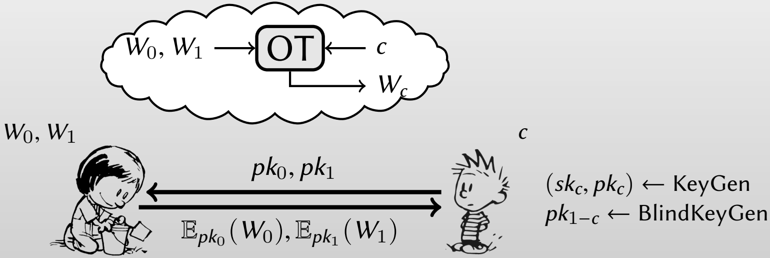
How to construct OT?



Need public-key encryption that supports **blind key generation**:

- sample a public key without knowledge of secret key
- E.g.: ElGamal (sample group element without knowing discrete log)

How to construct OT?



Need public-key encryption that supports **blind key generation**:

- sample a public key without knowledge of secret key
- E.g.: ElGamal (sample group element without knowing discrete log)

Yao's Protocol: overview

x

y

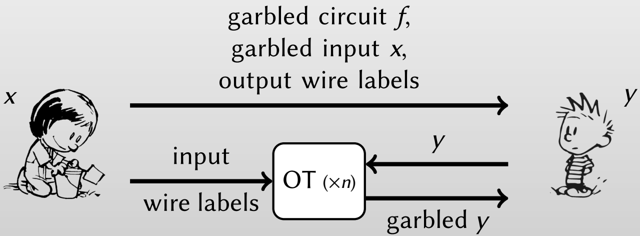
Yao's Protocol: overview

x

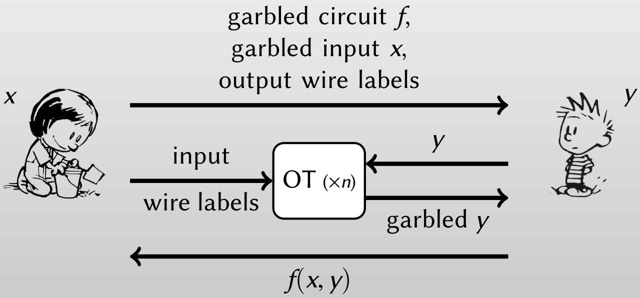
garbled circuit f,
garbled input x,
output wire labels
$\longrightarrow$

y

Yao's Protocol: overview



Yao's Protocol: overview



- Given garbled f + garbled inputs + all output labels $\Rightarrow$ Bob learns **only** $f(x, y)$

Summary so far

Secure Computation allows parties to perform a computation on private input, learning only the output.

- market clearing price, advertising revenue, . . .

Security: every attack against the protocol can be “simulated” in an ideal world interaction.

Yao’s protocol:

- Garbled lookup table for each gate of boolean circuit
- Oblivious transfer for each input wire

Next lectures:

2 **Garbled circuits** are extremely large

- How to reduce their size by $10\times$

3 Yao's protocol insecure against **malicious attacks:**

- How to harden the protocol against malicious adversaries

4 **Oblivious transfer** is prohibitively expensive:

- How to "amplify" OT instances using cheap crypto

5 **Special-purpose protocols** can be much faster than Yao's

- How to securely compute **set intersection**