Lecture 9: Privacy-Enhancing Technologies-3 -Secure Multiparty Computation

COMP 6712 Advanced Security and Privacy

Haiyang Xue

haiyang.xue@polyu.edu.hk

2023/3/18

Several slides are based on those of Mike Rosulek, Lindell, etc

Roadmap

- Recall zero-knowledge proof
- Introduction to Secure Multiparty computation (MPC)
- Yao's Garbled Circuits and GMW protocol
- Practical MPC: Private Set Intersection

Recall: Zero-knowledge proof

- Identification protocol and signature
- Sigma protocol
- Zero-knowledge proof
  - Non-interactive ZKP
  - zkSNARK

Schnorr Identification

$$u = g^{\alpha}$$

$P(\alpha)$		$V(u)$
$\alpha_t \xleftarrow{R} \mathbb{Z}_q,\ u_t \leftarrow g^{\alpha_t}$		
	$\xrightarrow{u_t}$	
		$c \xleftarrow{R} \mathcal{C}$
	$\xleftarrow{c}$	
$\alpha_z \leftarrow \alpha_t + \alpha c \mod q$		
	$\xrightarrow{\alpha_z}$	
		$g^{\alpha_z} \stackrel{?}{=} u_t \cdot u^c$

- Challenge space $\mathcal{C} = Z_q$
- Conversation: (u_t, c, α_z) is said to be valid if the verification passes

Schnorr Identification

$u = g^{\alpha}$

$\underline{P(\alpha)}$		$\underline{V(u)}$
$\alpha_t \xleftarrow{R} \mathbb{Z}_q,\ u_t \leftarrow g^{\alpha_t}$		
	$\xrightarrow{\quad u_t \quad}$	
		$c \xleftarrow{R} \mathcal{C}$
	$\xleftarrow{\quad c \quad}$	
$\alpha_z \leftarrow \alpha_t + \alpha c \mod q$		
	$\xrightarrow{\quad \alpha_z \quad}$	
		$g^{\alpha_z} \stackrel{?}{=} u_t \cdot u^c$

- **Correctness(Completeness):** If P and V execute the protocol honestly, the proof is accepted.
- **Soundness (proof-of-knowledge):** If the proof is accepted, we can extract the witness (discrete log) α
- **Honest verifier zero-knowledge** says that: without knowing the witness (discrete logarithm), we can generate (simulate) the valid transaction efficiently

Identification for Decisional Diffie-Hellman ID_{DDH}

$v = g^{\beta}, w = u^{\beta}$

$\underline{P(\beta, (u, v, w))}$		$\underline{V(u, v, w)}$
$\beta_t \xleftarrow{R} \mathbb{Z}_q,\ v_t \leftarrow g^{\beta_t},\ w_t \leftarrow u^{\beta_t}$		
	$\xrightarrow{v_t, w_t}$	
		$c \xleftarrow{R} \mathcal{C}$
	$\xleftarrow{c}$	
$\beta_z \leftarrow \beta_t + \beta c \ mod\ q$		
	$\xrightarrow{\beta_z}$	
		$g^{\beta_z} \stackrel{?}{=} v_t \cdot v^c$ and $u^{\beta_z} \stackrel{?}{=} w_t \cdot w^c$

Given $(g, u, v = g^{\beta}, w = u^{\beta})$ with witness β, P wants to prove that it knows β

Identification for Decisional Diffie-Hellman (DDH)

Given $(g, u, v = g^{\beta}, w = u^{\beta})$ with witness β, P wants to prove that it knows β

$$v = g^{\beta}, w = u^{\beta}$$

$P(\beta, (u, v, w))$		$V(u, v, w)$
$\beta_t \xleftarrow{R} \mathbb{Z}_q,\ v_t \leftarrow g^{\beta_t},\ w_t \leftarrow u^{\beta_t}$		
	$\xrightarrow{v_t, w_t}$	
		$c \xleftarrow{R} \mathcal{C}$
	$\xleftarrow{c}$	
$\beta_z \leftarrow \beta_t + \beta c \mod q$		
	$\xrightarrow{\beta_z}$	
		$g^{\beta_z} \stackrel{?}{=} v_t \cdot v^c$ and $u^{\beta_z} \stackrel{?}{=} w_t \cdot w^c$

- **Correctness(Completeness):** If P and V exact the protocol honestly, the proof is accepted.
- **Soundness (proof-of-knowledge):** If the proof is accepted, we can extract the witness (discrete log) α
- **Honest verifier zero-knowledge** says that: without knowing the witness (discrete logarithm), we can generate (simulate) the valid transaction efficiently

$$\beta_z \leftarrow Z_q, c \leftarrow Z_q, v_t = \frac{g^{\beta_z}}{v^c}, u_t = g^{\beta_z}/u^c$$

Assignment 2

- Task 1: zero knowledge proof for that
  - $(c_1, c_2) = (g^{\beta}, u^{\beta} \cdot g^1)$ and $(d_1, d_2) = (g^{\gamma}, u^{\gamma} \cdot g^0)$ are the encryption of 1 and 0 respectively
  - Hint: use the proof for DDH tuple

- Task 2: Implement the proof

- Write a report about this
- submit via Blackboard, Deadline: 10 Apr. 11:00 pm

Multiparty Computation (MPC)

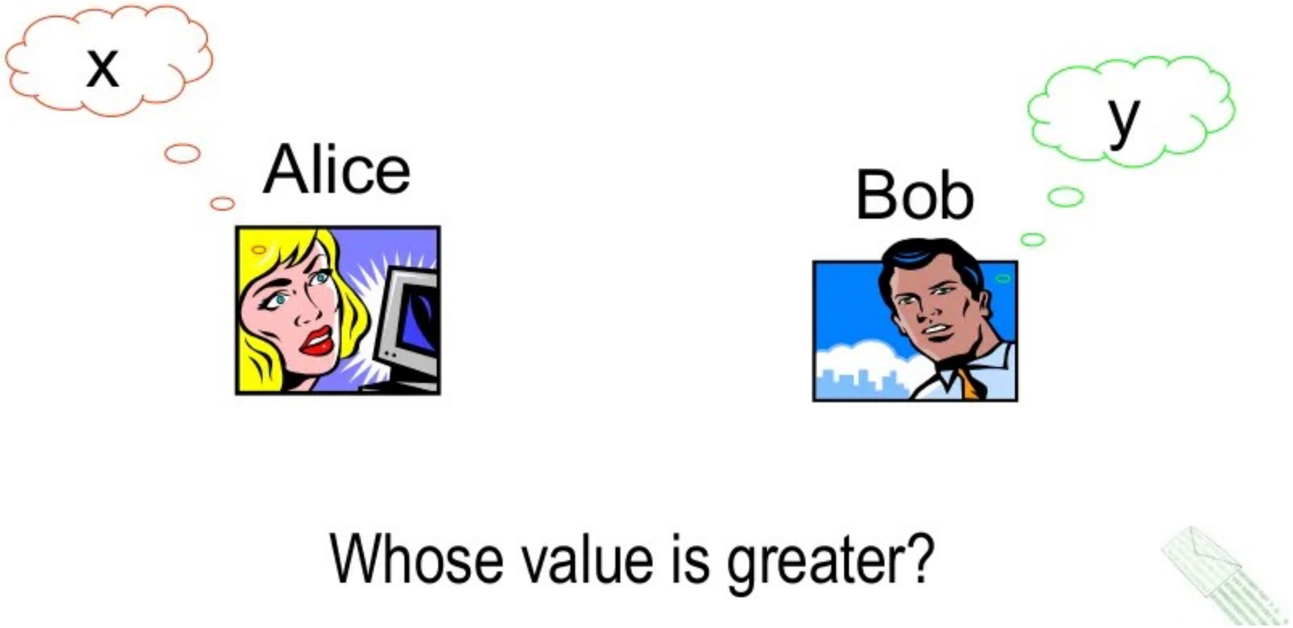
Our aim

1 **Secure computation**: Concepts & definitions

2 **General constructions**: Yao's protocol, and GMW

3 **Custom protocol**: private set intersection

Secure computation examples: Millionaires Problem

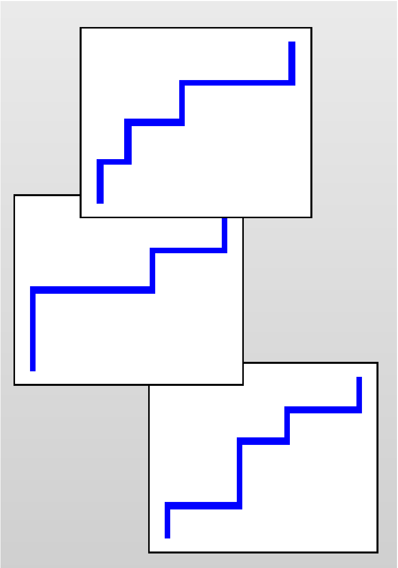


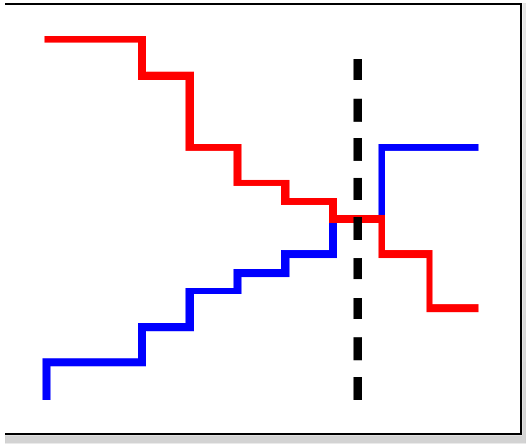
- Alice has money x
- Bob has money y
- X>y or not (but do not want to leak x or y to each other)

Andrew C. Yao, Protocols for Secure Computations.

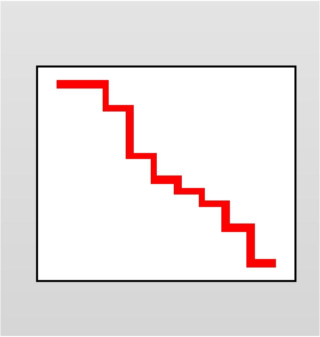
Secure computation examples: Sugar Bidding

Farmers



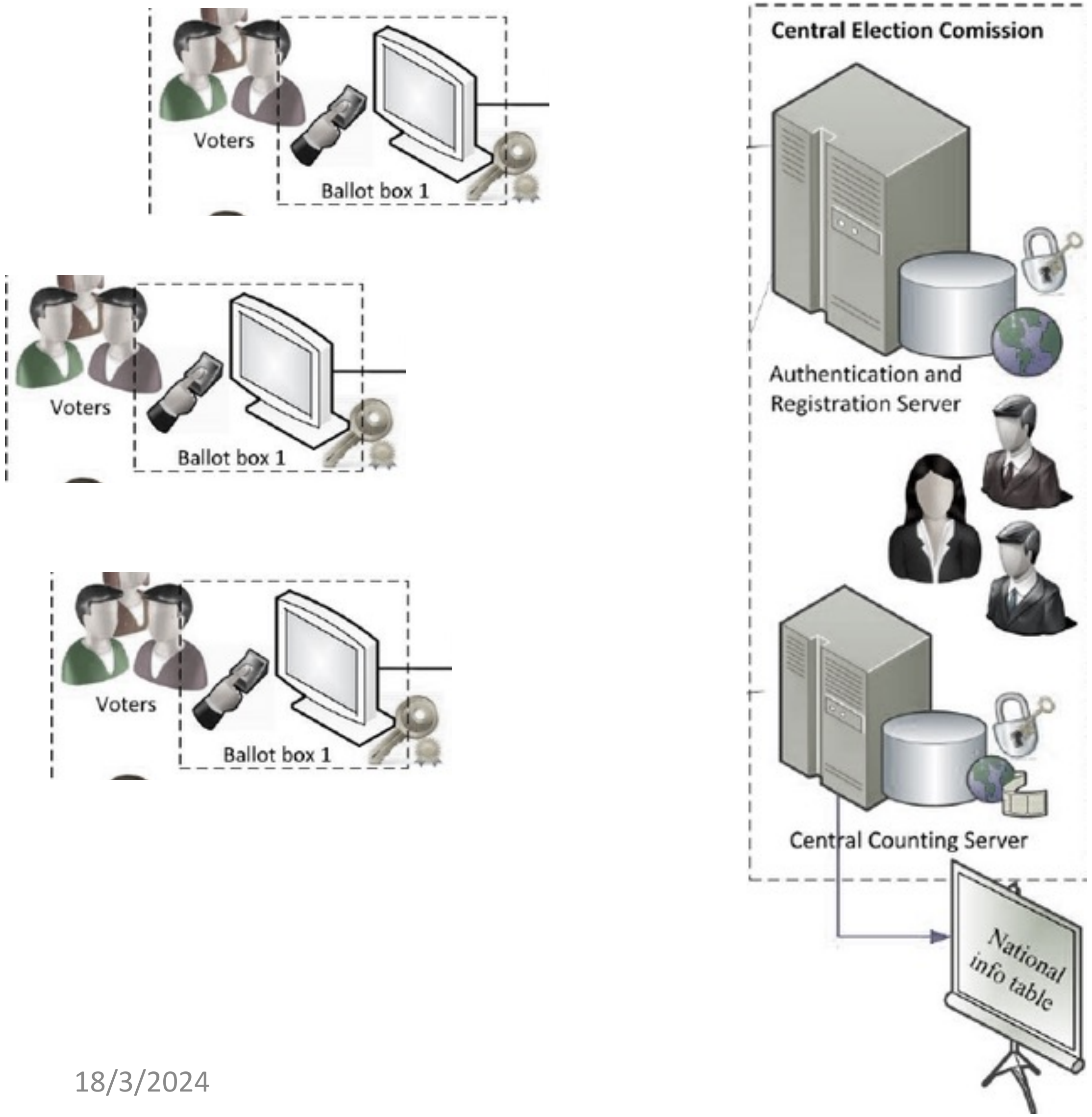


Purchaser



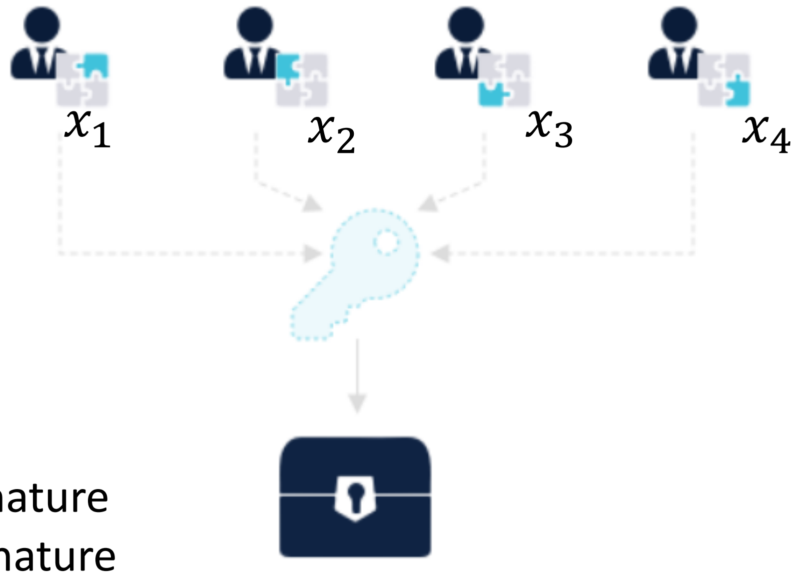
- Farmers make bids ("at price *X*, I will produce *Y* amount")
- Purchaser bids ("at price *X*, I will buy *Y* amount")
- Market clearing price (MCP): price at which total supply = demand

Secure computation examples: voting



- Secure electronic voting is simply computation of the addition function

Secure computation examples: Distribute signature



ECDSA Signature
or RSA signature

- Distribute (ECDSA) signature
- Split the secret signing key into several parts
- such that only they work together can generate the final signature

Secure computation examples: Ad conversion

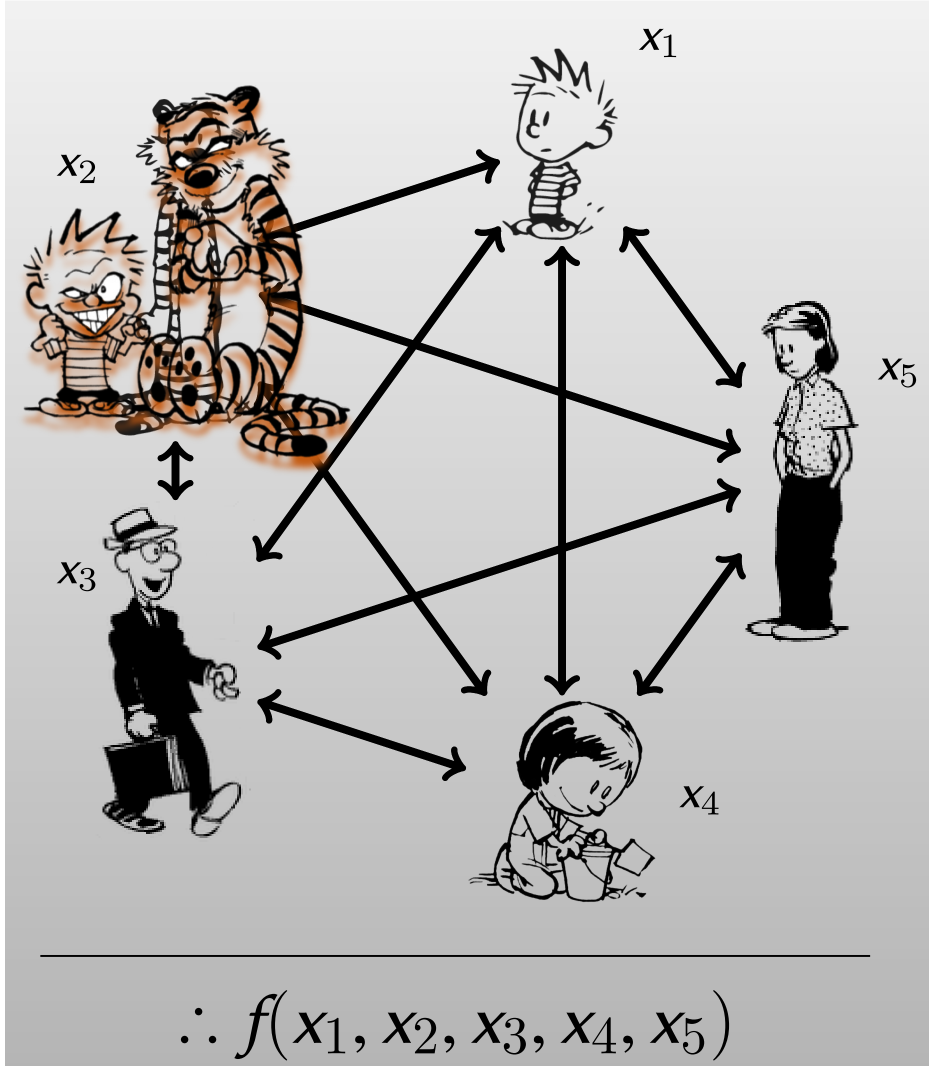
Ad impressions
alice@gmail.com
bob@gmail.com
charlie@gmail.com
dianne@gmail.com
edwin@gmail.com
frank@gmail.com
gina@gmail.com

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

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

- Computed with secure computation by Google and its customers

Secure computation



Premise:

- Mutually distrusting parties, each with a private input
- Learn the result of agreed-upon computation
- E.g, Millionaires Problem, sugar bidding, Ad conversion…

- Security
  - Privacy ("learn no more than" prescribed output)
  - Input independence
  - Etc…

Secure computation

Two or more parties want to perform some joint computation, while guaranteeing "security" against "adversarial behavior".

What does it mean to “security” when computing f?

Or How do we define security here?

Security lists for Bidding

Consider a secure secret Sugar bidding

- An adversary may wish to learn the bids of all parties – to prevent this, require **PRIVACY**
- An adversary may wish to win with a lower bid– to prevent this, require **CORRECTNESS**
- But, the adversary may also wish to ensure that it always gives the highest bid – to prevent this, require **INDEPENDENCE OF INPUTS**
- An adversary may try to abort the execution if its bid is not the highest – require **FAIRNESS**

General security requirement

- **Privacy**: only the output is revealed
- **Correctness**: the function is computed correctly
- **Independence of inputs**: parties cannot choose inputs based on others’ inputs
- **Fairness**: if one party receives output, all receive output

Defining security

- Option 1: analyze security concerns for each specific problem
  - Bidding: as in previous slide
  - E-voting: privacy, correctness and fairness only?

- Problems:
  - How do we know that all concerns are covered?
  - Definitions are application dependent and need to be redefined from scratch for each task

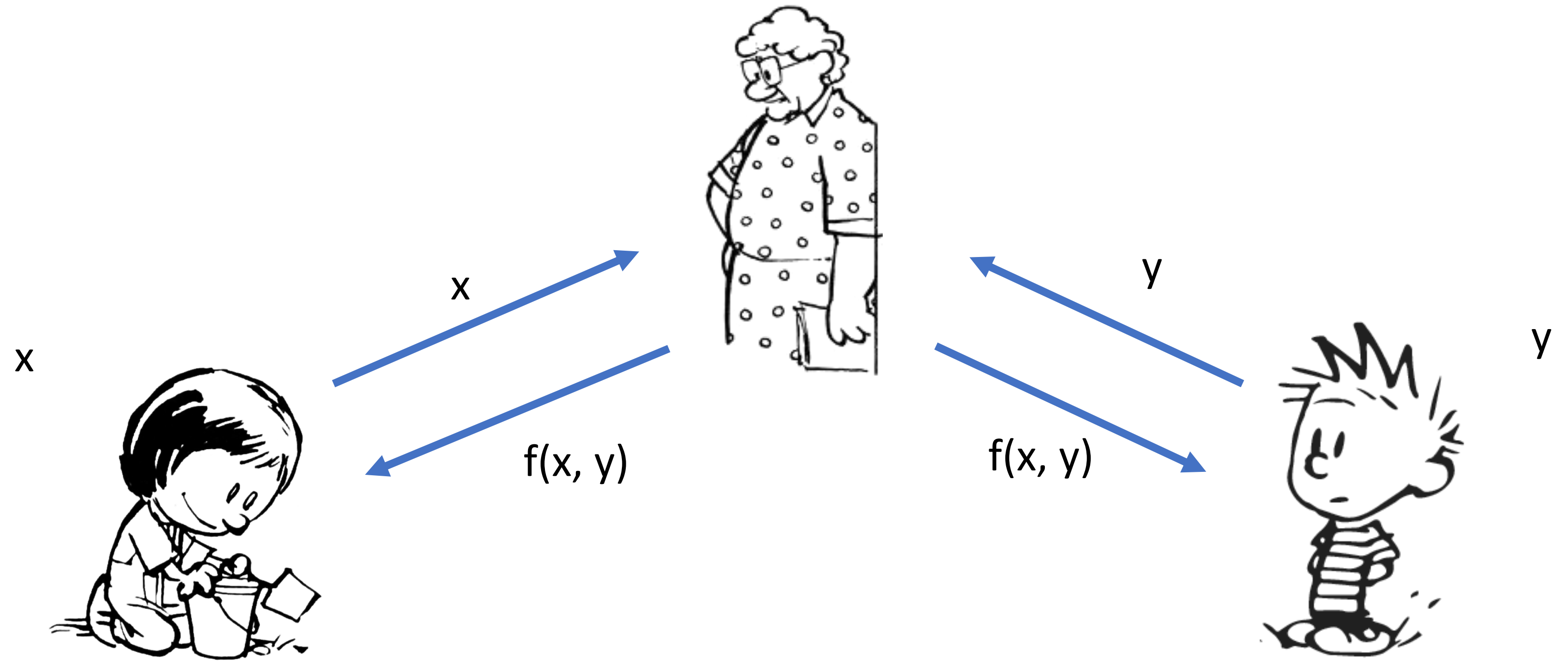
Defining security

- Option 2: general definition that captures all (most) secure computation tasks

- Properties of any such definition
  - Well-defined adversary model
  - Well-defined execution setting
  - Security guarantees are clear and simple to understand

  - How???

Defining security: ideal world



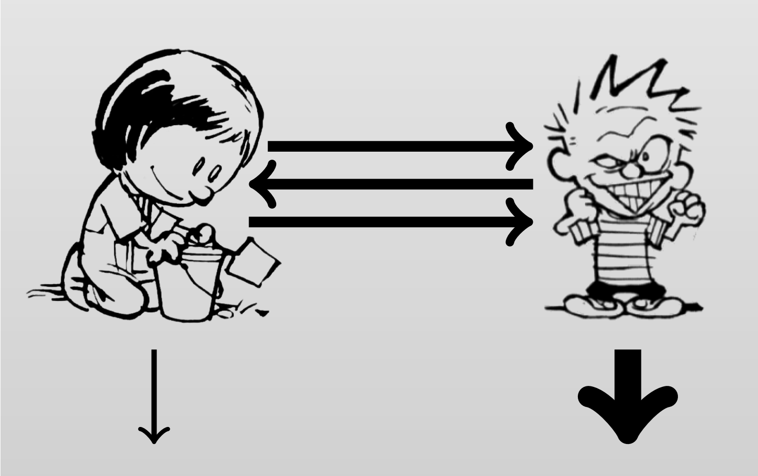
- What can a corrupt party do in this **ideal world?**
  - Choose any input *y* (independent of *x*)
  - Learn only *f*(*x*, *y*), and nothing more
  - Cause honest party to learn *f*(*x*, *y*)

Real-ideal paradigm [GoldwasserMicali84]

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

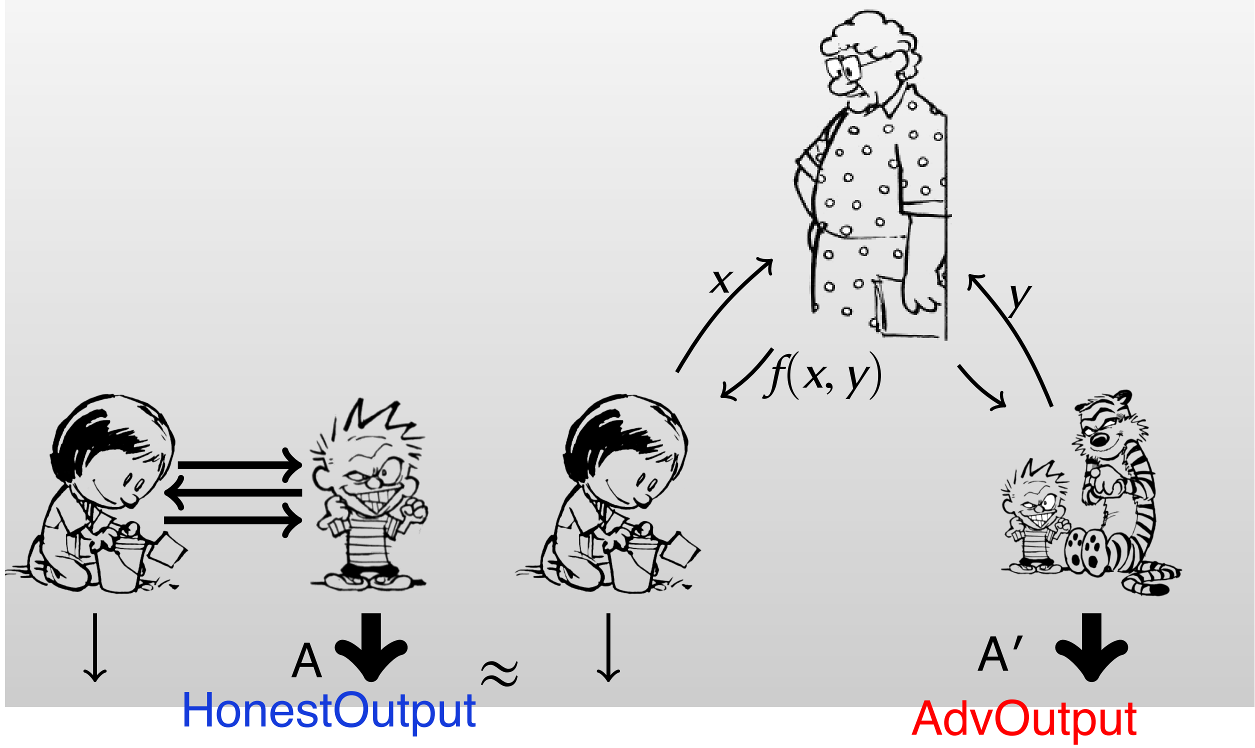
For every "attack" against real protocol, there is a way to achieve "same effect" in ideal world

What is the “effect” of a generic attack?



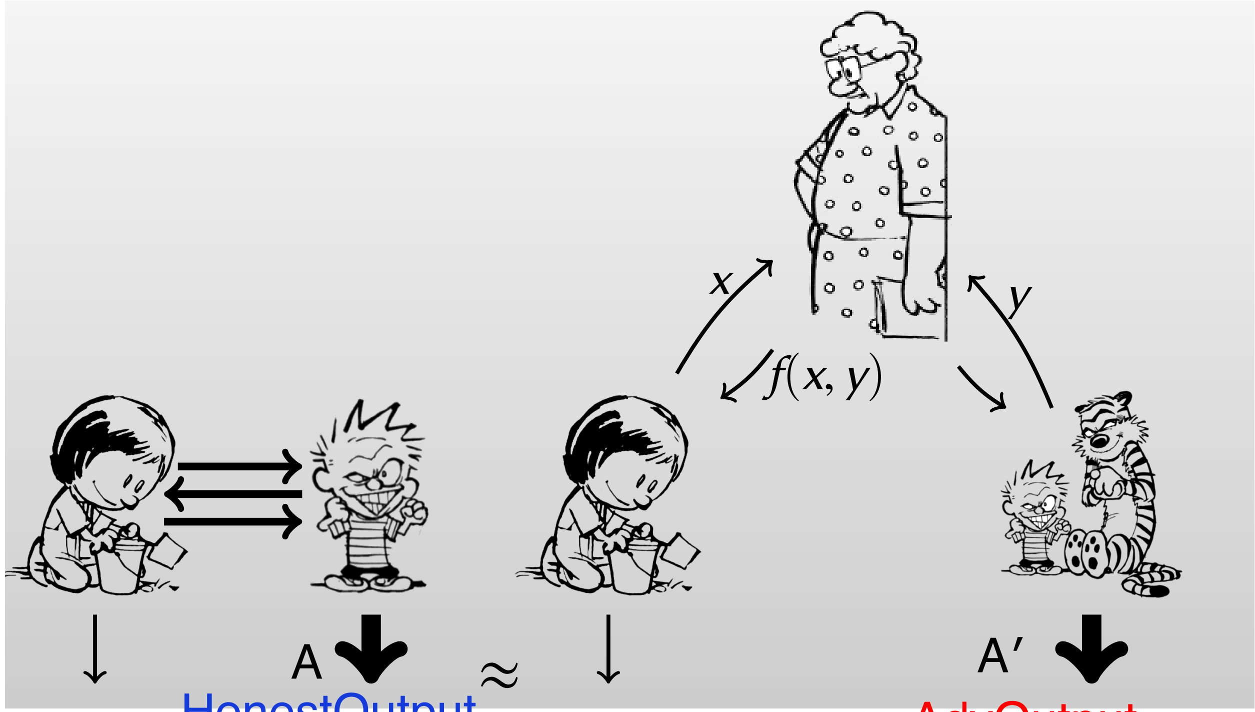
- Something the adversary learns / can compute about honest party
- Some influence on honest party’s output

Define Security



Security definition: For every real-world adversary A, there exists an ideal adversary A′ s.t. joint distribution (HonestOutput, AdvOutput) is indistinguishable

Define Security

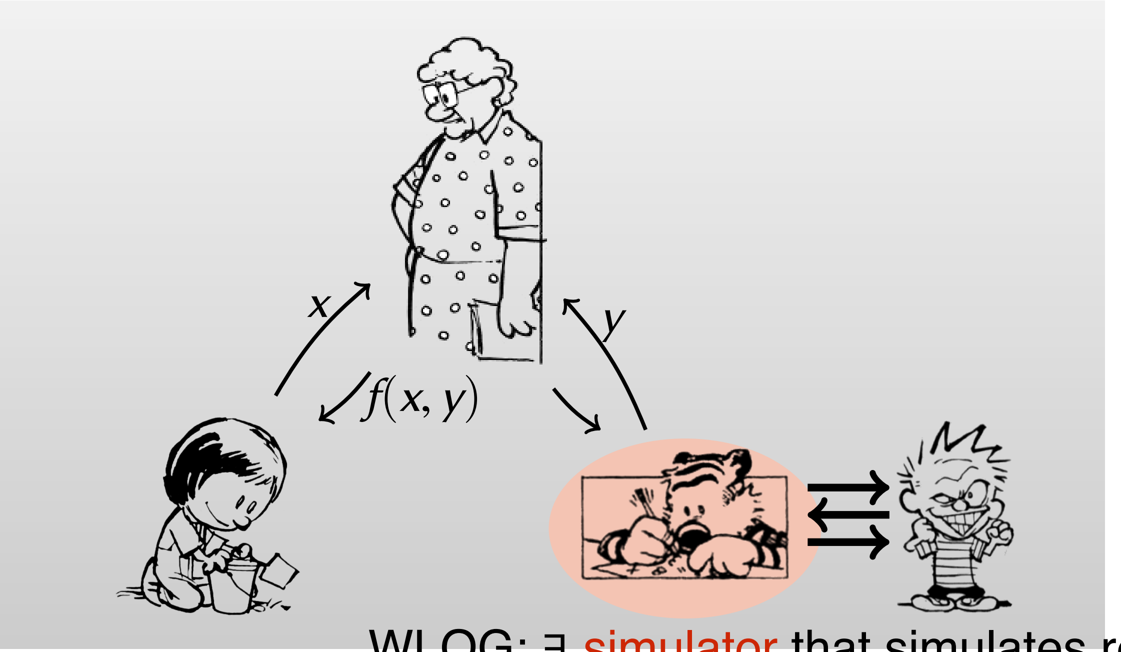


HonestOutput AdvOutput

Security definition: For every real-world adversary A, there exists an ideal adversary A′ s.t. joint distribution (HonestOutput, AdvOutput) is indistinguishable

WLOG: ∃ simulator that simulates real-world interaction in ideal world

Define Security



WLOG: ∃ simulator that simulates real-world interaction in ideal world

Rule 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 message
   - "Explains" the effect on honest party's output in terms of ideal world

Modeling of adversary

- Adversarial behavior
  - Semi-honest: follows the protocol specification
    Tries to learn more than allowed by inspecting transcript
  - Malicious: follows any arbitrary strategy

- Adversarial power
  - Polynomial-time
  - Computationally unbounded: information-theoretic security

Function: Yao's Millionaires' Problem

$$F(x, y) = \begin{cases} (0, 1), & x < y \\ (1, 0), & x \geq y \end{cases}$$

Function: Zero-knowledge proof (or SIGMA protocol)

A NP language $L \coloneqq \{y \mid \exists\, x, s.t. (x, y) \in R\}$ Corresponding Relation R

- Prover with input (x, y) wants to prove that it knows x such that $y \in L$

$$F((y, x), y) = (-, b), b = 1\; if\; (x, y) \in R$$

Why do we say SIGAMA is an honest verifier zero-knowledge?

Schnorr Identification

$$u = g^{\alpha}$$

$P(\alpha)$		$V(u)$
$\alpha_t \xleftarrow{R} \mathbb{Z}_q,\ u_t \leftarrow g^{\alpha_t}$		
	$\xrightarrow{\quad u_t \quad}$	
		$c \xleftarrow{R} \mathcal{C}$
	$\xleftarrow{\quad c \quad}$	
$\alpha_z \leftarrow \alpha_t + \alpha c \ \ mod\ q$		
	$\xrightarrow{\quad \alpha_z \quad}$	
		$g^{\alpha_z} \stackrel{?}{=} u_t \cdot u^c$

- **Correctness(Completeness):** If P and V execute the protocol honestly, the proof is accepted.

If Prover is an adversary

- **oof-of-knowledge):** If the proof is accepted, we can extract the witness (discrete log) α
- **Honest verifier zero-knowledge** says that: without knowing the witness (discrete logarithm), we can generate (simulate) the valid transaction efficiently

If Verifier is a Honest adversary

Function: Zero-knowledge proof (or SIGMA protocol)

A NP language $L \coloneqq \{y \mid \exists\, x, s.t. (x, y) \in R\}$ Corresponding Relation R

- Prover with input (x, y) wants to prove that it knows x such that $y \in L$

$$F((y, x), y) = (-, b), b = 1\; if\; (x, y) \in R$$

- If Prover is the adversary: View is simple; we can extract 'Prover''s input according to **Soundness (proof-of-knowledge):**

- If verifier is a Honest adversary: we can simulate the view according to **Honest verifier zero-knowledge**

Basic tool: Oblivious Transfer (OT)

It is theoretically equivalent to MPC as shown by Kilian (1988):

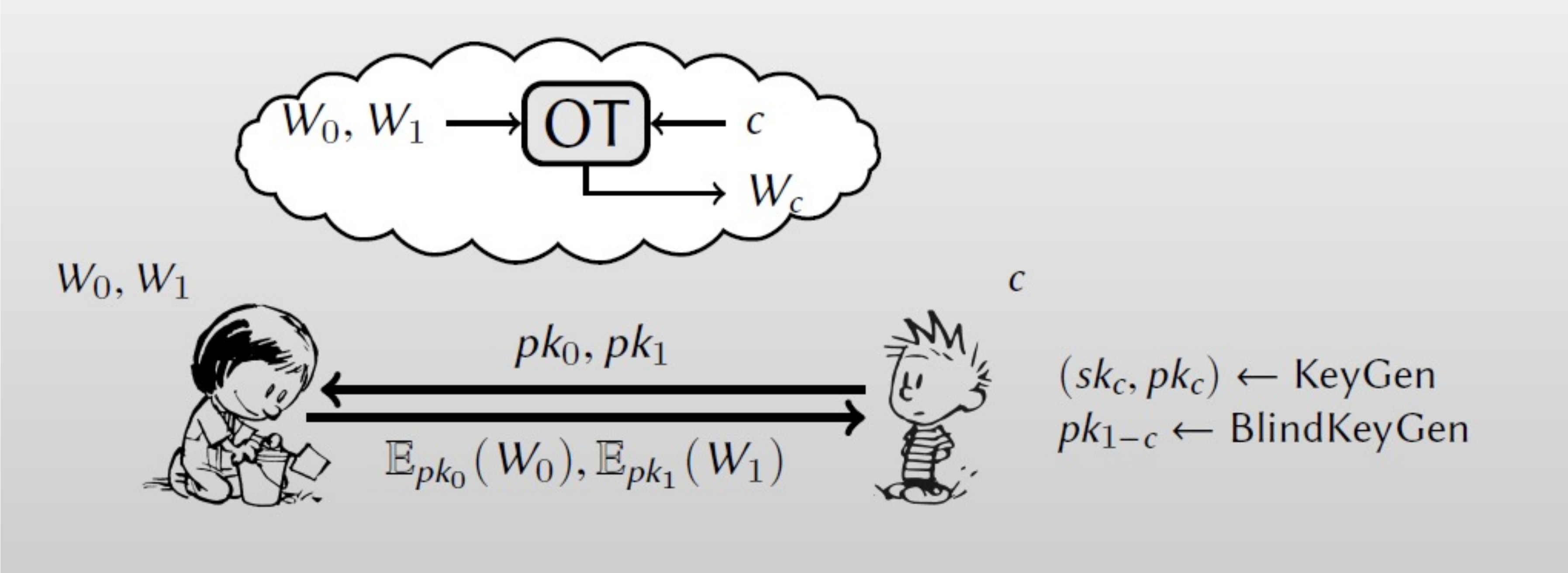
- Given OT, one can build MPC without any additional assumptions
- Similarly, one can directly obtain OT from MPC

Oblivious Transfer (OT)

- The standard definition of 1-out-of-2 OT involves two parties, a Sender S holding two secrets m_0, m_1, and a receiver R holding a choice bit b ∈ {0, 1}
- OT is a protocol allowing R to obtain m_b while learning nothing about the "other" secret m_{1-b}
- At the same time, S does not learn anything at all

How to construct OT?

- Semi-honest



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

- sample a public key without knowledge of the secret key
- E.g.: ElGamal

Function for OT

- A 1-out-of-2 OT is a cryptographic protocol securely implementing the functionality F^{OT} defined below:

- Parameters:

  Two parties: Sender S and Receiver R.

  S has input secrets $\mathrm{m}_0, \mathrm{m}_1$ and R has a selection bit b ∈ {0, 1}

Functionality F^{OT} :

S sends $\mathrm{m}_0, \mathrm{m}_1$ to F^{OT}, and R sends b to F^{OT}

R receives m_b, and S receives ⊥

Timetable: MPC

Diffie

Hellman

1976

New directions

Rivest

Shamir **Adelman**

1977

RSA

Rivest

Adelman **Dertouzos**

1978

Homomorphic Enc

Yao

1982

MPC

Goldwasser

Micali **Rackoff**

1985

Zero Knowledge

History of MPC

- The idea of secure computation was introduced by Andrew Yao in the early 1980s (Yao, 1982)

- Secure computation was primarily of only theoretical interest for the next twenty years

- In the early 2000s, algorithmic improvements and computing costs make it more realistic to build practical systems, e.g. Fairplay (Malkhi et al., 2004)

- Since then, the speed of MPC protocols has improved by more than five orders of magnitude

Our step

1 **Secure computation**: Concepts & definitions

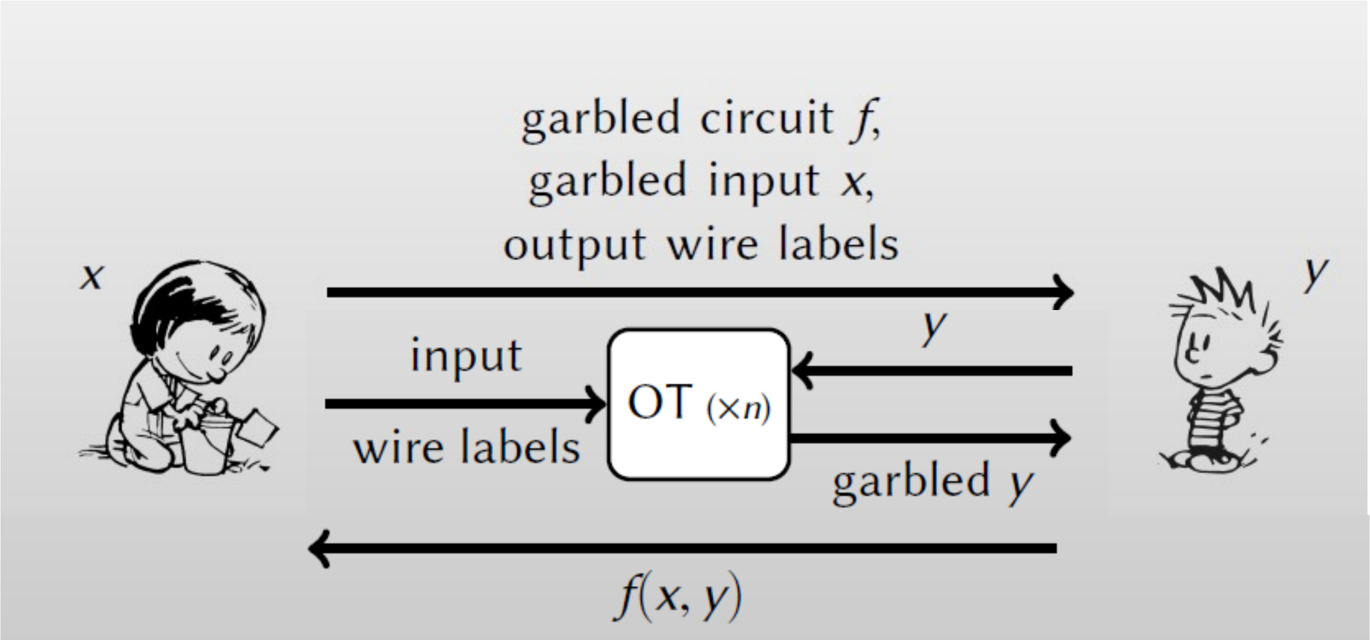
2 **General constructions**: Yao's protocol, and GMW

3 **Custom protocol**: private set intersection

First: Two-party computation

- Every computation of function could be transferred to computing a Boolean circuit.

- Yao’s protocol: semi-honest secure (2-party) computation for Boolean circuits



Before we start, , so we focus on semi-honest case

Malicious secure MPC for any circuit

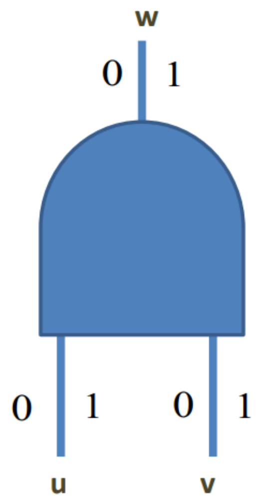
GMW compiler [GMW87]
Commitment
Zero-knowledge proof

Semi-honest secure MPC for any circuit
Goldreich-Micali-Wigderson (GMW)
Yao etc.

[GMW87]Goldreich, O., S. Micali, and A. Wigderson. 1987. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority".

Yao’s Garble Circuit (two-party, Boolean)

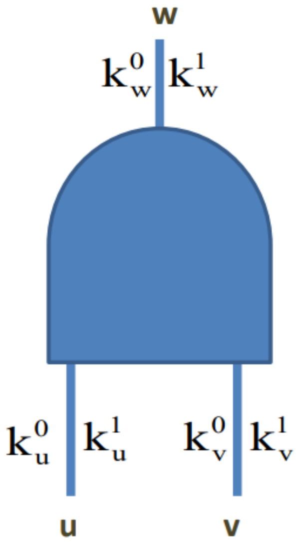
- Take AND gate for example
- $F(u, v) = (w, w)$



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

Yao's Garble Circuit (two-party, Boolean)

- $F(u, v) = (w, w)$

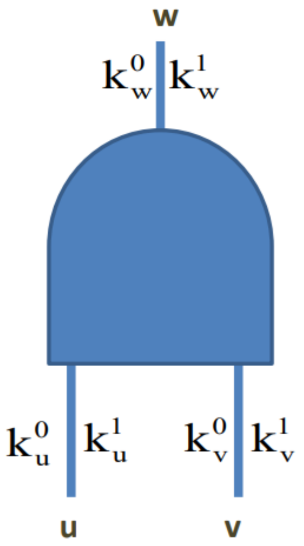


$E_{k_1}(E_{k_2}(m))$ is the double AES enc of m with k_1 and k_2

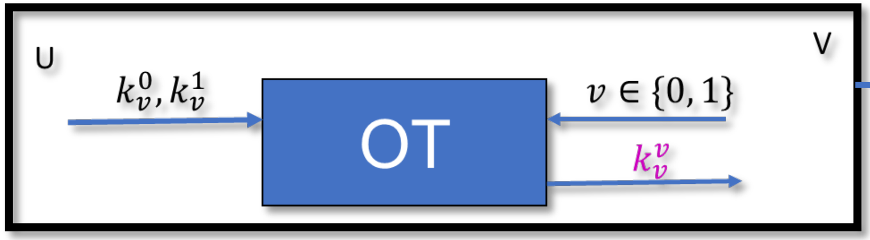
u	v	w
k_u^0	k_v^0	$E_{k_u^0}(E_{k_v^0}(k_w^0))$
k_u^0	k_v^1	$E_{k_u^0}(E_{k_v^1}(k_w^0))$
k_u^1	k_v^0	$E_{k_u^1}(E_{k_v^0}(k_w^0))$
k_u^1	k_v^1	$E_{k_u^1}(E_{k_v^1}(k_w^1))$

- U sends all the ciphertexts $E_{k'}(E_{k''}(k'''))$ in volume w to V
- U sends k_u^u to V
- U sends k_w^0, k_w^1 to V

Yao's Garble Circuit (two-party, Boolean)

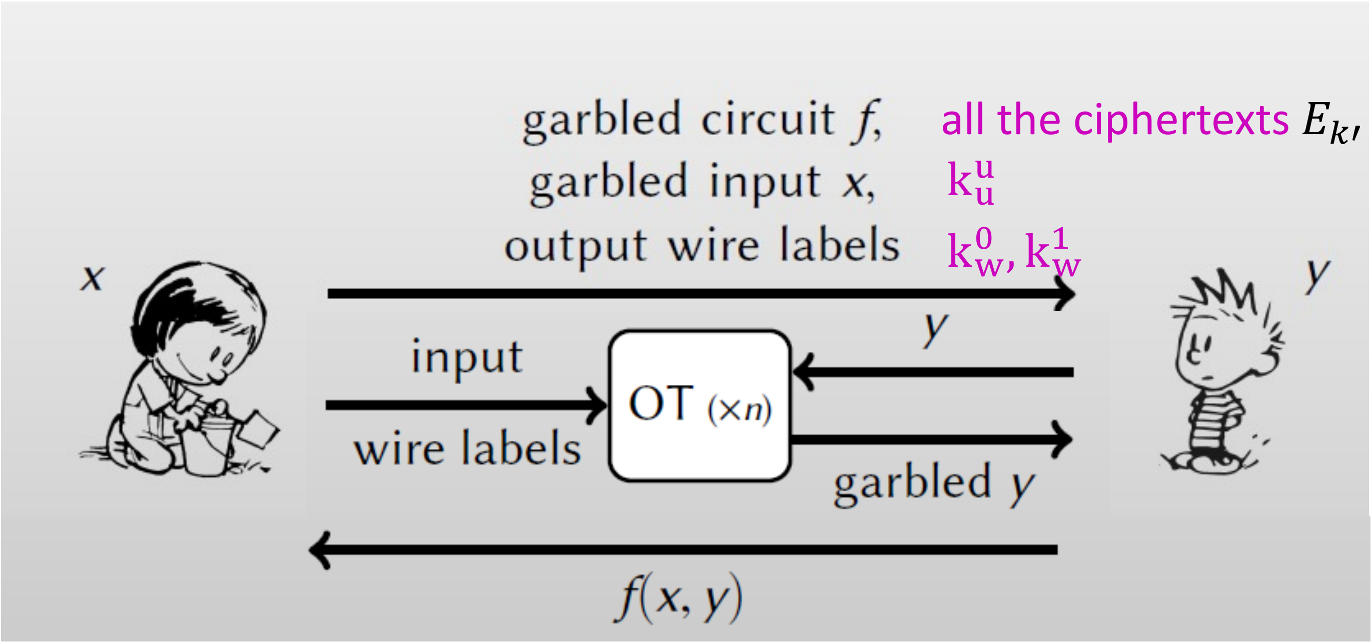


u	v	w
k_u^0	k_v^0	$E_{k_u^0}(E_{k_v^0}(k_w^0))$
k_u^0	k_v^1	$E_{k_u^0}(E_{k_v^1}(k_w^0))$
k_u^1	k_v^0	$E_{k_u^1}(E_{k_v^0}(k_w^0))$
k_u^1	k_v^1	$E_{k_u^1}(E_{k_v^1}(k_w^1))$



- all the ciphertexts $E_{k'}(E_{k''}(k'''))$ in volume w,
- k_u^u
- k_w^0, k_w^1

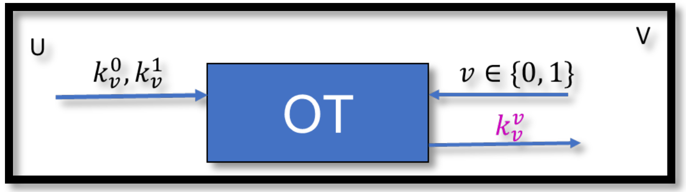
With k_u^u and k_v^v , V can decrypt k_w^w



all the ciphertexts $E_{k'}$ $(E_{k''}$ $(k'''))$ in volume w,

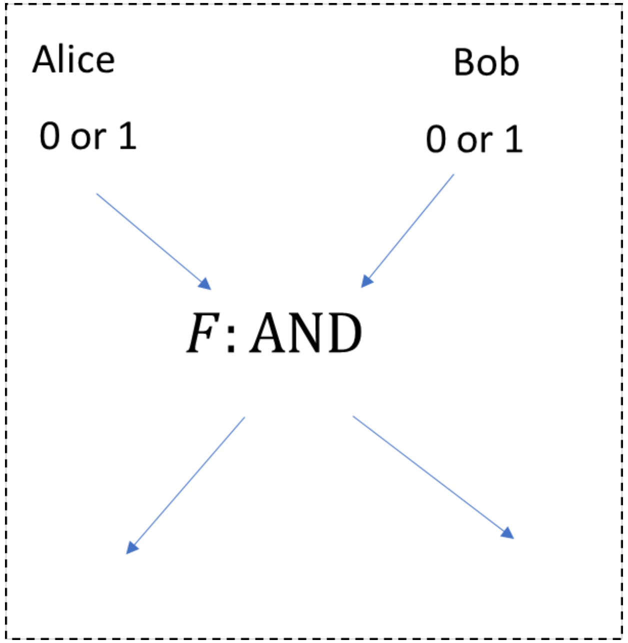
$\mathrm{k_u^u}$

$\mathrm{k_w^0, k_w^1}$



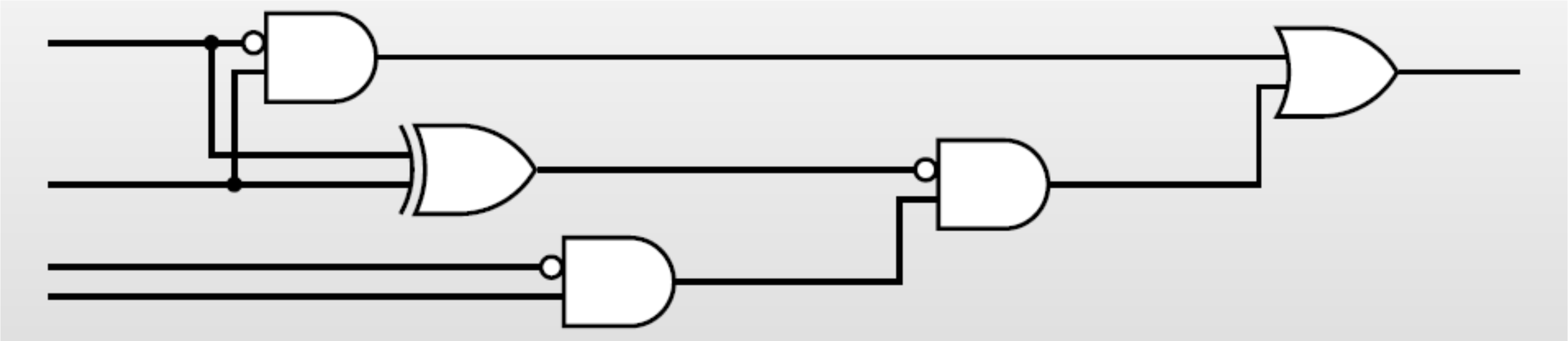
A fun application

- Bob and Alice want to check if they are interested in dating
  - If both are yes, the output is yes
  - If one is no, the output is no

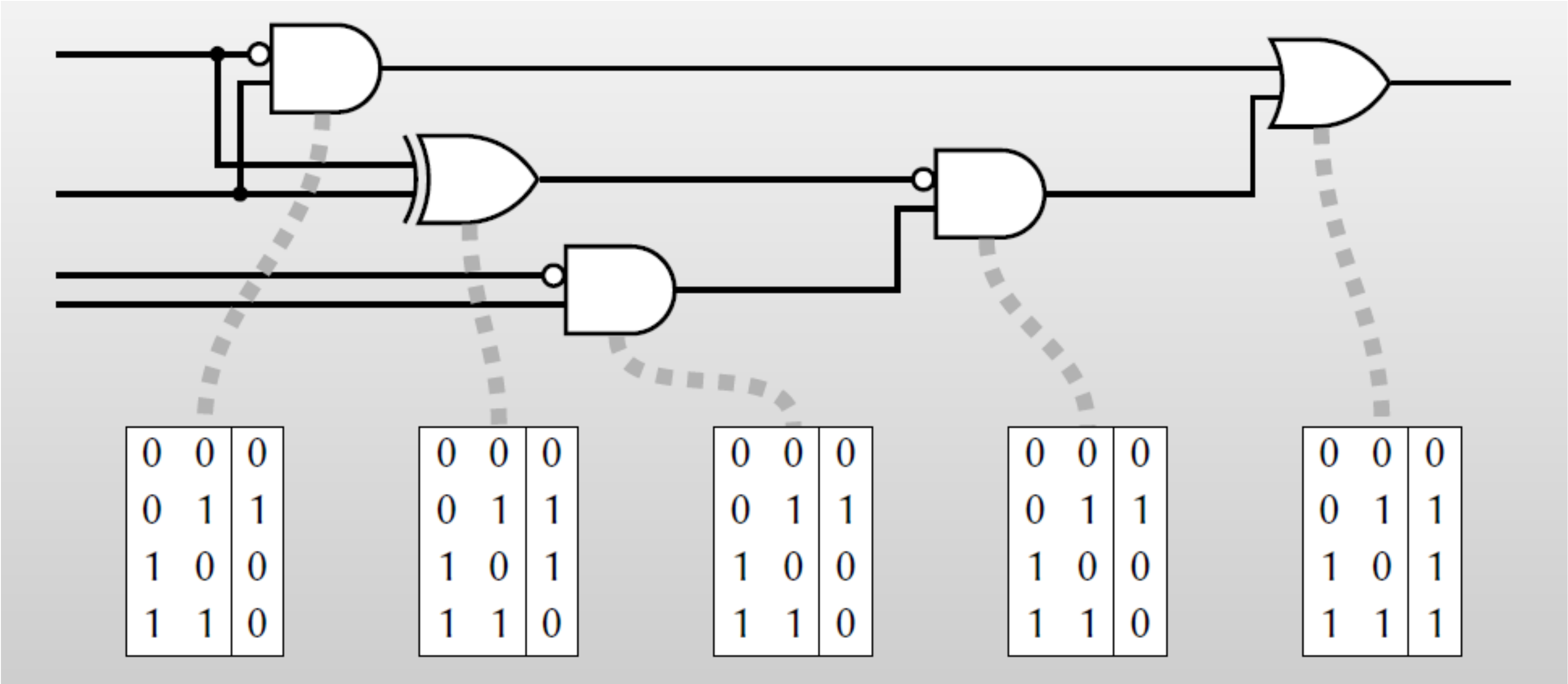


<Pride and Prejudice>

Garbled general circuit framework

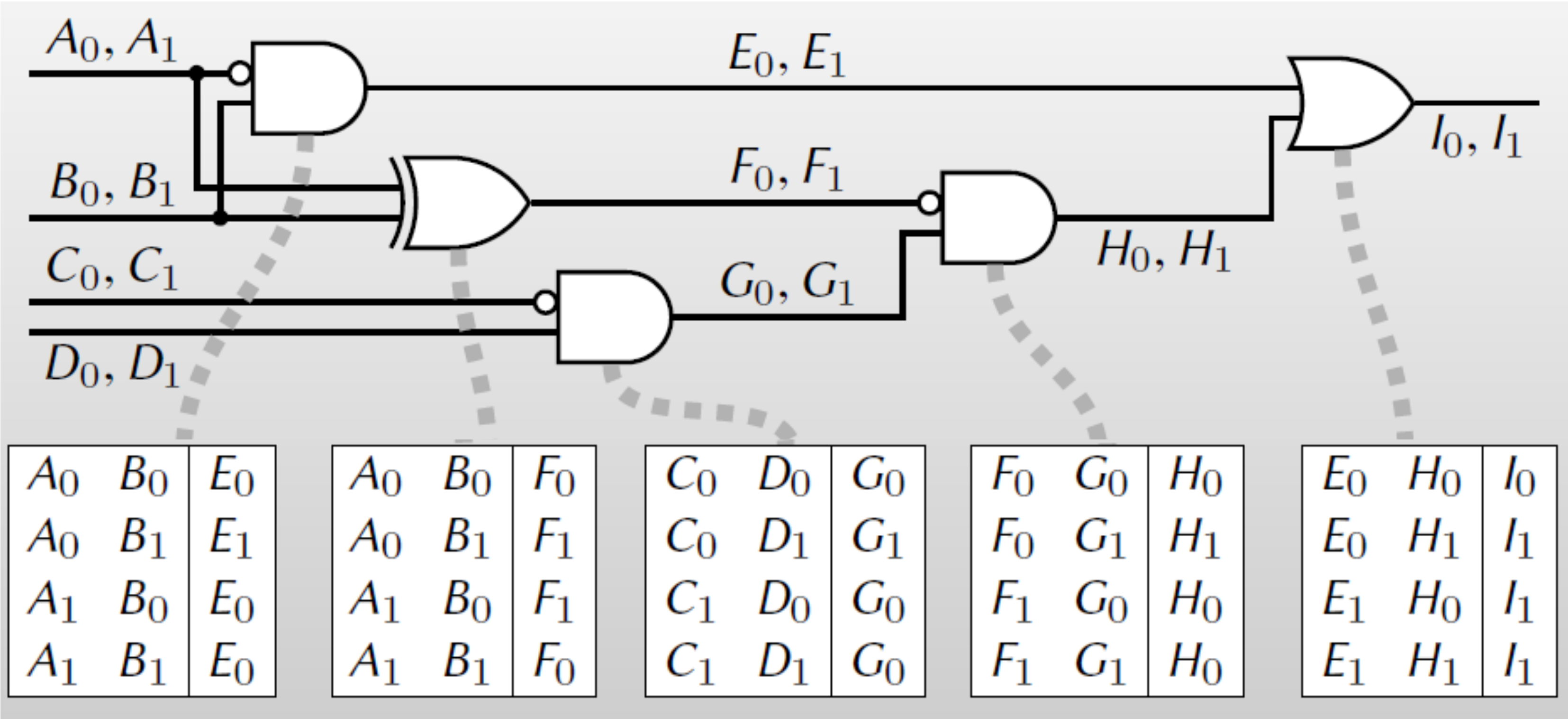


Garbled general circuit framework



Garbling a circuit:

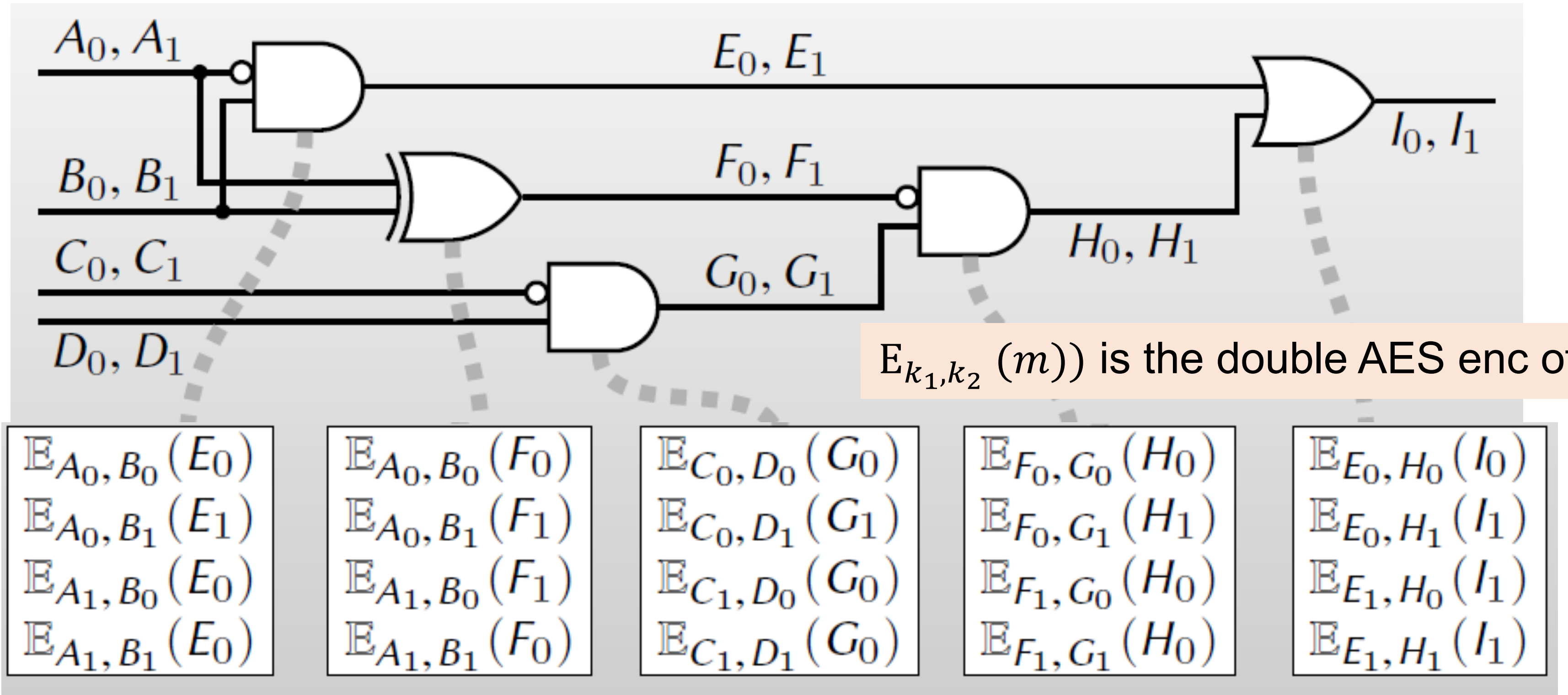
Garbled general circuit framework



Garbling a circuit:

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

Garbled general circuit framework

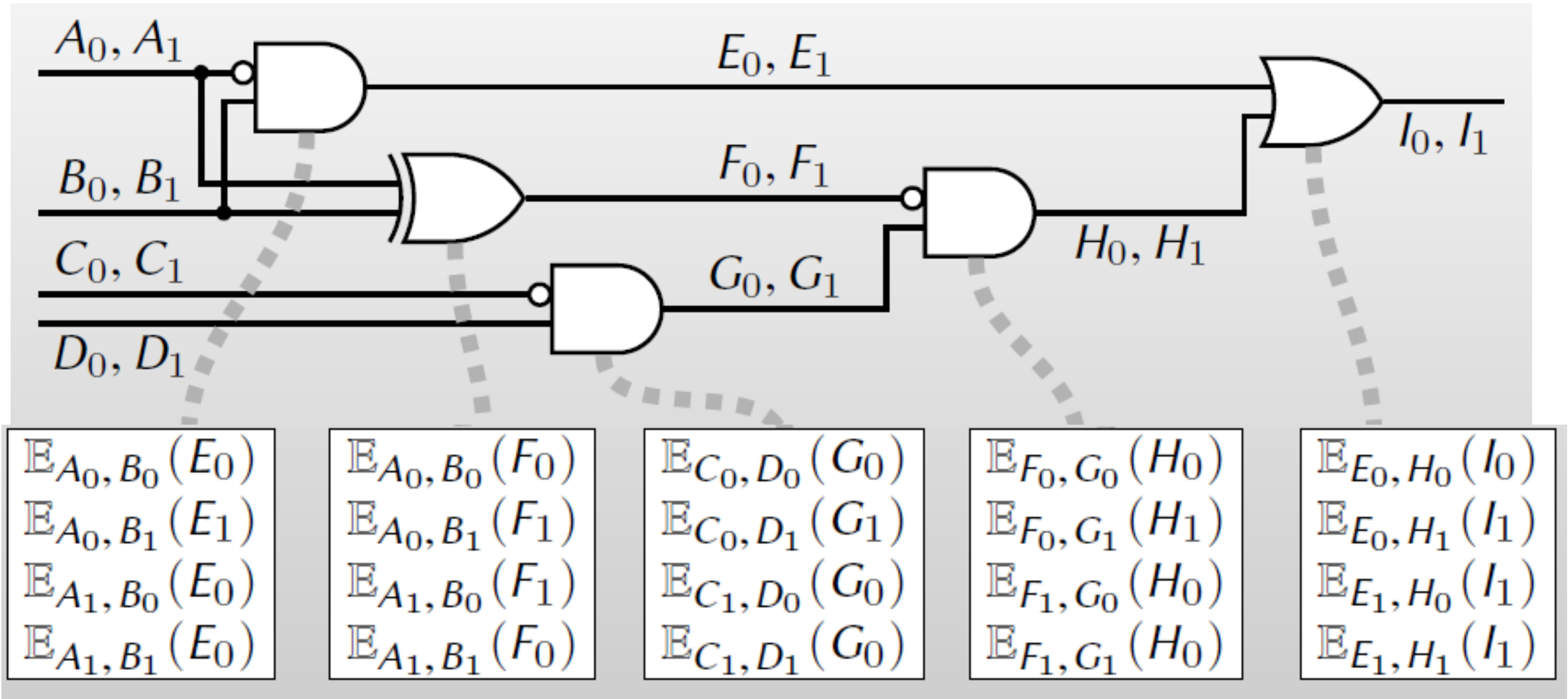


$\mathrm{E}_{k_1,k_2}(m))$ is the double AES enc of m with k_1 and k_2

Garbling a circuit:

- Pick random **labels** $W_0; \ W_1$ on each wire
- “Encrypt” truth table of each gate

Garbled general circuit framework

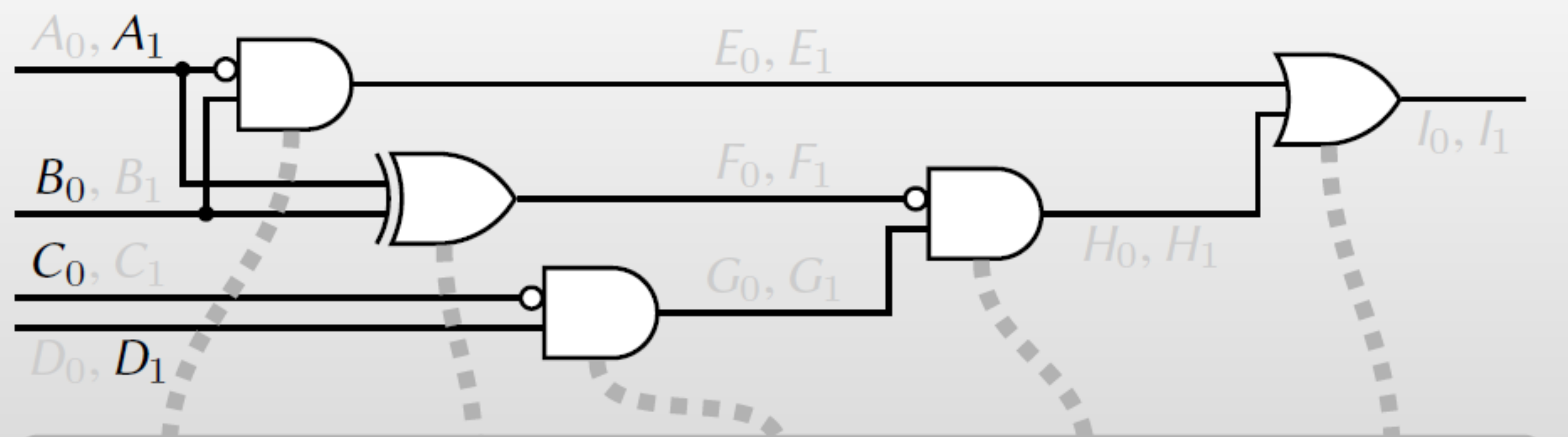


Garbling a circuit:

- Pick random **labels** W_0; W_1 on each wire
- “Encrypt” truth table of each gate

Garbled evaluation:

Garbled general circuit framework



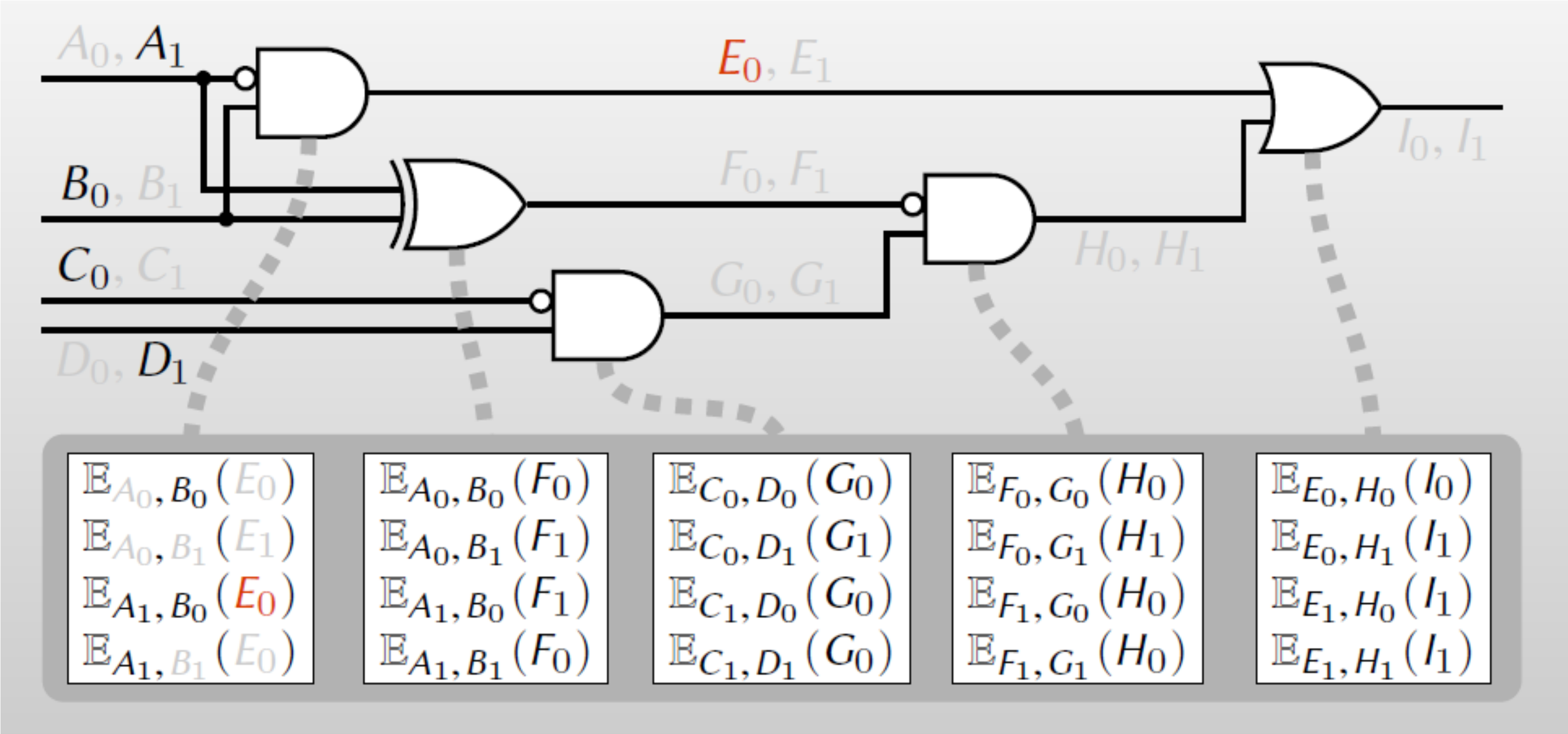
$\mathbb{E}_{A_0,B_0}(E_0)$	$\mathbb{E}_{A_0,B_0}(F_0)$	$\mathbb{E}_{C_0,D_0}(G_0)$	$\mathbb{E}_{F_0,G_0}(H_0)$	$\mathbb{E}_{E_0,H_0}(I_0)$
$\mathbb{E}_{A_0,B_1}(E_1)$	$\mathbb{E}_{A_0,B_1}(F_1)$	$\mathbb{E}_{C_0,D_1}(G_1)$	$\mathbb{E}_{F_0,G_1}(H_1)$	$\mathbb{E}_{E_0,H_1}(I_1)$
$\mathbb{E}_{A_1,B_0}(E_0)$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

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:

Garbled general circuit framework



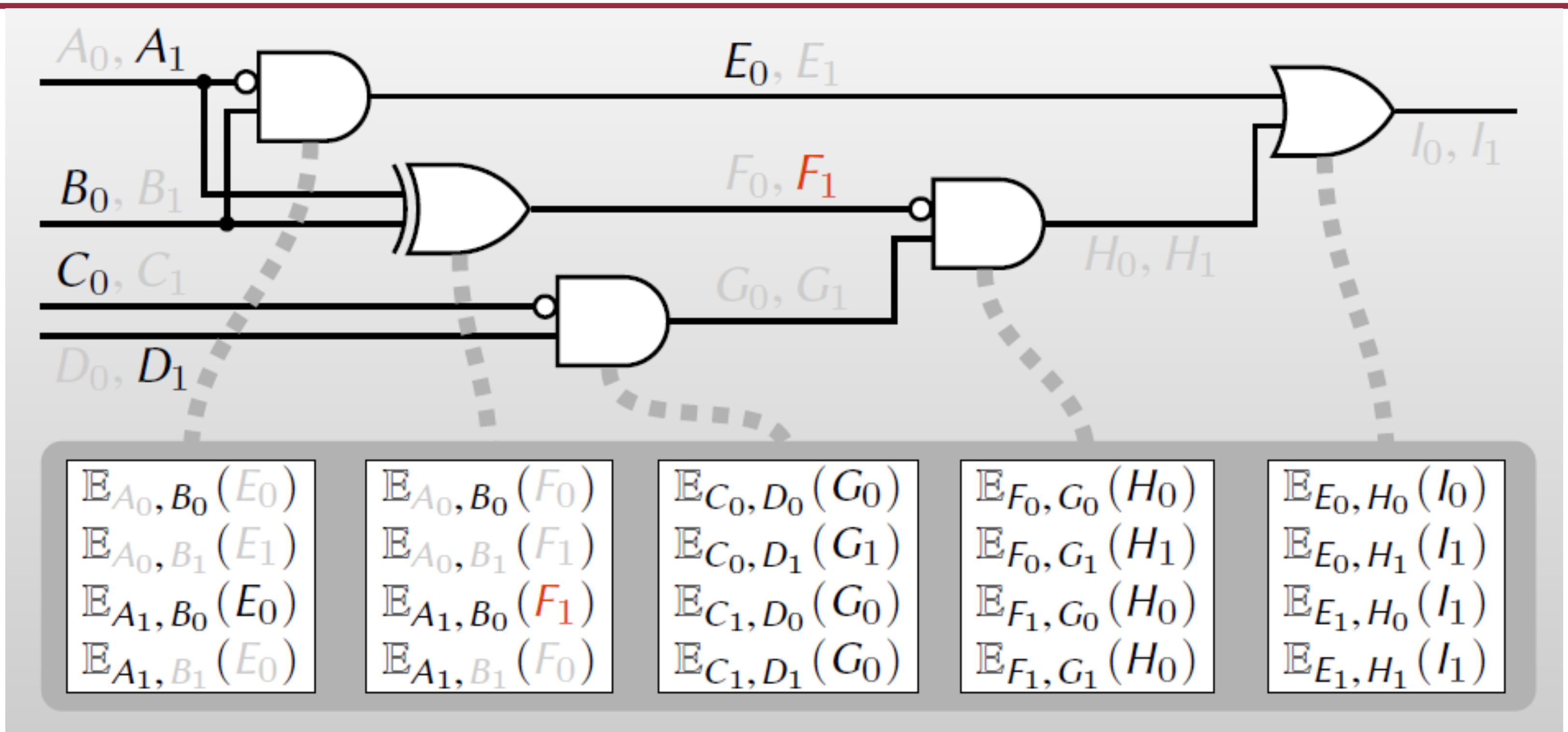
Garbling a circuit:

- 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 general circuit framework



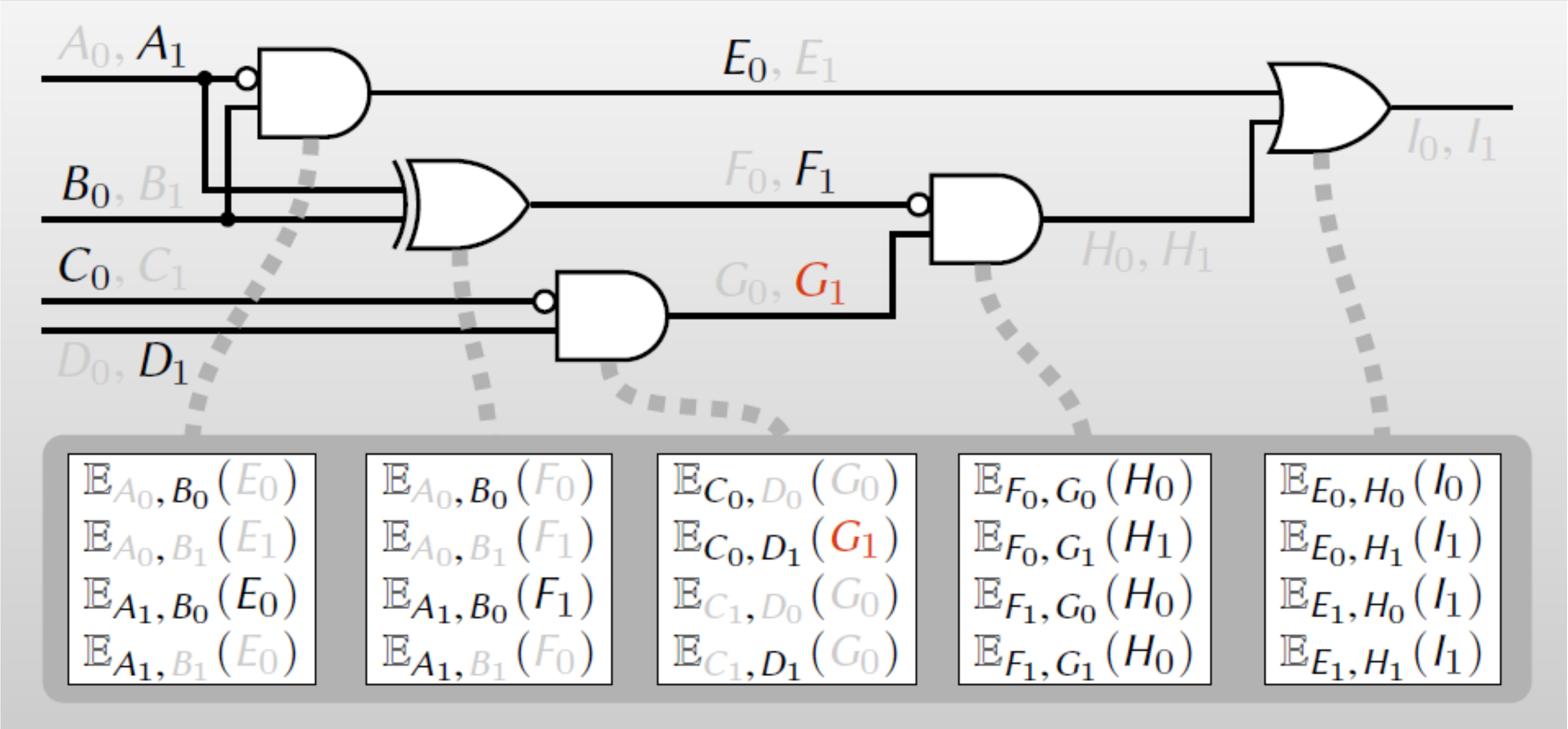
Garbling a circuit:

- 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 general circuit framework



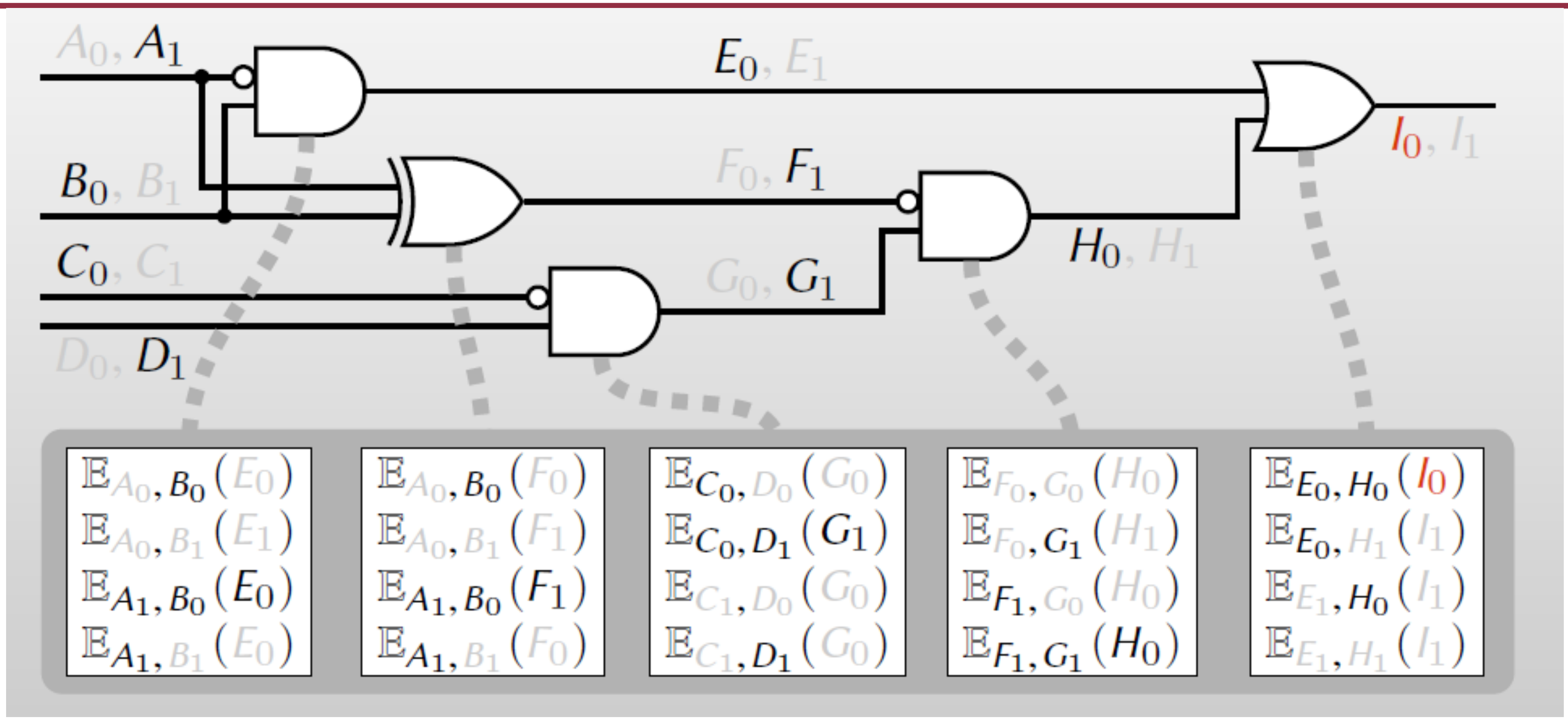
Garbling a circuit:

- 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 general circuit framework



Garbling a circuit:

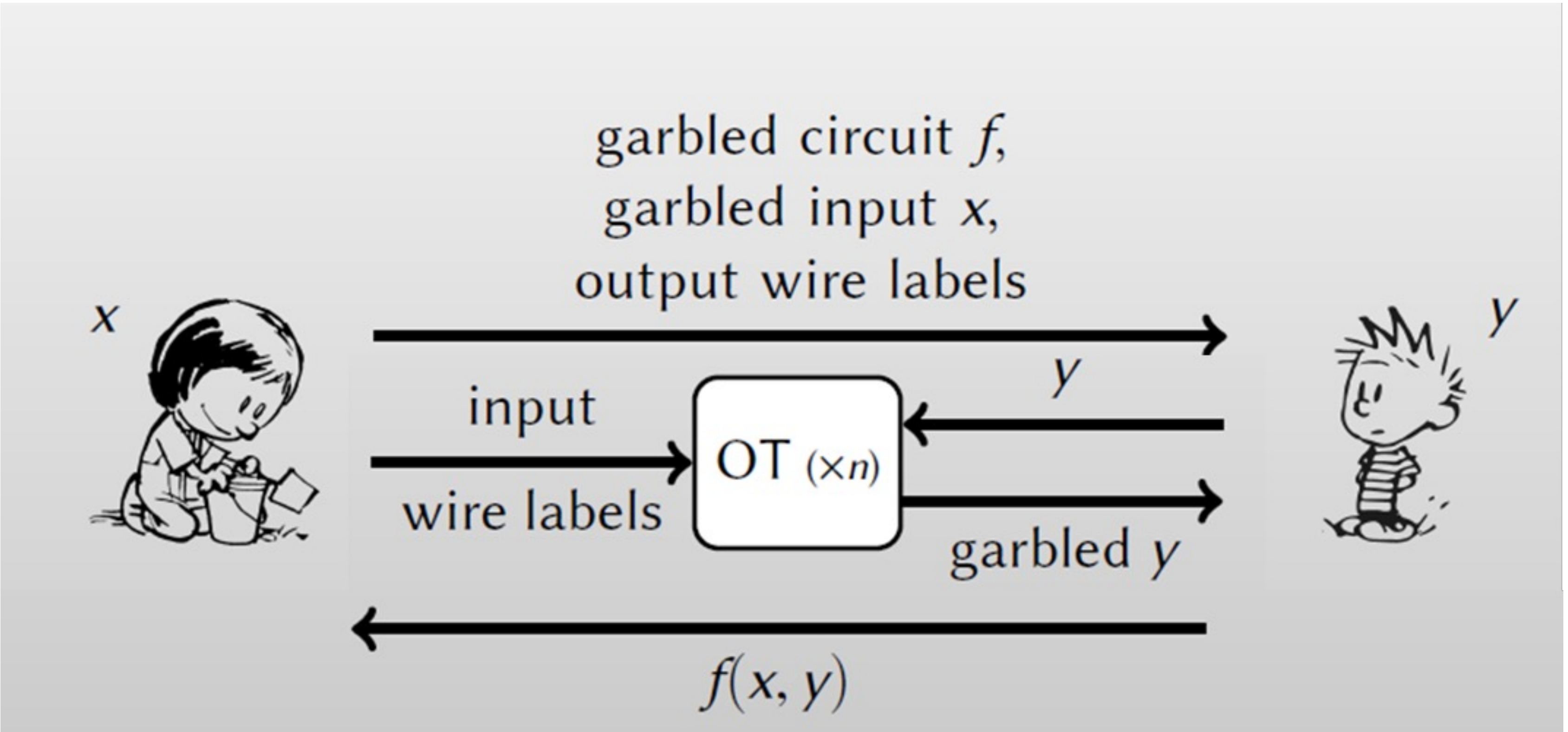
- Pick random **labels** $W_0;\ W_1$ on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** all encrypted gates
- **Garbled encoding** one label per wire

Garbled evaluation:

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

Security

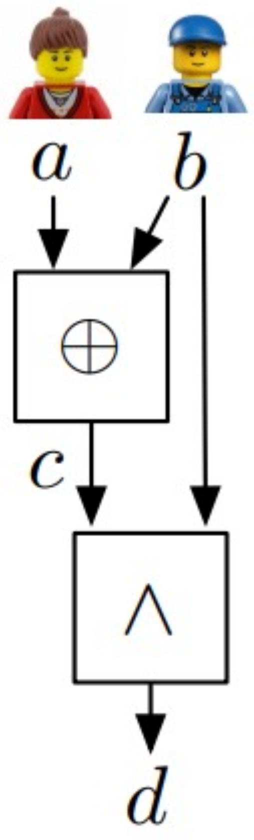
Yao's Protocol



- Two party
- For a Boolean circuit.

How about Multi-party and arithmetic / Boolean circuit?

GMW (multiparty, Boolean)



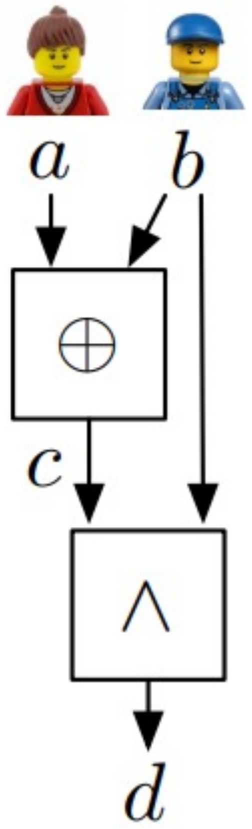
[GMW87]Goldreich, O., S. Micali, and A. Wigderson. 1987. “How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority”.

GMW (multiparty, Boolean)


Secret share inputs:

$a = a_1 \oplus a_2$

$b = b_1 \oplus b_2$



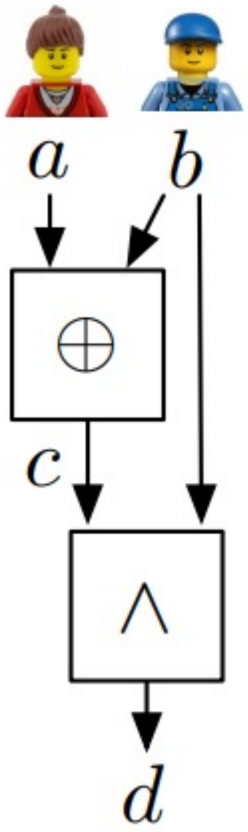
GMW (multiparty, Boolean)

Secret share inputs: $a = a_1 \oplus a_2$

$b = b_1 \oplus b_2$

Non-Interactive XOR gates: $c_1 = a_1 \oplus b_1$; $c_2 = a_2 \oplus b_2$

Interactive AND gates:



GMW (multiparty, Boolean)

Interactive AND gates:

- One AND gate requires the execution of 1-out-of-4 OT

$$d_2 = (c_1 \oplus c_2)(b_1 \oplus b_2) - d_1$$

$(c_1 \oplus 0)(b_1 \oplus 0) - d_1,$
$(c_1 \oplus 0)(b_1 \oplus 1) - d_1,$
$(c_1 \oplus 1)(b_1 \oplus 0) - d_1,$
$(c_1 \oplus 1)(b_1 \oplus 1) - d_1$

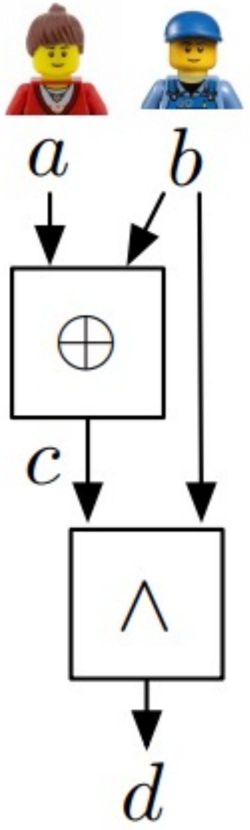
GMW (multiparty, Arithmetic/Boolean)

Secret share inputs: $a = a_1 \oplus a_2$

$b = b_1 \oplus b_2$

Non-Interactive XOR gates: $c_1 = a_1 \oplus b_1$; $c_2 = a_2 \oplus b_2$

Interactive AND gates:



Not difficult to extend to Multi-party by using 1-out-of-k OT

Our step

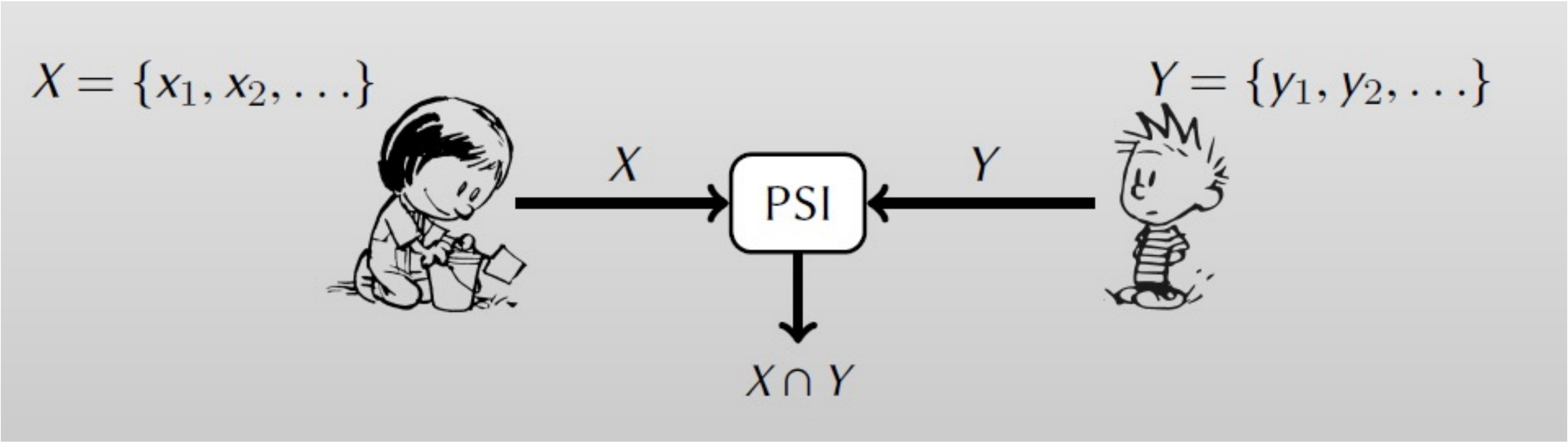
1 **Secure computation**: Concepts & definitions

2 **General constructions**: Yao's protocol, and others

3 **Custom protocol**: private set intersection

Custom protocol: private set intersection (PSI)

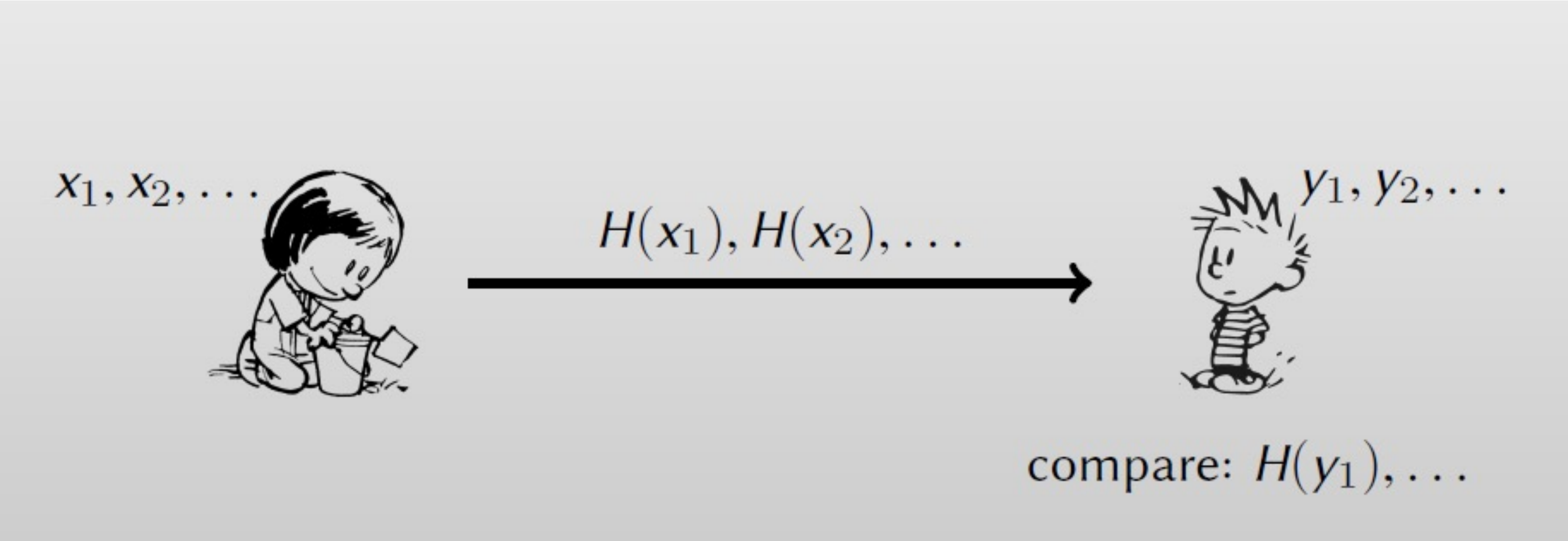
Special case of secure 2-party computation:



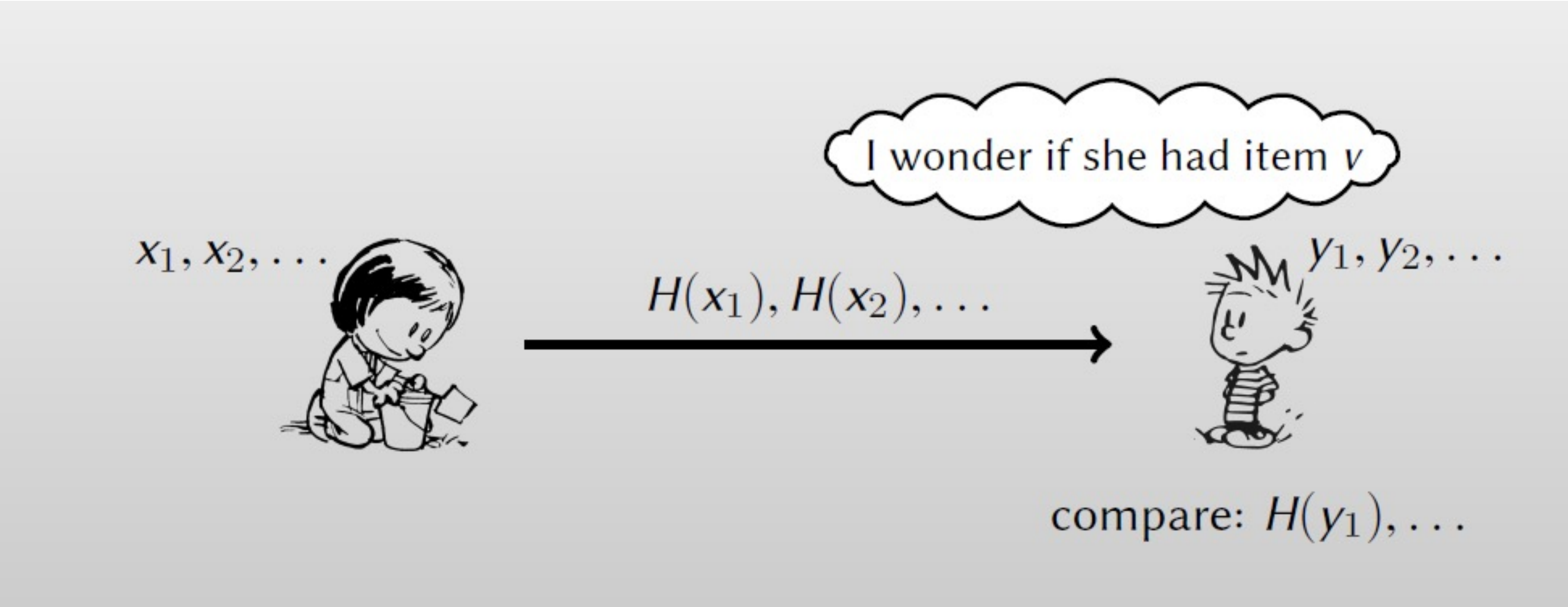
PSI applications

- Contact discovery, when signing up for WhatsApp
  - X = address book in my phone (phone numbers)
  - Y = WhatsApp user database
- Private scheduling
  - X = available timeslots on my calendar
  - Y = available timeslots on your calendar
- Ad conversion rate
  - X = users who saw the advertisement
  - Y = customers who bought the product
- etc

“Obvious” protocol

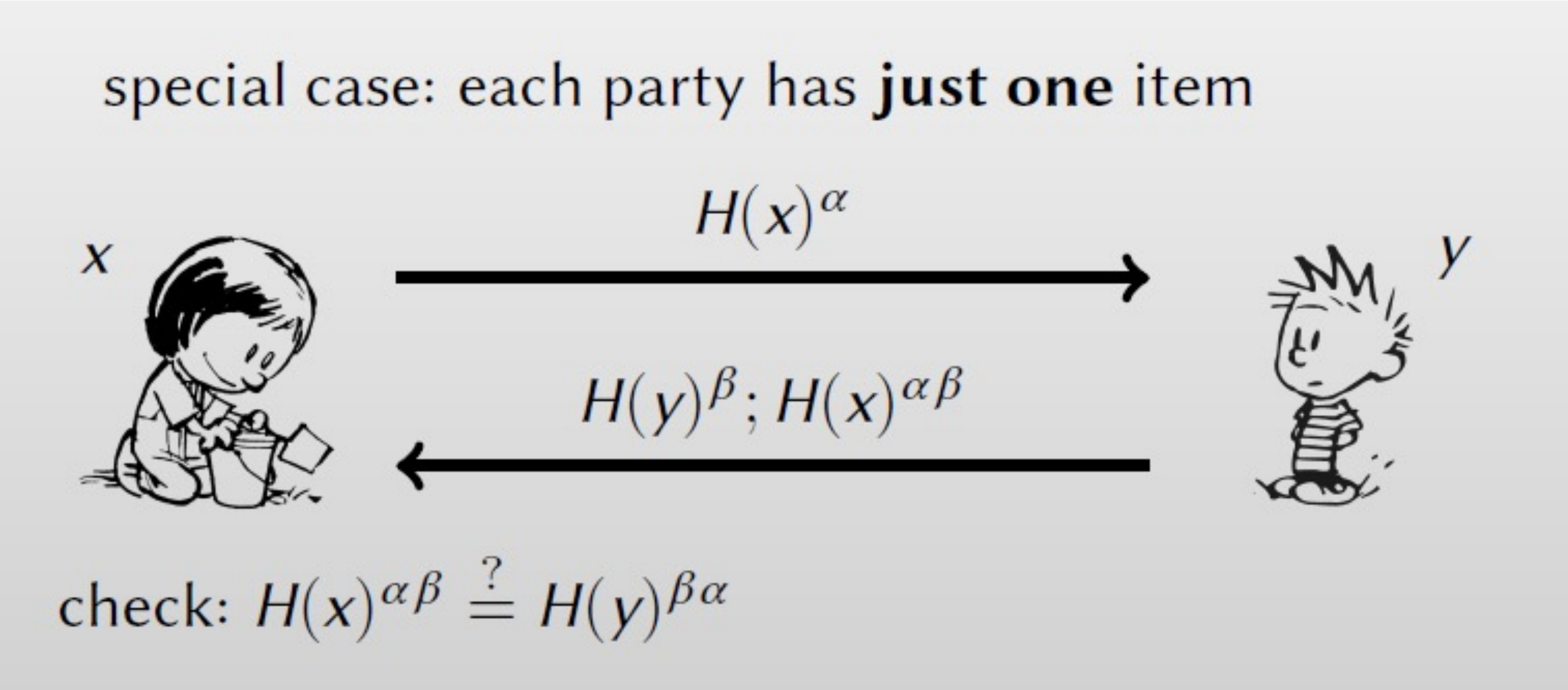


“Obvious” protocol



- INSECURE: Receiver can test any $v \in \{x_1, x_2, \cdots\}$ or not offline
- Problematic if items have low entropy (e.g., phone numbers)

Classical protocol: Diffie-Hellman



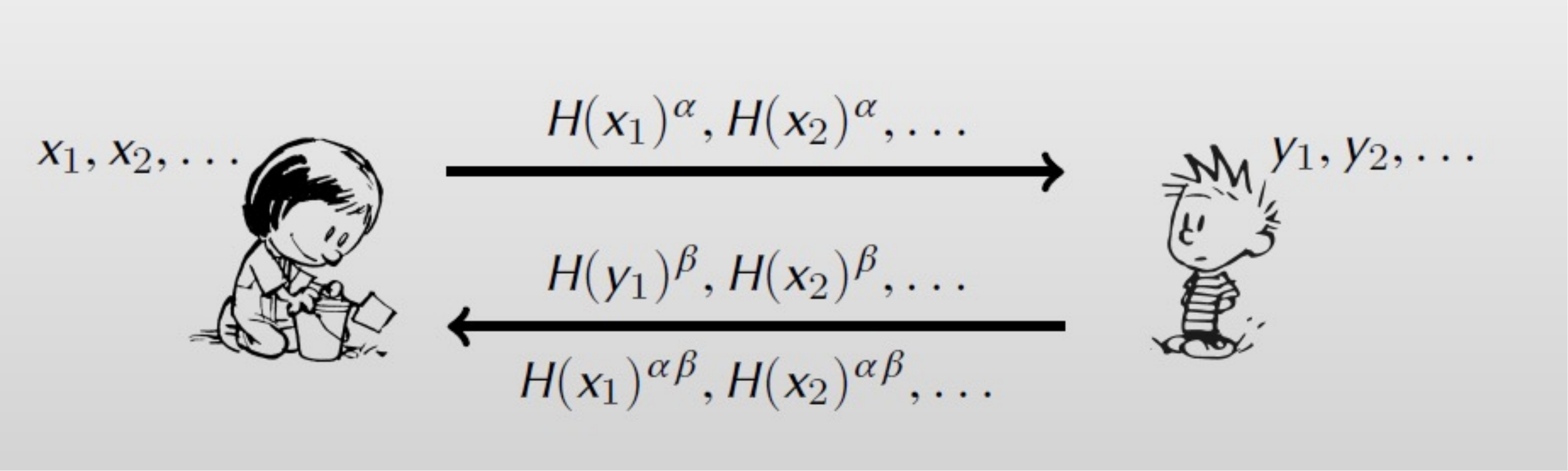
where H is a hash function with image of a group $G =< g >$

Idea:

- If $x = y, H(x)^{\alpha\beta} = H(y)^{\alpha\beta}$
- If $x \neq y$, they are random

Classical protocol: Diffie-Hellman

Drawback: *O*(*n*) **expensive** exponentiations



where H is a hash function with image of a group $G =< g >$

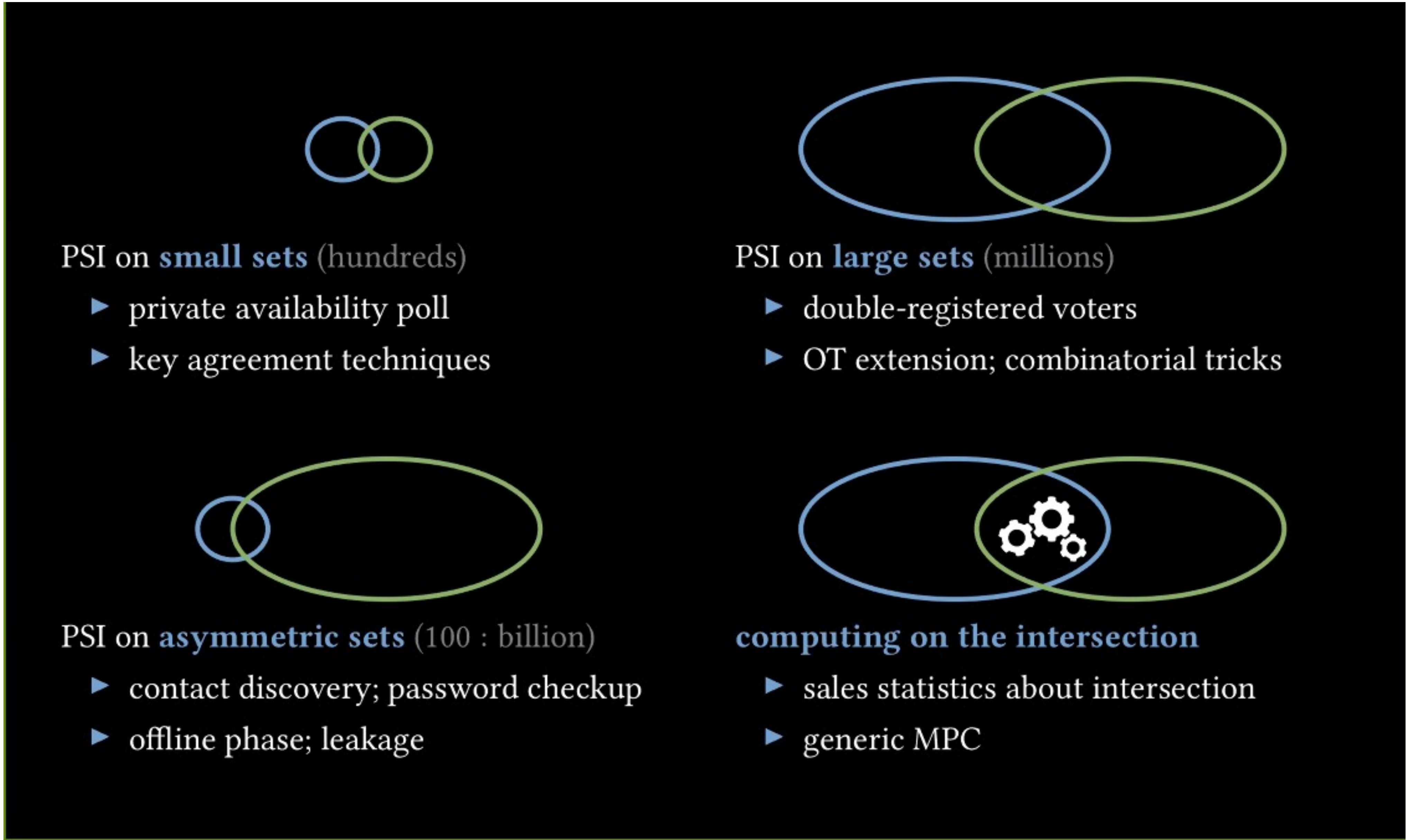
Idea:

- If $x = y, H(x)^{\alpha\beta} = H(y)^{\alpha\beta}$
- If $x \neq y$, they are random

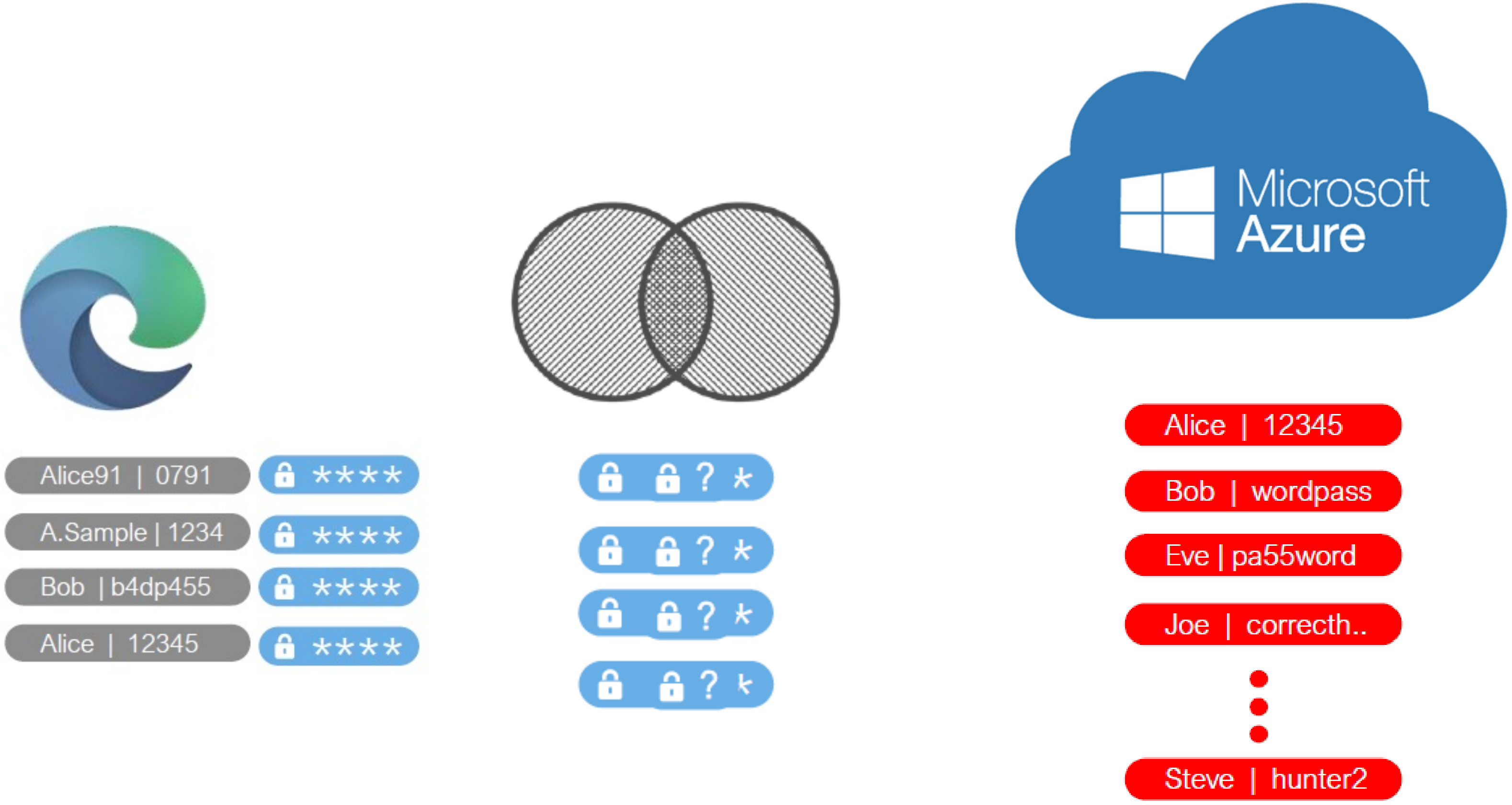
There are other solutions with trade-offs using

- Yao’s protocol
- OT
- Etc.

PSI


PSI on small sets (hundreds)
private availability poll
key agreement techniques
PSI on large sets (millions)
double-registered voters
OT extension; combinatorial tricks
PSI on asymmetric sets (100 : billion)
contact discovery; password checkup
offline phase; leakage
computing on the intersection
sales statistics about intersection
generic MPC

PSI: intersection of leaked password



Summary

1 **Secure computation**: Concepts & definitions

2 **General constructions**: Yao’s protocol, and GMW

3 **Custom protocol**: private set intersection

Depending on the definition of “Function F”, MPC could be very powerful

Materials

- David Evans, Vladimir Kolesnikov and Mike Rosulek, A Pragmatic Introduction to Secure Multi-Party Computation
- Dan Boneh and Victor Shoup, A Graduate Course in Applied Cryptography, Section 23

Lecture 9: Privacy-Enhancing technologies 3: MPC


Diffie


Rivest


Rivest


Yao


Goldwasser


Hellman


Shamir


Adelman


Adelman


Dertouzos


Micali


Rackoff

1976	1977	1978	1982	1985
New directions	RSA	Homomorphic Enc	MPC	Zero Knowledge

Thank you