MPC Blitz

- A brief introduction to basic MPC protocols

Table of content

Intro

- What is MPC?

  Secure Multi-Party Computation

- What does it do?

  Narrowly speaking, MPC enables multiple parties **jointly** compute the result of a **function without revealing** their respective inputs.

  e.g., n Parties each holding secret input x_i ($i \in \{1,2,\cdots,n\}$) wants to jointly compute the value of $f(x_1, x_2, \cdots, x_n)$ without revealing their respective x_i

- Why do we need it?

  Application scenarios: Machine learning, medical record analysis, voting or bidding system, etc.

Oblivious Transfer

Michael O. Rabin, 1981

Before everything starts…

- Oblivious transfer (OT), 1981

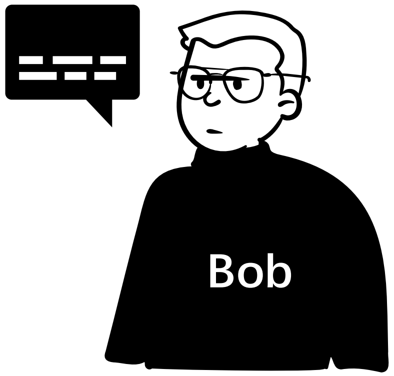
Allows a sender to transfer **one out of potentially many message** to receiver in such a way, that the **sender does not learn which specific piece** was received by receiver, and the **receiver does not learn which piece** was sent by sender.

Oblivious Transfer (OT)

- Imagine a scene: A patient Bob consults with doctor Alice.

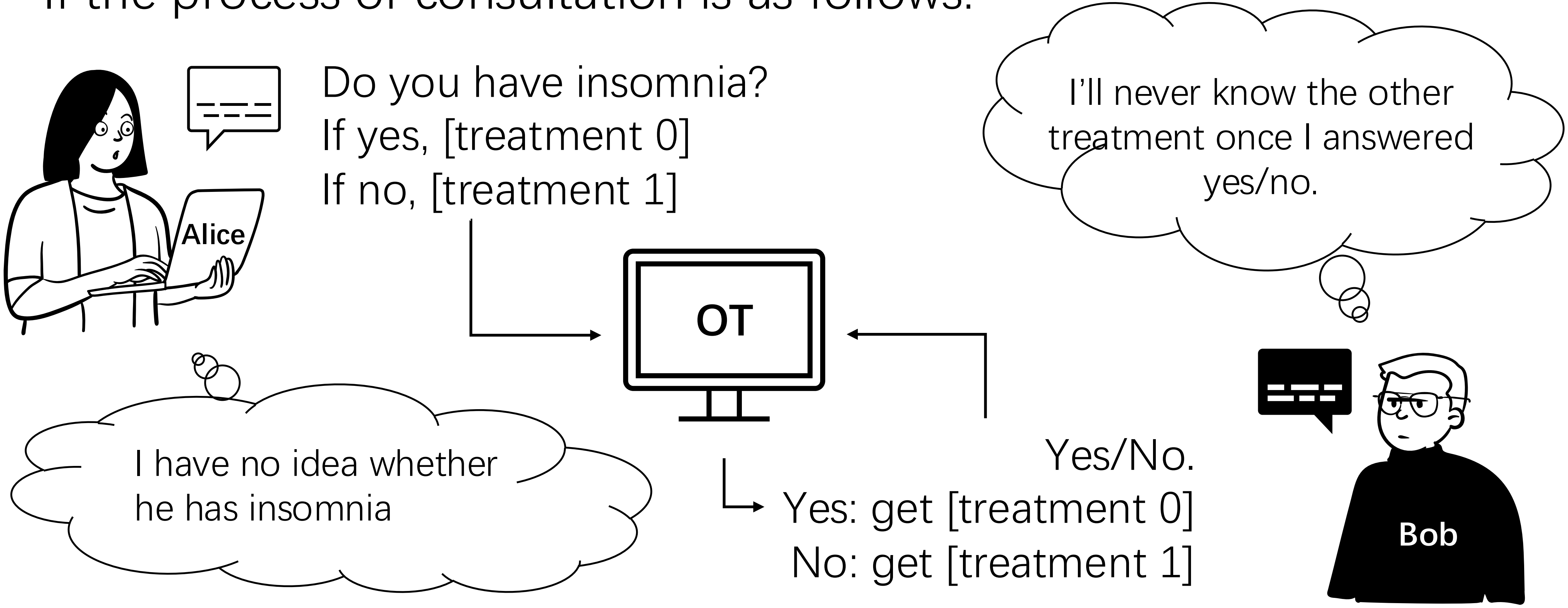
I have…
- A complete **therapy**
- **Different treatment** for different conditions
- The complete **therapy** is trade **secret**

I have…
- A **symptom**
- Ready for offering **details** of the condition
- Require **details not known** by Alice (for privacy)



Oblivious Transfer (OT)

- If the process of consultation is as follows:



Oblivious Transfer (OT)

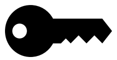
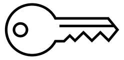
- How to achieve that?

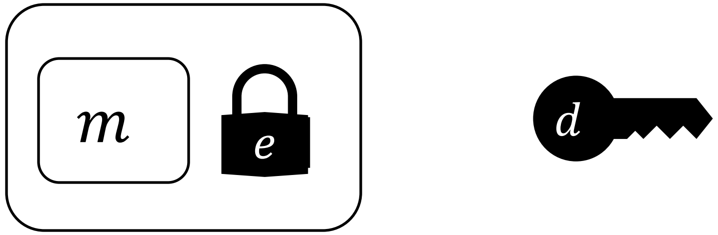
First let's recall public-key cryptography

A party has a public key (known to all) and private key (known to self). A message encrypted with his public key can only be decrypted with the corresponding private key.

Note that, the **process of decryption doesn't tell the correctness**. Decoder doesn't know the success or failure of decryption if he doesn't know whether he is using the right key, and the original message is meaningless (like random string representing another key).

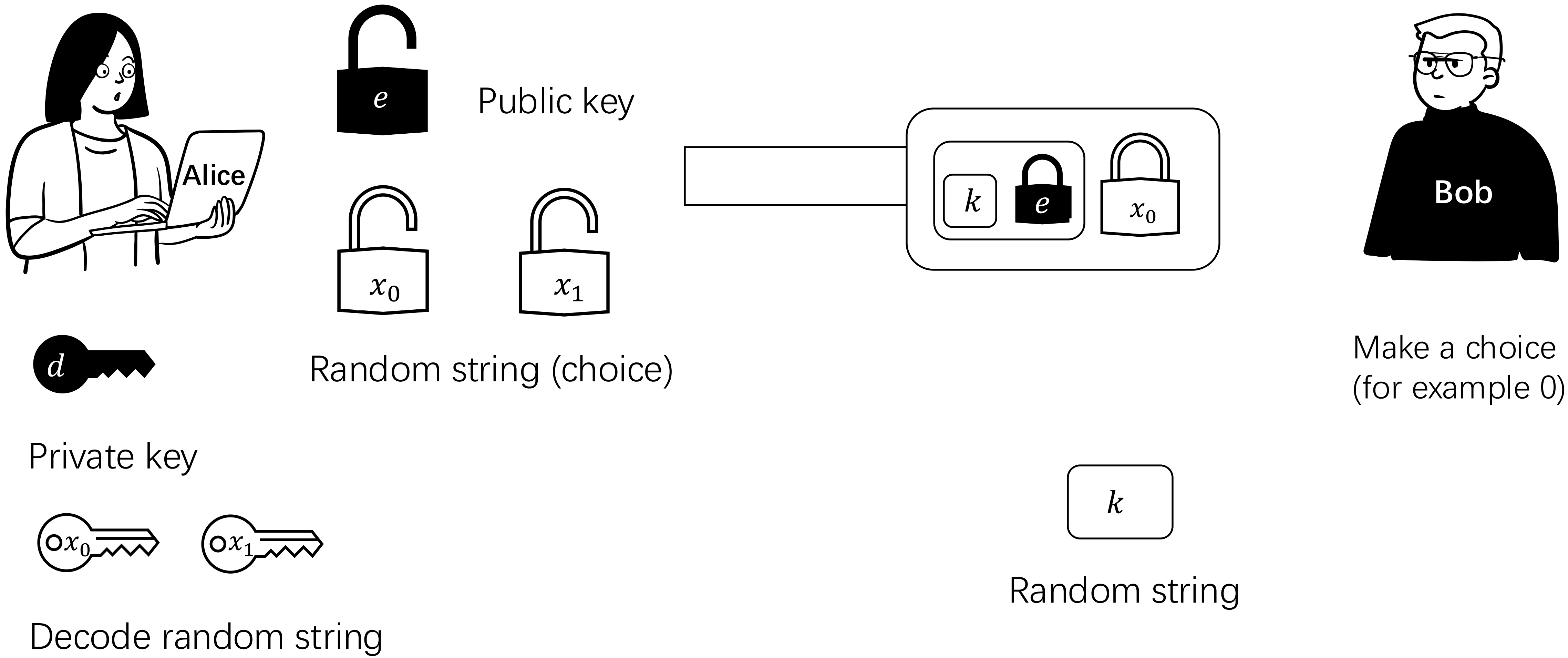
Oblivious Transfer (OT)

- Let  represent public key cryptography, corresponding key 
  Let  represent mask (to mask is to add), and  represent unmask (to unmask is to minus)
- For example, let

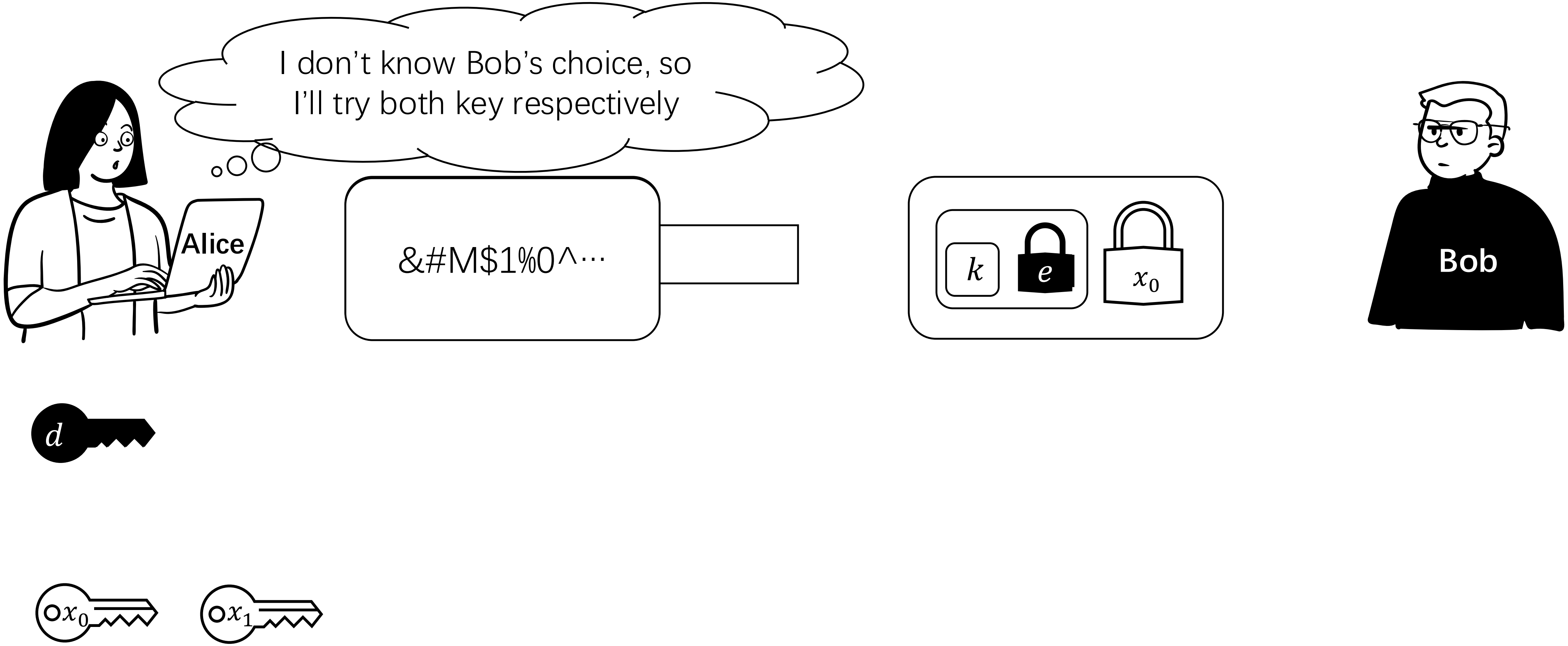


denotes message m encrypted by public key e and can be decrypted using private key d.

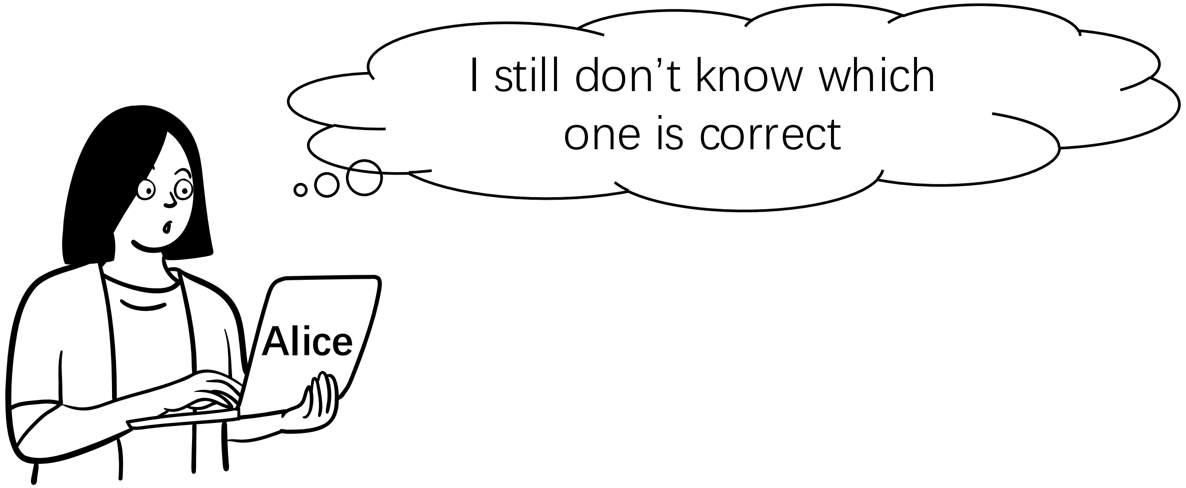
Oblivious Transfer (OT)

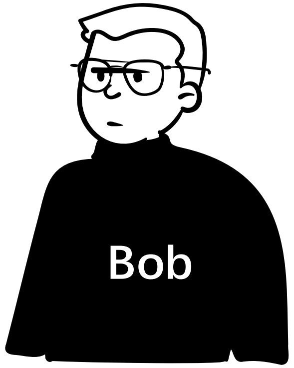


Oblivious Transfer (OT)



Oblivious Transfer (OT)





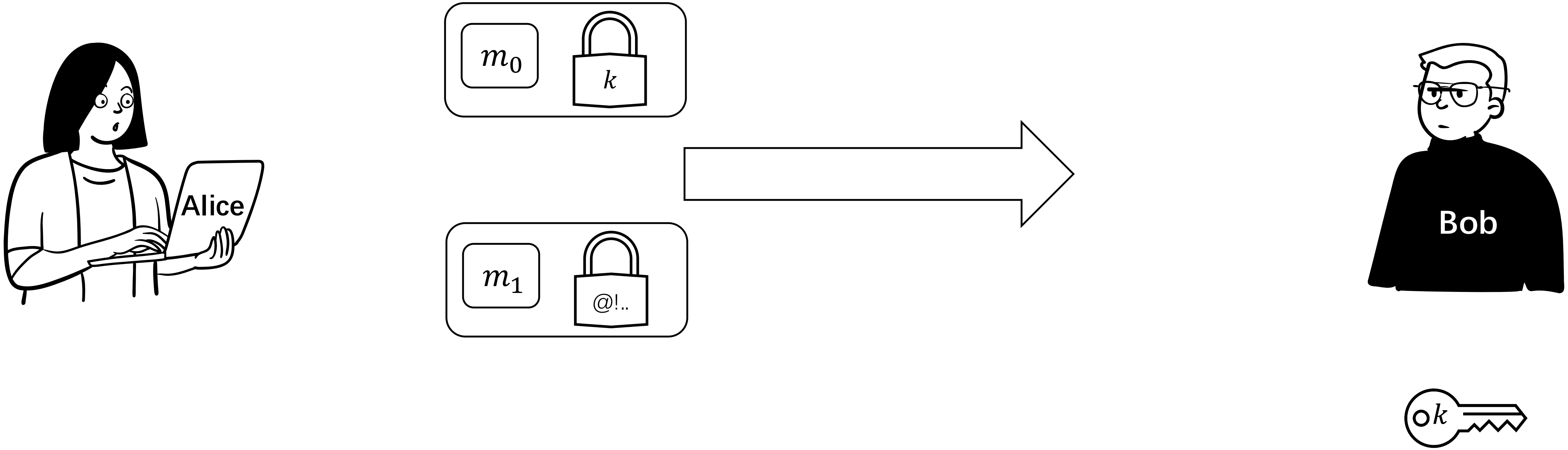
m_0 m_1

Treatment

k

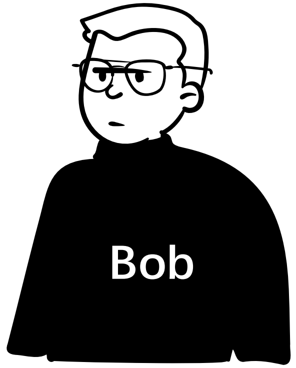
$A*%
#@

Oblivious Transfer (OT)



Oblivious Transfer (OT)

m_0

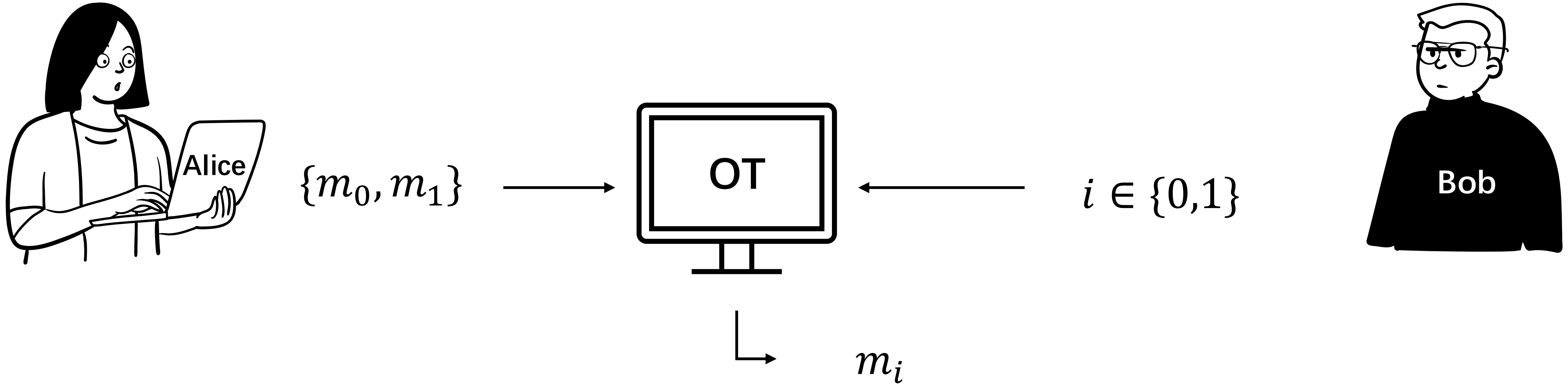


Oblivious Transfer (OT)

Problems

- How does Bob tell which one he decrypt is correct?
- Is it semi-honest or malicious?

Oblivious Transfer (OT)



OT extensions

- Correlated-OT and random-OT
- IKNP
- Standard OT
- …

Garbled Circuit

Andrew Chi-Chih Yao, 1986

Garbled Circuit (GC)

Allows 2 mistrusting parties **jointly compute a function** over their private inputs **without** a trusted third party.

For example, Alice holding x_0 and Bob holding x_1 want to jointly compute $f(x_0, x_1)$, with Alice know nothing about x_1, and Bob know nothing about x_0.

Garbled Circuit (GC)

- Origin: the millionaire problem, 1986

Alice and Bob want to figure out who has more money. How can they figure this out without revealing their bank statements?

Alice holding x_0 and Bob holding x_1 wants to compute

$$f(x_0, x_1) = \begin{cases} 0, x_0 \leq x_1 \\ 1, x_0 > x_1 \end{cases}$$

without revealing their own inputs.

Garbled Circuit (GC)

- Why ‘Circuit’?

  jointly compute -> communication

  secret input -> encryption

  computation over secret input -> **homomorphic encryption**

$$x_0 \to c_0 = Enc(x_0)$$
$$x_1 \to c_1 = Enc(x_1)$$
$$Dec(c_0 + c_1) = x_0 + x_1$$

  But! There’s no such technique then…

Garbled Circuit (GC)

- Why **Circuit**?

So, to directly encrypt private input is not possible.

But what else can be encrypt?

The process of computing

Computation on computer are performed over gates. So if we can encrypt gates, theoretically we can perform security computation.

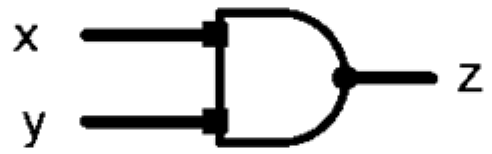
Garbled Circuit (GC)

- How to encrypt a gate?

Assume our circuit only consist of 1 gate. Alice holding one-bit x and Bob holding one-bit y wants to know the AND of their inputs:

$$z = f(x, y) = x \wedge y$$

Which can be represented by the circuit (or gate) below:



Garbled Circuit (GC)

This is the truth table of the gate:

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

To encrypt the gate is to encrypt the truth table. Our aim is to make the logic function, input, and output of the gate unclear.

Garbled Circuit (GC)

Truth table of encrypted AND gate:

x	y	z
$k_{x,0}$	$k_{y,0}$	$Enc(k_{x,0}, \left(Enc\left(k_{y,0}, k_{z,0}\right)\right))$
$k_{x,0}$	$k_{y,1}$	$Enc(k_{x,0}, \left(Enc\left(k_{y,1}, k_{z,0}\right)\right))$
$k_{x,1}$	$k_{y,0}$	$Enc(k_{x,1}, \left(Enc\left(k_{y,0}, k_{z,0}\right)\right))$
$k_{x,1}$	$k_{y,1}$	$Enc(k_{x,1}, \left(Enc\left(k_{y,1}, k_{z,1}\right)\right))$

Possible output

Encrypted output

Let the k be random strings representing inputs 0 or 1

Garbled Circuit (GC)

If only one party know the truth table, for the other party, the evaluating of the circuit is encrypted.

So, if we let one party craft the circuit (define the truth table), let the other party evaluate it (calculate the output of gates), the process of evaluating is **garbled** for the evaluator. (Because he has no idea what is the function of the gate, nor other possible outputs.)

x	y	z
$k_{x,0}$	$k_{y,0}$	$Enc(k_{x,0}, (Enc(k_{y,0}, k_{z,0})))$
$k_{x,0}$	$k_{y,1}$	$Enc(k_{x,0}, (Enc(k_{y,1}, k_{z,0})))$
$k_{x,1}$	$k_{y,0}$	$Enc(k_{x,1}, (Enc(k_{y,0}, k_{z,0})))$
$k_{x,1}$	$k_{y,1}$	$Enc(k_{x,1}, (Enc(k_{y,1}, k_{z,1})))$

Garbled Circuit (GC)

- What Alice know

Inputs/outputs	keys
$x = 0$	$k_{x,0}$
$x = 1$	$k_{x,1}$
$y = 0$	$k_{y,0}$
$y = 1$	$k_{y,1}$
$z = 0$	$k_{z,0}$
$z = 1$	$k_{z,1}$

x	y	z
$k_{x,0}$	$k_{y,0}$	$Enc(k_{x,0}, \left(Enc\left(k_{y,0}, k_{z,0}\right)\right))$
$k_{x,0}$	$k_{y,1}$	$Enc(k_{x,0}, \left(Enc\left(k_{y,1}, k_{z,0}\right)\right))$
$k_{x,1}$	$k_{y,0}$	$Enc(k_{x,1}, \left(Enc\left(k_{y,0}, k_{z,0}\right)\right))$
$k_{x,1}$	$k_{y,1}$	$Enc(k_{x,1}, \left(Enc\left(k_{y,1}, k_{z,1}\right)\right))$

Garbled Circuit (GC)

- What Bob know

  Alice's input $k_{x,i}$ $(i \in \{0,1\})$, for example $k_{x,0}$

  Bob's input $k_{y,i}$ $(i \in \{0,1\})$, for example $k_{y,1}$

  But not the corresponding of $\boldsymbol{i}$ and $\boldsymbol{k_{x,i}}$ or $\boldsymbol{k_{y,i}}$

  All encrypted outputs:

$$z_0 = Enc(k_{x,0}, \left(Enc\left(k_{y,0}, k_{z,0}\right)\right))$$
$$z_1 = Enc(k_{x,0}, \left(Enc\left(k_{y,1}, k_{z,0}\right)\right))$$
$$z_2 = Enc(k_{x,1}, \left(Enc\left(k_{y,0}, k_{z,0}\right)\right))$$
$$z_3 = Enc(k_{x,1}, \left(Enc\left(k_{y,1}, k_{z,1}\right)\right))$$

Garbled Circuit (GC)

- How Bob evaluate

$$Dec\left(k_{y,1}, Dec(k_{x,0}, z_0)\right) = FAIL$$
$$Dec\left(k_{y,1}, Dec(k_{x,0}, z_1)\right) = k_{z,0}$$
$$Dec\left(k_{y,1}, Dec(k_{x,0}, z_2)\right) = FAIL$$
$$Dec\left(k_{y,1}, Dec(k_{x,0}, z_3)\right) = FAIL$$

Without the other keys, Bob has no way of knowing which truth table is used and gets no information about which values the input keys represent.

Garbled Circuit (GC)

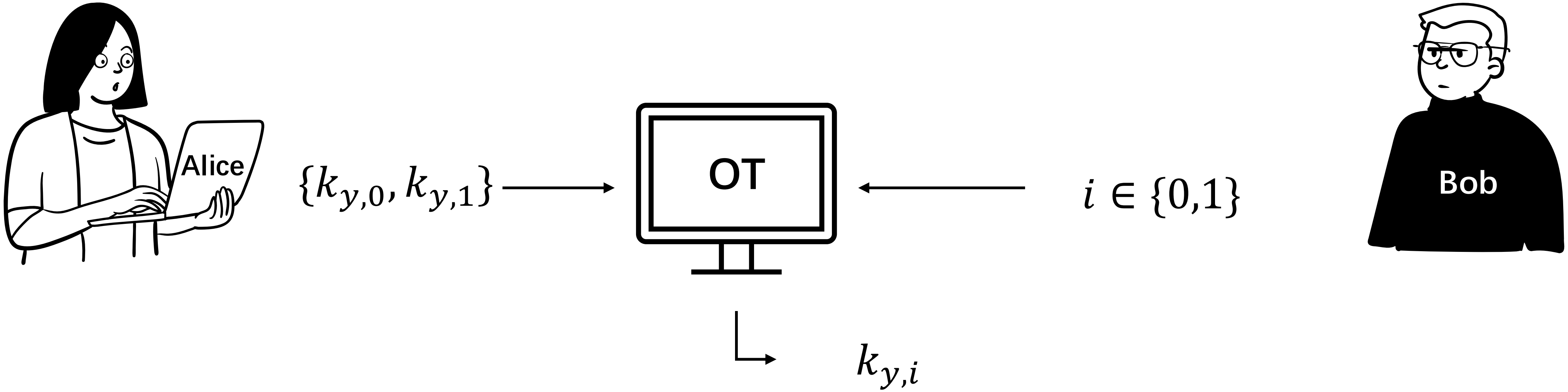
- Problem: How does Bob know the input keys?

Alice can directly send the keys corresponding to her own inputs to Bob, because Bob doesn't know the corresponding relation. But what about Bob's input?

Is there any technique that enables Bob to get the key corresponding to his input from Alice, but reveal nothing about his input?

OT

Garbled Circuit (GC)

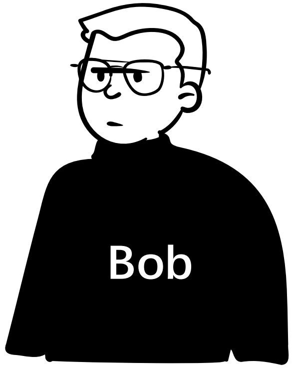


All the building blocks are ready, time to construct the protocol

Garbled Circuit (GC)

1. Alice craft a **circuit**, and encrypt it (keep the truth table and corresponding relation)

2. Alice send her **inputs (keys)** and the **circuit** to Bob (circuit includes the **encrypted outputs**)



Truth table
Corresponding table
Final output
result

3. Bob get **his input keys** from Alice through **OT**

4. Bob **evaluate** the circuit, and send the **final output** to Alice

Garbled circuit
Alice’s input keys
Bob’s input keys
Final output
result

5. Alice **reveal** the result

Still confused?

Garbled Circuit (GC)

1. Alice craft a **circuit**, and encrypt it (keep the truth table and corresponding relation)

Inputs/outputs	keys
$x = 0$	1001
$x = 1$	1010
$y = 0$	0001
$y = 1$	1101
$z = 0$	1110
$z = 1$	1111

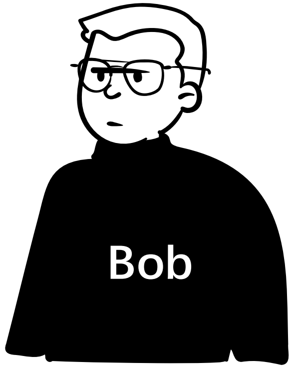
Let $Enc(a, b) = a \oplus b$

x	y	z
$k_{x,0}$ 1001	$k_{y,0}$ 0001	$Enc\left(k_{x,0}, Enc\left(k_{y,0}, k_{z,0}\right)\right)$ $1001 \oplus 0001 \oplus 1110 = 0110$
$k_{x,0}$ 1001	$k_{y,1}$ 1101	$Enc\left(k_{x,0}, Enc\left(k_{y,1}, k_{z,0}\right)\right)$ $1001 \oplus 1101 \oplus 1110 = 1010$
$k_{x,1}$ 1010	$k_{y,0}$ 0001	$Enc\left(k_{x,1}, Enc\left(k_{y,0}, k_{z,0}\right)\right)$ $1010 \oplus 0001 \oplus 1110 = 0101$
$k_{x,1}$ 1010	$k_{y,1}$ 1101	$Enc\left(k_{x,1}, Enc\left(k_{y,1}, k_{z,1}\right)\right)$ $1010 \oplus 1101 \oplus 1111 = 1000$

Garbled Circuit (GC)

2. Alice send her **inputs (keys)** and the **circuit** to Bob (circuit includes the **encrypted outputs**)

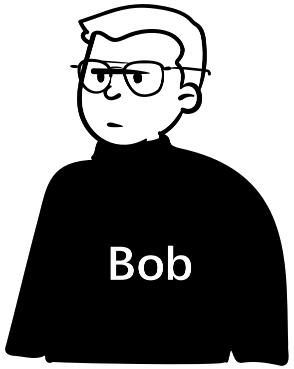
Alice input:
1001
Circuit:
$Gate(\{0110,1010,0101,1000\})$



Inputs/outputs	keys
$x = 0$	1001
$x = 1$	1010
$y = 0$	0001
$y = 1$	1101
$z = 0$	1110
$z = 1$	1111

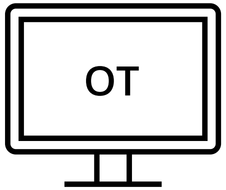
Garbled Circuit (GC)

3. Bob get **his input keys** from Alice through **OT**



$k_{y,0} = 0001$

$k_{y,1} = 1101$



$i = 1$

$k = 1101$

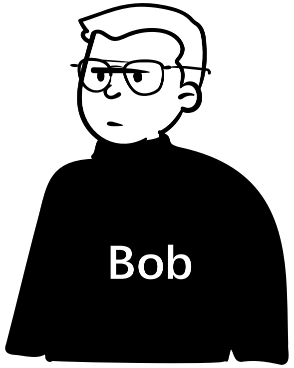
Alice input:
1001

Circuit:
$Gate(\{0110, 1010, 0101, 1000\})$

Inputs/outputs	keys
$x = 0$	1001
$x = 1$	1010
$y = 0$	0001
$y = 1$	1101
$z = 0$	1110
$z = 1$	1111

Garbled Circuit (GC)

4. Bob **evaluate** the circuit, and send the **final output** to Alice



$$1001 \oplus 1101 \oplus 0110 = 0010 \quad (FAIL)$$
$$1001 \oplus 1101 \oplus 1010 = 1110$$
$$1001 \oplus 1101 \oplus 0101 = 0001 \quad (FAIL)$$
$$1001 \oplus 1101 \oplus 1000 = 1100 \quad (FAIL)$$

Inputs/outputs	keys
$x = 0$	1001
$x = 1$	1010
$y = 0$	0001
$y = 1$	1101
$z = 0$	1110
$z = 1$	1111

Question: How does Bob know whether he decrypts correctly?

1. Alice add pre-negotiated info in the possible outputs, for example a string of 0
2. **point-and-permute**: the last n-bit serve as a pointer to the permuted table, indicating which row to be decrypted.

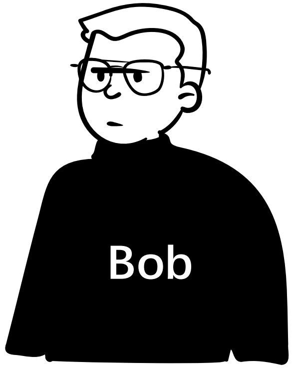
Alice input:
1001
Bob input:
1101
Circuit:
$Gate(\{0110, 1010, 0101, 1000\})$

Garbled Circuit (GC)

4. Bob **evaluate** the circuit, and send the **final output** to Alice

Final output:
1110

Final output:
1110



Inputs/outputs	keys
$x = 0$	1001
$x = 1$	1010
$y = 0$	0001
$y = 1$	1101
$z = 0$	1110
$z = 1$	1111

Alice input:
1001
Bob input:
1101
Circuit:
$Gate(\{0110,1010,0101,1000\})$
Final output:
1110

Garbled Circuit (GC)

5. Alice reveal the result

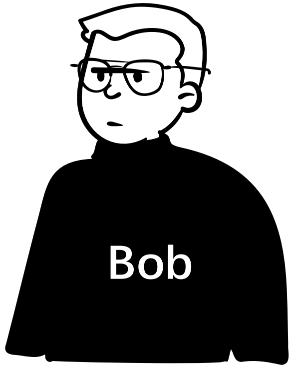
Result:

$z = 0$

Final output:

1110

Inputs/outputs	keys
$x = 0$	1001
$x = 1$	1010
$y = 0$	0001
$y = 1$	1101
$z = 0$	1110
$z = 1$	1111



Alice input:

1001

Bob input:

1101

Circuit:

$Gate(\{0110, 1010, 0101, 1000\})$

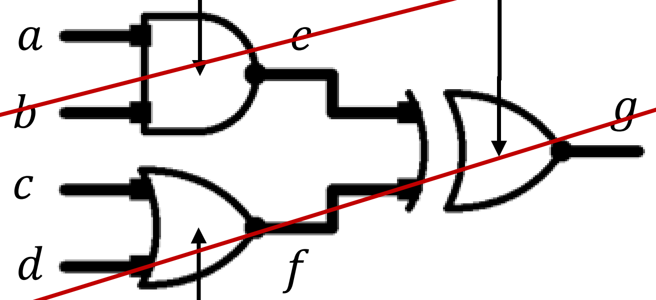
Final output:

1110

Garbled Circuit (GC)

- What about multiple gates?

i/o	keys
$in_a = 0$	a_0
$in_a = 1$	a_1
$in_b = 0$	b_0
$in_b = 1$	b_1
$out_c = 0$	c_0
$out_c = 1$	c_1



i/o	keys
$in_a = 0$	c_0
$in_a = 1$	c_1
$in_b = 0$	f_0
$in_b = 1$	f_1
$out_c = 0$	g_0
$out_c = 1$	g_1

i/o	keys
$x = 0$	d_0
$x = 1$	d_1
$y = 0$	e_0
$y = 1$	e_1
$z = 0$	f_0
$z = 1$	f_1

Garbled Circuit (GC)

- Optimizations and extensions

  Point-and-permute

  Free XOR

  Multi-party GC: BMR

  …

Garbled Circuit (GC)

If you are interested in the implementation of GC using programming language, here I got a demo in Python on my GitHub:

https://github.com/thewatertells/demoGC

Notes: I'm a noob coder and the TOY program only focus on IMPLEMENTATION but not efficiency and stability. The codes may seem dull, and definitely need optimization in many places.

Goldreich-Micali-Wigderson protocol

Oded Goldreich, Silvio Micali, Avi Wigderson, 1987

GMW

Similar with GC, GMW protocol allows 2 mistrusting parties **jointly evaluate a circuit** over their private inputs **without** a trusted third party.

Core concept: **additive secret sharing**

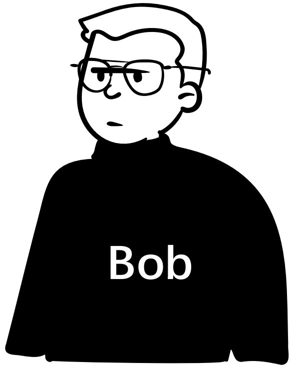
GMW

- What is additive secret sharing?

There is a secret x (private to a third party)
Let $x = x_0 \oplus x_1$
Then send x_0 to Alice, send x_1 to Bob

Each party doesn't know the secret, but they can **reconstruct** the secret by adding their shares

Additive sharing holds for **both single bits and bit strings**



x_0

x_1

GMW

I have a secret bit $x \in \{0,1\}$

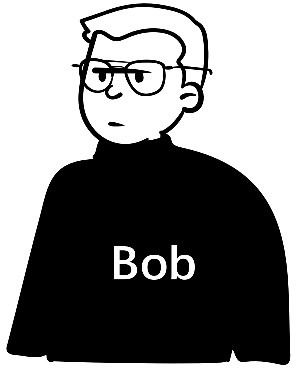
Let $x = x_0 \oplus x_1$

I keep x_0,
then send x_1 to Bob

I have a secret bit $y \in \{0,1\}$

Let $y = y_0 \oplus y_1$

I keep y_1,
then send y_0 to Alice



x_0

y_0

Alice won't know y without Bob,
Bob won't know x without Alice

y_1

x_1

GMW

- Secure computation.

Consider a functionally complete set $\{\wedge\ (AND), \neg(NOT), \oplus\ (XOR)\}$

x_0

y_0

z_0

XOR: To compute

$$z = x \oplus y$$

Note that:

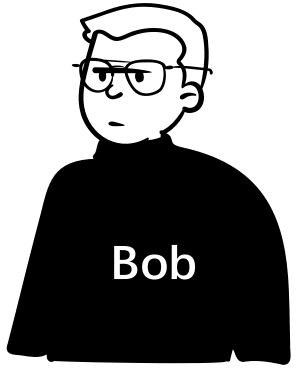
$$z = x \oplus y = x_0 \oplus x_1 \oplus y_0 \oplus y_1$$

If we share z as

$$z = z_0 \oplus z_1$$

Then

$$z = (x_0 \oplus y_0) \oplus (x_1 \oplus y_1)$$
$$z_0 = x_0 \oplus y_0$$
$$z_1 = x_1 \oplus y_1$$



y_1

x_1

z_1

GMW

- Secure computation.

Consider a functionally complete set $\{\wedge\ (AND), \neg(NOT), \oplus\ (XOR)\}$

x_0

y_0

z_0

NOT: To compute

$$z = \bar{x}$$

Note that:

$$z = \overline{x_0 \oplus x_1} = x_0 \oplus \overline{x_1} = \overline{x_0} \oplus x_1$$

If we share z as

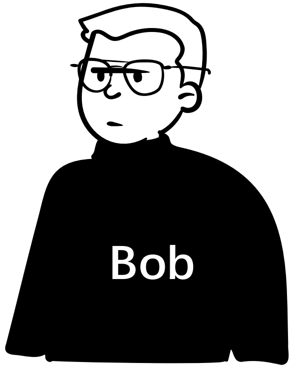
$$z = z_0 \oplus z_1$$

Then

$$z_0 = x_0, z_1 = \overline{x_1}$$

Or

$$z_0 = \overline{x_0}, z_1 = x_1$$



y_1

x_1

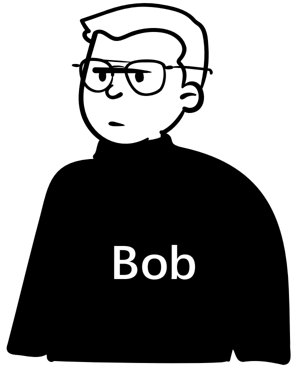
z_1

GMW

- Secure computation.

Consider a functionally complete set $\{\wedge\ (AND), \neg(NOT), \oplus\ (XOR)\}$

XOR and NOT gates can be evaluated **without any interaction**
Evaluating an AND gate requires interaction and uses
1-out-of-4 OT



x_0

y_0

y_1

x_1

GMW

- Secure computation.

Consider a functionally complete set $\{\wedge\,(AND), \neg(NOT), \oplus\,(XOR)\}$

AND: To compute

$$z = x \wedge y$$

Note that:

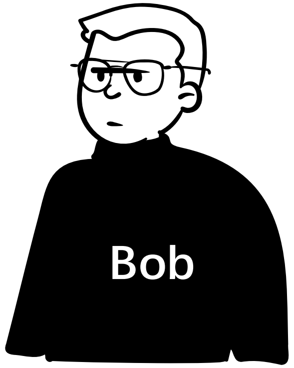
$$z = (x_0 \oplus x_1) \wedge (y_0 \oplus y_1)$$

Alice choose a random bit r as a mask,

Why mask?

and enumerate all possible x_1 and y_1

$$w_{00} = r \oplus (x_0 \oplus 0) \wedge (y_0 \oplus 0)$$
$$w_{01} = r \oplus (x_0 \oplus 0) \wedge (y_0 \oplus 1)$$
$$w_{10} = r \oplus (x_0 \oplus 1) \wedge (y_0 \oplus 0)$$
$$w_{11} = r \oplus (x_0 \oplus 1) \wedge (y_0 \oplus 1)$$



y_1

x_1

2-bit str: $x_1 y_1$

1-4 OT

$z_1 = r \oplus (x_0 \oplus x_1) \wedge (y_0 \oplus y_1)$

x_0

y_0

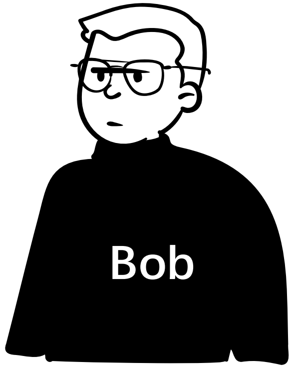
$z_0 = r$

GMW

- Secure computation.

Consider a functionally complete set $\{\wedge\ (AND), \neg(NOT), \oplus\ (XOR)\}$

After evaluating all gates, parties reveal to each other the shares of the final output to obtain the output of the entire computation.



GMW

- Generalization to more than 2 parties

  For XOR gates, the parties locally XOR their share.

  For NOT gates, one of the parties flip his share.

  For AND gates, consider the following equation:

  ($\sum$ here is the summation of XOR)

$$c = a \wedge b = (a_1 \oplus \cdots \oplus a_n) \wedge (b_1 \oplus \cdots \oplus b_n)$$

$$= \left(\sum_{i=1}^{n} a_i \wedge b_i \right) \oplus \left(\sum_{i \neq j} a_i \wedge b_j \right)$$

Computed locally ← (first term)

(second term) → 1-4 OT with every other party

Ben-Or Goldwasser Wigderson protocol

Michael Ben-Or, Shafi Goldwasser, Avi Wigderson, 1988

BGW

Although differ in many perspectives, GC and GMW both focus on **Boolean circuits**. The encryption or sharing is on **single bits**.

BGW protocol can be used to evaluate an **arithmetic circuit** (over a finite field), whose encryption and sharing is operated on **numbers**, consisting of addition, multiplication (by secrets and by constant numbers).

BGW

- Recall Shamir secret sharing:

A secret can be represented as the **constant term of a polynomial**, the values of the polynomial at different points can be considered as **shares** of the secret.

A **threshold** number of shares can be used reconstruct the polynomial and the secret through Lagrange interpolation.

BGW

- Recall Shamir secret sharing:
  - Choose secret and define the polynomial:
    Suppose a secret s shared among n parties with a threshold t.
    Generate $f(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1}$ on finite field $GF(q)$
  - Distribute shares:
    For each party i (public, $i \in \{1,2,\ldots,n\}$) calculate $\mathrm{s_i} = f(x_i)$ (private), send (i, s_i) as share.
  - Reconstruct secret:
    Choose t shares $\{(i_1, s_{i_1}), (i_2, s_{i_2}), \ldots, (i_t, s_{i_t})\}$ and calculate:

$$a_0 = s = (-1)^k \sum_{j=1}^{k} f(i_j) \prod_{l=1, l \neq j}^{k} \frac{i_l}{i_j - i_l} \; mod\; q$$

BGW

- Core concept of BGW:

Similar with GMW, BGW protocol enables parties to evaluate an arithmetic circuit using “Shared values”. Evaluation may involve calculating shared values locally (when doing addition), or communication with several parties (when doing multiplication).

BGW

- What does BGW do:

t parties $i \in (1,2,\ldots,t)$ each holding secret x^i. Now the parties want to jointly compute a polynomial

$$f(x^1, x^2, \ldots, x^t).$$

Superscript represents **secret**, subscripts represents **share**.

Each party i share his secret x^ias $\{x_1^i, x_2^i, \ldots, x_t^i\}$ using Shamir secret sharing, and distribute each x_j^i to party j.

So now, every party i is holding $\{x_i^1, x_i^2, \ldots, x_i^t\}$

BGW

Consider the polynomial

$$f(x^1, x^2, \dots, x^t)$$

There may be:

- addition of secrets
- multiplication of secrets
- multiplication of secrets and constant number.

BGW

- Addition

Assume we have 2 secrets x, y shared among t parties as:

$$x \xrightarrow{share} \{x_1, x_2, \dots x_t\}, \qquad y \xrightarrow{share} \{y_1, y_2, \dots, y_t\}$$

Now we want to secretly compute $z = x + y$, and share z

Then for each party, locally compute $z_i = x_i + y_i$, and all z_i reconstructs z.

$$z \xleftarrow{reconstruct} \{z_1, z_2, \dots, z_t\}$$

BGW

according to Shamir's method, 2 polynomials can be reconstructed as follow:

x is shared using polynomial:

$$f(u) = a_0 + a_1 u + a_2 u^2 + \cdots + a_{t-1} u^{t-1}$$

where $a_0 = x$. Each share $x_i = f(i)$.

y is shared using polynomial:

$$g(v) = b_0 + b_1 v + b_2 v^2 + \cdots + b_{t-1} u^{t-1}$$

where $b_0 = y$. Each share $y_i = g(i)$.

BGW

For each party i calculate

$$z_i = x_i + y_i = f(\alpha_i) + g(\alpha_i)$$

Then all z_i will reconstruct a function

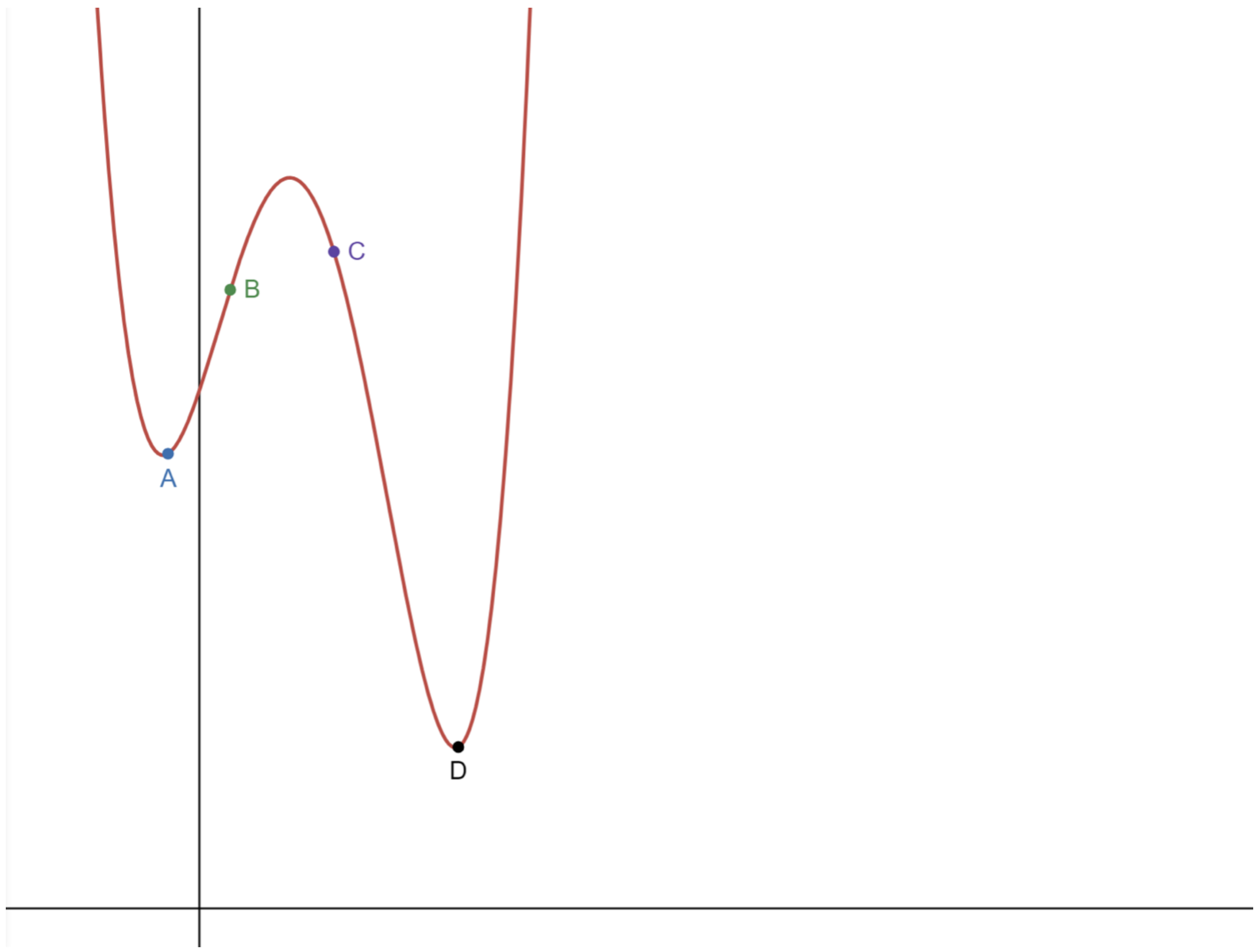
$$h(\cdot) = f(\cdot) + g(\cdot)$$

Which is:

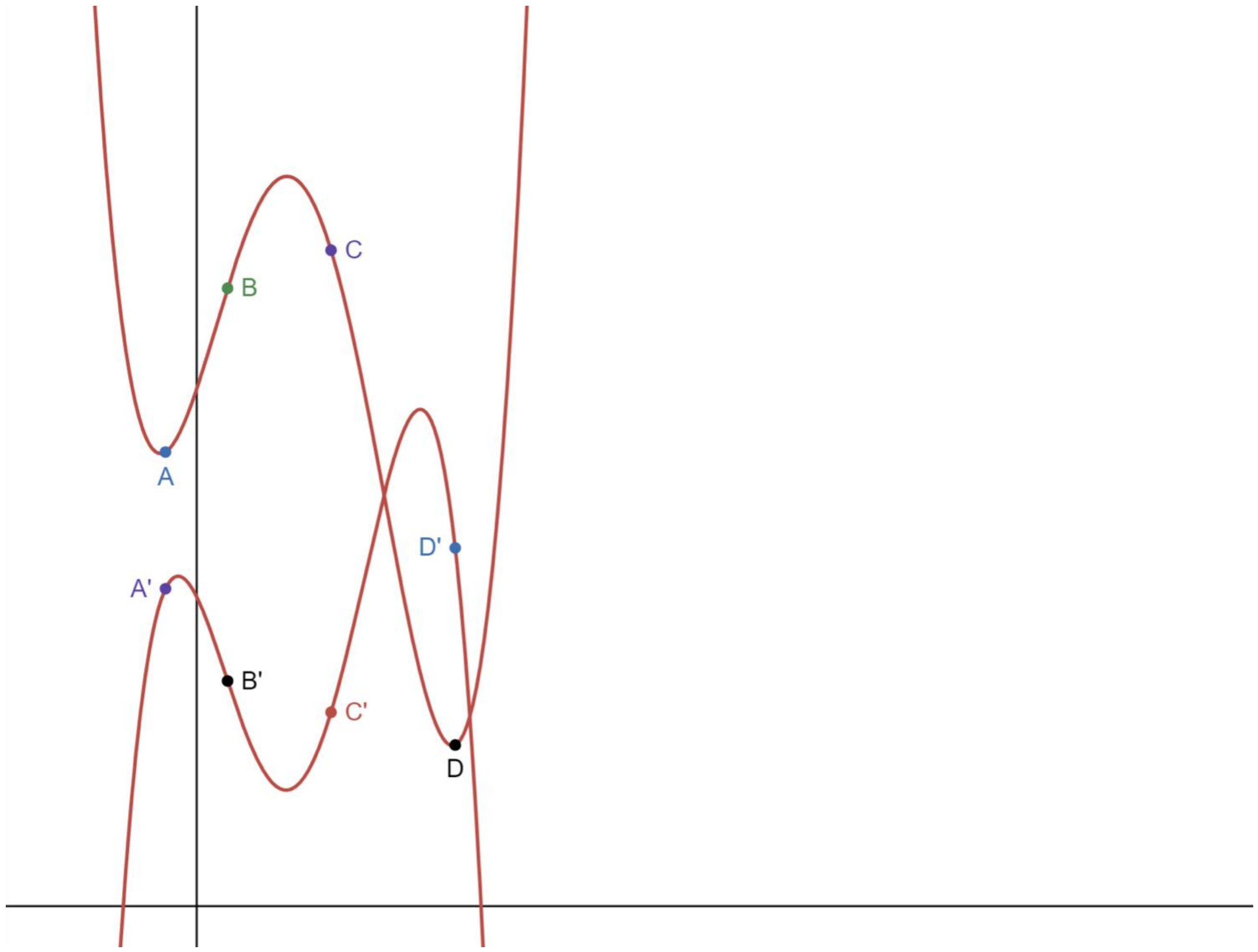
$$h(w) = (a_0 + b_0) + (a_1 + b_1)w + \cdots + (a_{t-1} + b_{t-1})w^{t-1}$$

Whose constant term $a_0 + b_0 = x + y = z$

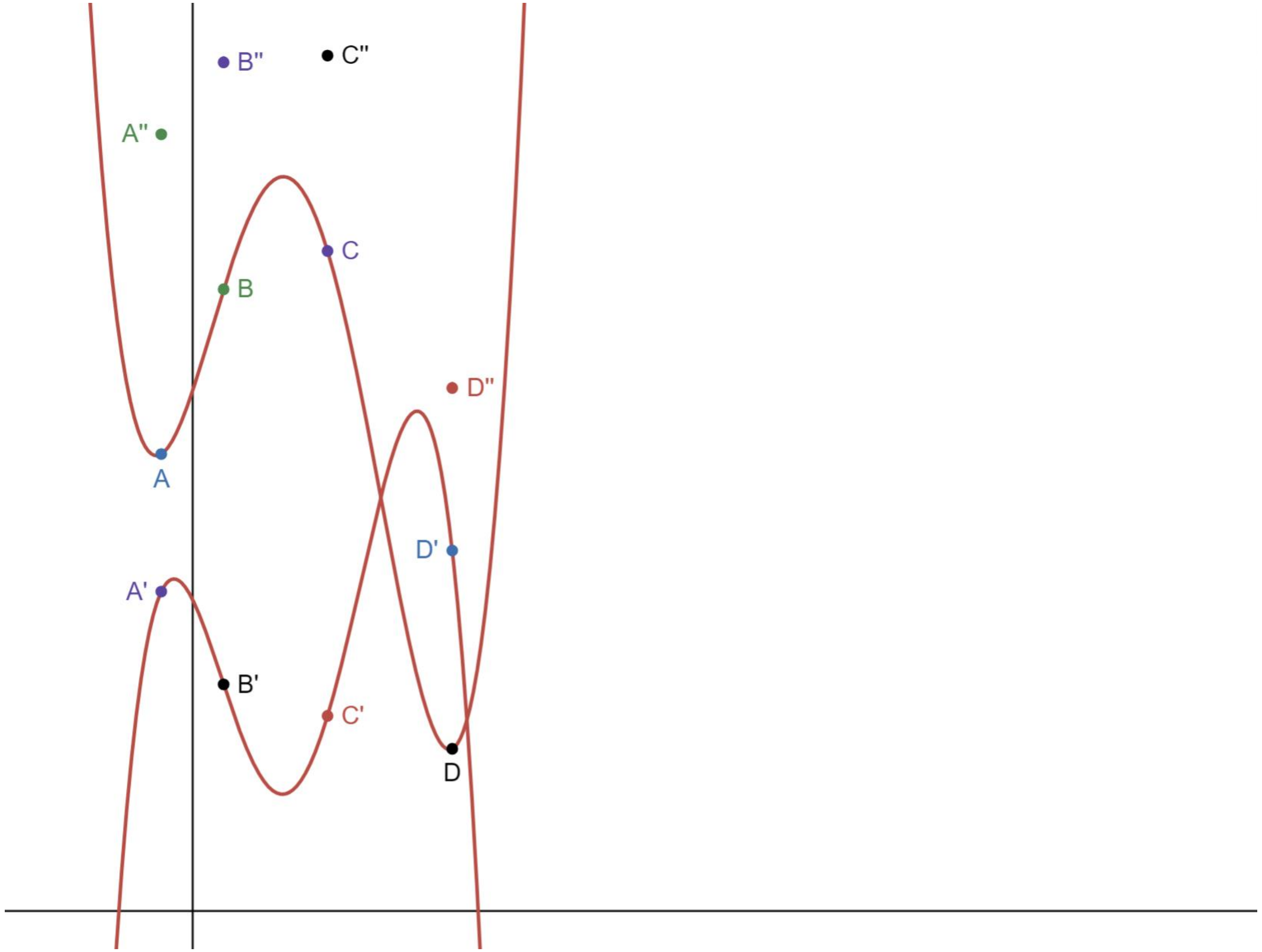
BGW



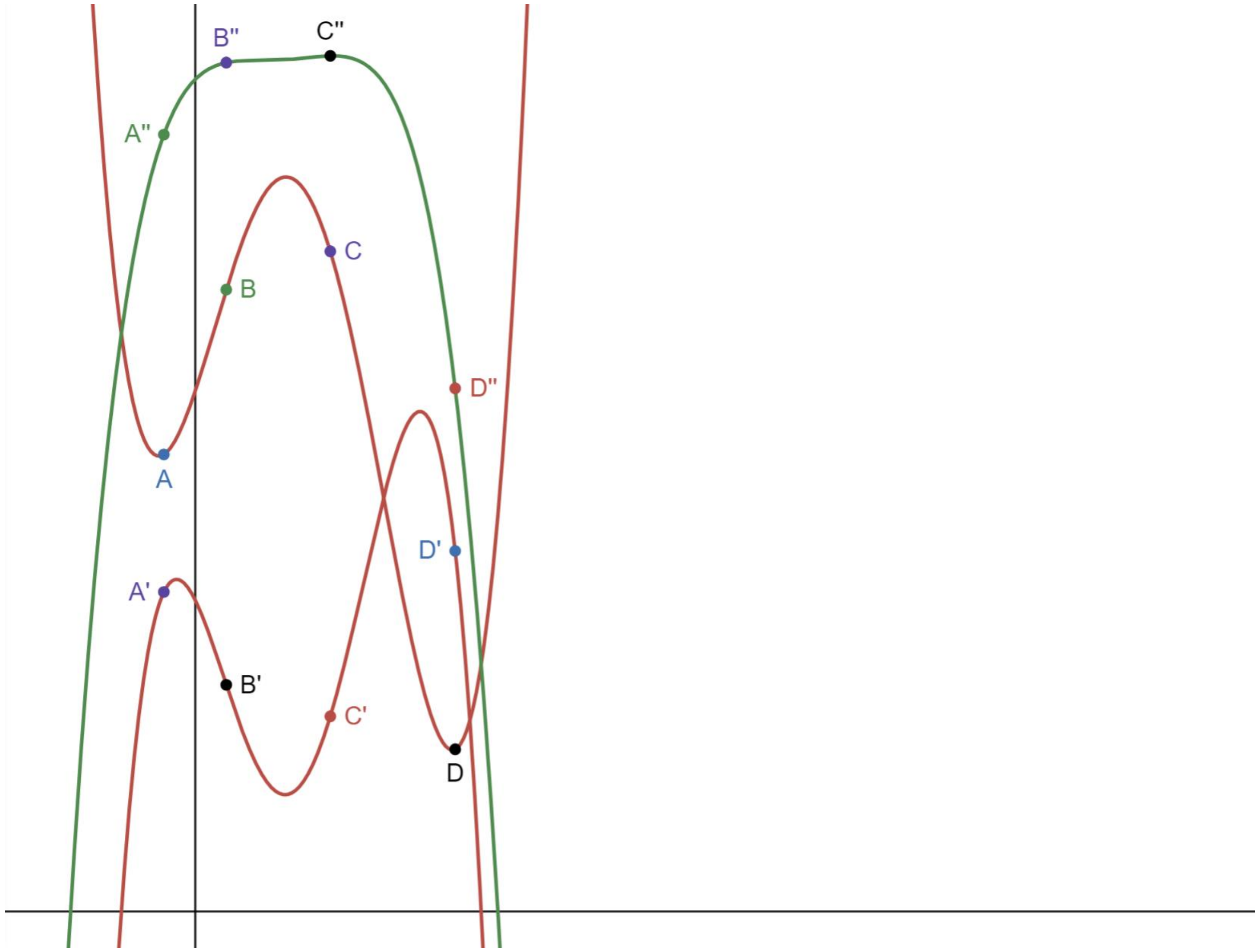
BGW



BGW



BGW



BGW

- Multiplication with constant number

Assume we have a secret x shared by t parties:

$$x \xrightarrow{share} \{x_1, x_2, \dots x_t\}$$

Now we want to secretly compute $z = c \cdot x$ and share z.

Then for each party i, locally compute $z_i = c \cdot x_i$, and all z_i reconstructs z.

$$z \xleftarrow{reconstruct} \{z_1, z_2, \dots, z_t\}$$

BGW

- Multiplication of secrets

  Assume we have 2 secrets x, y shared among n parties as:

$$x \xrightarrow{share} \{x_1, x_2, \dots x_n\}, \qquad y \xrightarrow{share} \{y_1, y_2, \dots, y_n\}$$

  Now we want to secretly compute $z = x \cdot y$, and share z

  If we use methods similar with secret addition, we may encounter problems as follow

BGW

- Problems:
  - If we simply let shares of product $z_i = x_i \cdot y_i$, the polynomial we're about to reconstruct is $h(\cdot) = f(\cdot)g(\cdot)$, whose constant term is $x \cdot y$, but of degree $2t - 1$.

$$h(u) = (a_0 + b_0) + \left(\sum_{i+j=1} a_i b_i\right) x + \left(\sum_{i+j=2} a_i b_i\right) x^2 + \cdots + \left(\sum_{i+j=2t-1} a_i b_i\right) x^{2t-1}$$

If number of parties $n > 2t - 1$ we do can reconstruct the polynomial using $2t - 1$ shares. But for consistency, we want the share of the product has the same threshold as multiplicated polynomials, which is t.

BGW

- Solution: degree reduction
  - Our intention is to **truncate** a polynomial of degree $2t-1$ to $t-1$ with the same constant term, and re-share the truncated polynomial.

NOTE

In this protocol, we only deal with cases when $n \geq 2t-1$. This protocol **cannot achieve secret multiplication when** $n < 2t-1$. It can only **truncate and reduce the degree** of the product polynomial corresponding to the secret multiplication when the number of parties is sufficient, making the threshold match the factor polynomials.

BGW

- Solution: degree reduction
  - Each party $i \in [1, n]$ holds $\left(\alpha_i, x_i = f(\alpha_i), y_i = g(\alpha_i)\right)$, α_i is public.
  - For every party i, let
    $$s_i = x_i \cdot y_i = h(\alpha_i) = f(\alpha_i)g(\alpha_i)$$
  - The original product polynomial is:
    $$h(u) = h_0 + h_1 u + h_2 u^2 + \cdots + h_{2t-1} u^{2t-1}$$
  - Define the truncation of $h(\cdot)$ to be:
    $$k(u) = h_0 + h_1 u + h_2 u^2 + \cdots + h_{t-1} u^{t-1}$$
  - Then $k(\cdot)$ is the polynomial we want to re-share and reconstruct. We want to distribute $r_i = k(\alpha_i)$ to every party i, so that all r_i reconstructs $k(\cdot)$.

BGW

- Solution: degree reduction
  - Luckily, there's relationship between r_i and s_i:

  Let $S = (s_0, s_1, \dots, s_n)$, and $R = (r_0, r_1, \dots, r_n)$, there is a $n \times n$ matrix A that:

  $$R = S \cdot A$$

  Our goal is to secretly find A.

BGW

- Solution: degree reduction
  - Let H be a $1 \times n$ vector consists of coefficients of $h(\cdot)$
    $$H = (h_0, h_1, \dots, h_{t-1}, \dots, h_{2t-1}, 0, \dots, 0)$$
  - Let K be a $1 \times n$ vector consists of coefficients of $k(\cdot)$
    $$K = (h_0, h_1, \dots h_{t-1}, 0, \dots 0)$$
  - Let B be an $n \times n$ Vandermonde matrix, where $b_{ij} = \alpha_j^i$,
    P be a $n \times n$ linear projection that $P(x_1, \dots, x_n) = (x_1, \dots, x_{t-1}, 0, \dots, 0)$

$$B = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \alpha_3 & \cdots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \alpha_3^2 & \cdots & \alpha_n^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{n-1} & \alpha_2^{n-1} & \alpha_3^{n-1} & \cdots & \alpha_n^{n-1} \end{bmatrix}, \qquad P = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{bmatrix}$$

BGW

- Solution: degree reduction
  - Now we have:

$$HB = S$$

$$[h_0 \ h_1 \cdots \ h_{2t-1} \ 0 \cdots 0] \cdot \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \alpha_3 & \cdots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \alpha_3^2 & \cdots & \alpha_n^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{n-1} & \alpha_2^{n-1} & \alpha_3^{n-1} & \cdots & \alpha_n^{n-1} \end{bmatrix} = [s_1 \ s_2 \cdots s_n]$$

$h(\alpha_1)$

BGW

- Solution: degree reduction
  - Now we have:

$$HP = K$$

$$[h_0 \;\; h_1 \cdots \;\; h_{2t-1} \;\; 0 \cdots 0] \cdot \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{bmatrix} = [h_0 \;\; h_1 \cdots h_{t-1} \;\; 0 \cdots 0]$$

BGW

- Solution: degree reduction
  - Now we have:

$$KB = R$$

$$[h_0 \ h_1 \cdots h_{t-1} \ \ 0 \cdots 0] \cdot \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \alpha_3 & \cdots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \alpha_3^2 & \cdots & \alpha_n^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{n-1} & \alpha_2^{n-1} & \alpha_3^{n-1} & \cdots & \alpha_n^{n-1} \end{bmatrix} = [r_1 \ r_2 \cdots r_{t-1} \ \ 0 \cdots 0]$$

$k(\alpha_1)$

BGW

- Solution: degree reduction
  - Now we have:

$$HB = S$$
$$HP = K$$
$$KB = R$$

So we'll get:

$$R = S(B^{-1}PB)$$

The $B^{-1}PB$ is the A we want to find, i.e.

$$R = SA$$

because matrix B and P does not involve secrets, A can be computed locally by each party.

BGW

- We successfully find a projection from s_i to r_i that makes only t secrets is needed for reconstruction of secret $z = x \cdot y$

$$x \xrightarrow{share} \{x_1, x_2, \dots, x_t\}, \qquad y \xrightarrow{share} \{y_1, y_2, \dots, y_t\}$$

$$s_i = x_i \cdot y_i$$

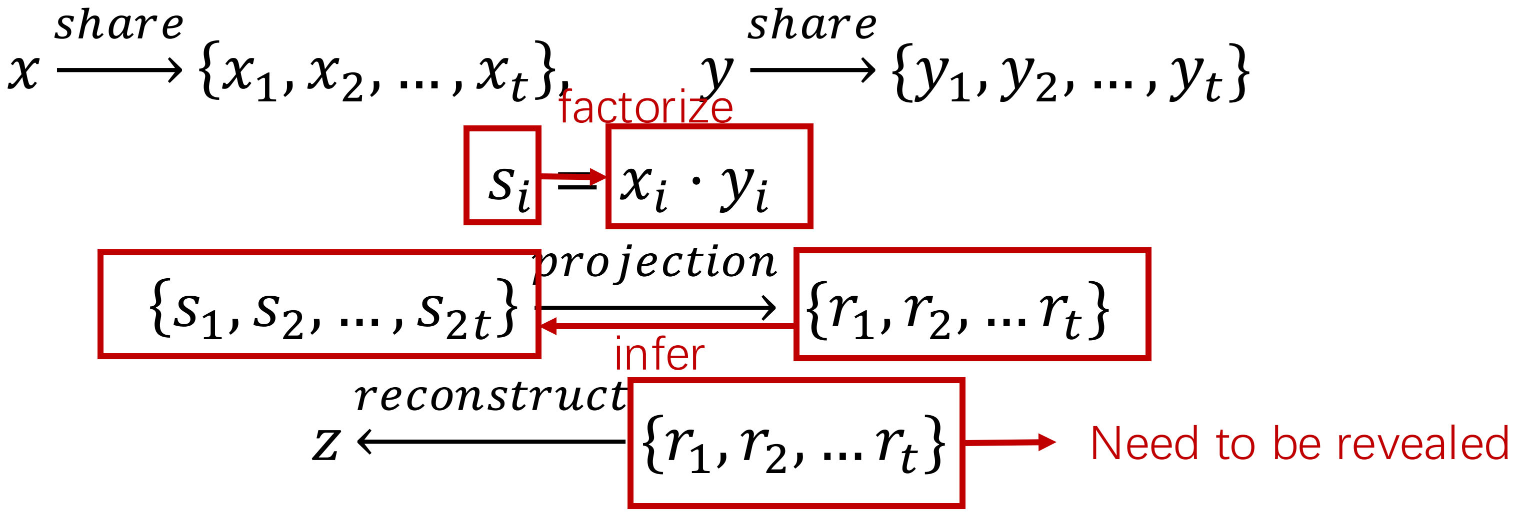
$$z \xleftarrow{reconstruct} \{s_1, s_2, \dots, s_{2t}\}, \qquad z \nleftarrow \{s_1, s_2, \dots, s_t\}$$

$$\{s_1, s_2, \dots, s_{2t}\} \xrightarrow{projection} \{r_1, r_2, \dots r_t\}$$

$$z \xleftarrow{reconstruct} \{r_1, r_2, \dots r_t\}$$

BGW

Is it safe?



BGW

- Solution: randomization

  Before the degree reduction:

  - Let every party generate a polynomial $q_i(\cdot)$ of degree $2t-1$ with constant term 0, and reveal the polynomial.
  - Then for every party P_i, calculate:

$$h'(\alpha_i) = \boxed{h(\alpha_i)} + \boxed{\sum_{j=1}^{n} q_j(\alpha_i)}$$

Random number

Secret that can be factorized

as his share s_i

BGW

- Whole process

$$x \xrightarrow{share} \{x_1, x_2, \dots, x_t\}, \qquad y \xrightarrow{share} \{y_1, y_2, \dots, y_t\}$$

$$s'_i = x_i \cdot y_i = f(\alpha_i) \cdot g(\alpha_i)$$

$$s_i = s'_i + \sum_{j=1}^{n} q_j(\alpha_i)$$

$$\{s_1, s_2, \dots, s_{2t}\} \xrightarrow{projection} \{r_1, r_2, \dots r_t\}$$

$$z \xleftarrow{reconstruct} \{r_1, r_2, \dots r_t\}$$

Beaver’s Multiplication Triple

Donald Beaver, 1992

Multiplication Triple (MT)

Beaver's Multiplication Triple is a protocol for secret multiplication on additive shared arithmetic circuit.

- Additive shared: A secret x is shared by Alice and Bob. Alice has a share x_0, and Bob has a share x_1, that $x_0 + x_1 = x$.
- Arithmetic circuit: Secrets is shared and computed on an arithmetic level instead of bit level.

Alice holding secret x, Bob holding secret y, they want to jointly compute $f(x, y)$

Multiplication Triple (MT)

- Sharing:
  - Alice sample a random x_0, let $x_1 = x - x_0$, send x_1 to Bob.
  - Bob sample a random y_1, let $y_0 = y - y_1$, send y_0 to Alice.

- Reconstruction:
  - There is a secret z shared by Alice and Bob, each holding z_0, z_1
  - Alice reveal z_0, Bob reveal z_1, both party compute $z = z_0 + z_1$.

Multiplication Triple (MT)

- Addition

Secret addition can be computed locally.

Alice holding secret x, Bob holding secret y, they want to jointly compute $z = x + y$.

- Alice sample a random x_0, let $x_1 = x - x_0$, send x_1 to Bob.
- Bob sample a random y_1, let $y_0 = y - y_1$, send y_0 to Alice.
- Alice holding x_0, y_0 compute $z_0 = x_0 + y_0$
- Bob holding x_1, y_1 compute $z_1 = x_1 + y_1$
- Both party reveal shares of z and reconstruct

Multiplication by constant numbers can be computed using same method.

Multiplication Triple (MT)

- Multiplication

  Alice holding secret x, Bob holding secret y, they want to jointly compute $z = x \cdot y$.

  Let $[x], [y], [z]$ denote the shared value of x, y, z

  - Assume we can pre-produce random triples:

$$a, b, c$$

Where $a \cdot b = c$ and a, b, c are shared by Alice and Bob as a_0, b_0, c_0 and a_1, b_1, c_1, respectively. Let $[a], [b], [c]$ denote the shared value.

(This production can be done in an offline phase using one of the previous methods)

Multiplication Triple (MT)

- Multiplication

  To multiply $[x]$ and $[y]$, we take a new Beaver Triple (i.e. $[a], [b], [c]$) and:
  - Both party compute $[x] - [a] = [d]$, disclose $[d]$ and reconstruct d.
  - Both party compute $[y] - [b] = [e]$, disclose $[e]$ and reconstruct e.
  - Both party compute:

$$[z] = [c] + d \cdot [b] + e \cdot [a] + (d \cdot e)^*$$

  ***: Only one party need to add $d \cdot e$ to his share.**

  $[z]$ is the share of $z = x \cdot y$. If they don't need further computation, they reconstruct z.

Multiplication Triple (MT)

- Multiplication: Proof
  - Alice

$$z_0 = c_0 + d \cdot b_0 + e \cdot a_0 + de$$
$$= c_0 + (x-a) \cdot b_0 + (y-b)a_0 + (x-a)(y-b)$$
$$= c_0 + xb_0 - ab_0 + a_0 y - a_0 b + xy - bx - ay + ab$$

  - Bob

$$z_1 = c_1 + d \cdot b_1 + e \cdot a_1 + de$$
$$= c_1 + (x-a) \cdot b_1 + (y-b)a_1$$
$$= c_1 + xb_1 - ab_1 + a_1 y - a_1 b$$

  - Reconstruction

$$z_0 + z_1 = \quad + xy$$
$$= xy$$

Multiplication Triple (MT)

- Problem: How to distribute MTs without TTP(Trustful Third Party)?
  - HE: Homomorphic Encryption
  - OLE: Oblivious Linear Evaluation

Multiplication Triple (MT)

- Distribute MTs via HE
  - Paillier Homomorphic Encryption for brief:

Let x, y be secrets (in $\mathbb{Z}_{2^l}$), $Enc(x), Enc(y)$ be ciphertexts using Paillier encryption.

Then we have:

$$Enc_k(x + y) = Enc_k(x) \cdot Enc_k(y)$$
$$Enc_k(x \cdot y) = Enc_k(x)^y$$

i.e.

$$Dec_k\big(Enc_k(x) \cdot Enc_k(y)\big) = x + y$$
$$Dec_k(Enc_k(x)^y) = x \cdot y$$

Multiplication Triple (MT)

- Distribute MTs via HE
  - The 2 parties first additively share their inputs:
    $$P_0: x_0, y_0, \qquad P_1: x_1, y_1$$
  - P_0 samples r randomly and compute (which is his share):
    $$z_0 = x_0 \cdot y_0 - r$$
  - P_1 send P_0:
    $$Enc_{k1}(x_1), \qquad Enc_{k1}(y_1)$$
  - P_0 send P_1:
    $$d = \left(Enc_{k1}(x_1)\right)^{y_0} \cdot \left(Enc_{k1}(y_1)\right)^{x_0} \cdot Enc_{k1}(r)$$
  - P_0's share:
    $$z_1 = x_1 \cdot y_1 + Dec_{k1}(d) = x_1 \cdot y_1 + x_1 \cdot y_0 + y_1 \cdot x_0 + r$$

Multiplication Triple (MT)

- Distribute MTs via OT
  - To generate $x \cdot y = z$, observe that we can write:
    $$x \cdot y = (x_0 + x_1) \cdot (y_0 + y_1) = x_0 y_0 + x_0 y_1 + x_1 y_0 + x_1 y_1$$
  - P_0 randomly generate x_0, y_0, P_1 randomly generate x_1, y_1
  - $x_0 y_0, x_1 y_1$ can be computed locally, so we need to compute $x_0 y_1, x_1 y_0$
  - Take the computation of $x_0 y_1$ for example, since the other one can be computed symmetrically by reversing the parties' roles.

Multiplication Triple (MT)

- Distribute MTs via OT
  - We want to compute $x_0 y_1$, but plain $x_0 y_1$ known by one party will cause leak of information.
  - So we compute the sharing of it. Find u_0, u_1 that:

$$u_0 + u_1 = x_0 y_1$$

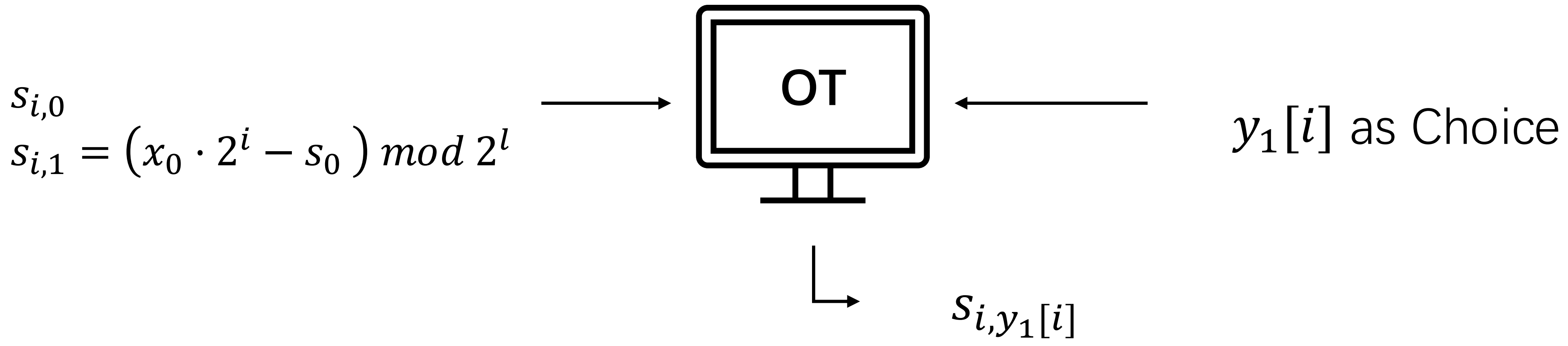
Which will be held by P_0, P_1 as shared values respectively.

Multiplication Triple (MT)

- Distribute MTs via OT

  P_0, P_1 start a C-OT(l, l). In i-th C-OT:

  - P_1 inputs y_1's i-th bit: $y_1[i]$ as the choice bit
  - P_0 inputs the correlation function $f_{\Delta_i}(x) = \left(x_0 \cdot 2^i - x\right) mod\ 2^l$
  - Every round P_0 samples $s_{i,0}$ randomly.



Multiplication Triple (MT)

- Distribute MTs via OT
  - Now, P_0 has: $\{s_{0,0}, s_{1,0}, s_{2,0}, \dots, s_{l-1,0}\}$
  - Now, P_1 has: $\{s_{0,y[0]}, s_{1,y[1]}, s_{2,y[2]}, \dots, s_{l-1,y[l-1]}\}$
  - Then P_0 's share $u_0 = \sum_{i=0}^{l-1} s_{i,0}$
  - Then P_1 's share $u_1 = \sum_{i=0}^{l-1} s_{i,y[i]}$

Multiplication Triple (MT)

- Distribute MTs via OT

  Proof (or example):

  - P_0 has input $x_0 = 1101$
  - P_1 has input $y_1 = 1001$
  - 0-th OT: P_0 samples $s_{0,0} = 1000$, then $s_{0,1} = f_{\Delta,0}(s_{0,0}) = 1101 \cdot 1 - 1000$

    P_1 obtains $s_{0,y[0]} = s_{0,1} = 0101$
  - 1-th OT: P_0 samples $s_{1,0} = 1001$, then $s_{1,1} = f_{\Delta,0}(s_{1,0}) = 1101 \cdot 10 - 1001$

    P_1 obtains $s_{1,y[1]} = s_{1,0} = 1001$
  - …

Comparison & Conclusion

Protocols including OT, GC, GMW, BGW, MT

Comparison:

	Circuit Type	Parties/ scalability	Communication rounds	Computation cost (offline phase)	sensitivity to latency (online)
OT	/	2 party;	2-3 rounds; 1-2 in OT extension	/	/
GC	Boolean	2 party; Poor scalability	Constant rounds ($O(n_{input})$)	High in encryption	Low due to constant rounds
GMW	Boolean	Multi-party; Scales well	Logarithmic in circuit depth($O(\log n_{dep})$)	Low	High due to many rounds
BGW	Arithmetic	Multi-party; Scales well	Linear in circuit depth($O(n_{dep})$)	High in reconstruction	High due to many rounds
MT	Arithmetic	2 party; Scales well	Logarithmic in Mul gates($O(\log n_{Mul})$)	High (preprocess) in generating triples	Moderate sensitivity

References

[1] https://en.wikipedia.org/wiki/Oblivious_transfer

[2] https://zhuanlan.zhihu.com/p/126396795

[3] https://www.youtube.com/watch?v=wE5cl8J27Is

[4] https://www.cs.umd.edu/~jkatz/gradcrypto2/s21

[5] Evans, David, Vladimir Kolesnikov, and Mike Rosulek. "A pragmatic introduction to secure multi-party computation." Foundations and Trends® in Privacy and Security 2.2-3 (2018): 70-246.

[6] Lindell, Yehuda. "How to simulate it–a tutorial on the simulation proof technique." Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich (2017): 277-346.

[7] Yao, Andrew Chi-Chih. "How to generate and exchange secrets." 27th annual symposium on foundations of computer science (Sfcs 1986). IEEE, 1986.

[8] Micali, Silvio, Oded Goldreich, and Avi Wigderson. "How to play any mental game." Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC. New York: ACM, 1987.

References

[9] Ben-Or, Michael, Shafi Goldwasser, and Avi Wigderson. "Completeness theorems for non-cryptographic fault-tolerant distributed computation." Providing sound foundations for cryptography: on the work of Shafi Goldwasser and Silvio Micali. 2019. 351-371.

[10] Beaver, Donald. "Efficient multiparty protocols using circuit randomization." Advances in Cryptology—CRYPTO'91: Proceedings 11. Springer Berlin Heidelberg, 1992.

[11] Asharov, Gilad, et al. "More efficient oblivious transfer and extensions for faster secure computation." Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. 2013.

[12] Demmler, Daniel, Thomas Schneider, and Michael Zohner. "ABY-A framework for efficient mixed-protocol secure two-party computation." NDSS. 2015.

[13] Rabin, Michael O. "How to exchange secrets with oblivious transfer." Cryptology ePrint Archive (2005).

[14] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in EUROCRYPT'99, ser. LNCS, vol. 1592. Springer,1999, pp. 223–238.