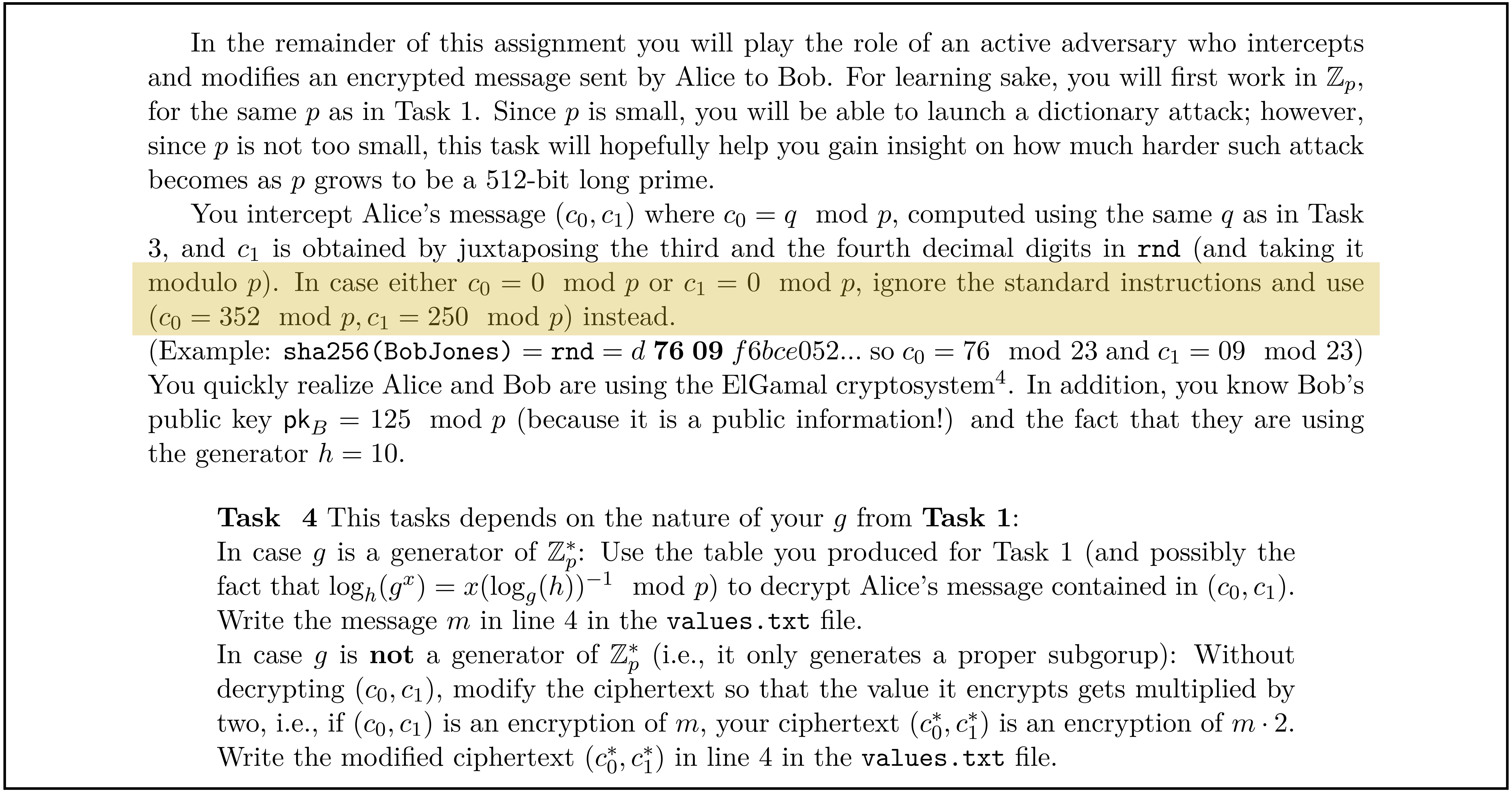
CRYPTOGRAPHY

(lecture 9)

Literature:

"A Graduate Course in Applied Cryptography" (ch 11.6.1,11.6.2)
"Lecture Notes on Cryptography" (**ch 12.3.1**)
"Lecture Notes on Introduction to Cryptography" by V. Goyal (ch10, **10.2.1, 10.3.1, 13.4**)
"Lecture Notes on Cryptographic Protocols" by Schoenmakers (ch **6.1.1,6.2.2**, all ch 6)
"Cryptography Made Simple" by N. Smart: (Part 4 ch 20)
"Efficient Secure Two-Party Protocols" by C. Hazay & H. Lindell (ch7.0-7.2.1, DDH-based OT)

Announcements



In the remainder of this assignment you will play the role of an active adversary who intercepts and modifies an encrypted message sent by Alice to Bob. For learning sake, you will first work in $\mathbb{Z}_p$, for the same p as in Task 1. Since p is small, you will be able to launch a dictionary attack; however, since p is not too small, this task will hopefully help you gain insight on how much harder such attack becomes as p grows to be a 512-bit long prime.

You intercept Alice's message (c_0, c_1) where $c_0 = q \mod p$, computed using the same q as in Task 3, and c_1 is obtained by juxtaposing the third and the fourth decimal digits in `rnd` (and taking it modulo p). In case either $c_0 = 0 \mod p$ or $c_1 = 0 \mod p$, ignore the standard instructions and use $(c_0 = 352 \mod p, c_1 = 250 \mod p)$ instead.
(Example: `sha256(BobJones)` = `rnd` = d **76 09** $f6bce052$... so $c_0 = 76 \mod 23$ and $c_1 = 09 \mod 23$)
You quickly realize Alice and Bob are using the ElGamal cryptosystem[4]. In addition, you know Bob's public key $\mathsf{pk}_B = 125 \mod p$ (because it is a public information!) and the fact that they are using the generator $h = 10$.

Task 4 This tasks depends on the nature of your g from **Task 1**:
In case g is a generator of $\mathbb{Z}_p^*$: Use the table you produced for Task 1 (and possibly the fact that $\log_h(g^x) = x(\log_g(h))^{-1} \mod p$) to decrypt Alice's message contained in (c_0, c_1). Write the message m in line 4 in the `values.txt` file.
In case g is **not** a generator of $\mathbb{Z}_p^*$ (i.e., it only generates a proper subgorup): Without decrypting (c_0, c_1), modify the ciphertext so that the value it encrypts gets multiplied by two, i.e., if (c_0, c_1) is an encryption of m, your ciphertext (c_0^*, c_1^*) is an encryption of $m \cdot 2$. Write the modified ciphertext (c_0^*, c_1^*) in line 4 in the `values.txt` file.

Module 3: Agenda

Introduction to Multi-Party Computation

- What Is MPC
- Terminology
- An Intuition of Security

Secret Sharing Scheme

- Motivation
- Definition
- A Simple Construction
- Shamir Secret Sharing Scheme **HA3**
- An Application: Threshold Cryptography

Commitment Schemes Interlude

- Pedersen Commitment - Proof **HA3**
- A Security Note - Proof

Back to Sharing

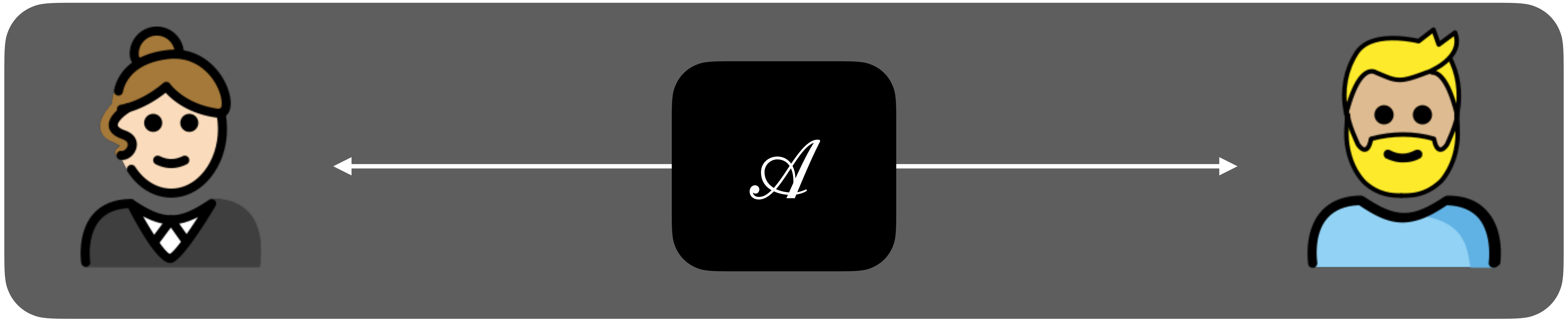
- Verifiable Secret Sharing (Pedersen)

Oblivious Transfer

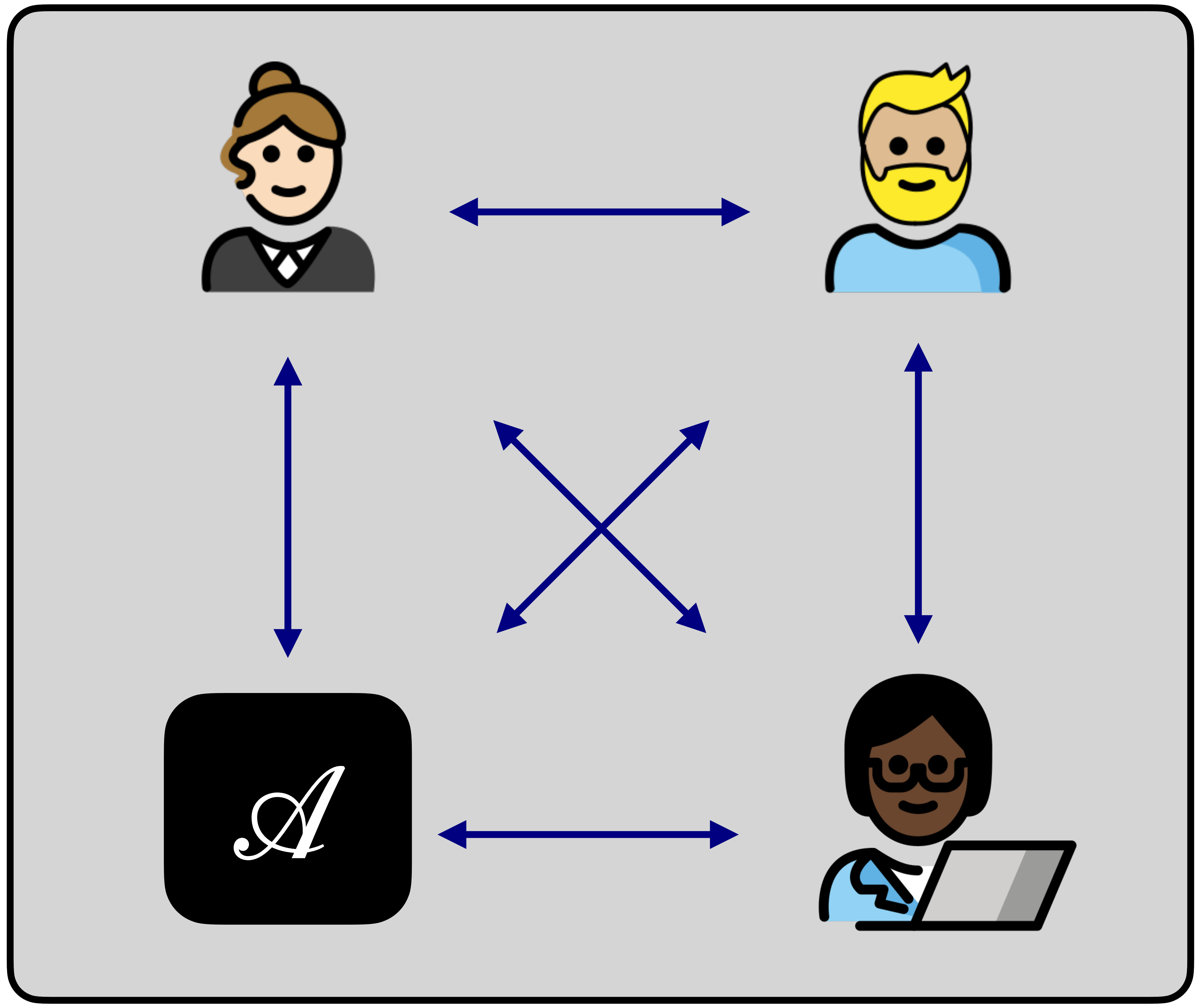
- Example
- DDH-Based Construction

Zero-Knowledge Proofs **HA3**
Σ-Protocols **HA3**
Generic 2 Party Computation
Garbled Circuits

In Module 1 and 2



In Module 3



SECURE MULTI-PARTY COMPUTATION

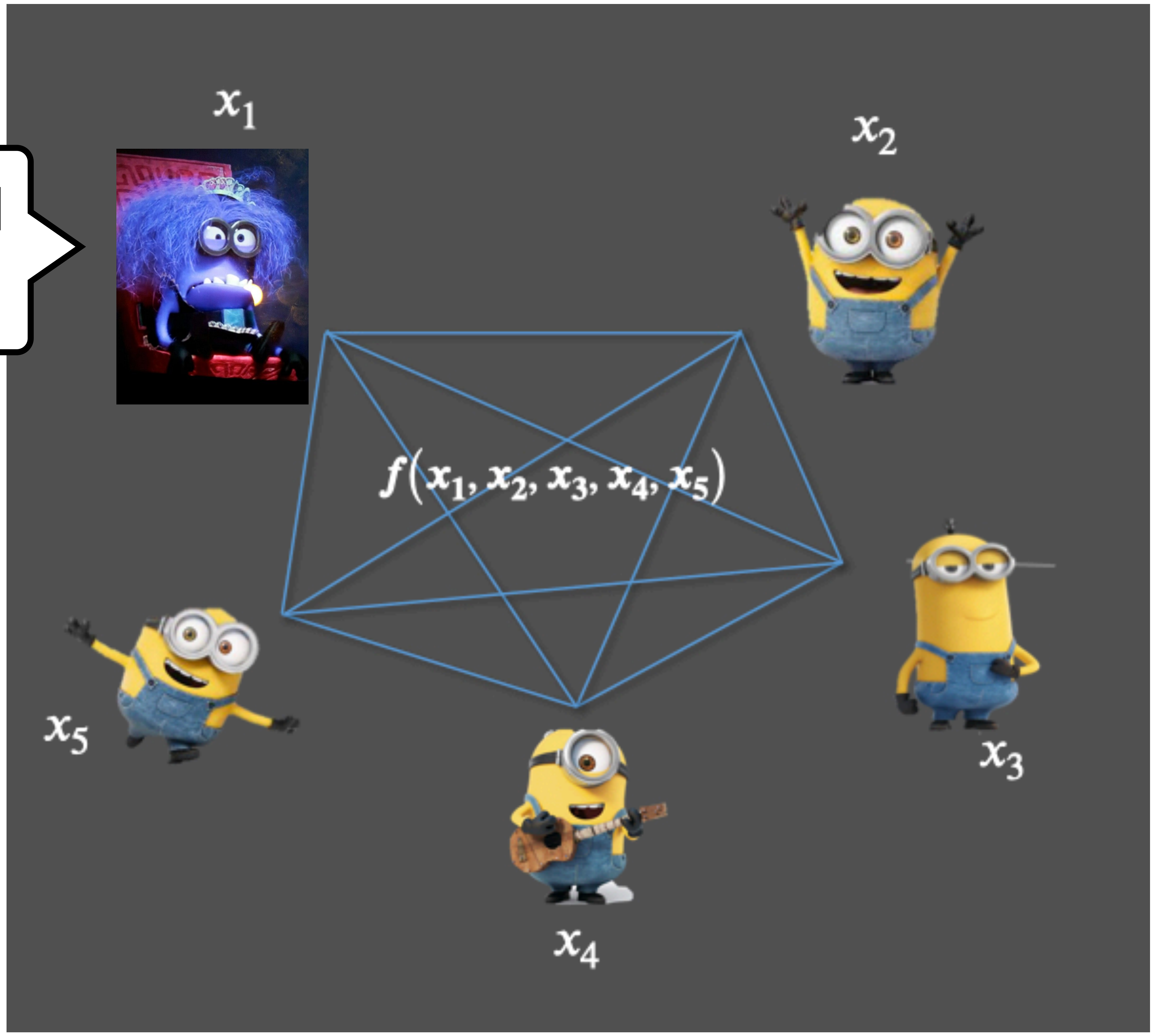
Aim: realise *security* for **advanced** *settings*, often relying on **basic** *tools* (*primitives*) from Module 1 and 2.

What Is Multi-Party Computation?

"everything is MPC"

The aim of **secure multiparty computations** is to enable a number of distinct, yet connected, computing devices (or *parties*) to carry out a joint *computation* of some *function* in a *secure* manner.

corrupted party



Key Agreement
Blind Signatures
. . .
Untreatable eCash
Private Set Intersection
Private Information Retrieval
. . .
Mental Poker
(Socialist) Millionaire Problem
. . .
Secret Sharing
Threshold Cryptography

Security Requirements in MPC

Privacy: No party should learn anything more than its prescribed output.

◉ *the only information that should be learned about other parties' inputs is what can be inferred from the output itself.*

Example: An auction where the only bid revealed is the one of the highest bidder. Inferred information: all other bids were lower than the winning bid. However, this should be the only information revealed about the losing bids.

Correctness: Each party is guaranteed that the output that it receives is correct.

Guaranteed Output Delivery (GOD): Corrupted parties should not be able to prevent honest parties from receiving their output.

Fairness: Corrupted parties should receive their outputs if and only if the honest parties also receive their outputs.

The Adversary’s Power

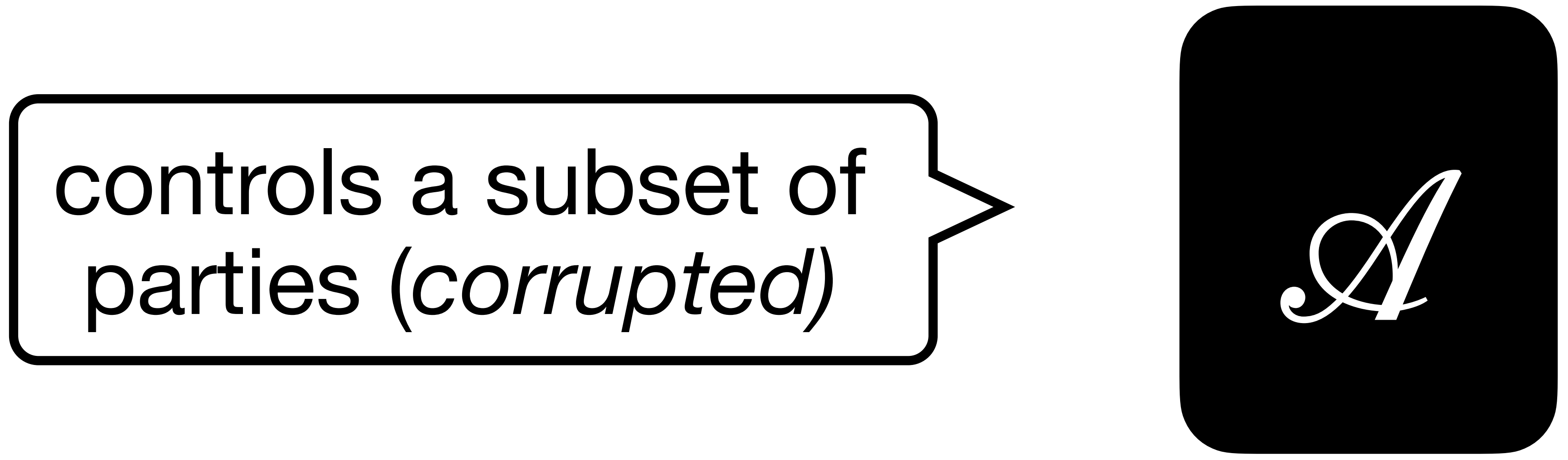
controls a subset of parties (*corrupted*)

$\mathscr{A}$

Corruption Strategies: when or how parties come under the “control” of the adversary

- **Static**: $\mathscr{A}$ is given a fixed set of parties whom it controls. Honest parties remain honest throughout and corrupted parties remain corrupted.
- **Adaptive**: $\mathscr{A}$ can corrupt parties during the computation. The choice of whom to corrupt, and when, can be arbitrarily decided by the adversary and may depend on what it has seen throughout the execution. Once a party is corrupted, it remains corrupted from that point on.
- **Proactive/Dynamic**: $\mathscr{A}$ can corrupt parties for a certain period of time only. Not only honest parties may become corrupted throughout the computation, but also corrupted parties may also become honest.

The Adversary’s Power



Adversarial Behaviour: the actions that corrupted parties are allowed to take.

◉ **Semi-honest:** corrupted parties correctly follow the protocol specification. However, $\mathscr{A}$ obtains the internal state of all the corrupted parties (including the transcript of all the messages received), and attempts to use this to learn information that should remain private. *Semi-Honest = Honest-but-Curious = Passive*

◉ **Malicious**: corrupted parties can arbitrarily deviate from the protocol specification, following $\mathscr{A}$’s instructions. *Malicious = Active*

◉ **Covert:** $\mathscr{A}$ may behave maliciously but if it does so, it will be caught cheating by the honest parties, with some given probability. *Covert = Rational*

The Adversary's Power

controls a subset of parties (*corrupted*)

$\mathscr{A}$

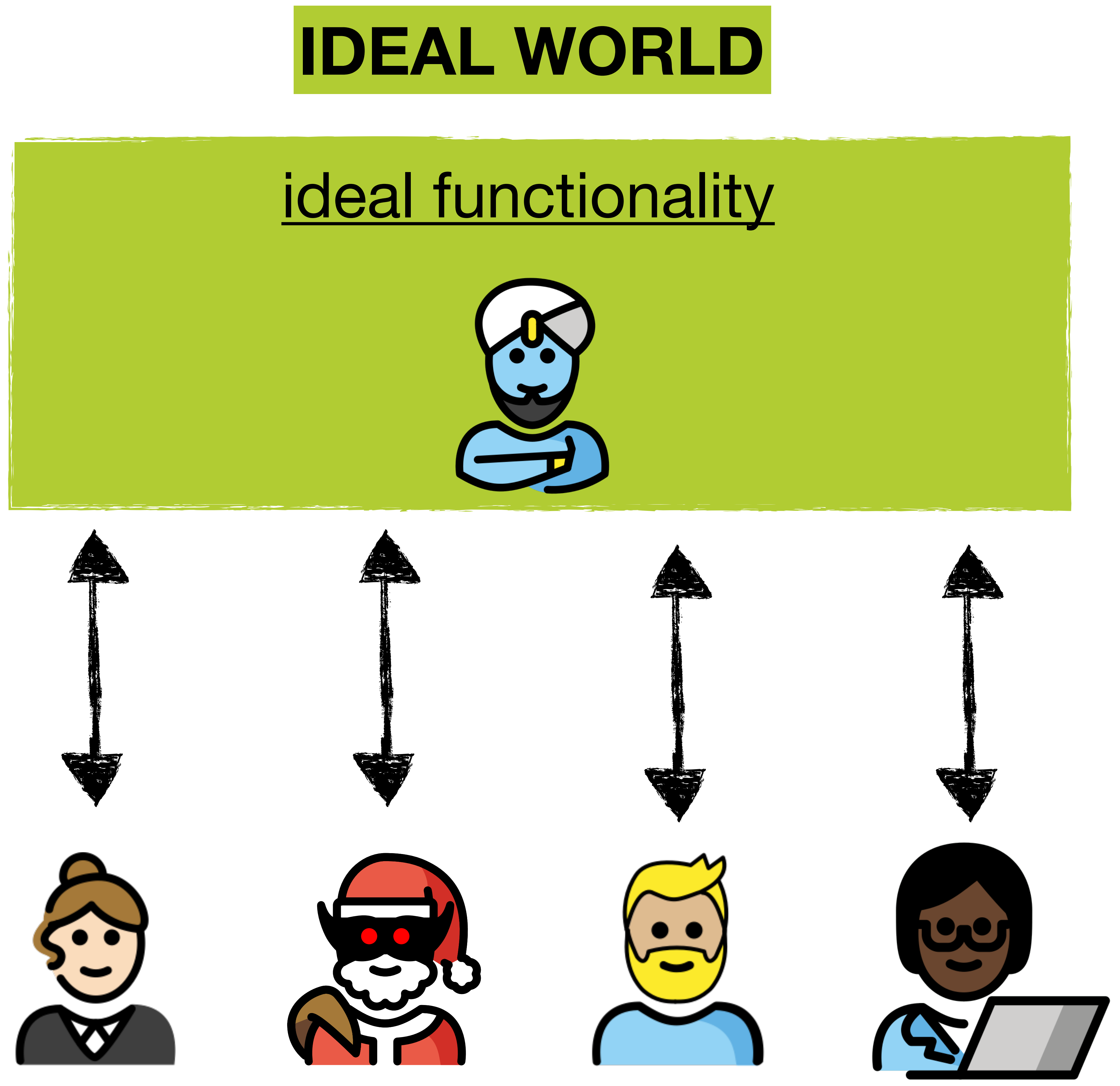
(Computational) **Complexity:**

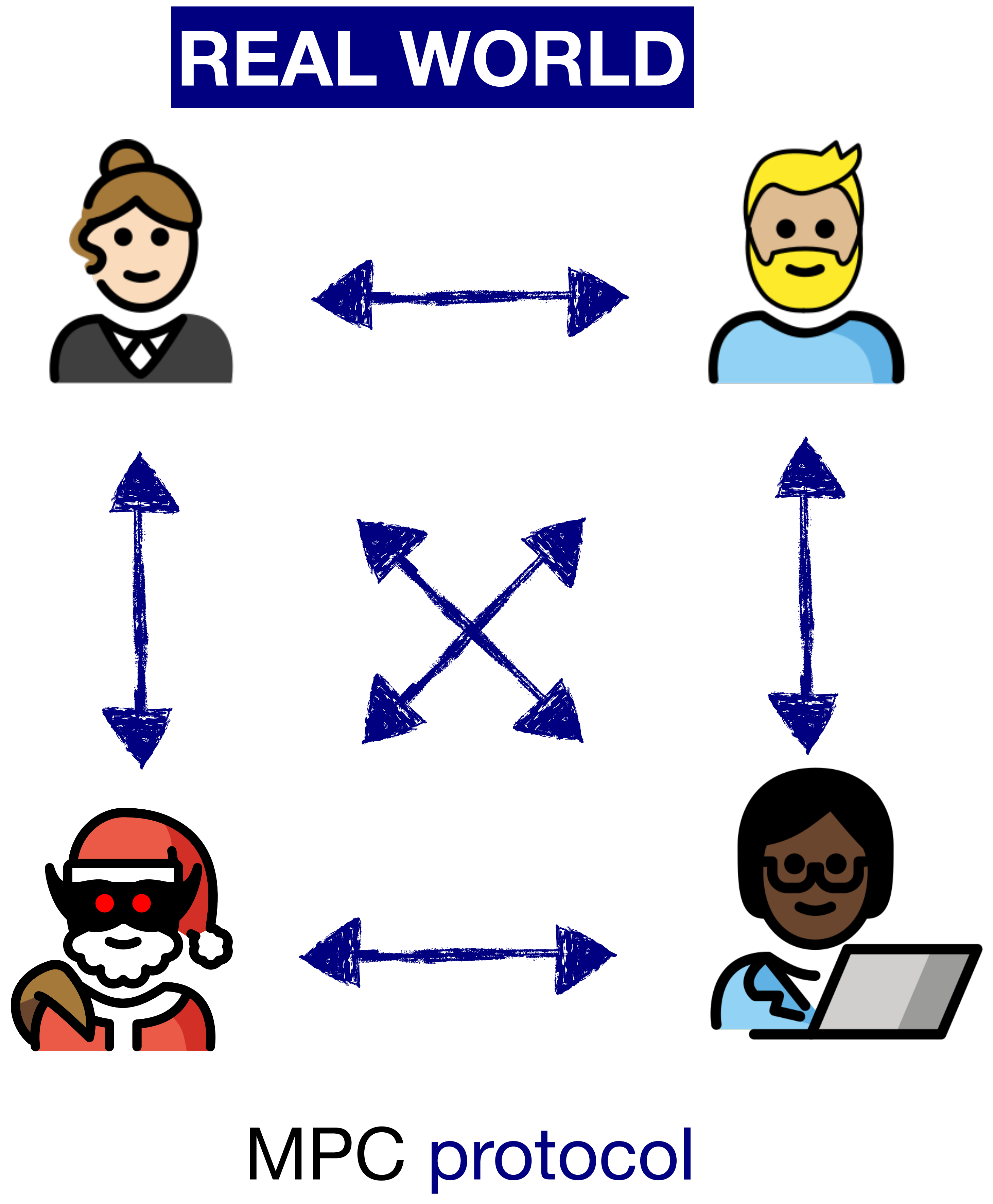
◉ **Polynomial time:** $\mathscr{A}$ is allowed to run in (probabilistic) polynomial time (and sometimes, expected polynomial time).

◉ **Computationally unbounded**: $\mathscr{A}$ has no computational limits whatsoever, is not bound to any complexity class and is not assumed to run in polynomial time.

Security Models in MPC: computational vs. information-theoretic (unconditional)

The Core Idea of Secure Multi-Party Computation (MPC)

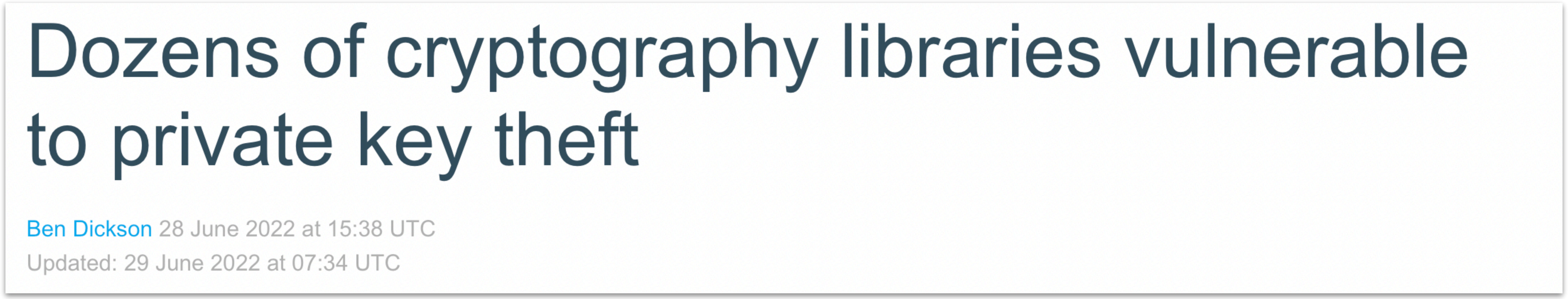




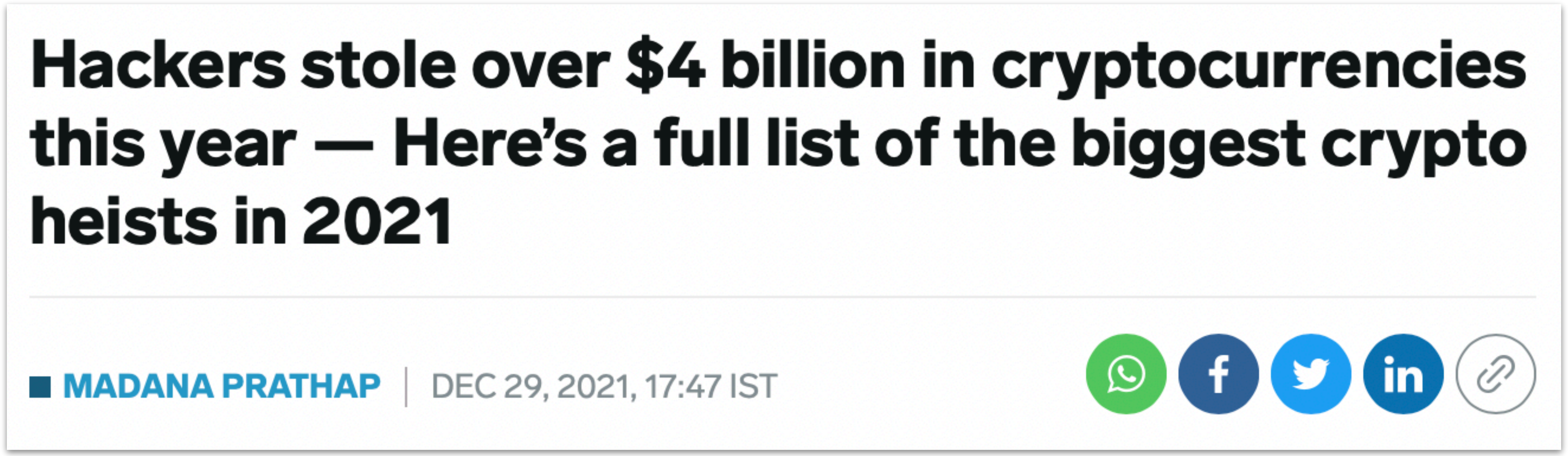
MPC protocols allow multiple parties to jointly compute a function over private inputs (the input of each party remains unknown to the other parties)

Module 3: Agenda

Sharing a Secret…Why?

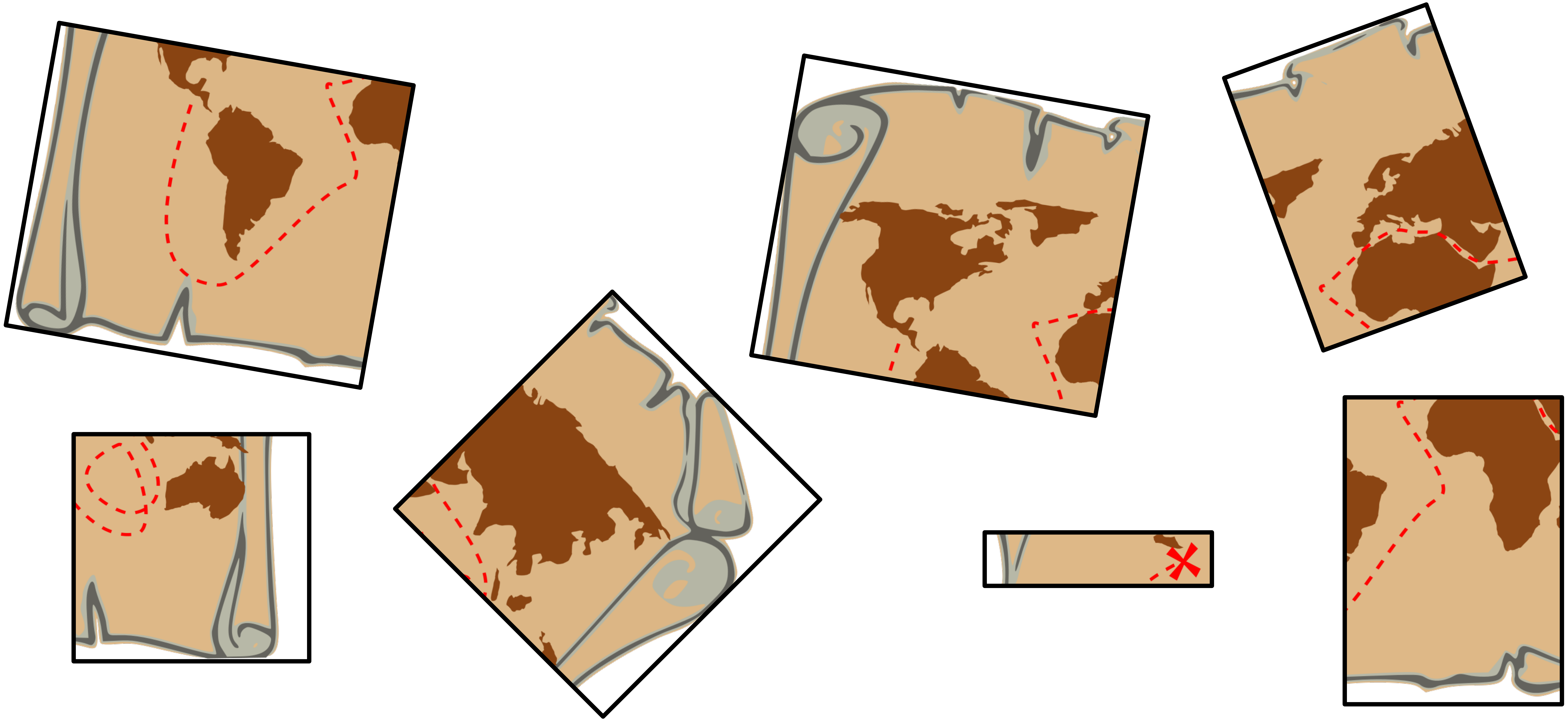

Dozens of cryptography libraries vulnerable to private key theft

Ben Dickson 28 June 2022 at 15:38 UTC
Updated: 29 June 2022 at 07:34 UTC


Hackers stole over $4 billion in cryptocurrencies this year — Here's a full list of the biggest crypto heists in 2021

MADANA PRATHAP | DEC 29, 2021, 17:47 IST

Secret Sharing



How To Formalise This Notion?

Definition: Secret Sharing Scheme

An (n, t)-secret sharing scheme is a method by which a dealer distributes shares of a secret value s to a set of $n \geq 2$ parties so that:

1) $t-$**correctness**: any set of t parties can reconstruct s (together)
2) **privacy**: no single party alone learns anything about s
3) $(t-1)$**-security**: any subset of less than t parties cannot compute s

To define a secret sharing scheme it suffices to explain how two algorithms work: $Distribute$ (run by the dealer) and $Reconstruct$ run by the share holders (parties)

A Simple Construction: $(n\ out\ of\ n)$ Secret Sharing

Functionality : Given a secret value $s \in \mathbb{Z}_N$, generate n values $x_1, \ldots, x_n$s.t. $s = f(x_1, \ldots, x_n)$ for some public function $f(\cdot)$

A trivial realisation

$$x_1, \ldots, x_{n-1} \leftarrow \$ - \mathbb{Z}_N$$

$$x_n = s - (x_1 + \ldots + x_{n-1}) \mod N$$

$$f(x_1, \ldots, x_n) = (x_1 + \ldots + x_n) \mod N$$

✓ 1) $t-$**correctness**: any set of $t\ (= n)$ parties can reconstruct s (together)

✓ 2) **privacy**: no single party alone learns anything about s

✓ 3) $(t-1)$-**security**: any subset of less than n parties cannot compute s

A More Interesting Construction: 2 Out of 3 Secret Sharing

Functionality : Given a secret value $s \in \mathbb{Z}_N$, generate n values $x_1, \ldots, x_n$ s.t. for any subset $T \subseteq \{1,\ldots,n\}$ with cardinality $|T| = t$ it holds that $s = f_T(x_i : i \in T)$ for some public function $f_T(\cdot)$

A trivial realisation fix $t = 2, n = 3$

$y_1, y_2 \leftarrow \$ - \mathbb{Z}_N$

$y_3 = s - y_1 - y_2 \mod N$

$x_1 = \{y_2, y_3\}, x_2 = \{y_1, y_3\}, x_3 = \{y_1, y_2\}$

$f_{1,2}(x_1, x_2) = y_1 + y_2 + y_3 \mod N$

👍 any set of $t = 2$ parties can reconstruct the secret s

👍 one party alone cannot retrieve s

👎 (not optimal) to secret share ONE value in $\mathbb{Z}_N$ each party needs to store TWO values in $\mathbb{Z}_N$

🧐 *can we do better than this?*

Shamir Secret Sharing Scheme $(t \text{ out of } n)$

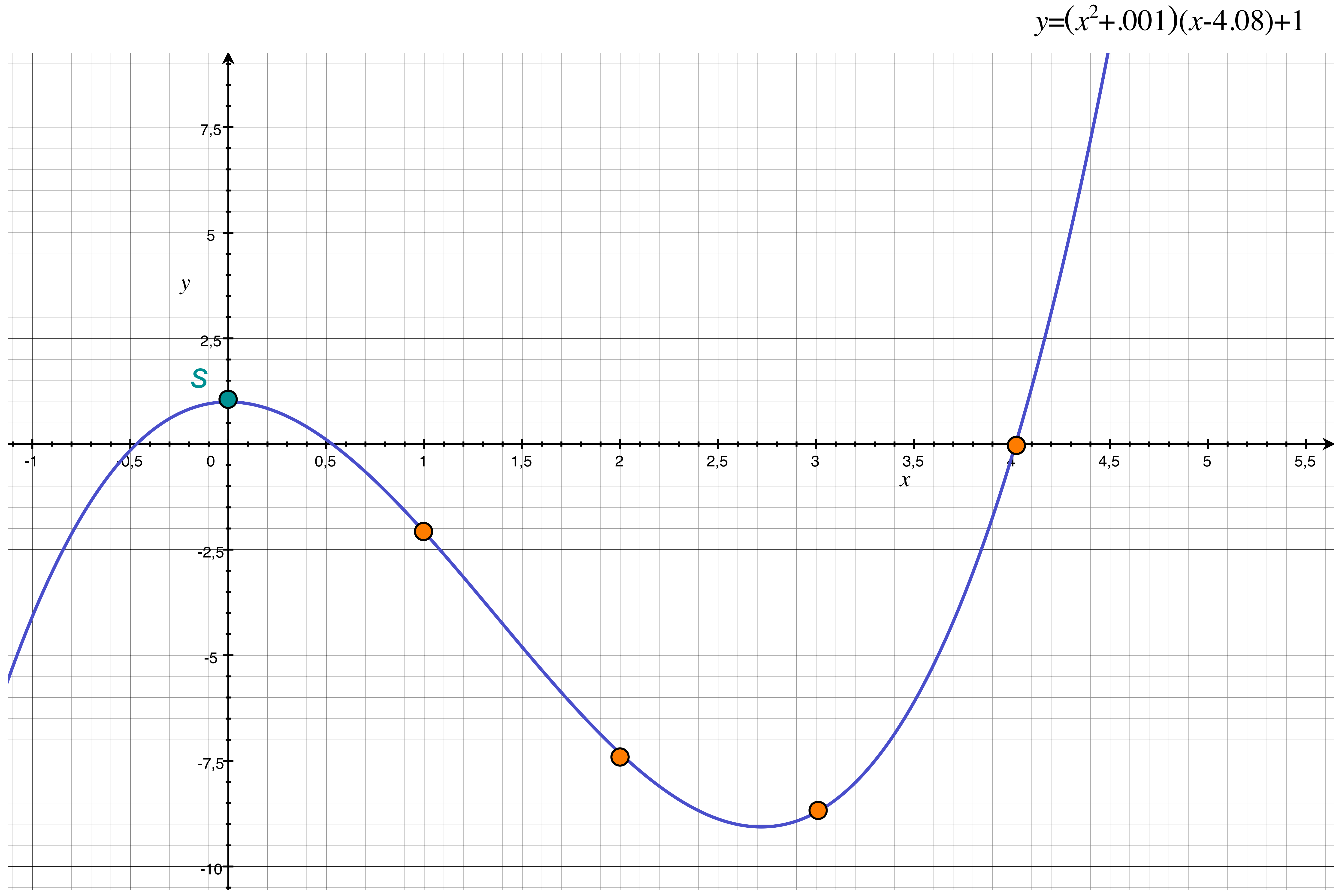
let *s* = *1* be the secret value

construct a random polynomial of degree $t - 1$ passing by the point (0, *s*)

compute the share for party **P*ᵢ*** as *sᵢ = f(i) mod p* for i ∈ {1, 2, 3, .. n }

👀 the shares *sᵢ* look completely random

but if you interpolate $t = 4$ shares, there exists only one polynomial of degree $3 = t - 1$ that passes by all of them.



the polynomial is exactly *f*, and *f (0) = s* is exactly the secret (reconstructed by interpolation from the shares)

Shamir Secret Sharing Scheme $(t\ out\ of\ n)$

How to reconstruct *s* from the *shares*?

let there be *t* parties **P₁ P₂ … Pₜ**

each holding its share $s_i = f(i) \bmod p$

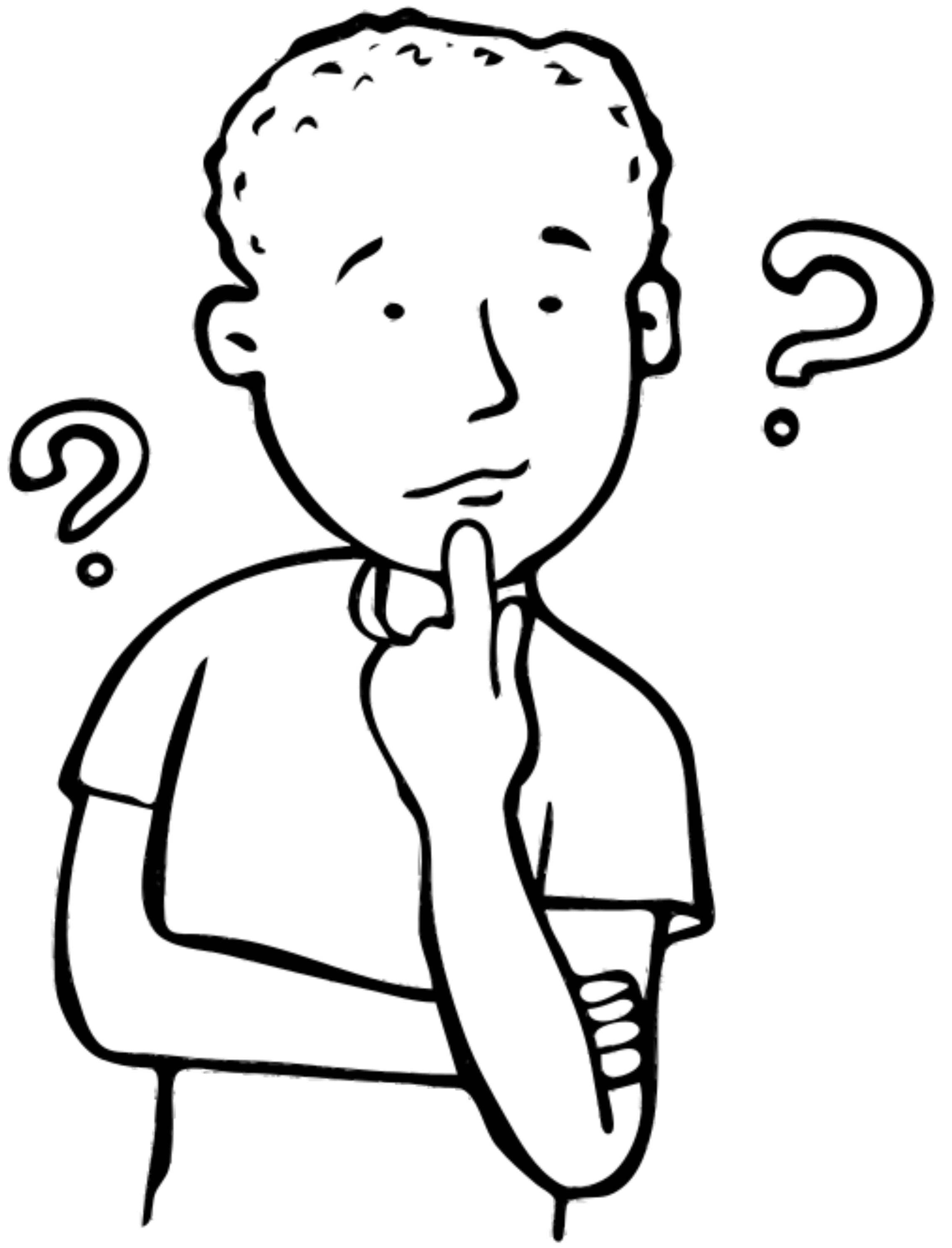
the set of parties can reconstruct the secret *s* by computing:

$s = s_1\delta_1 + s_2\delta_2 + \ldots + s_t\delta_t$

where $\delta_i = \prod_{j \in \{1,\ldots,t\} \setminus \{i\}} \frac{j}{j-i} \bmod p$

Lagrange coefficients

this expression is actually a number!



e.g. (for P_1, P_2, P_3) $\delta_1 = 3$, $\delta_2 = -3$, $\delta_3 = -1$

e.g. (for P_1, P_4) $\delta_1 = 4/3$, $\delta_4 = -1/3$ ***(inverses to be computed mod p)***

Shamir Secret Sharing - Generic Reconstruction

Lagrange Interpolation

For any polynomial $f(x) \in \mathbb{Z}_p[x]$ of degree $d = (t-1) > 0$, given $t = d + 1$ distinct points $I = \{i_1, i_2, \ldots, i_t\} \subseteq \mathbb{Z}_p$ it holds that:

$$f(x) = f(i_1)\delta_{i_1}(x) + f(i_2)\delta_{i_2}(x) + \ldots + f(i_t)\delta_{i_t}(x) \in \mathbb{Z}_p[x]$$

where $\delta_i(x) = \prod_{j \in I \setminus \{i\}} \frac{x-j}{i-j} \mod p$

the $\delta_i(x)$ are called Lagrange basis polynomials, sometimes the set I is specified as $\delta_i^I(x)$

🧐 *why does this work?*

[polynomial identity: $f(x)$ and $\sum_{i \in I} f(i)\delta_i^I(x)$ coincide on $d+1$ points]

👀 the polynomials $\delta_i(x)$ only depend on $i, j \in I$ and not on $f(x)$

👀 the polynomials $\delta_i(x)$ exist for every i and every set of $I \subseteq \mathbb{Z}_p$ of cardinality t

👀 setting $x = 0$ we get $s = f(0) = f(i_1)\delta_{i_1}(0) + f(i_2)\delta_{i_2}(0) + \ldots + f(i_t)\delta_{i_t}(0) \mod p$ (correctness)

Shamir (n, t) Secret Sharing Scheme (Formal)

$Distribute(q, n, t, s)$: to (n, t) secret-share the value $s \in \mathbb{Z}_p$ among n parties, the dealer does the following:

- sample t random values $a_1, \ldots, a_t \leftarrow \$\mathbb{Z}_q$
- construct the polynomial $f(x) = s + a_1 x + a_2 x^2 + \cdots + a_t x^t \in \mathbb{Z}_q[x]$
- compute the n shares by evaluating $f(\cdot)$ on n distinct points: $s_i = f(i)$ for $i \in \{1, 2, \ldots, n\}$
- send the share s_i to party P_i (via a secure channel)

$Reconstruct(I, \{s_i\}_{i \in I})$: to reconstruct the secret given $t = |I|$ distinct shares do:

- compute the Lagrange coefficients for the set I: $\delta_i^I(0) \in \mathbb{Z}_q$
- Recover the secret $s = \sum_{i \in I} s_i \cdot \delta_i^I(0) \mod p$

Is Shamir’s Scheme a Secure Secret Sharing?

✓ 1) $t-$**correctness**: any set of t parties can reconstruct s (together)

✓ 2) **privacy**: no single party alone learns anything about s

✓ 3) $(t-1)$**-security**: any subset of less than t parties cannot compute s

Yes! And all properties hold unconditionally (information theoretically)

But for the scheme to work, we rely on the fact that the n parties have peer-to-peer, confidential and possibly authenticated channels (that can be built with techniques from Module 1 and 2).

And Now What?

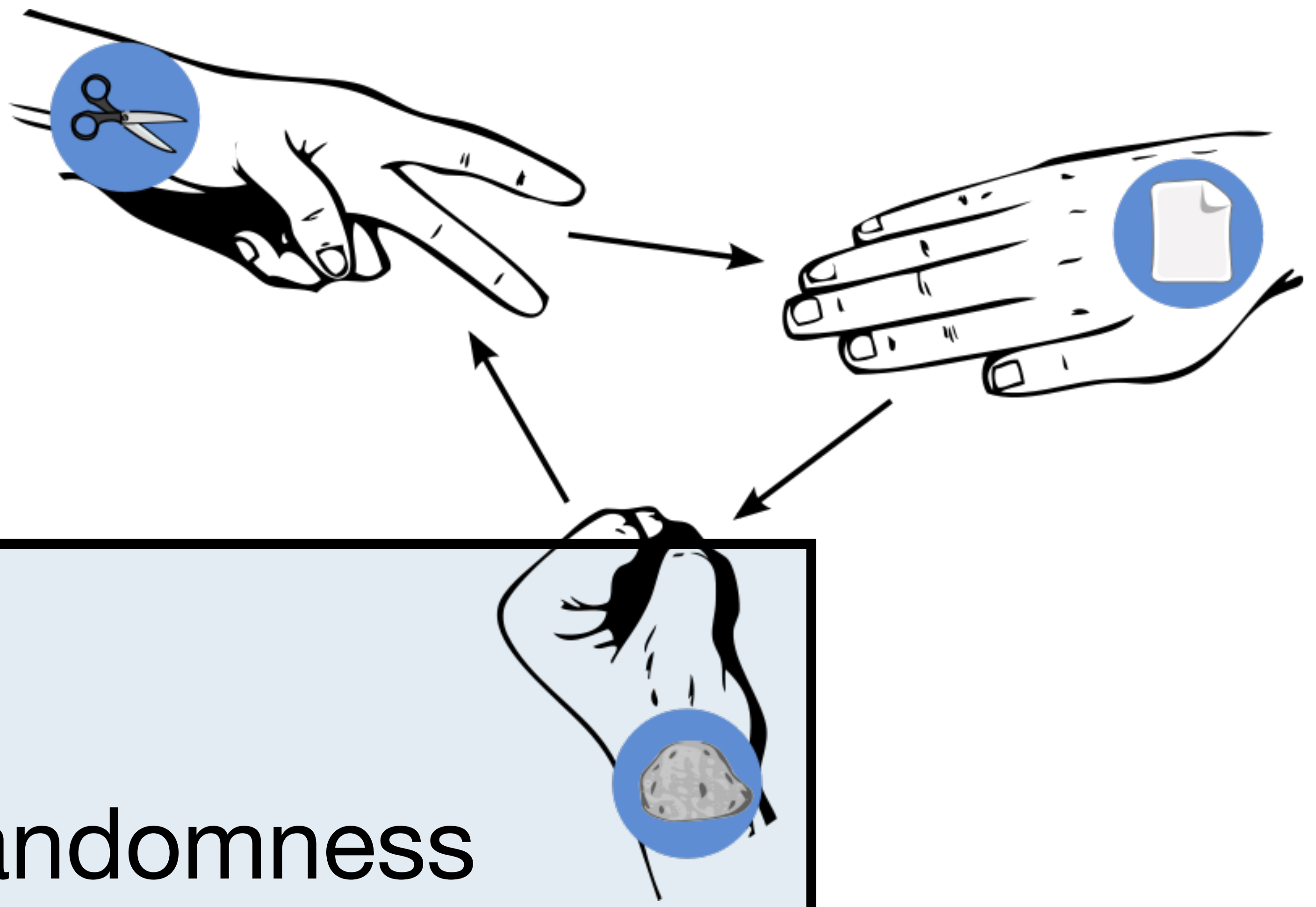
Threshold Cryptography

Threshold El-Gamal's Cryptosystem

- $KeyGen(sec\,.\,par)$: [same setting as ElGamal] Sample $x \leftarrow \$\mathbb{Z}_q^*$. Set $\mathsf{pk} = g^x$.
  Use Shamir secret sharing to compute $(\mathsf{sk}_1, \ldots \mathsf{sk}_n) \leftarrow Distribute(q, n, t, x)$.
  Output pk and send sk_i to party i.
- $Enc(\mathsf{pk}, m)$: Sample $r \leftarrow \$\mathbb{Z}_q^*$. Output the ciphertext $c = (g^r, m \cdot \mathsf{pk}^r)$.
- $Dec'(\mathsf{sk}_i, c)$: Parse $c = (c_1, c_2)$. Compute $d_i = c_1^{\mathsf{sk}_i}$.
- $Recover(I, \{d_i\}_{i\in I}, c)$:
  Compute $D = \prod_{i\in I} d_i^{\delta_i(0)} = g^{r\sum_{i\in I}\mathsf{sk}_i\,\cdot\,\delta_i(0)} = g^{rx}$. Parse $c = (c_1, c_2)$.
  Output $m = c_2 \cdot D^{-1}$.

Module 3: Agenda

Commitment Schemes Definitions



Syntax

A commitment scheme over a set of messages $\mathscr{M}$, a set of keys/randomness $\{0,1\}^n$ and a set of commit values C is defined by the two following PPT algorithms:

- Commit$(m, r) = c$ is a deterministic algorithm that takes in input a message m and a random string r; and outputs a commitment c to m.
- Open$(m, r, c) \in \{0,1\}$ this is a deterministic algorithm that takes in input a message m and a random string r, and a commitment c, and returns 1 (accept) if c is a valid commitment (for m, r); and 0 (reject) otherwise.

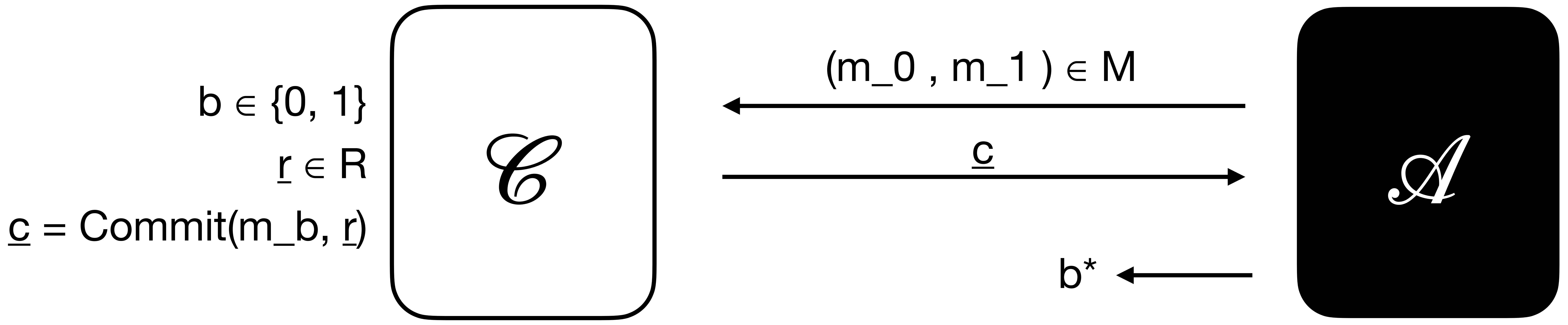
… and satisfying the **binding** and **hiding** properties (given next)

Commitment Schemes

🧐 *Is it possible to realize a Commitment Scheme using any Encryption Scheme?*

IND-CPA security for Encryption Schemes !

have you seen this game before? 👇



|Prob[b* = b] - 1/2| ≤ negligible

Pedersen Commitment Scheme

Setup(sec.par) → ($\mathbb{G}$, q, g, h)
Commit(m,r) = $g^m h^r$ mod q =: c
Open(m,r,c) = 1 if c = $g^m h^r$ mod q, and 0 otherwise

Setting: g,h are two distinct generators of a group $\mathbb{G}$ of order q

Binding? yes, *computationally*

Prob[Commit(m , r) = c = Commit(m*, r*) | m≠m*] ≤ negligible

Proof by reduction:
Given the values {c, (m,r) , (m*,r*)} we can extract the discrete logarithm of h with respect to g via

$$dLog_g(h) = \frac{m* - m}{r - r*}$$

Hiding? yes, information-theoretically

|Prob[b* = b] - 1/2| ≤ negligible

Proof: for every m_1 there exist an r* s.t. commit(m_1,r*)=commit(m_0,r) [similar to the one time pad]

An (Important) Security Note

Theorem There exists no commitment scheme which is *both* information-theoretically concealing *and* binding. *[Lemma 20.3. from the book]*

Proof. (by reduction to absurd)

Suppose we have a scheme which is both information-theoretically concealing and binding,
(now we want to conclude that this contradicts the statement of the theorem)
and suppose the committing party (the sender) generates a commitment c = Commit(m, r).
The information-theoretical hiding property implies that there must exist values m* (≠ m) and r* such that c = Commit(m*, r*);
why?
otherwise an infinitely powerful receiver could break the concealing property (find the unique pair (m,r) that generates the commitment c).
But now we know that there exists an m* s.t. Commit(m*,r*)=c=Commit(m,r) which means the commitment is not binding (and an infinitely powerful sender can find such m*)
So we conclude that if the commitment scheme is information-theoretically hiding it cannot be binding for a computationally unbounded sender, which contradicts the statement of the theorem. The conclusion is that it was absurd to assume the existence of a commitment scheme that is both hiding and binding in a information-theoretically way. $\square$

Module 3: Agenda

Pedersen Verifiable Secret Sharing (VSS)

- $Distribute(s)$: *[this algorithm is run by the dealer, who holds the secret $s \in \mathbb{Z}_p$]*

  Sample at random $a_1, \ldots, a_{t-1}, b_0, \ldots, b_{t-1} \leftarrow \$\mathbb{Z}_p$. Define the polynomials $a(x) = s + a_1 x + \ldots + a_{t-1}x^{t-1}$ and $b(x) = b_0 + b_1 x + \ldots + b_{t-1}x^{t-1}$.

  Compute the share for party P_i as: $s_i = (a(i), b(i))$ *[s_i is sent to P_i via a secure channel]*

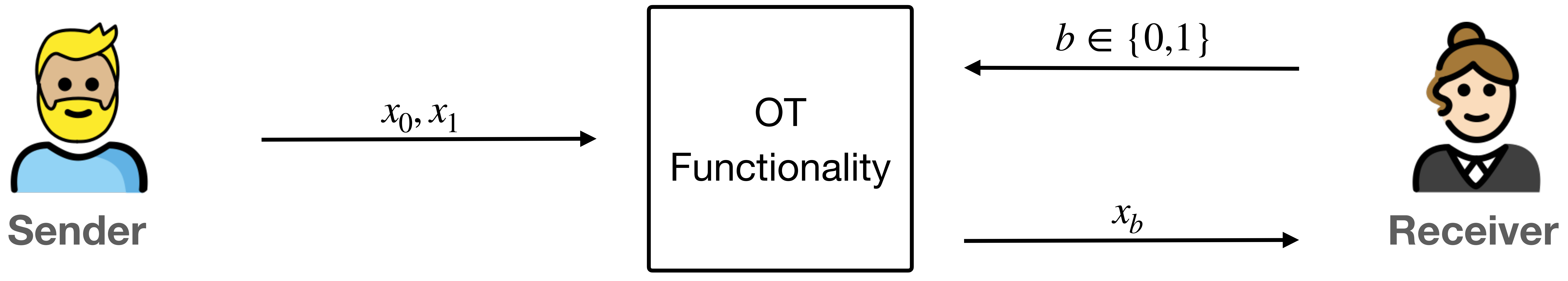
  Publish a collection $\{C_i\}_{n=0}^{t}$ of commitments to each share: $C_i = g^{a_i}h^{b_i}$, where $a_0 = s$

- $Verify(s_i, \{C_j\})$: Check share consistency $g^{s_i[1]} \cdot h^{s_i[2]} = \prod_{j=0}^{t} (C_j)^{i^j}$

- $Reconstruct(I, \{s_i\}_{i \in I}, \{C_i\}_{i \in i})$: For a subset $|I| = t$ do like in Shamir secret sharing to reconstruct $a(0) = s$ and $b(0) = b_0$ *[Lagrange interpolation with evaluation at 0]* and verify the consistency $g^s \cdot h^{b_0} = C_0$

Module 3: Agenda

Oblivious Transfer an Intuition



$$\mathscr{F}_{OT}(\{x_0, x_1\}, b) \mapsto (\emptyset, x_b)$$

Sender learns nothing
Recevier only learns Sender's b-th input

Oblivious Transfer Definitions

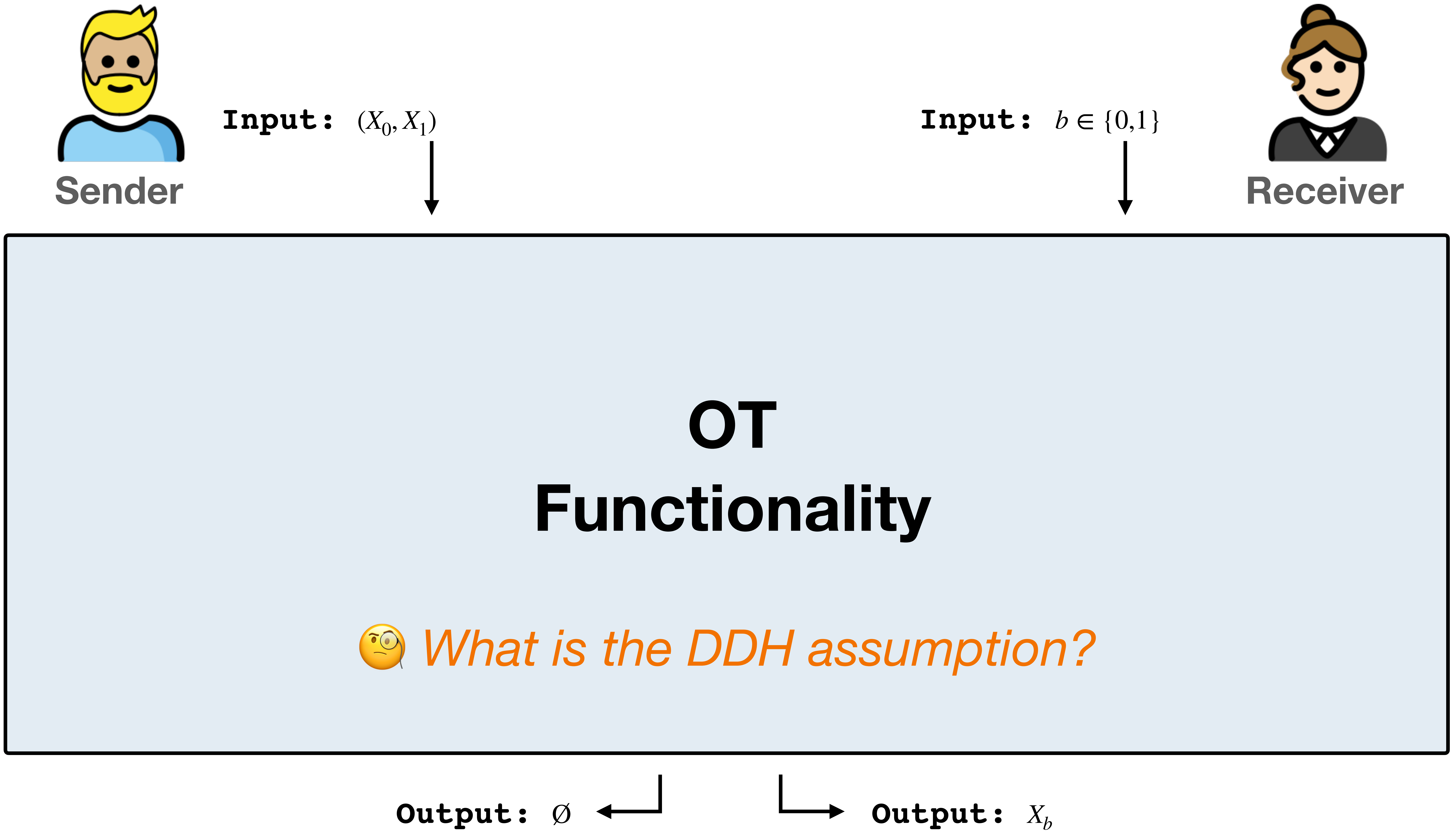
Syntax A 1-out-of-2 oblivious transfer (OT) is an interactive protocol between a Sender and a Receiver that realizes the following:

- Input: Sender knows (x_0, x_1); Receiver knows $b \in \{0,1\}$.
- Output: Sender knows (x_0, x_1); Receiver knows $b \in \{0,1\}$ **and** x_b.

… in particular:
the Receiver learns nothing about x_{1-b}, and the Sender learns nothing about b

👀 : one can build 1ooN OT by running N parallel instances of 1oo2 OT

Construction 1: OT Based on the DDH Assumption



🧐 *What is the DDH assumption?*

Construction 1: OT Based on the DDH Assumption

Auxiliary inputs: $(\mathbb{G}, g, q)$
(public parameters)

Sender

Receiver

Input: $(X_0, X_1) \in \mathbb{G}$

Input: $b \in \{0,1\}$

Receiver:

$pick \quad \alpha, \beta, \gamma \leftarrow \$\{1,\ldots,q\}$

$if\ b = 0\ set\ a = (g^\alpha, g^\beta, \boxed{g^{\alpha\beta}}, \boxed{g^\gamma})$

$if\ b = 1\ set\ a = (g^\alpha, g^\beta, \boxed{g^\gamma}, \boxed{g^{\alpha\beta}})$

$$\xleftarrow{a \in \mathbb{G}^4}$$

Sender:

Consistency checks:
$a = (a[0], a[1], z[0], z[1]) \in \mathbb{G}^4$
and $z[0] \neq z[1]$ (otherwise abort)

$pick \quad u_0, u_1, v_0, v_1 \leftarrow \$\{1,\ldots,q\}$

```
for  $d \in \{0,1\}$  do
    $w_d = a[0]^{u_d} \cdot g^{v_d}$
    $k_d = \boxed{z[d]^{u_d}} \cdot a[1]^{v_d}$
    $c_d = X_d \cdot k_d$
```

$$\xrightarrow{(w_0, c_0, w_1, c_1)}$$

Receiver:

$compute \quad k_b = w_b^\beta$

$compute \quad X_b = c_b \cdot (k_b)^{-1}$

inverse in $\mathbb{G}$

Output: $\varnothing$

Output: X_b

Construction 1: OT Based on the DDH Assumption

```
Consistency checks:
$a = (a[0], a[1], z[0], z[1]) \in \mathbb{G}^4$
and $z[0] \neq z[1]$ (otherwise abort)
```

$pick \;\; \alpha, \beta, \gamma \leftarrow \$\{1,\ldots,q\}$

$if \;\; b = 0 \;\; set \;\; a = (g^\alpha, g^\beta, \boxed{g^{\alpha\beta}}, \boxed{g^\gamma})$

$if \;\; b = 1 \;\; set \;\; a = (g^\alpha, g^\beta, \boxed{g^\gamma}, \boxed{g^{\alpha\beta}})$

$$\xleftarrow{\quad a \in \mathbb{G}^4 \quad}$$

$pick \;\; u_0, u_1, v_0, v_1 \leftarrow \$\{1,\ldots,q\}$

```
$for \;\; d \in \{0,1\} \;\; do$
    $w_d = a[0]^{u_d} \cdot g^{v_d}$
    $k_d = \boxed{z[d]^{u_d}} \cdot a[1]^{v_d}$
    $c_d = X_d \cdot k_d$
```

$$\xrightarrow{\quad (w_0, c_0, w_1, c_1) \quad}$$

$compute \;\; k_b = w_b^\beta$

$compute \;\; X_b = c_b \cdot (k_b)^{-1}$

Correctness (the Receiver gets the intended output)

Sender's Privacy (security against a malicious Receiver)

Receiver's Privacy (security against a malicious Sender)

whiteboard
proofs

Adversary: is static, malicious, PPT | **Privacy**: No party should learn anything other than its prescribed output