DATA PRIVACY AND SECURITY

Prof. Daniele Venturi

Master's Degree in Data Science
Sapienza University of Rome

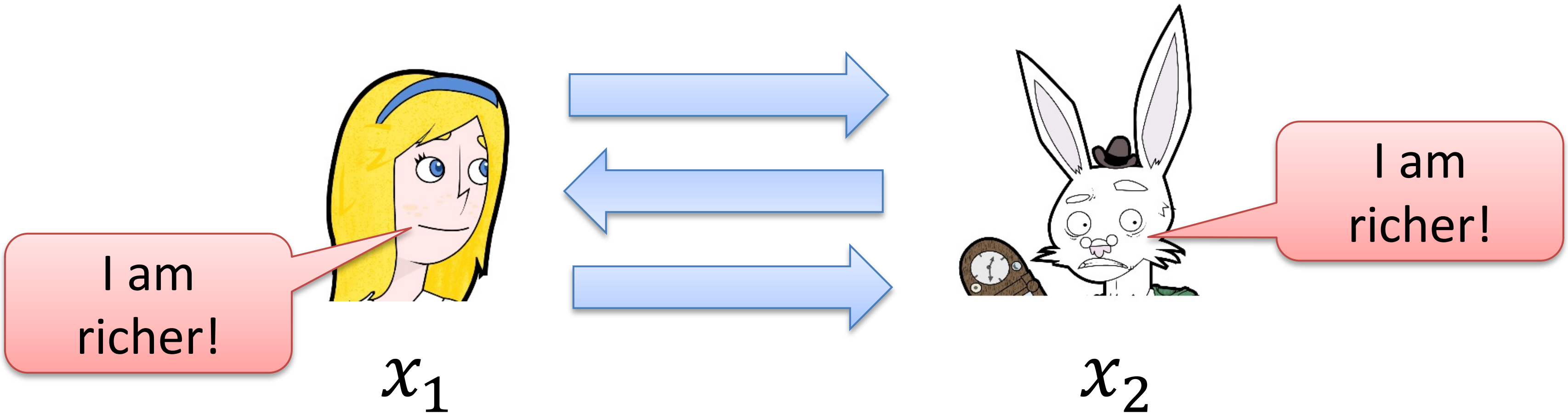
CIS SAPIENZA
RESEARCH CENTER FOR CYBER INTELLIGENCE AND INFORMATION SECURITY

<u>CHAPTER 8:</u> Multi-Party Computation

MPC Protocols

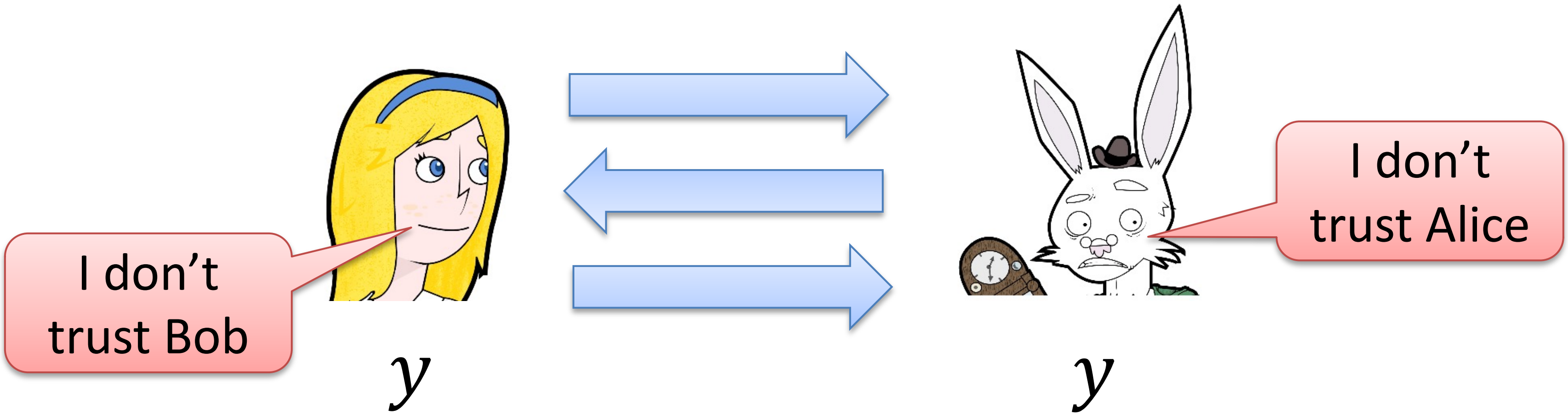
- Multi-Party Computation (MPC): Protocols where the players do not trust each other
- Yet they want to achieve a **common goal**
  - Typically, expressed as a function on the parties' **secret inputs** (say # of players $= n$)

Example: The Millionaires' Problem



$$f(x_1, x_2) = \begin{cases} 1 \text{ if } x_1 > x_2 \\ 0 \text{ if } x_2 \geq x_1 \end{cases}$$

Example: Coin Tossing

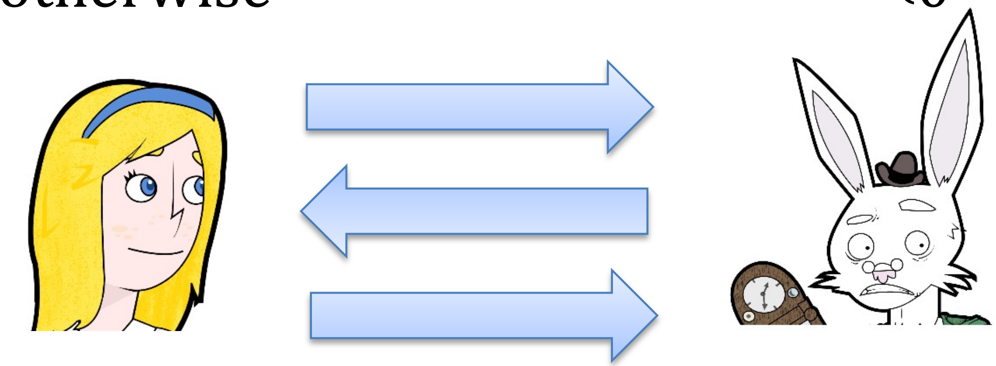


$$y = \begin{cases} 0 \text{ w.p. } 1/2 \\ 1 \text{ w.p. } 1/2 \end{cases}$$

Example: Secure Dating

$$x_1 = \begin{cases} 1 \text{ if Alice loves Bob} \\ 0 \qquad\qquad \text{otherwise} \end{cases}$$

$$x_2 = \begin{cases} 1 \text{ if Bob loves Alice} \\ 0 \qquad\qquad \text{otherwise} \end{cases}$$



y $\qquad$ y

$$y = x_1 \cdot x_2$$

Possible Applications

- Cloud computing
- Digital auctions
- Online gambling (poker)
- Electronic voting
- …

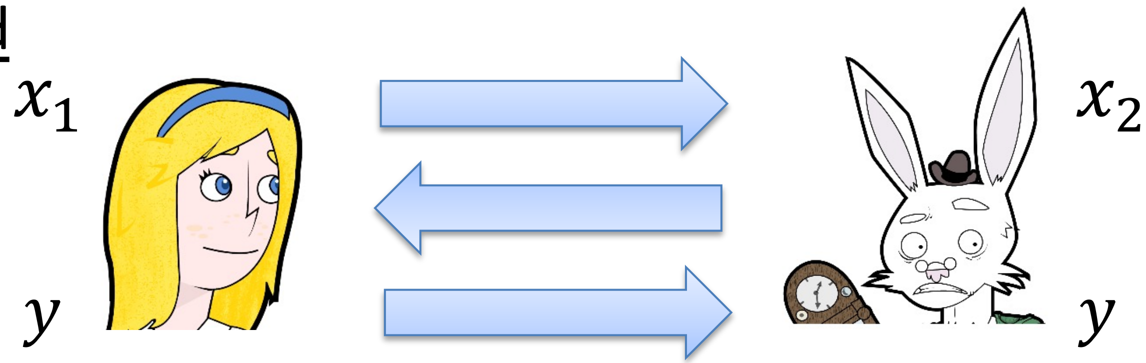
But do such protocols exist?

Ideal and Real World

- Trivial assuming a **trusted third party**

Ideal World

Real World



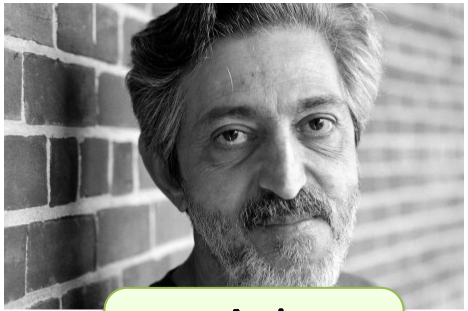
Every Function can be Computed Securely


Manuel Blum


Andrew Yao


Silvio Micali


Oded Goldreich


Avi Widgerson

Every trusted party can be **"simulated"** in a secure manner (under some assumptions)

The Age of Optimism

80s 90s 00s 10s 20s

PKE
MPC — Invented

PKE — Practical

PKE — Ubiquitous

MPC — Feasible

MPC — Practical

MPC — Ubiquitous

Security Requirements (1/4)

- Consider a secure auction with secret bids
- Attacker may wish to learn the bids
  - Require **privacy** of inputs
- Attacker may wish to win using a bid lower than the highest
  - Require **correctness** of the output

Security Requirements (2/4)

- Attacker may wish to ensure his bid is always the highest
  - Require **independence of inputs**
- Attacker may wish to abort the protocol if he is not the winner
  - Require **fairness**

Security Requirements (3/4)

- **Privacy:** Only the output is revealed
- **Correctness:** The desidered function is computed correctly
- **Independence of inputs:** Parties can't choose inputs based on other parties' inputs

Security Requirements (4/4)

- **Fairness:** If one party receives the output, all parties receive the output
- **Guaranteed output delivery:** Corrupted parties can't prevent honest parties to receive the output

Defining Security (1/2)

- First option: Define **specific** properties for **each** scenario
  - Auctions: As in previous slide
  - Elections: Only privacy, correctness and fairness
- Problem:
  - How do we know **all possible concerns** are covered?
  - Definitions are application dependent and need to be redefined from scratch for each task

Defining Security (2/2)

- Second option: Have a **general definition** that works for **all possible** scenarios
  - Need well-defined adversarial model and execution setting
  - Security guarantees are **simple** to understand

On the Power of the Adversary

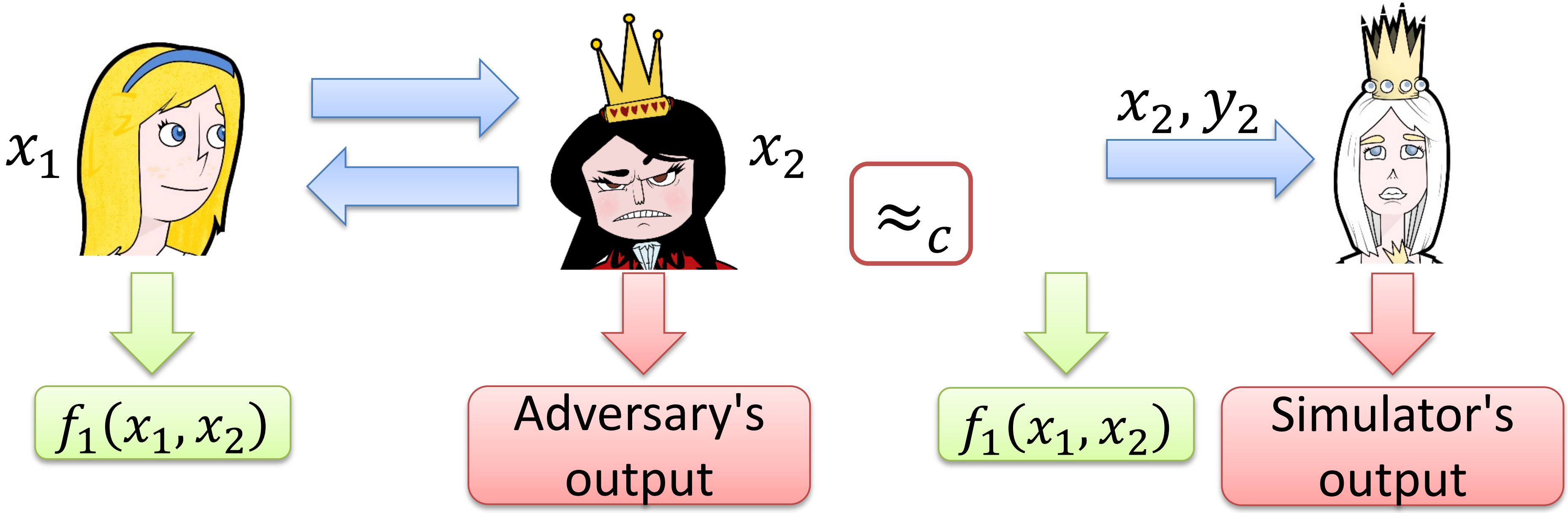
- The adversary can **corrupt** a subset of players
  - Threshold adversary: Corrupts $t < n$ players
  - Monolithic adversary: **Single adversary** corrupting all parties
- Semi-honest vs. malicious
  - Semi-honest: Follows the protocol
  - Malicious: Behaves **arbitrarily**
- Non-adaptive vs. adaptive
  - Non-adaptive: Identity of corrupted parties **fixed before the protocol starts**

Execution Setting

- Standalone execution
  - Consider only a **single** execution
  - Allows for **sequential composition**
- Concurrent and universal composition
  - **Concurrent:** Different instances of the same protocol are run concurrently
  - **Universal:** Arbitrary protocols are executed concurrently
- Universal composability is the true goal
  - Allows for **arbitrary composition**

Security by Simulation

$$f(x_1, x_2) = \left(f_1(x_1, x_2), f_2(x_1, x_2)\right) = (y_1, y_2)$$



- Given input and output **can generate the adversary's view**
- Inputs are well defined (**semi-honest** case)

Properties

- Correctness, independence of inputs, fairness **not** a concern in the semi-honest model
- What about **privacy**?
  - The attacker's view can be generated given only the input and output
  - So whatever the adversary has learned he could have also learned by talking to the simulator, which **does not know** the honest party's input
  - Without even running the protocol!

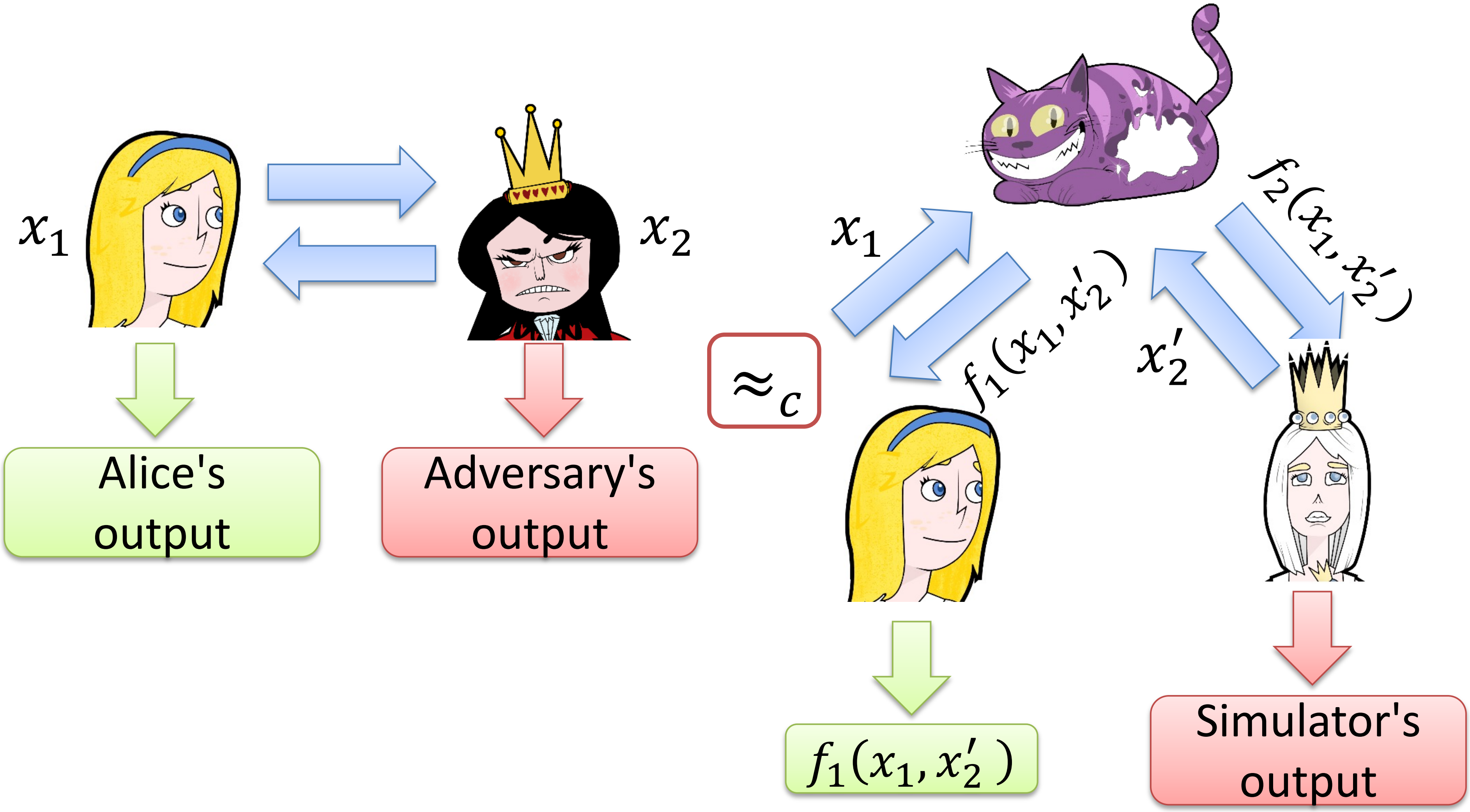
Malicious Adversaries

- First attempt: Require the existence of a simulator as before
  - The simulator should simulate the attacker's view given the input/output for the malicious party
- Problem: What is the input used by the adversary?
  - In fact, the input **might not even exist!**
- Moreover, independence of inputs, correctness, and fairness are **not implied** by the ability to simulate the adversary's view

Trusted Third Parties

- Best option: An **incorruptible** trusted party
  - All players send their inputs to the trusted party
  - The trusted party computes the outputs and gives them to the players
  - In this sense, this is an **ideal world**
- What can the adversary do?
  - Only **change its input**
- Security now says that an execution of the real protocol should be like in the ideal world

The Real/Ideal Paradigm



Properties

- All properties are satisfied in the ideal world
  - **Privacy:** As before
  - **Correctness:** Because honest parties get the correct output
  - **Independence of inputs:** Because the simulator does not know the honest party's input
  - **Fairness:** Because the honest party always receives the output
  - **Guaranteed output delivery:** Same as fairness

Sequential Composition

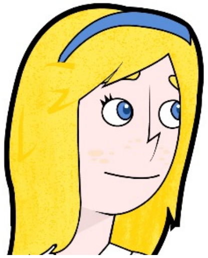
- Secure protocols run **sequentially**, with arbitrary messages in between
- Why is this interesting?
  – Helpful tool for analyzing security of protocols
- Formalization: The **hybrid model**
  – Replace each protocol with the corresponding ideal functionality
  – Real messages (exchanged by the parties)
  – Ideal messages (sent to the ideal functionalities)

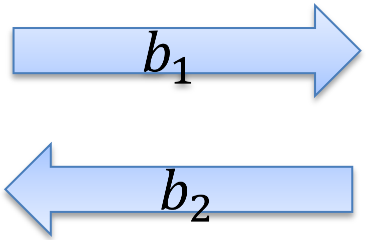
Universal Composability

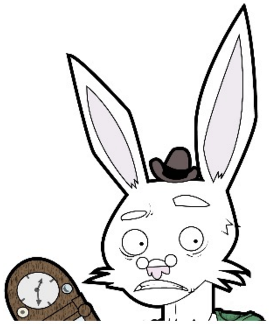
- Sequential composition does not model settings (like, e.g., the Internet) where protocols are **run concurrently**
  - With different instances of the same protocol and other protocols
- **Universal composability** captures this
  - R. Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". 2001

Coin Tossing

How to Realize Coin Tossing?



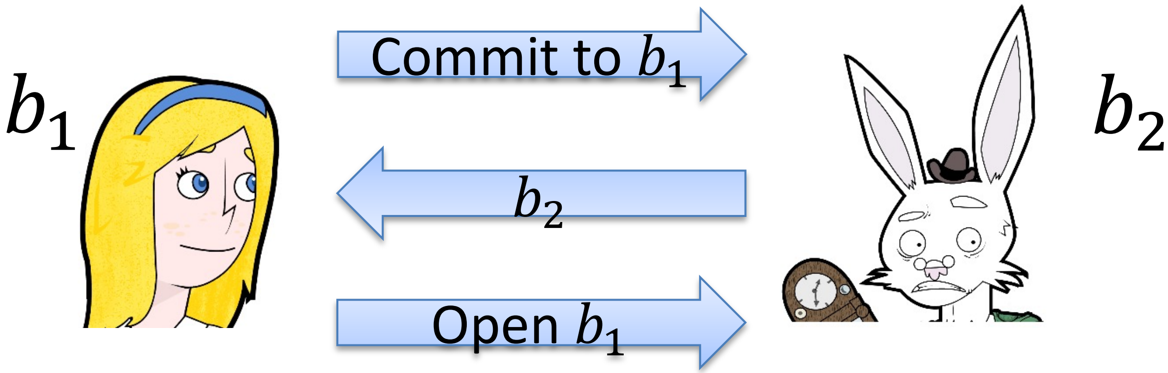




$y = b_1 \oplus b_2$ $\quad\quad$ $y = b_1 \oplus b_2$

- But the bits should be sent at the **same time**
  - Otherwise parties can easily cheat
  - Seems hard to realize this in the internet

Solution Using Bit Commitments



$y = b_1 \oplus b_2$ $\qquad$ $y = b_1 \oplus b_2$

- Digital commitment satisfies two properties
  - **Binding:** Alice cannot commit to b and later open the commitment to $b' \neq b$
  - **Hiding:** The commitment hides b

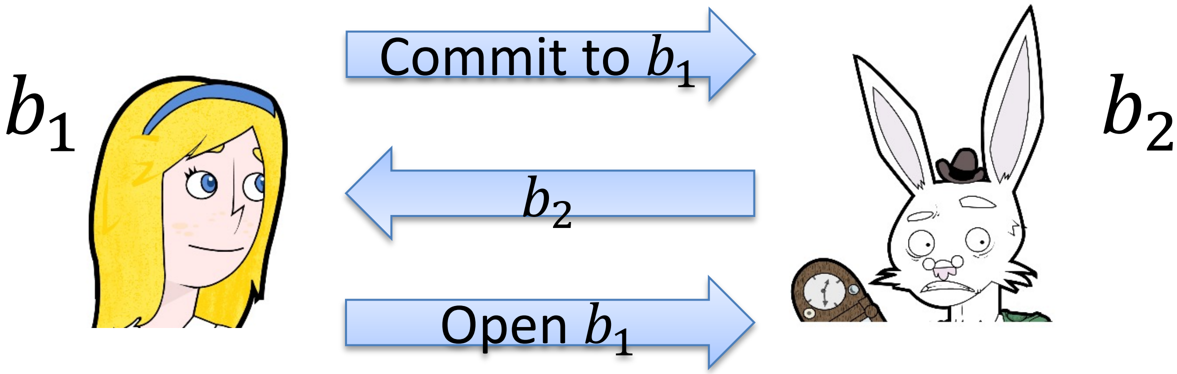
Hash-Based Commitments

- Hash function $\mathbf{H}$ (modeled as **random oracle**)
  - In practice, could be SHA-256
- To commit to $b \in \{0,1\}$, pick random $r \in \{0,1\}^k$ and output $\mathbf{H}(b||r)$
- To open b, send (b, r)
  - **Hiding:** The function's outputs look random
  - **Binding:** Finding $(0, r_0)$ and $(1, r_1)$ such that $\mathbf{H}(0||r_0) = \mathbf{H}(0||r_1)$ is hard

The Limitations

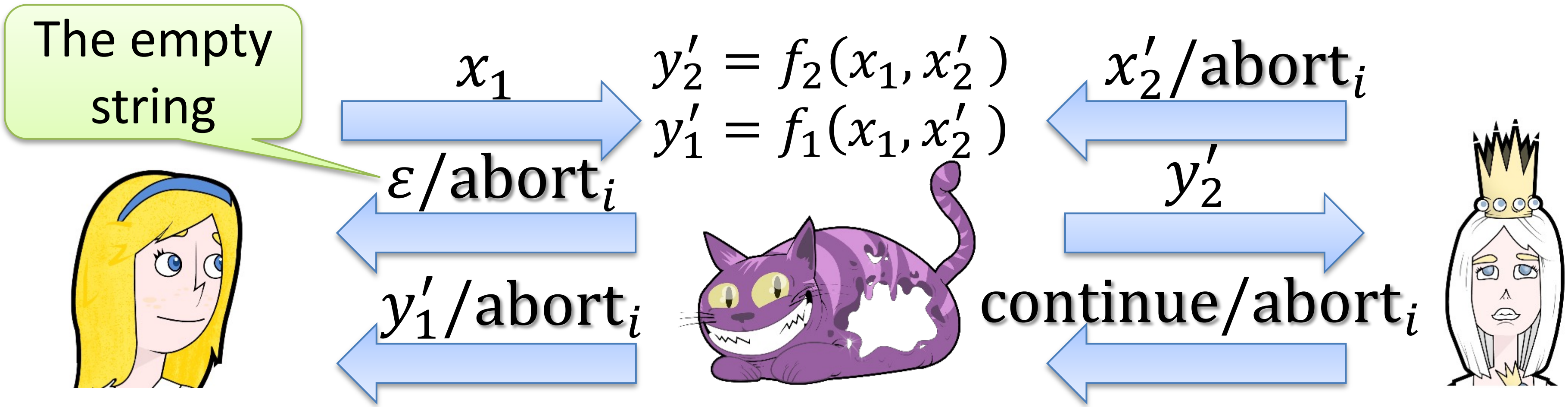
- **Lack of fairness** when there is **no honest majority** (see following slides)
  - Partial remedies exist
- No way to force parties to use **true inputs** and to **respect the outcome**
- We can deal with these problems using Bitcoin!
  - M. Andrychowicz, S. Dziembowski, D. Malinowski, L. Mazurek. "Secure Multiparty Computations on Bitcoin." 2014

Problem 1



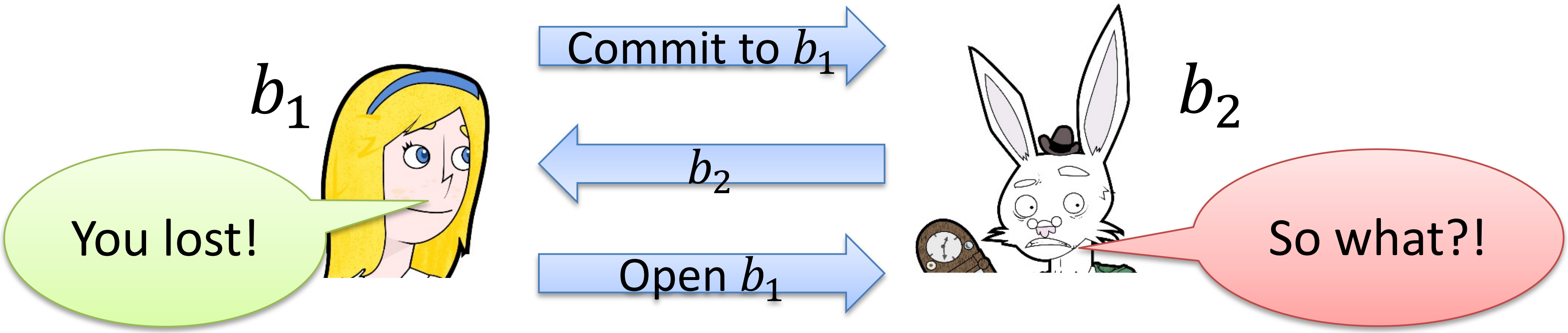
- Lack of fairness
  - Alice can refuse to open the commitment
- **Inherent issue** in most of the interesting MPC protocols

Security with Aborts



- The simulator **can abort** either at the beginning, or after seeing the output (before the honest party)
- This yields a weaker notion known as **security with aborts**

Problem 2

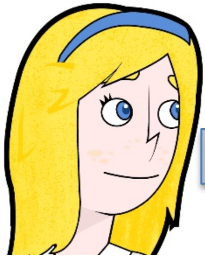


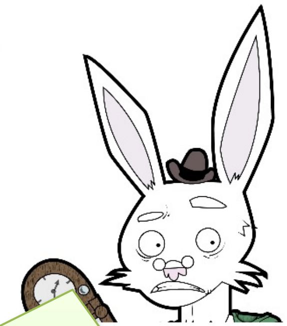
- This is the problem of forcing the parties to **respect the output**
- **Inherent** even in the ideal world specification

Main Idea

Commit to b_1

Commits to bit b_1





b_2

Transaction "commit":

- Has value 1 BTC
- Can be redeemed by Alice
- Claiming the transaction requires revealing b_1

If Alice **didn’t redeem** "commit", I can do it after one day!

CIS SAPIENZA
RESEARCH CENTER FOR CYBER INTELLIGENCE AND INFORMATION SECURITY

How to do it?

- Using the Bitcoin scripting language
- **Hash-locked** transactions
  - Let $\mathbf{H}$ be a hash function and $Y = \mathbf{H}(X)$
  - A Y-hash-locked transaction can be redeemed only by publishing X (in our case $X = (b_1, r)$)

$T_2 =$

T_1	1 BTC	Can be spent using Bob's signature and X such that $Y = \mathbf{H}(X)$	Alice's signature

Alice's Commitment

"commit" = | T | 1 BTC | Can be spent using Alice's signature and (b_1, r) such that $Y = \mathbf{H}(b_1, r)$ or using both Alice's and Bob's signatures | Alice's signature |

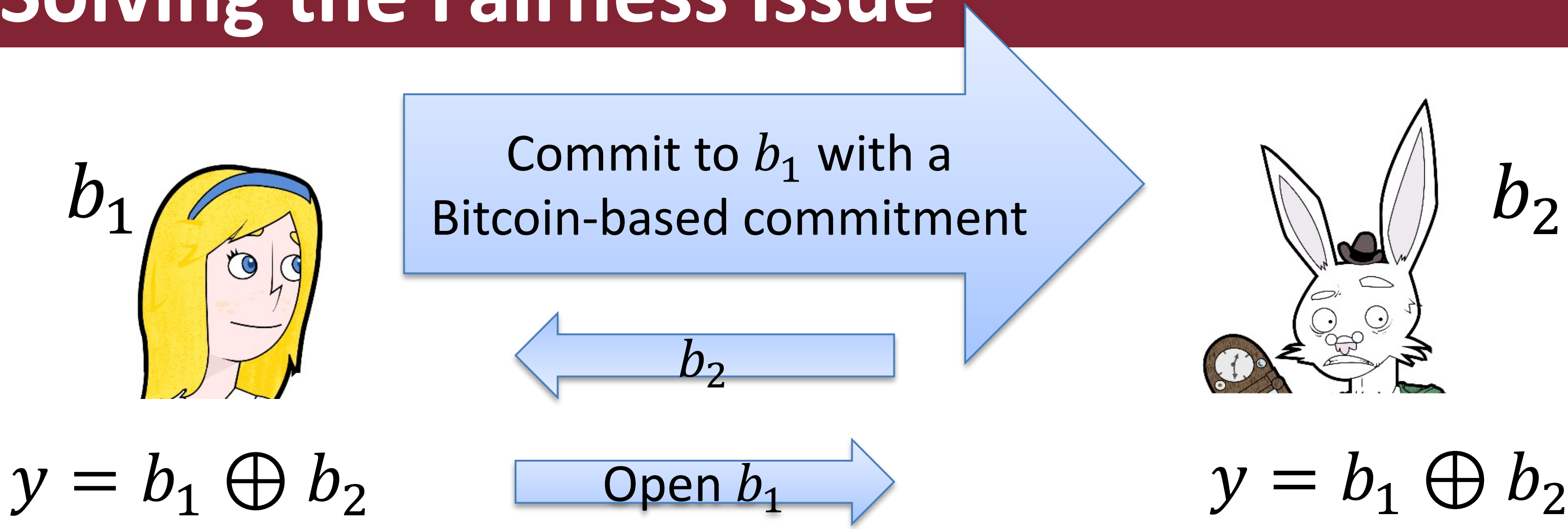
Posted on the blockchakin

Earlier transaction of Alice

Send to Bob a "refund" transaction

"refund" = | "commit" | 1 BTC | Can be spent using Bob's signature <u>after one day</u> | Alice's signature |

Solving the Fairness Issue



- If Alice **does not open** the commitment within one day, Bob can get 1BTC by posting the "refund" transaction
- Otherwise Alice gets her 1 BTC back

A Commitment Contract in Ethereum

```
contract Commitment{
    bytes32 commitment;
    uint timeout;
    address owner;
    function commit(bytes32 c) payable {
        hash = h;
        timeout = now + 10 minutes;
        owner = msg.sender;
    }
    function open (uint d) {
      if (sha3(d) == commitment)
           selfdestruct(msg.sender);
    }
    function refund (){
        if (timeout < now)
        self-destruct(owner);
    }
}
```

1. Challenger deposits coins

2. Solver opens commitment

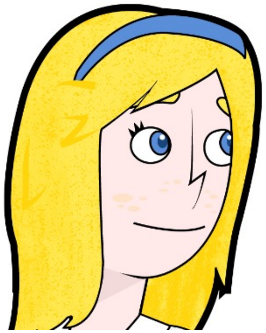
3. Refund coins

Final Result

- **Any two-party stateless** functionality can be simulated in this way
- The simulation enforces **financial consequences**
- Generalization to **multi-party reactive** functionalities by Kumaresan, Moran, Bentov
- Example: Selling secret information
  - Set union plus a money transfer between Alice and Bob for each new element that they learned

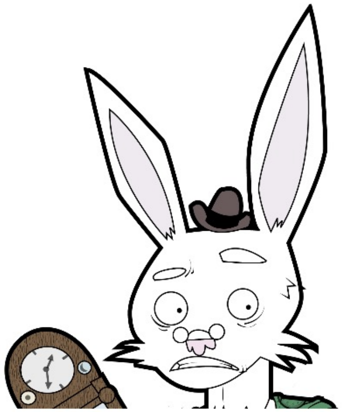
Zero Knowledge

Motivating Example: ID Schemes



Random message m ←

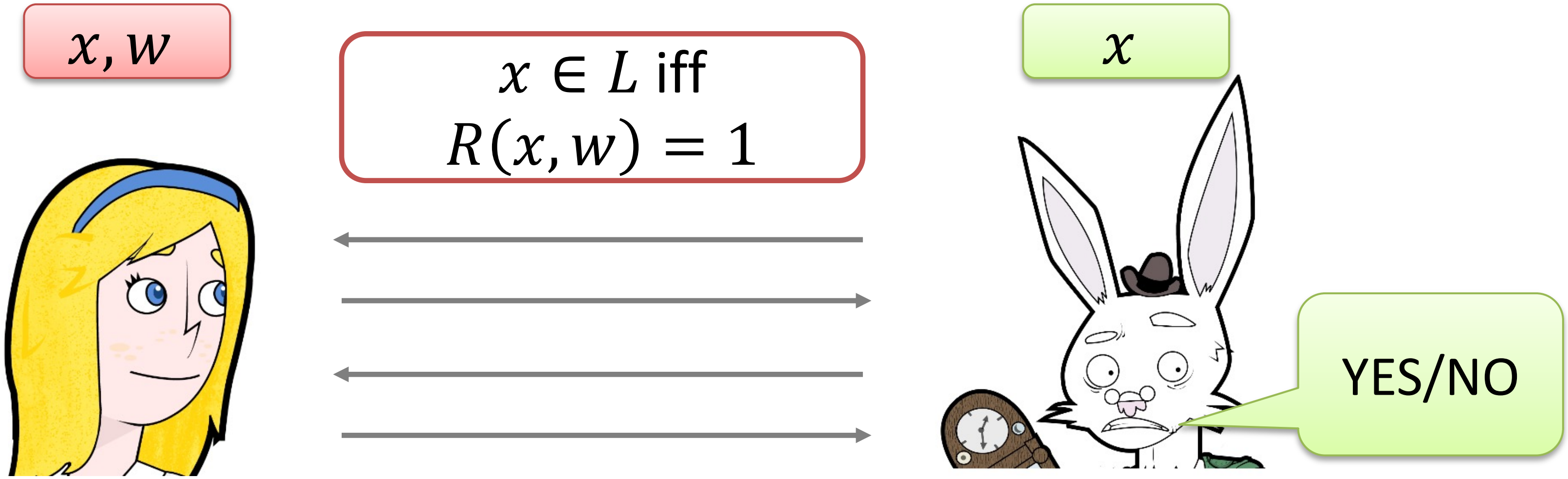
Signature on m →



pk, sk pk

- Protocol is **not deniable**: Signature is a **proof** that someone has talked to the prover
- Can we have a protocol where the verifier does not learn **anything**?

Interactive Proofs

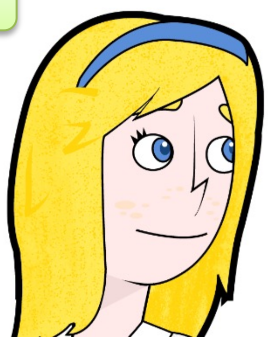


- **Completeness:** Honest prover always convinces the verifier
- **Soundness:** No malicious prover can convince the verifier in case $x \notin L$

The Schnorr Protocol

$L = \{x = g^w : w \in \mathbb{Z}_q\}$

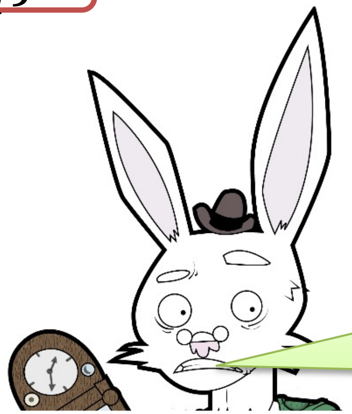
x, w



$\alpha = g^a$ →

← $\beta \leftarrow_\$ \mathbb{Z}_q$

$\gamma = \beta \cdot w + a$ →



x

YES iff $g^\gamma = \alpha \cdot x^\beta$

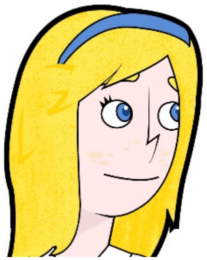
- **Completeness:** $g^\gamma = g^{\beta \cdot w + a} = g^a \cdot (g^w)^\beta$
- **Soundness:** Follows from the DL assumption
- **Honest-Verifier Zero-Knowledge:** Pick random β, γ such that $\alpha = g^\gamma \cdot x^{-\beta}$

What Can be Proven in Zero Knowledge?

- Assuming OWFs exist **every language in NP!**
  - O. Goldreich, S. Micali, A. Widgerson. "Proofs that yield nothing but their validity." 1986
- The above is achieved by showing a zero-knowledge proof for an **NP-complete** language
  - E.g., 3-coloring or graph Hamiltonicity

Zero Knowledge from FHE

$(pk, sk) \leftarrow_\$ \mathbf{K}(1^\lambda)$
$\vec{c} \leftarrow_\$ \mathbf{E}(pk, \vec{w})$
$d = \mathbf{D}(sk, c')$

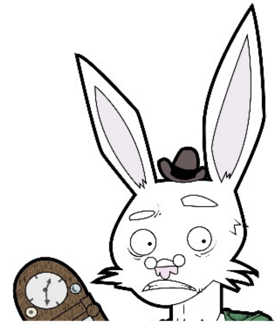


x, w

$pk, \vec{c}$ →

← c'

d →



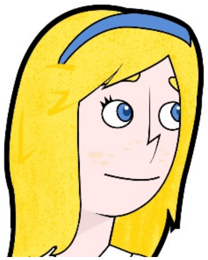
$x \in L$

$c' \leftarrow_\$ \mathbf{C}(pk, f_{R,x}, \vec{c})$

- Let $L \in NP$ with relation R
  - Consider the circuit $f_{R,x}(w) = R(x, w)$
- The above protocol is **not sound!**
  - Can you say why?

Adding Soundness (1/2)

$(pk, sk) \leftarrow_\$ \mathbf{K}(1^\lambda)$
$\vec{c} \leftarrow_\$ \mathbf{E}(pk, \vec{w})$
$d = \mathbf{D}(sk, c')$

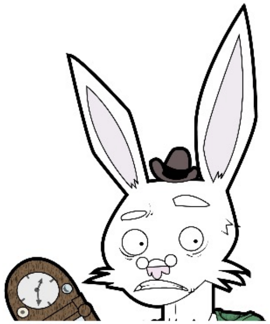


x, w

$pk, \vec{c}$ →

← c'

d →



$x \in L$

$\beta \leftarrow_\$ \{0,1\}$

$$c' \leftarrow_\$ \begin{cases} \mathbf{C}(pk, f_{R,x}, \vec{c}) & \text{if } \beta = 1 \\ \mathbf{E}(pk, 0) & \text{if } \beta = 0 \end{cases}$$

Check $\beta = d$

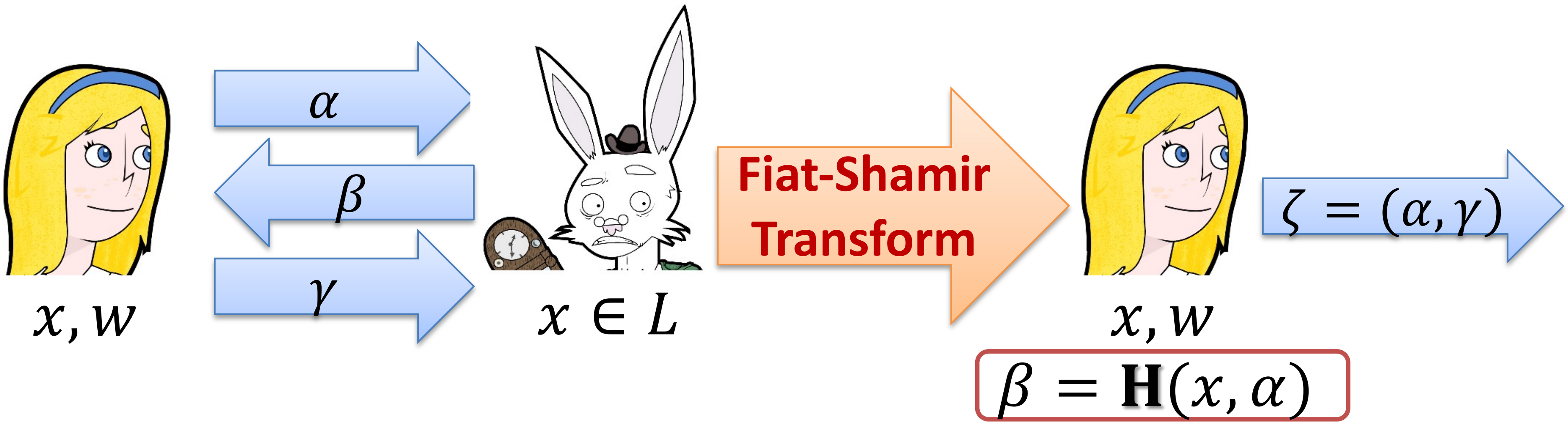
Adding Soundness (2/2)

- Soundness follows by the fact that, for $x \notin L$, both ciphertexts will be **encryptions of zero**
  - Thus, Alice can cheat with probability 1/2
- However, we need to ensure that $pk, \vec{c}$ are **well formed**
  - Alice generates pk_1, pk_2 and Bob asks her to "open" one at random
  - With the other key Alice encrypts $\vec{w}_1, \vec{w}_2$ s.t. $\vec{w}_1 \oplus \vec{w}_2 = \vec{w}$, and Bob asks her to "open" one of the encryptions at random

Adding Zero Knowledge

- The previous protocol is only **honest-verifier zero-knowledge**
  – In fact, malicious Bob could send to Alice the first ciphertext in the vector $\vec{c}$, so that d reveals the first bit of w
- This can be fixed using **commitments**
  – Namely, Alice sends a commitment to d
  – Hence, Bob must reveal his randomness in order to prove he run the computation as needed
  – Finally, Alice opens the commitment revealing d

The Fiat-Shamir Transformation



- **Non-Interactive zero knowledge**
  - The proof now consists of a **single message**
- Security relies on the assumption that hash function $\mathbf{H}$ behaves as a **random oracle**

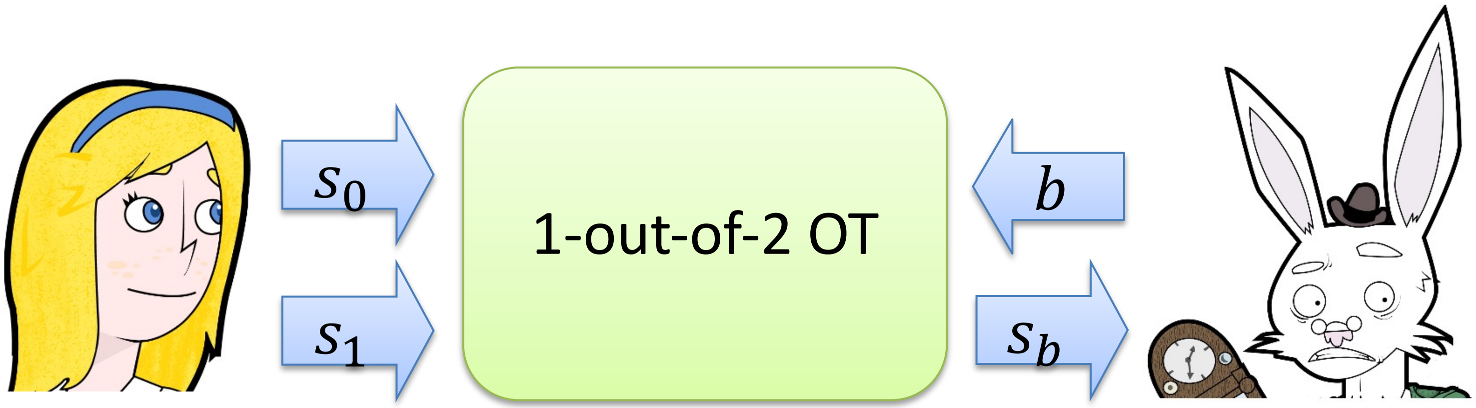
Applications

- Suppose $m = m_1||m_2$ is signed by Bob with $\sigma = \mathbf{S}(sk, m)$ and Alice wants to reveal to Carol m_2 while keeping m_1, σ **secret**
  - $L = \{m_2 : \exists m_1, \sigma \text{ s.t. } \mathbf{V}(pk, m_1||m_2, \sigma) = 1\}$
- Alice holds an ID card signed by some authority and wants to prove she is 18 **without revealing her age**
- Ubiquitous primitive in advanced cryptographic constructions

Oblivious Transfer

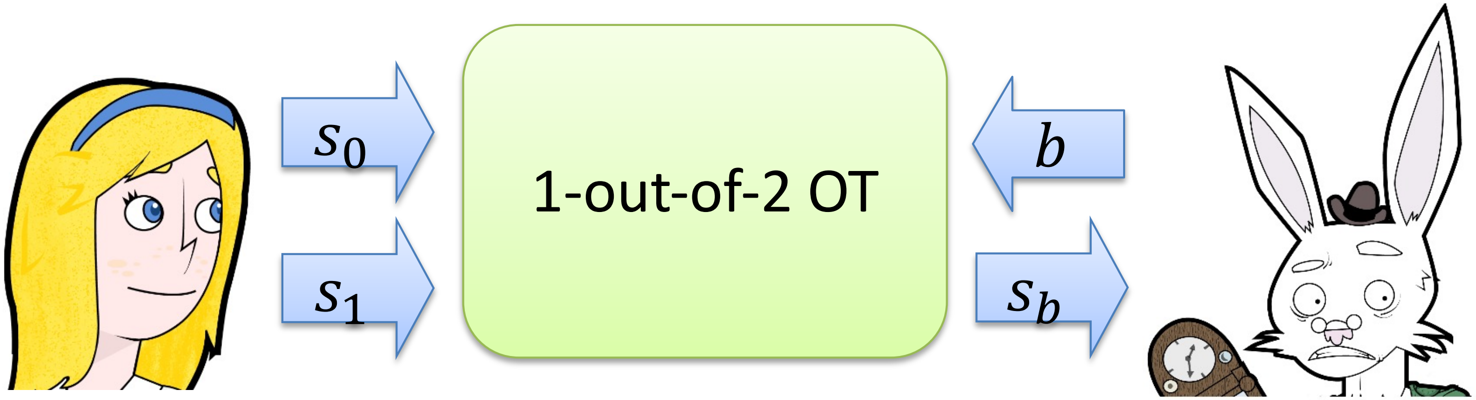
Oblivious Transfer

- Introduced by Rabin in 1981



- Properties
  - Sender **learns nothing** about b
  - Receiver **learns nothing** about s_{1-b}

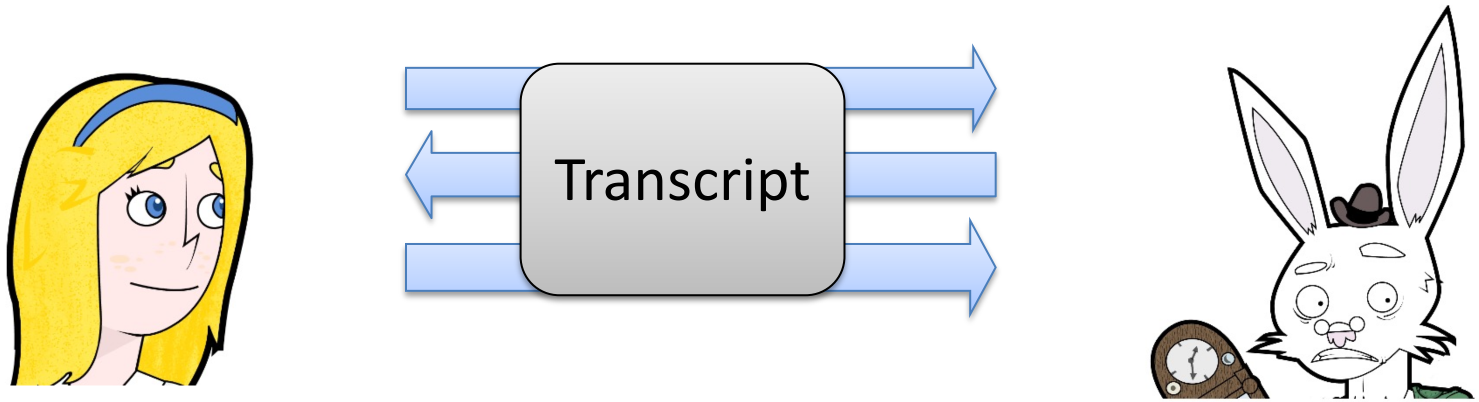
Why is it Useful?



$(s_0, s_1) = (0, b')$

- Bob's output is 1 iff $b = b' = 1$ (so it is equivalent to computing $b \cdot b'$)
- **Impossible** to compute AND with **information theoretic** security (even for **passive** security)

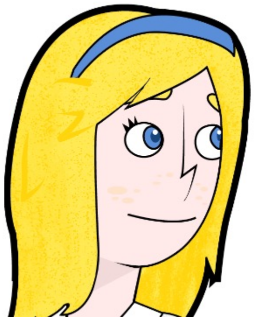
Protocol Transcript

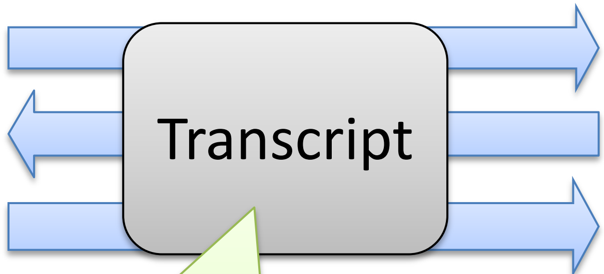


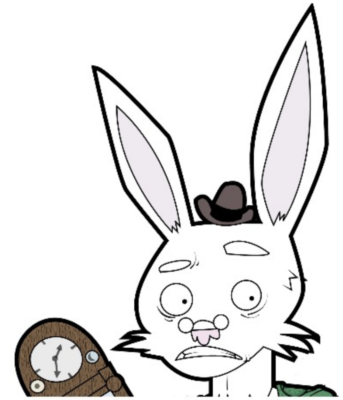
x_1, r_1 x_2, r_2

- Transcript T is **consistent** with x_1 if **there exist** values r_1 and (x_2, r_2) such that T is a transcript of the protocol with inputs
  – (x_1, r_1) for Alice
  – (x_2, r_2) for Bob

Suppose $x_1 = 0$ and $x_2 = 0$





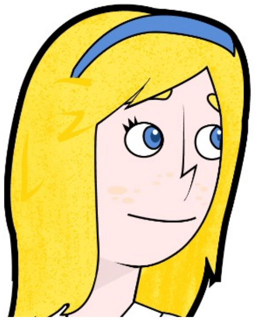


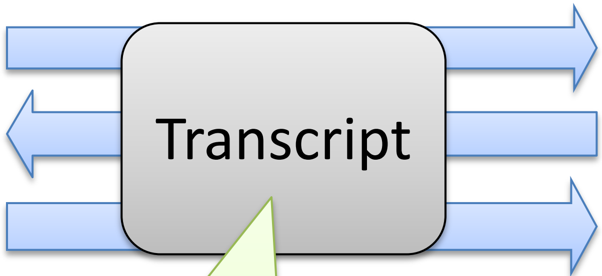
$x_1 = 0, r_1$

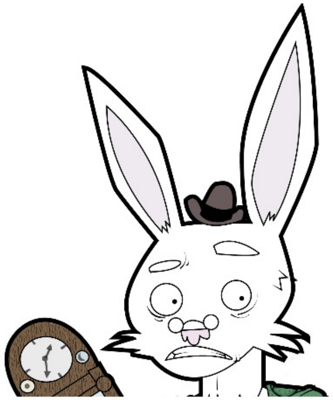
Has to be consistent with $x_1 = 1$, otherwise malicious Bob can learn x_1

$x_2 = 0, r_2$

Suppose $x_1 = 0$ and $x_2 = 1$







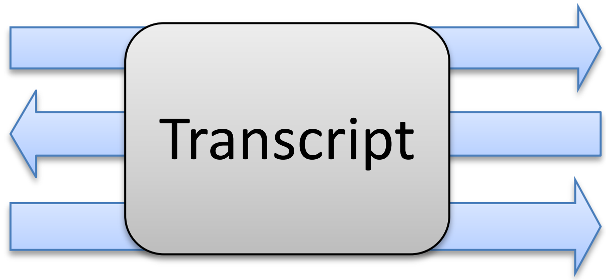
$x_1 = 0, r_1$

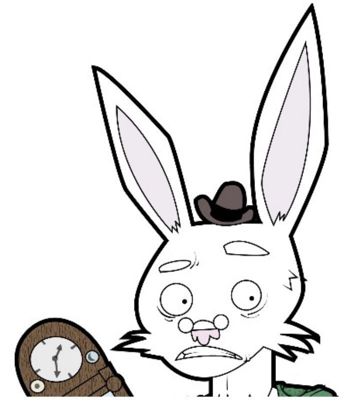
$x_2 = 1, r_2$

Cannot be consistent with $x_1 = 1$, because the output of the protocol has to be different in the following cases

- $x_1 = 0, x_2 = 1$
- $x_1 = 1, x_2 = 1$

The Attacker





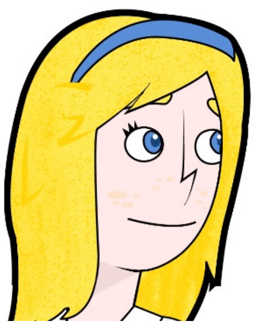
$x_1 = 0, r_1$

x_2, r_2

- Check if T is **consistent** with $x_1 = 1$
  - If it is, $x_2 = 0$
  - Else, $x_2 = 1$
- **Corollary:** Any secure protocol for AND must rely on **computational assumptions**

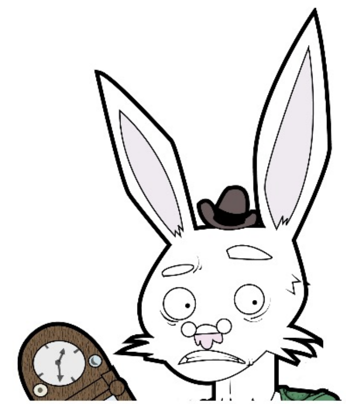
OT with Passive Security

- Recall the Elgamal PKE
  - Ciphertext is $c = (g^r, h^r \cdot m)$ for $h = g^x$
  - **Oblivious** key generation: Can generate h **without knowing** the secret key x



(h_0, h_1) ←

(c_0, c_1) →



s_0, s_1

$c_0 = (g^{r_0}, h_0^{r_0} \cdot s_0)$

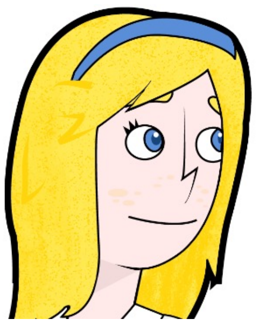
$c_1 = (g^{r_1}, h_1^{r_1} \cdot s_1)$

$(h_b = g^x, x), h_{1-b}$

Decrypt c_b using x

OT with Active Security

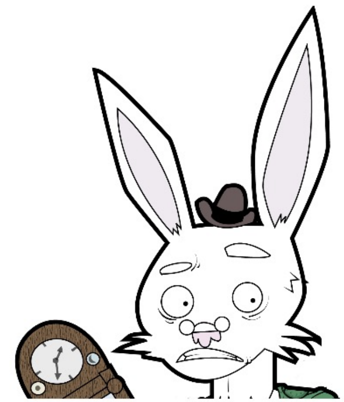
- Let $(\mathbf{K}, \mathbf{E}, \mathbf{D})$ be a PKE and $(\mathbf{E}', \mathbf{D}')$ be an SKE



(pk_0, pk_1) →

← $c \leftarrow_\$ \mathbf{E}(pk_b, k)$

$(c_0' = \mathbf{E}'(k_0, s_0), c_1' = \mathbf{E}'(k_1, s_1))$ →



$(pk_0, sk_0) \leftarrow_\$ \mathbf{K}$

$(pk_1, sk_1) \leftarrow_\$ \mathbf{K}$

$k_0 = \mathbf{D}(sk_0, c)$

$k_1 = \mathbf{D}(sk_1, c)$

Random k

$s_b = \mathbf{D}'(k, c_b')$

	$\mathbf{b = 0}$	$\mathbf{b = 1}$
k_0	k	$$
k_1	$$	k

Oblivious Transfer for Strings

- What if the sender inputs (s_0, s_1) consist of a **sequence** of strings $s_b = \left(s_b^1, \ldots, s_b^t\right)$?
- **Passive case:** Just apply basic OT to each (s_0^j, s_1^j) separately (with the same b)
- **Active case:** It's more complicated
  - But a generic construction also exists

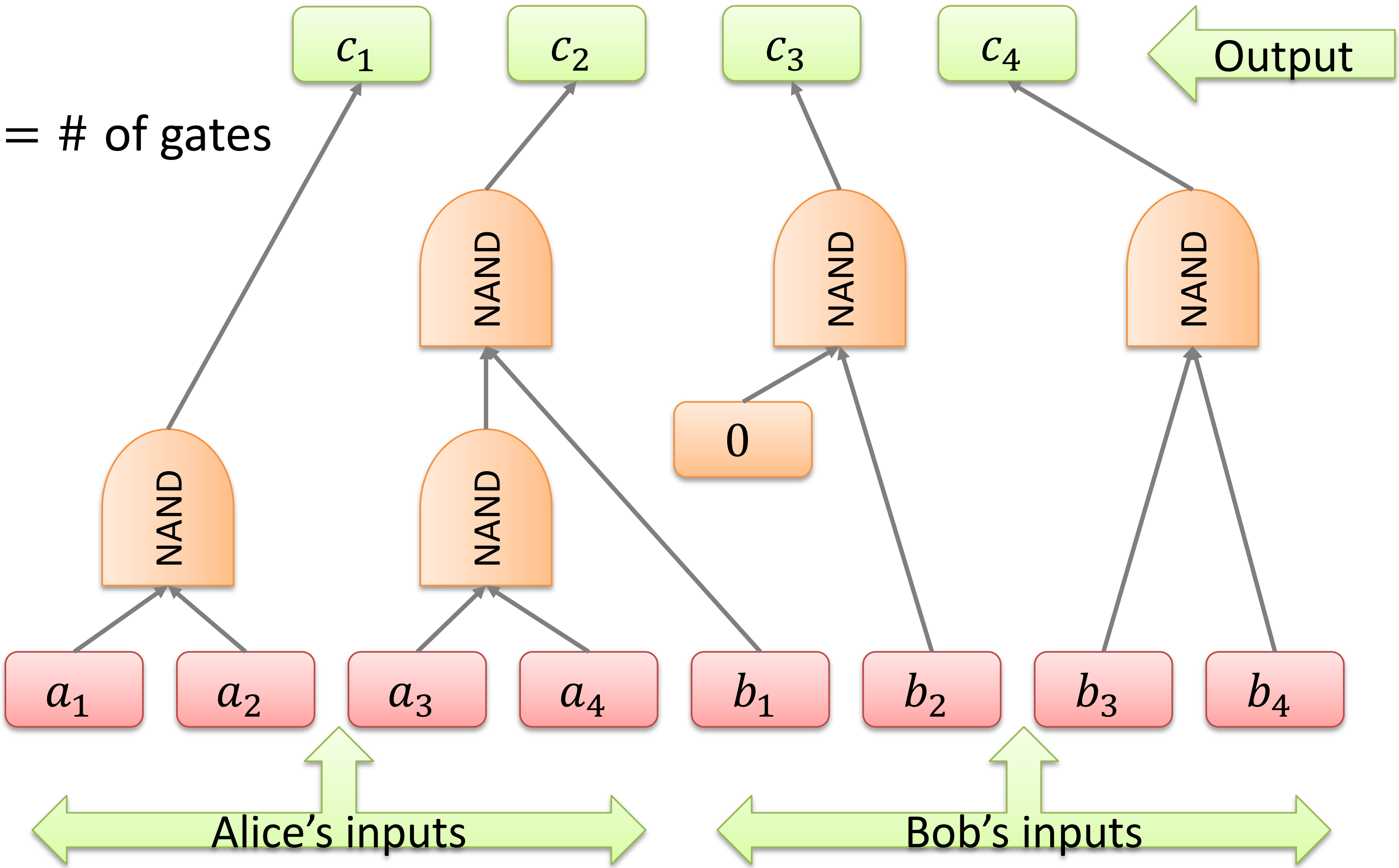
Garbled Circuits

Protocols for Arbitrary Functions

- We now show how Alice and Bob can compute **any function** securely
  - I.e., a general solution for the problem of secure two-party computation
  - We start with the simpler case of **passive security**
  - Also assume only **one party gets the output** (we will see how to generalize it later)
- Main idea: Represent the function as a Boolean circuit
  - Recall: NAND gate is complete

Boolean Circuits

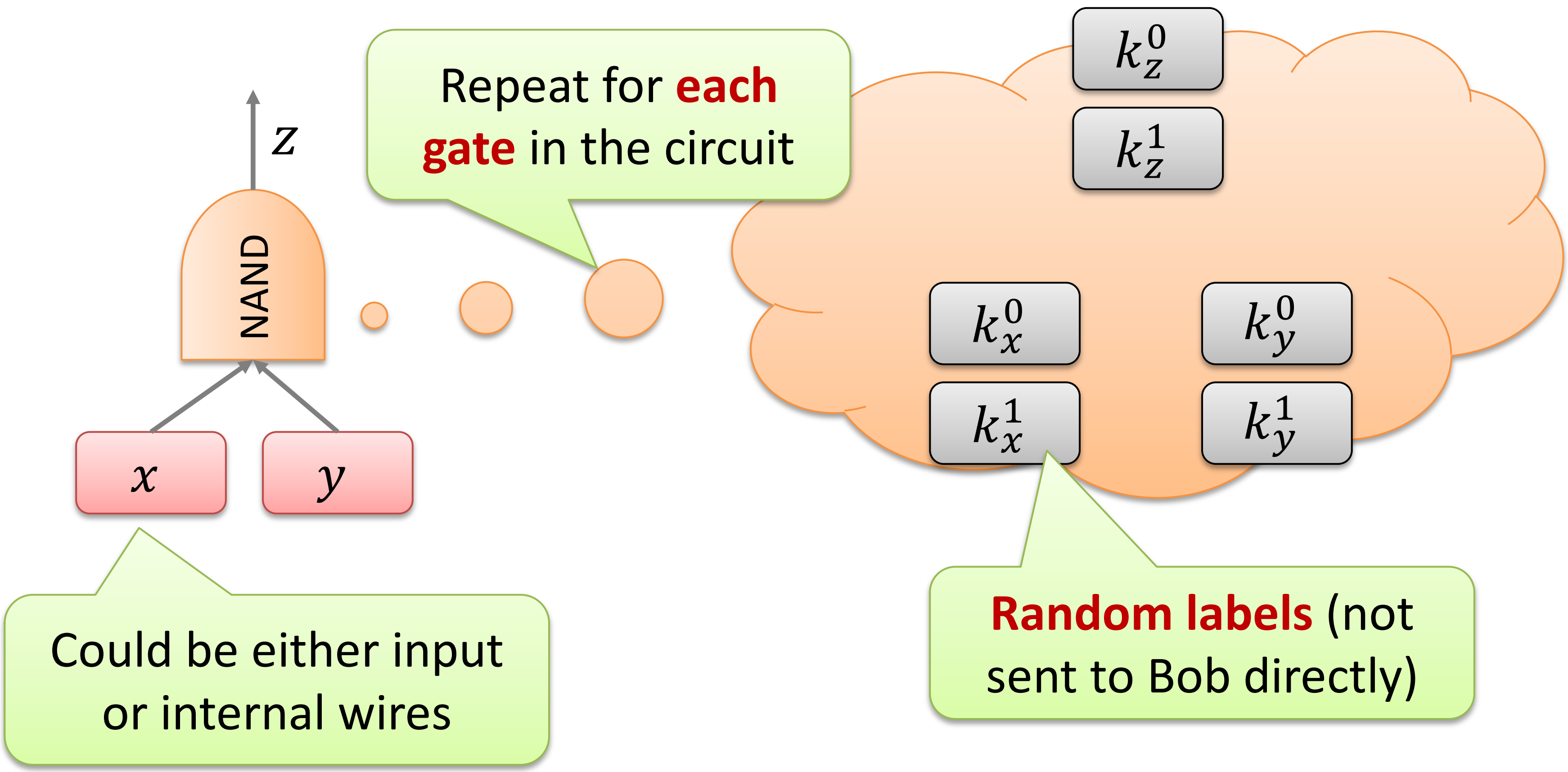
Size = # of gates



High-Level Idea

- Alice **encrypts (garbles)** the circuit together with her input and sends it to Bob
- Bob adds its own input and evaluates the encrypted circuit **gate by gate**
- The above must be done in such a way that the values for the input and internal gates **remain secret**
  – Except for the output gates

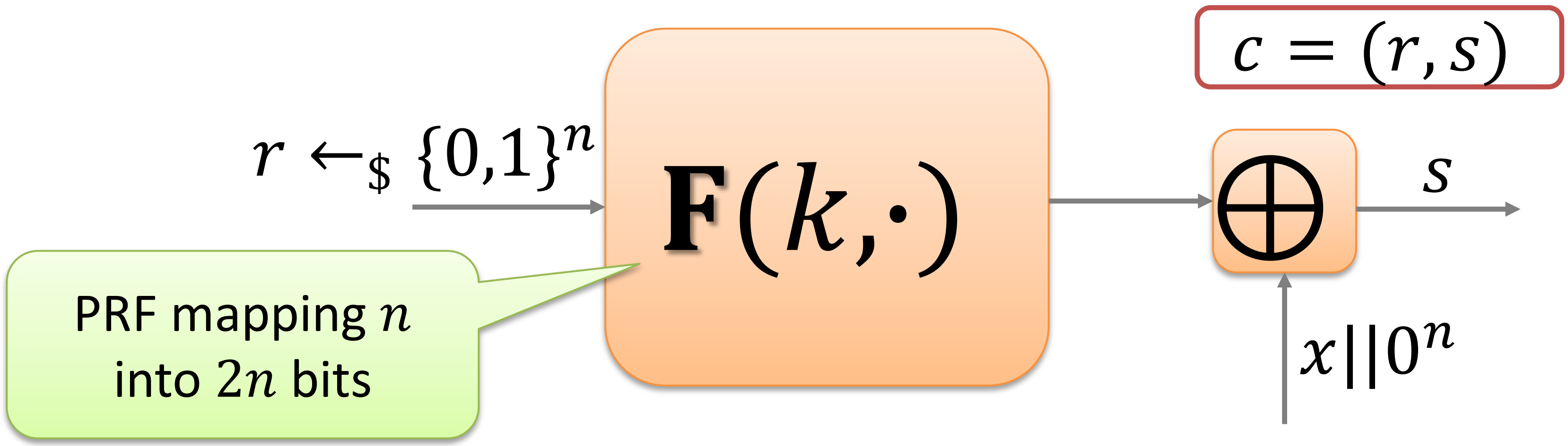
Step 1: Key Generation



Double Encryption (1/2)

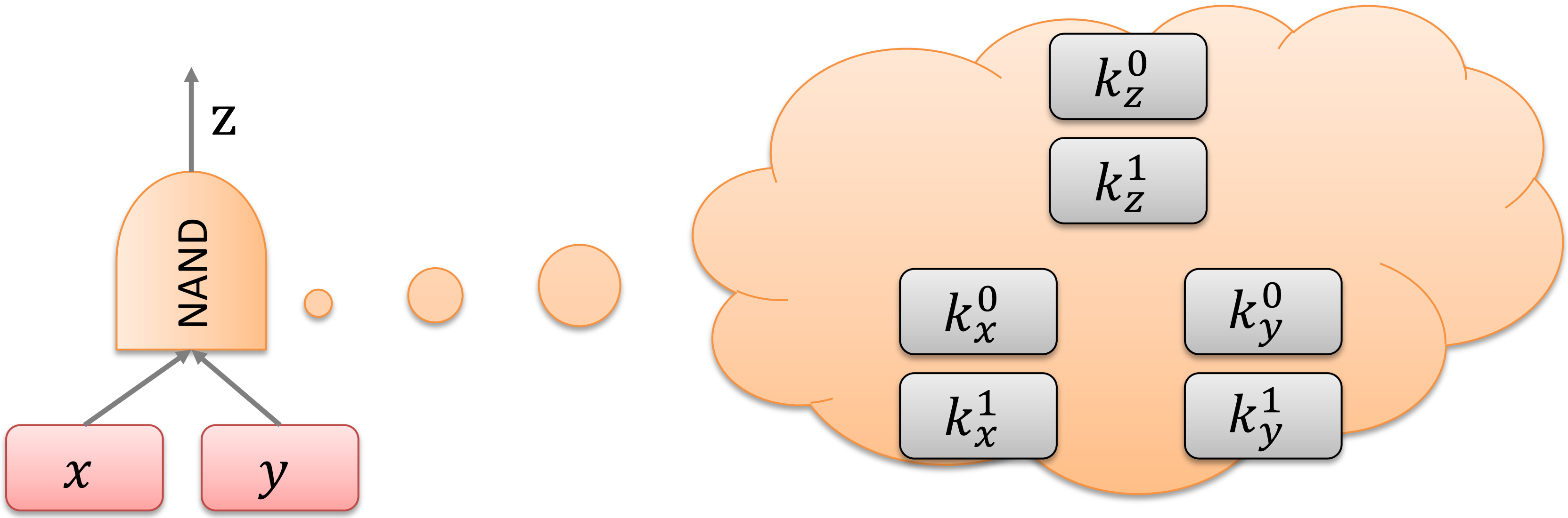
- How to encrypt a message m in such a way that in order to decrypt it one needs to know **two keys** k_0, k_1?
  - Encrypt twice, i.e. $\mathbf{E}(k_0, \mathbf{E}(k_1, m))$
- Special properties
  - **Elusive range:** Hard to generate a valid ciphertext without knowing the key k
  - **Verifiable range:** Given k, c it is easy to test if c is in the output range of $\mathbf{E}(k, \cdot)$

Double Encryption (2/2)



- **Elusive range:** Hard to find r s.t. it is possible to predict the last n bits of $\mathbf{F}(k, r)$
- **Verifiable range:** Given k and (r, s) can compute $\mathbf{F}(k, r)$ and check that the last n bits equal the last n bits of s

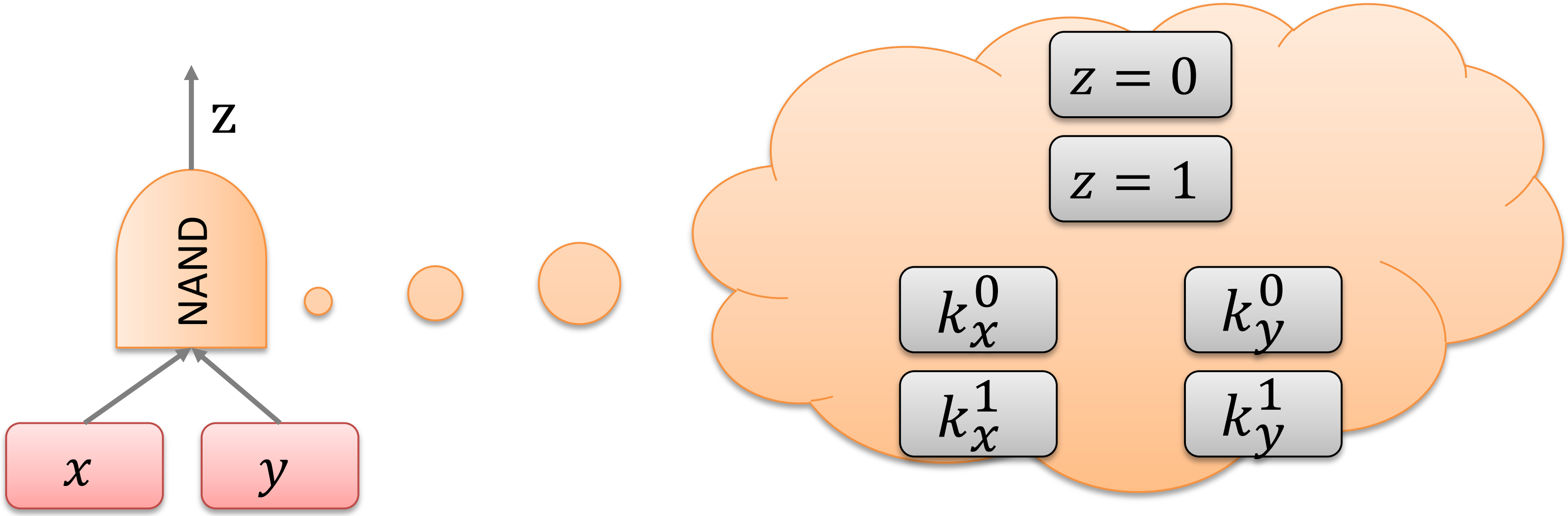
Step 2: Garbling Gates



Given k_x^a, k_x^b it is possible to decrypt only k_z^c such that $c = a \text{ NAND } b$ (all other entries yield **invalid outcome**)

x	y	x NAND y	Garbled Output
0	0	1	$\mathbf{E}(k_x^0, \mathbf{E}(k_y^0, k_z^1))$
0	1	1	$\mathbf{E}(k_x^0, \mathbf{E}(k_y^1, k_z^1))$
1	0	1	$\mathbf{E}(k_x^1, \mathbf{E}(k_y^0, k_z^1))$
1	1	0	$\mathbf{E}(k_x^1, \mathbf{E}(k_y^1, k_z^0))$

Garbling Output Gates

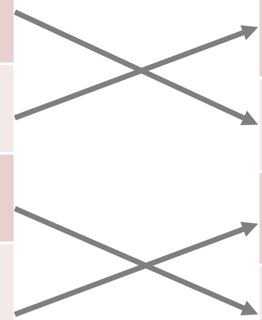


x	y	x NAND y	Garbled Output
0	0	1	$\mathbf{E}(k_x^0, \mathbf{E}(k_y^0, 1))$
0	1	1	$\mathbf{E}(k_x^0, \mathbf{E}(k_y^1, 1))$
1	0	1	$\mathbf{E}(k_x^1, \mathbf{E}(k_y^0, 1))$
1	1	0	$\mathbf{E}(k_x^1, \mathbf{E}(k_y^1, 0))$

Step 3: Sending Garbled Gates

- For every gate Alice sends the encrypted labels in **randomly permuted order**
  - So for each gate Bob knows 4 ciphertexts

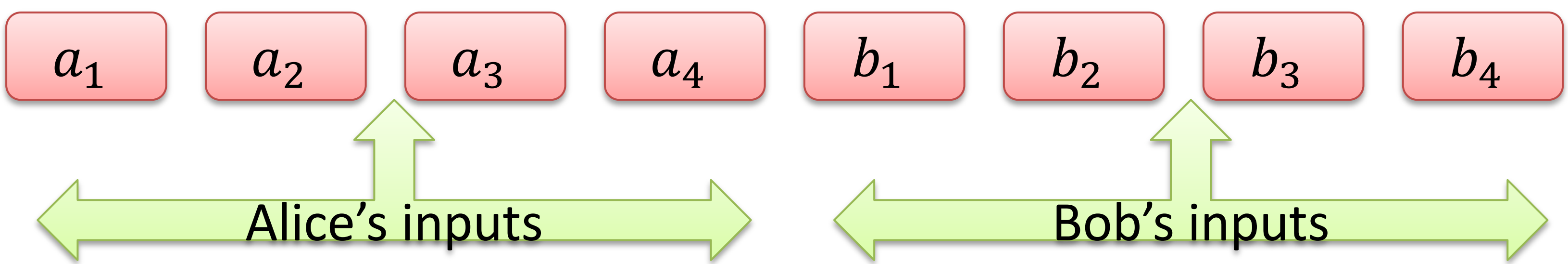
x	y	x NAND y	Garbled Output
0	0	1	$\mathbf{E}(k_x^0, \mathbf{E}(k_y^0, 1))$
0	1	1	$\mathbf{E}(k_x^0, \mathbf{E}(k_y^1, 1))$
1	0	1	$\mathbf{E}(k_x^1, \mathbf{E}(k_y^0, 1))$
1	1	0	$\mathbf{E}(k_x^1, \mathbf{E}(k_y^1, 0))$



To Bob
c_z^1
c_z^2
c_z^3
c_z^4

Step 4: Garbled Circuit Evaluation (1/3)

- Bob needs to evaluate the circuit bottom-up to obtain the keys that **reveal the output**
- To do so, he needs the **labels corresponding to the inputs**
  - Recall that part of the input is from Alice and part is from Bob

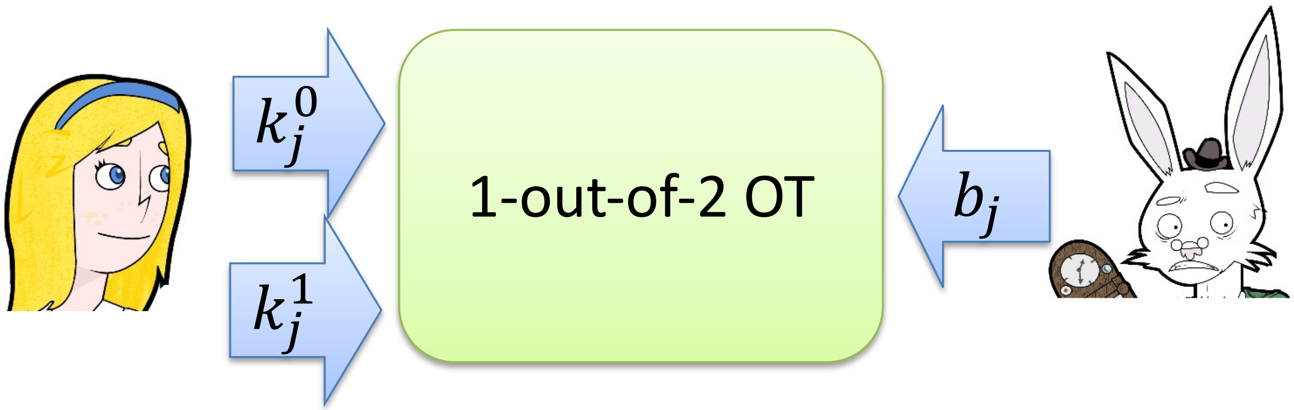


Step 4: Garbled Circuit Evaluation (2/3)

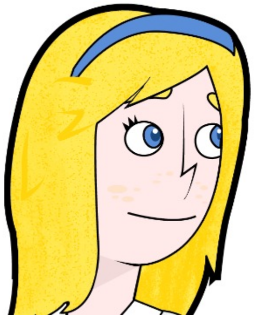
- Alice can simply **send** the labels $k_i^{a_i}$ corresponding to **her inputs**
  - The labels are clearly **independent** of the inputs
- Moreover, since the gates are permuted Bob **does not learn** whether he received the label corresponding to 0 or to 1

Step 4: Garbled Circuit Evaluation (3/3)

- But how can Bob get the labels corresponding to his inputs?
  - He **cannot reveal** the input to Alice
  - Alice **cannot send both labels**, otherwise Bob could compute the function on **multiple inputs**
- Solution: Use 1-out-of-2 OT!



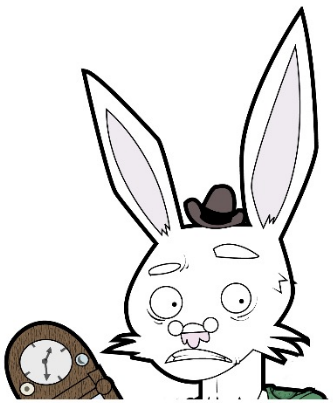
Yao's Protocol Overview



$a_1, \dots, a_n$

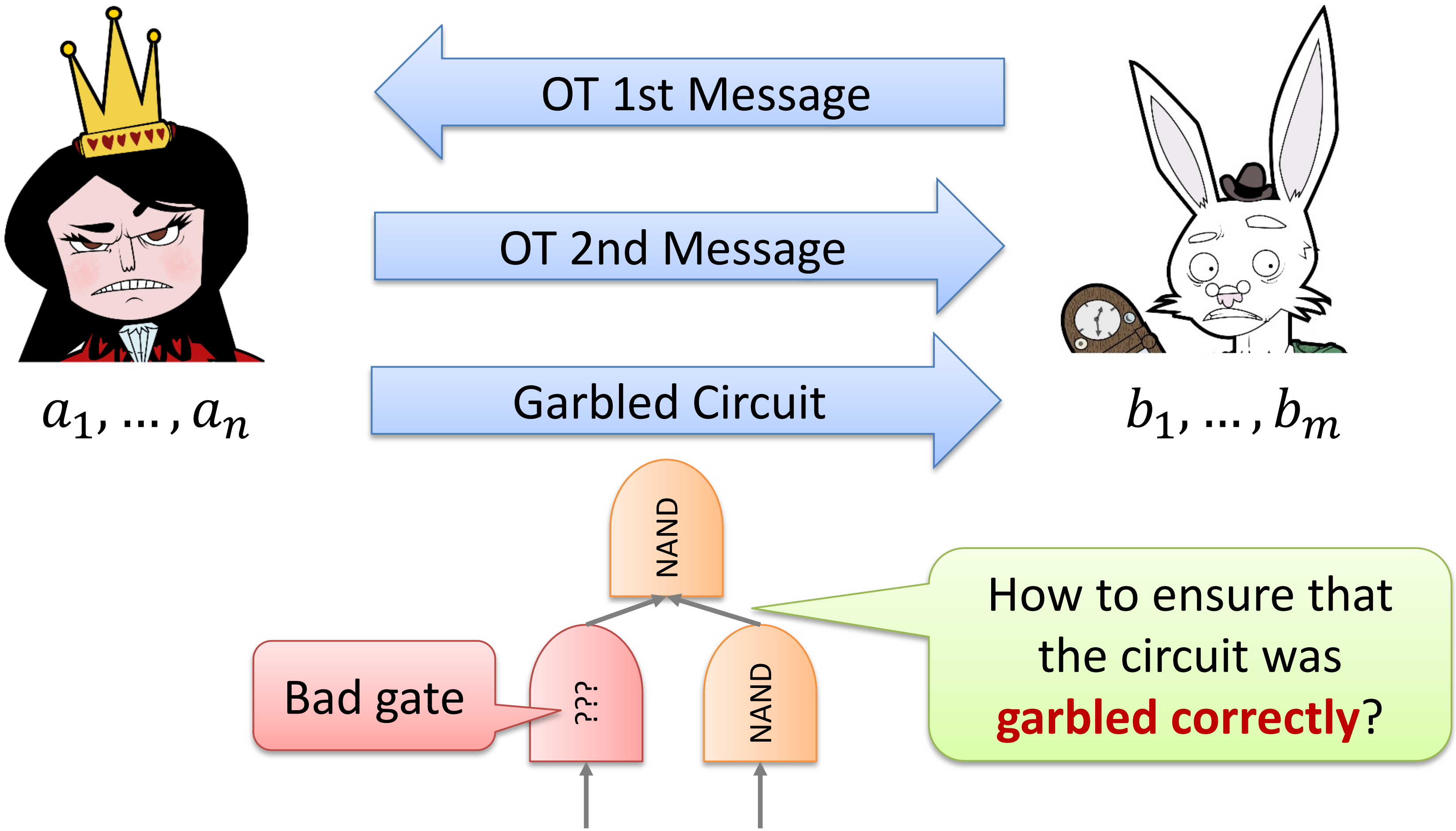
- Garbled circuit corresponding to f
- Labels for $a_1, \dots, a_n$

m times OT for each $b_1, \dots, b_m$

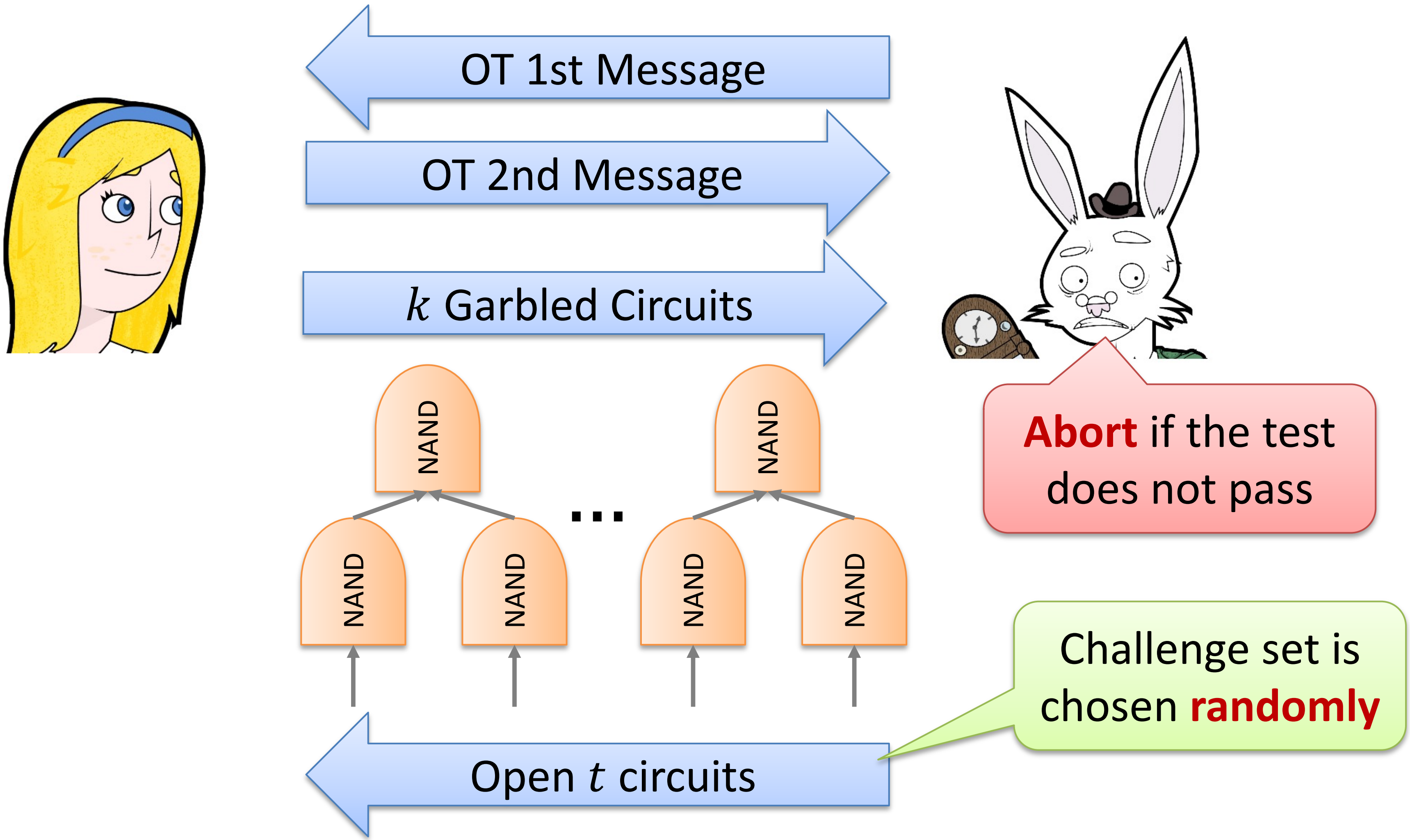


$b_1, \dots, b_m$

What Can Go Wrong?



Cut & Choose


OT 1st Message
OT 2nd Message
k Garbled Circuits
NAND
NAND
NAND
...
NAND
NAND
NAND
Open t circuits
Abort if the test does not pass
Challenge set is chosen **randomly**

Balls and Bins

- Say k circuits in total, out of which c are **corrupted** and t are **tested** by the evaluator

$$\# \text{ of ways to pick only good} = \binom{k-c}{t}$$

$$\# \text{ of ways to pick } t = \binom{k}{t}$$

- Probability that garbler **succeeds**

Setting $t = k/2$

$$\frac{\binom{k-c}{t}}{\binom{k}{t}} = \frac{k/2 \cdot (k/2 - 1) \cdot \dots \cdot (k/2 - c)}{k \cdot (k-1) \cdot \dots \cdot (k-c)} < 2^{-c}$$

Consequences

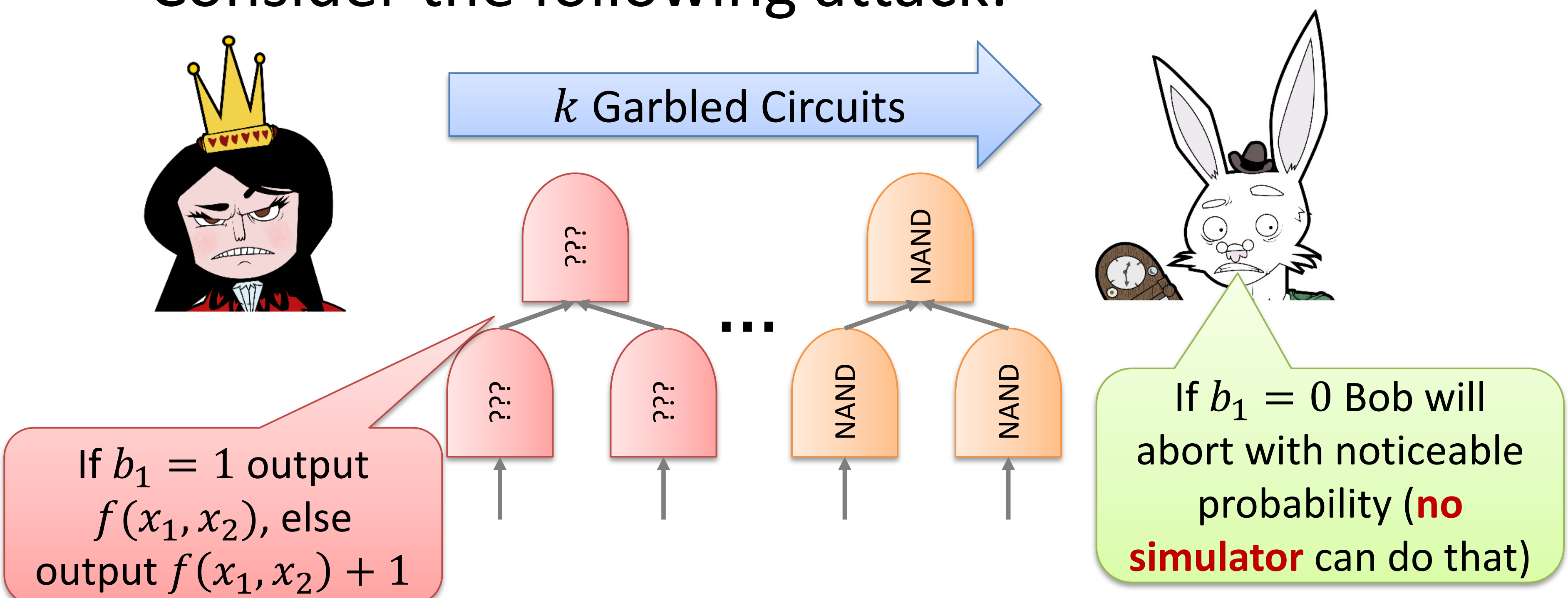
- The above equation implies that the probability that the **test passes** in case
  - $O(k)$ circuits are corrupted is **negligible**
  - $O(1)$ circuits are corrupted is **noticeable**

$$\left(\frac{1}{2}-\frac{c}{k}\right)^c \leq \frac{\binom{k-c}{t}}{\binom{k}{t}} < 2^{-c}$$

Since $\frac{k/2-c}{k-c} \geq \frac{k/2-c}{k} = \left(\frac{1}{2}-\frac{c}{k}\right)$

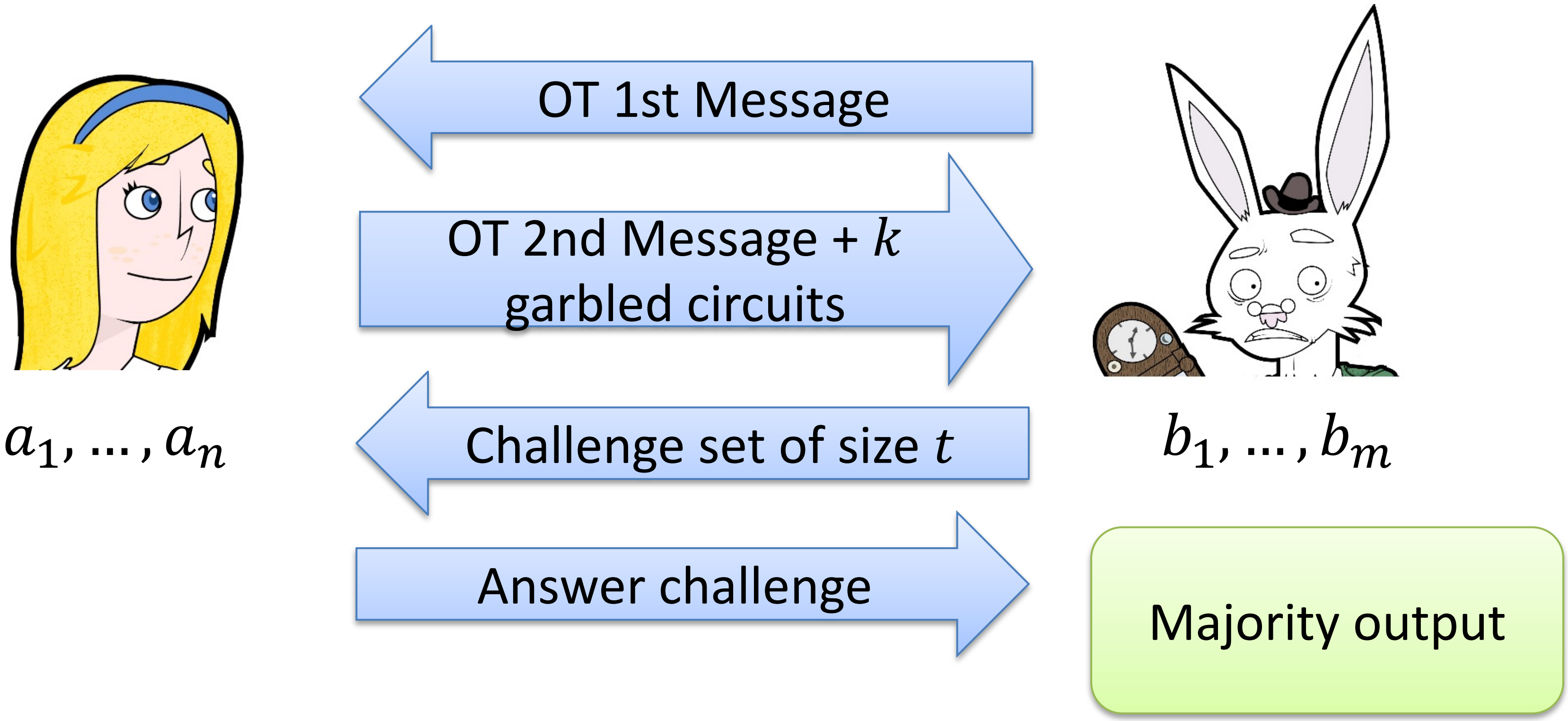
First Idea: Aborting

- Bob evaluates all **unopened** garbled circuits
- If some of the outputs **differ**, abort
- Consider the following attack:

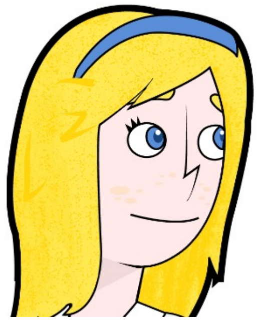


Second Idea: Take Majority

- If some of the outputs **differ**, define the output to be the **majority** of the outputs



Another Problem

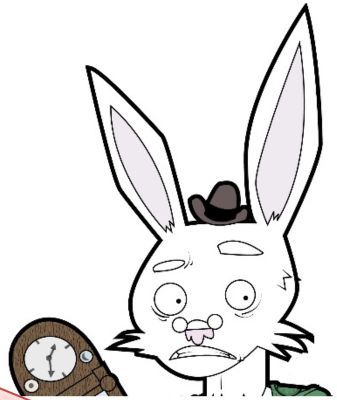


OT 1st Message

OT 2nd Message + k garbled circuits

Challenge set of size t

Answer challenge

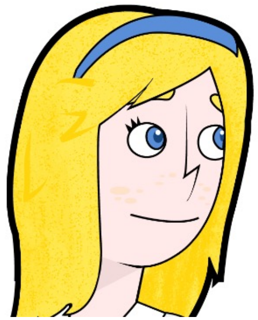


Can't send the labels for Alice's inputs here

Send them here instead!

Majority output

Input Consistency

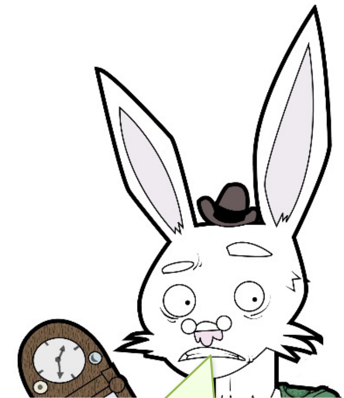


OT 1st Message

OT 2nd Message + k garbled circuits

Challenge set of size t

Answer challenge + $k_{\text{in}}^{1}, \dots, k_{\text{in}}^{\ell}$

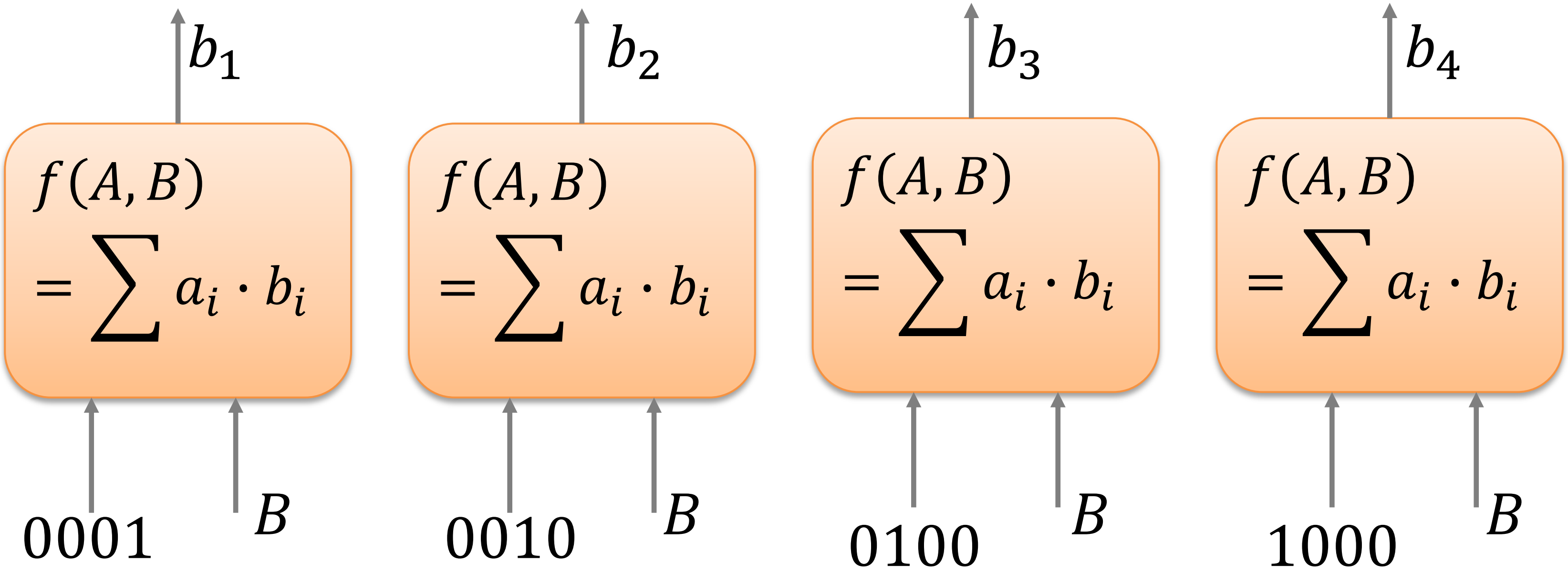


Need to evaluate $\ell = k - t$ garbled circuits

What if the keys do not correspond to the same input?

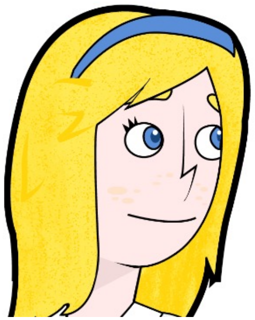
Majority output

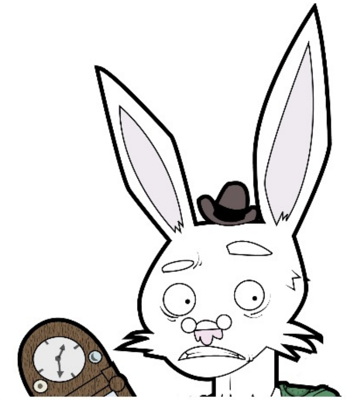
Input Consistency Attack



Protocol output: $\mathbf{Maj}(b_1, b_2, b_3, b_4)$

Need to Prove Input Consistency





OT 1st Message

OT 2nd Message + k garbled circuits + π_1

Commit to the input labels and prove the input is the same

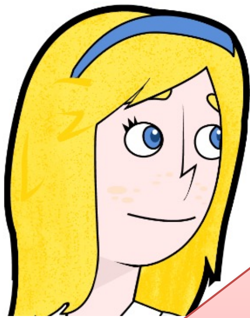
Challenge set of size t

Answer challenge + $k_{\text{in}}^1, \ldots, k_{\text{in}}^{\ell} + \pi_2$

Prove these labels are consistent with the commitment

Majority output

Problem: Malicious OT

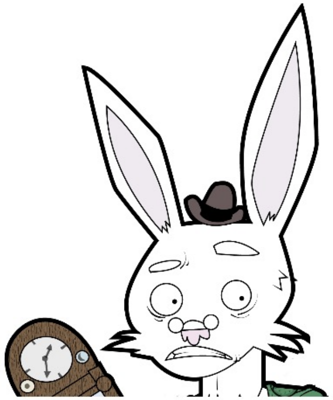


OT 1st Message

OT 2nd Message + k
garbled circuits + π_1

Challenge set of size t

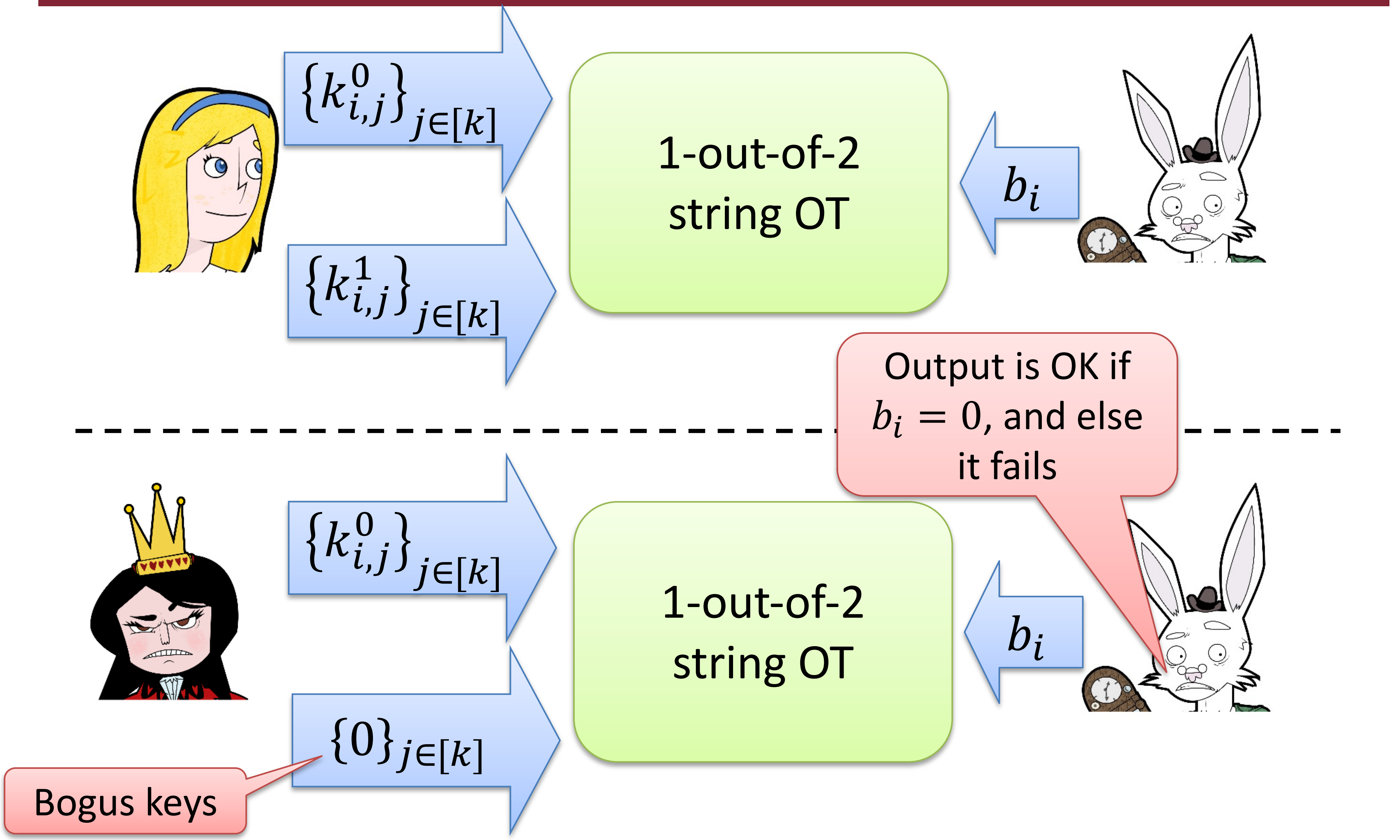
Answer challenge +
$k_{\text{in}}^{1}, \dots, k_{\text{in}}^{\ell} + \pi_2$



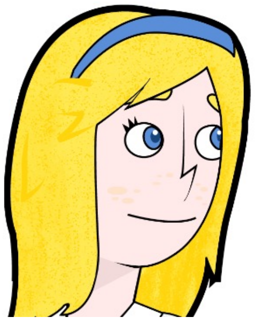
OT protocol must be secure against **malicious adversaries**

Majority output

Selective Failure Attack



OT on Committed Inputs



OT + π'_1

k garbled circuits + π_1 + π'_2

Challenge set of size t

Answer challenge + $k_{\text{in}}^1, \ldots, k_{\text{in}}^{\ell}$ + π_2

Prove consistency between input to the circuits and in the OT

OT on committed inputs

Majority output

Randomized Functionalities

- Let $f(x_1, x_2)$ be a **randomized** functionality
  - Write $f(x_1, x_2; r)$ for a run with randomness r
  - Consider $g\big((x_1, r_1), (x_2, r_2)\big) = f(x_1, x_2; r_1 \oplus r_2)$
- Given a secure protocol for g we construct secure protocol for f:
  - Alice picks random r_1 and Bob picks random r_2
  - Alice and Bob run the protocol for g
  - If one party is honest $r = r_1 \oplus r_2$ is random
- Works both for **passive/active** security

2-Output Functionalities (Semi-Honest)

- Let $f(x_1, x_2) = (f_1(x_1, x_2), f_2(x_1, x_2))$
  - I.e., Alice and Bob get **different outputs**
- Given a secure protocol for **1-output** functions
  - Alice picks random r_1 and Bob picks random x_2
  - Alice and Bob run the protocol for
    $$f'\big((x_1, r_1), (x_2, r_2)\big) = f_1(x_1, x_2) \oplus r_1 || f_2(x_1, x_2) \oplus r_2$$
  - Bob obtains $u||v$, sends u to Alice and outputs $v \oplus r_2$
  - Alice outputs $u \oplus r_1$

2-Output Functionalities (Malicious)

- Let $f(x_1, x_2) = (f_1(x_1, x_2), f_2(x_1, x_2))$
  - Alice picks random x_1, α, β
  - Alice and Bob run the **1-output** protocol for
    $f'\big((x_1, r_1, \alpha, \beta), x_2\big) = c_1 \| f_2(x_1, x_2) \| \gamma$
    $c_1 = f_1(x_1, x_2) \oplus r_1$
    $\gamma = \alpha \cdot c_1 + \beta$
- Bob gets $u||v||w$, sends $u||w$ to Alice and outputs v
- Alice outputs $u \oplus r_1$ iff $w = \alpha \cdot u + \beta$

Performances

Protocol	Security	# Gates	Gates/Sec
Fairplay (‘04)	HBC	4k	600
C&C (‘08)	MAL	1k	4
AES Circuit (‘09)	MAL	40k	35
C&C + ZK (‘11)	MAL	40k	130
C&C + ZK + Parallel (‘11)	MAL	6B	130
C&C + Parallel (‘13)	MAL	1B	1M

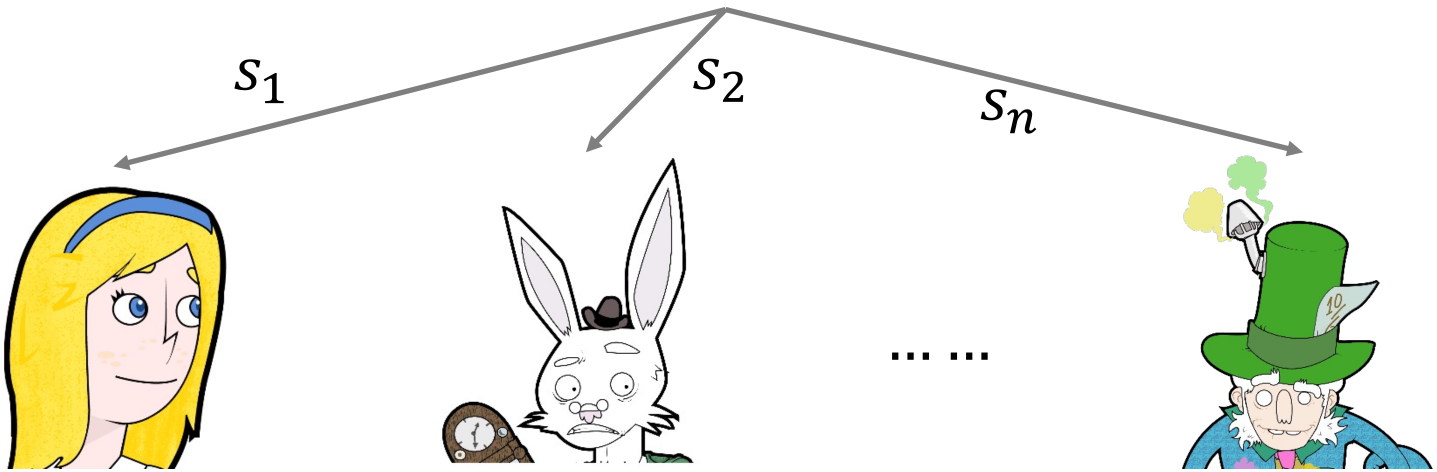
MPC with Honest Majority

How to Share a Secret?

- A dealer wants to **share** a **secret** m between a set of parties in such a way that
  - Any coalition of t parties has zero information about m
  - Any set of at least $t+1$ parties can reconstruct the secret m
  - The adversary is **passive but all powerful**
- The above is called a t-out-of-n **secret sharing** scheme

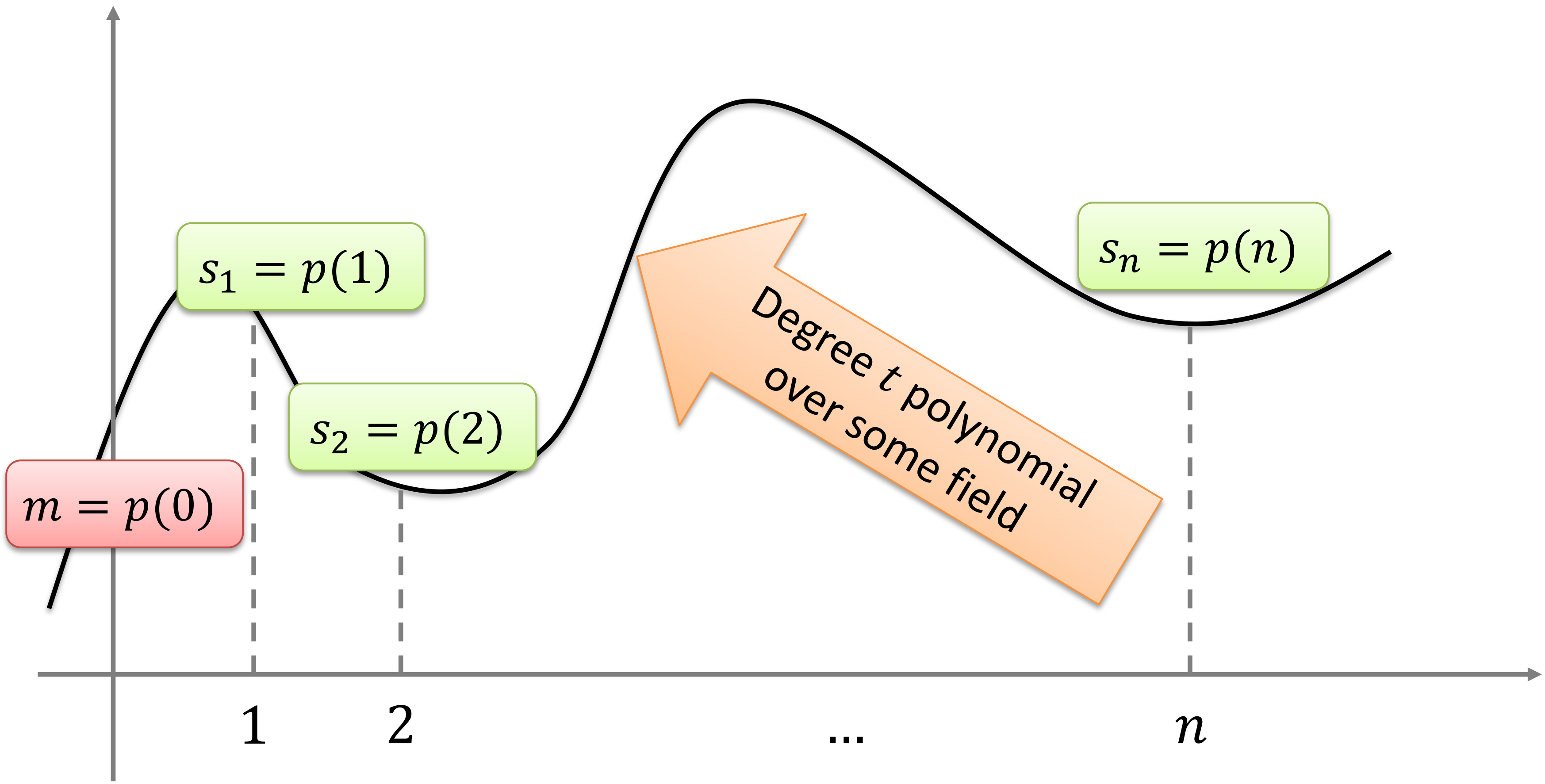
Simple Construction for t $= n - 1$

$$m \in \{0,1\}^k$$
$$s_2, \dots, s_n \leftarrow_\$ \{0,1\}^k$$
$$s_1 = m \oplus s_2 \oplus \dots \oplus s_n$$



$$m = s_1 \oplus s_2 \oplus \dots \oplus s_n$$

Shamir's Secret Sharing (1/4)



CIS SAPIENZA
Research Center for Cyber Intelligence and Information Security

Shamir's Secret Sharing (2/4)

- Sharing
  - The dealer chooses a **random polynomial** $p(X) = m + \sum_{i=1}^{t} a_i \cdot X^i$ over some finite field $\mathbb{F}$, and distributes $s_i = p(i)$ to the i-th player

Shamir's Secret Sharing (3/4)

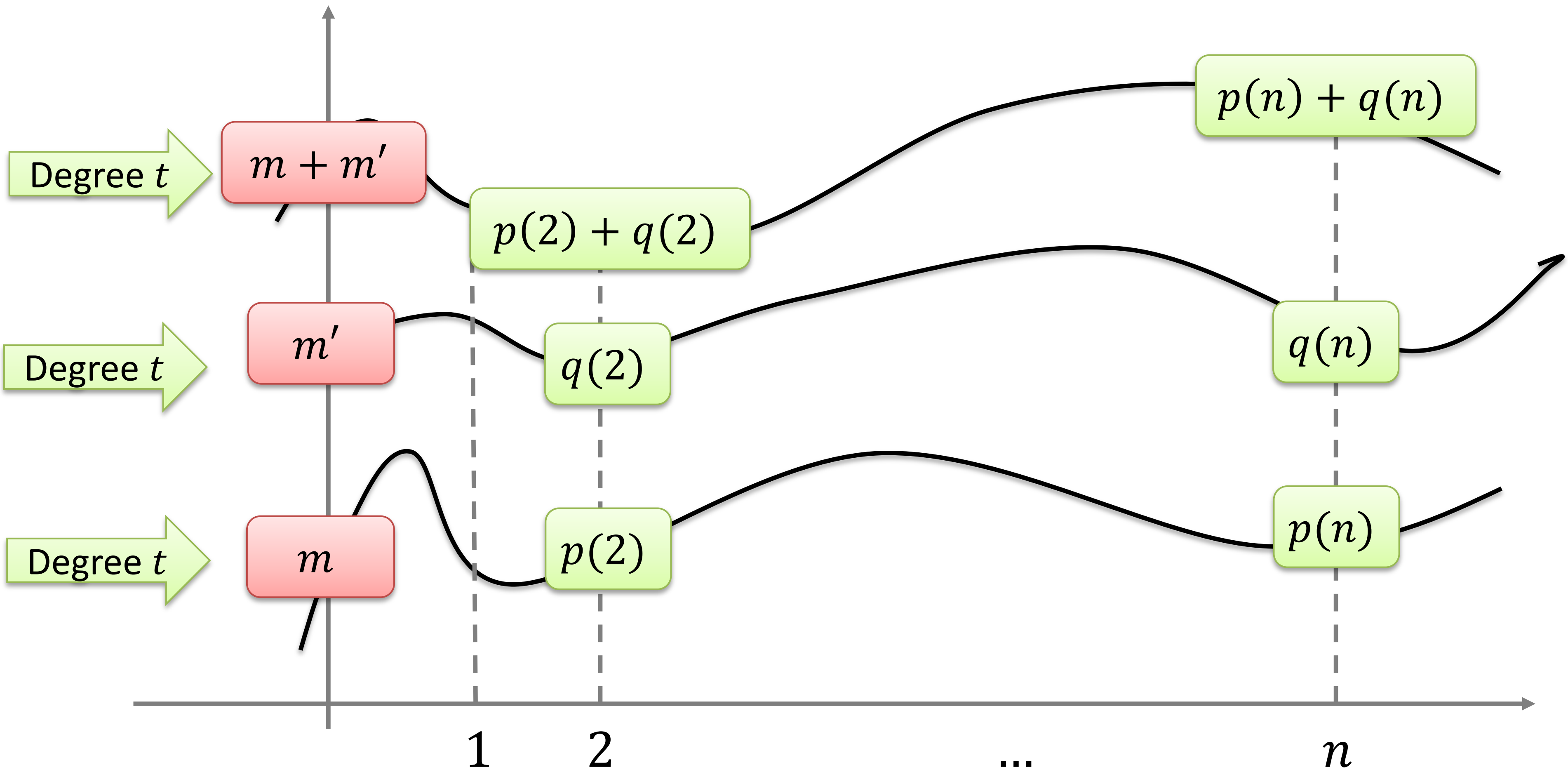
- Reconstruction
  - Given $t + 1$ points $(x_0, y_0), \ldots, (x_t, y_t)$ one can **interpolate** the polynomial and recover the secret
  - **Lagrange interpolation:** Define $p(X) = \sum_{i=0}^{t} y_i \cdot p_i(X)$ where we let $p_i(X) = \prod_{i \neq j} (X - x_j)/(x_i - x_j)$ so that $m = p(0) = \sum_{i=0}^{t} y_i \cdot p_i(0)$

Shamir's Secret Sharing (4/4)

- Privacy
  - For **any distribution** M, any non-zero $x_1, \ldots, x_t \in \mathbb{F}$, and any $y_1, \ldots, y_t \in \mathbb{F}$ we have that once we fix $p(0) = a_0 = m$

$$\mathbb{P}[p(x_1) = y_1, \ldots, p(x_t) = y_t | M = m] = 1/|\mathbb{F}|^t$$

Additive Homomorphism



More on Secret Sharing

- Computational secret sharing
  - **Computational** vs. unconditional security
- General **access structures**
  - Richer sets of authorized players
- Verifiable secret sharing
  - Allows to deal with **malicious dealers** handing wrong shares
- Robust and non-malleable secret sharing
  - Malicious players handing **wrong shares**

Threshold Cryptography

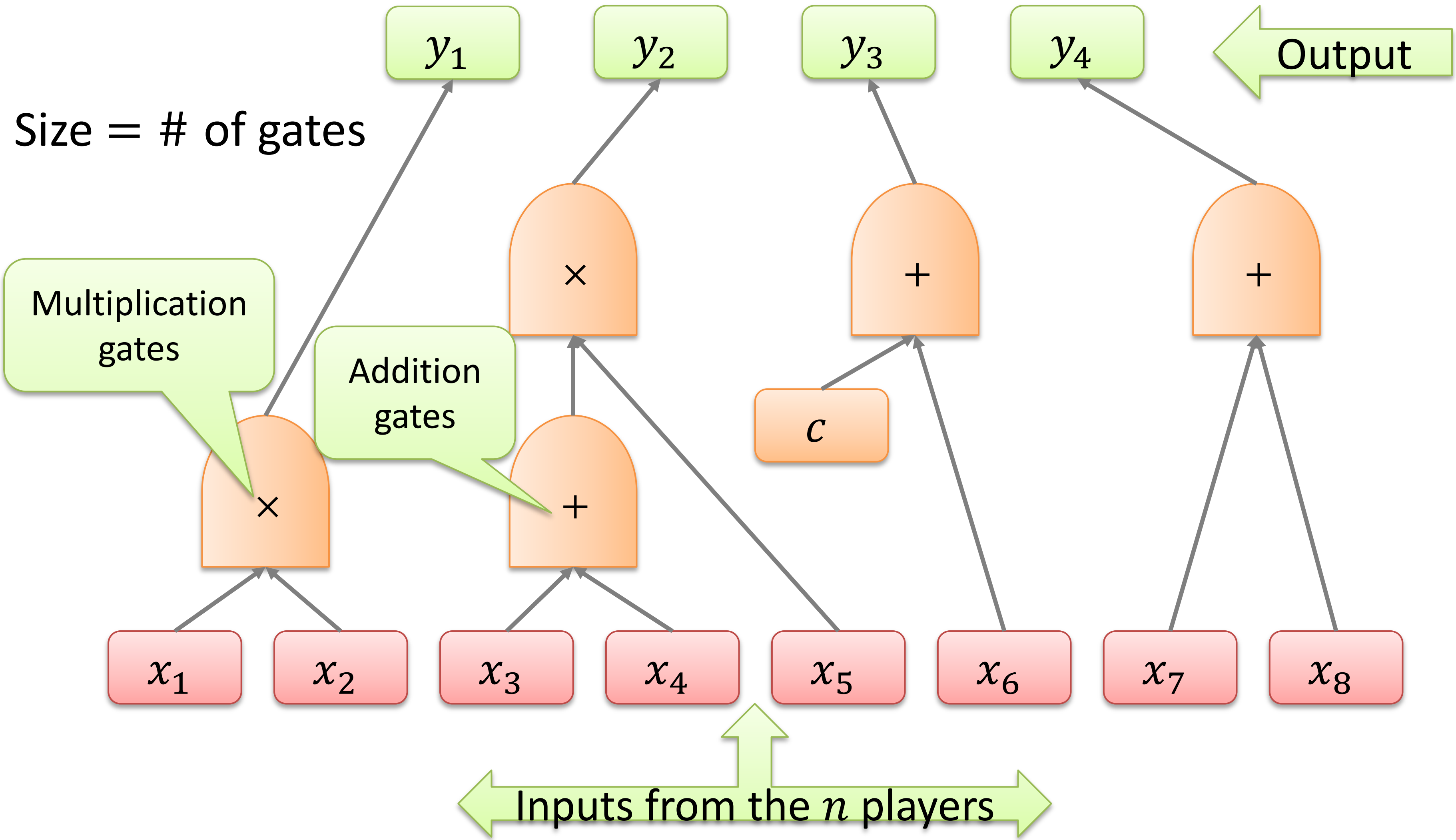
- Suppose we have a secret key sk for a signature scheme, but we don't want to store it on a machine
- Solution:
  - Share sk within n machines
  - Sign in a **distributed manner** (without ever reconstructing sk)
- Useful in cryptocurrencies to **protect users' wallets from thefts**

From Secret Sharing to MPC

- We now describe a protocol for computing any n-party functionality
- High-level idea
  - We represent the function as an **arithmetic circuit**
  - Each party **shares** its input with the other parties
  - Evaluate the circuit gate by gate (invariant: the values of the **intermediary gates** are shared between the parties)
  - Reconstruct the output

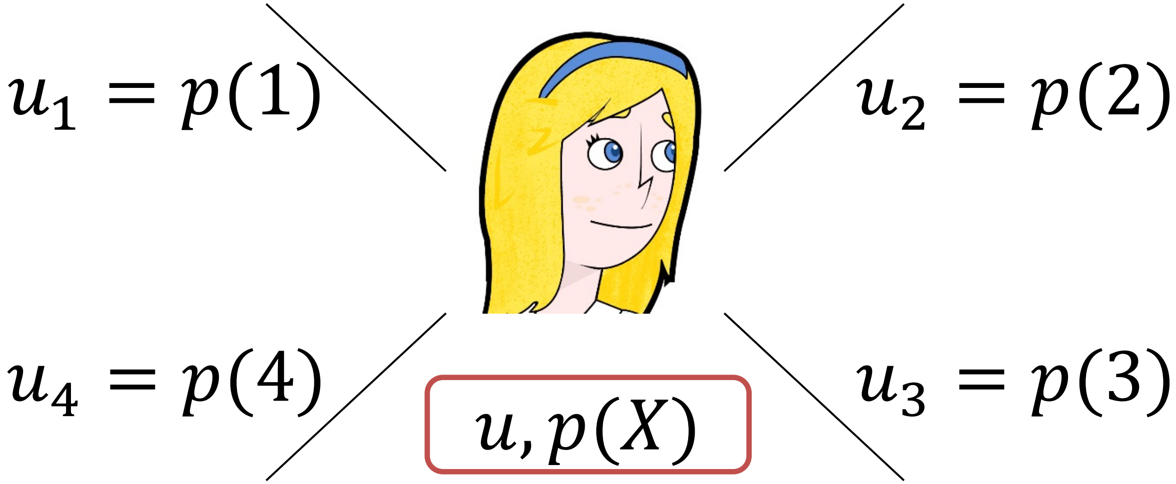
Arithmetic Circuits

Size = # of gates



Step 1: Share Inputs

- Each player secret shares its own input u by picking a **random polynomial** $p(X)$ of degree $\leq t$ such that $p(0) = u$
- At the end of this phase, each party thus holds **one share** for each of the inputs

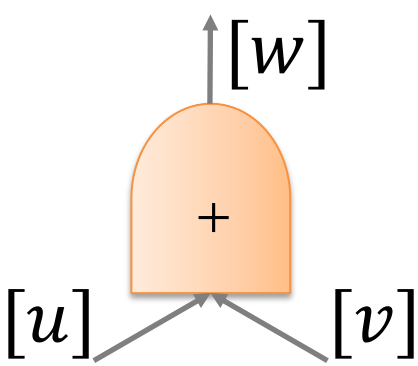


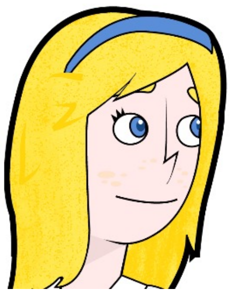
Step 2: Addition Gates (1/2)

- Given secret sharing $[u] = (u_1, \ldots, u_n)$ and $[v] = (v_1, \ldots, v_n)$ we want to compute a secret sharing $[w]$ of the output $w = u + v$
- By **additive homomorphism** each player can **locally** compute $w_i = u_i + v_i$

Step 2: Addition Gates (2/2)

- Since $[u] = (p(1), \dots, p(n))$ and $[v] = (q(1), \dots, q(n))$ for **random polynomials** p, q s.t. $u = p(0)$ and $v = q(0)$, it also holds that $[w] = ((p + q)(1), \dots, (p + q)(n))$ satisfies $w = (p + q)(0)$





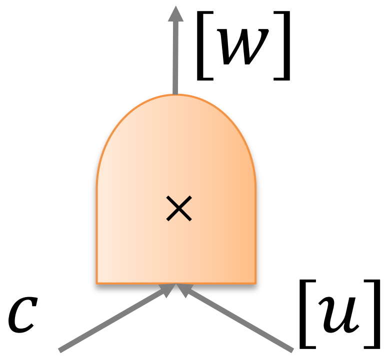
$$w_i = u_i + v_i$$

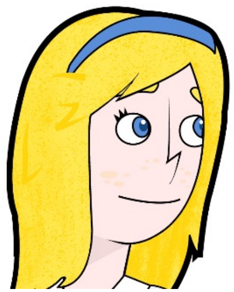
Step 2: Multiplication by a Constant (1/2)

- Given secret sharing $[u] = (u_1, \dots, u_n)$ we want to compute a secret sharing $[w]$ of the output $w = c \cdot u$
- By **additive homomorphism** each player can **locally** compute $w_i = c \cdot u_i$

Step 2: Multiplication By a Constant (2/2)

- Since $[u] = (p(1), \dots, p(n))$ for **random polynomial** p s.t. $u = p(0)$, it holds that $[w] = (c \cdot p(1), \dots, c \cdot p(n))$ satisfies $w = c \cdot p(0)$





$$w_i = c \cdot u_i$$

Step 2: Multiplication Gates (1/6)

- Given secret sharing $[u] = (u_1, \ldots, u_n)$ and $[v] = (v_1, \ldots, v_n)$ we want to compute a secret sharing $[w]$ of the output $w = u \times v$
- Each player can **locally** compute $w_i = u_i \times v_i$

Step 2: Multiplication Gates (2/6)

- Since $[u] = (p(1), \ldots, p(n))$ and $[v] = (q(1), \ldots, q(n))$ for **random polynomials** p, q s.t. $u = p(0)$ and $v = q(0)$, it also holds that $[w] = ((p \times q)(1), \ldots, (p \times q)(n))$ satisfies $w = (p \times q)(0)$
  - Note that the degree of $(p \times q)(X)$ is now $2t$, but as long as $n > 2t$ we can still **uniquely** reconstruct the secret

Step 2: Multiplication Gates (3/6)

- Unfortunately, after another multiplication the degree would become $4t$, which is **too large** if we just want to assume **honest majority**
  - To handle this problem, we use a trick to **reduce the degree**

Step 2: Multiplication Gates (4/6)

- Each party first lets $[z] = [u]\times[v] = (z_1, \ldots, z_n)$, and then creates a **fresh secret sharing** of each $[z_i] = (z_{i,1}, \ldots, z_{i,n})$
  - That is, it picks random $p_i(X)$ of degree $\leq t$ s.t. $p_i(0) = z_i$ and $z_{i,j} = p_i(j)$, and sends $z_{i,j}$ to the j-th player

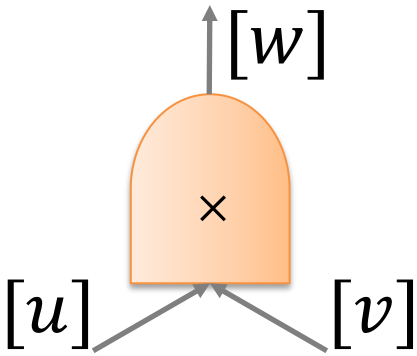
Step 2: Multiplication Gates (5/6)

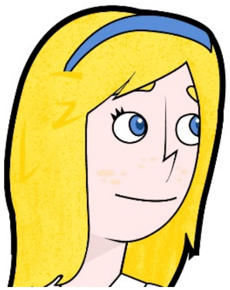
- Now, let

$$[w] = \sum_{i=1}^{n} \alpha_i \cdot [z_i]$$
$$= \left(\sum_{i=1}^{n} \alpha_i \cdot z_{i,1}, \dots, \sum_{i=1}^{n} \alpha_i \cdot z_{i,n}\right)$$

- Here α_i are the **lagrange coefficients** for the reconstruction of $z = \sum \alpha_i \cdot z_i$
  - Hence, $[w] = (p^*(1), \dots, p^*(n))$ where $p^*(X) = \sum_i \alpha_i \cdot p_i\,(X)$ is a degree $\leq t$ polynomial s.t. $p^*(0) = \sum_i \alpha_i \cdot p_i\,(0) = w$

Step 2: Multiplication Gates (6/6)





$$w_j = p^*(j) = \sum_i \alpha_i \cdot p_i(j)$$

Step 3: Output Reconstruction

- At the end of the protocol, each player owns a **share of the output** wire $[y]$ which it sends to each other player
- Thus, each player can **obtain the output**

Feasibility of Maliciously Secure MPC

- Given an MPC protocol secure against **passive adversaries**, can we compile it into an MPC protocol secure against **active adversaries**?
- Main idea:
  - Each player behaves as in the semi-honest protocol, but also
  - Each player proves in **zero-knowledge** that the messages it sends are **computed correctly**
  - O. Goldreich, S. Micali, A. Wigderson. "How to play any mental game." 1987

Efficient MPC with Malicious Security

- I. Damgård, V. Pastro, N. P. Smart, S. Zakarias. "Multiparty computation from somewhat homomorphic encryption." 2012
- N. Chandran, J. A. Garay, P. Mohassel, S. Vusirikala. "Efficient, constant-round and actively secure MPC: Beyond the three-party case." 2017
- X. Wang, S. Ranellucci, J. Katz: "Global-scale secure multiparty computation." 2017

Redactable Blockchain

Blockchain Technology

- Many **applications** beyond cryptocurrencies
  - Healthcare
  - Identity and Reputation Management
  - IoT Devices
  - Smart Grid
  - Supply Chain Management
  - Post-trade Services (US cash equities)
- HYPE?

Necessity of Hard Forks

- Resolve **human errors**
  - Accommodate legal and regulatory requirements, and address bugs, and mischief
- General Data Protection Regulation (**GDPR**)
  - Privacy violations lead to hefty fines: 4 percent of a company's annual revenue or EUR 20 million
- Smart contracts require **flexibility**
  - The DAO had $60 million worth of cryptocurrency stolen

Recent Developments (1/3)

- The "right to be **forgotten**"
  - A real case has stalled after the European Court of Justice found a Dutch man's identity information was uploaded on the Bitcoin blockchain
- The Open Data Institute (**ODI**) Report:
  - "Immutable data storage in blockchains may be incompatible with legislation which requires changes to the official truth"
  - "Even if personal data is not stored on a blockchain, metadata can be sufficient to reveal information"

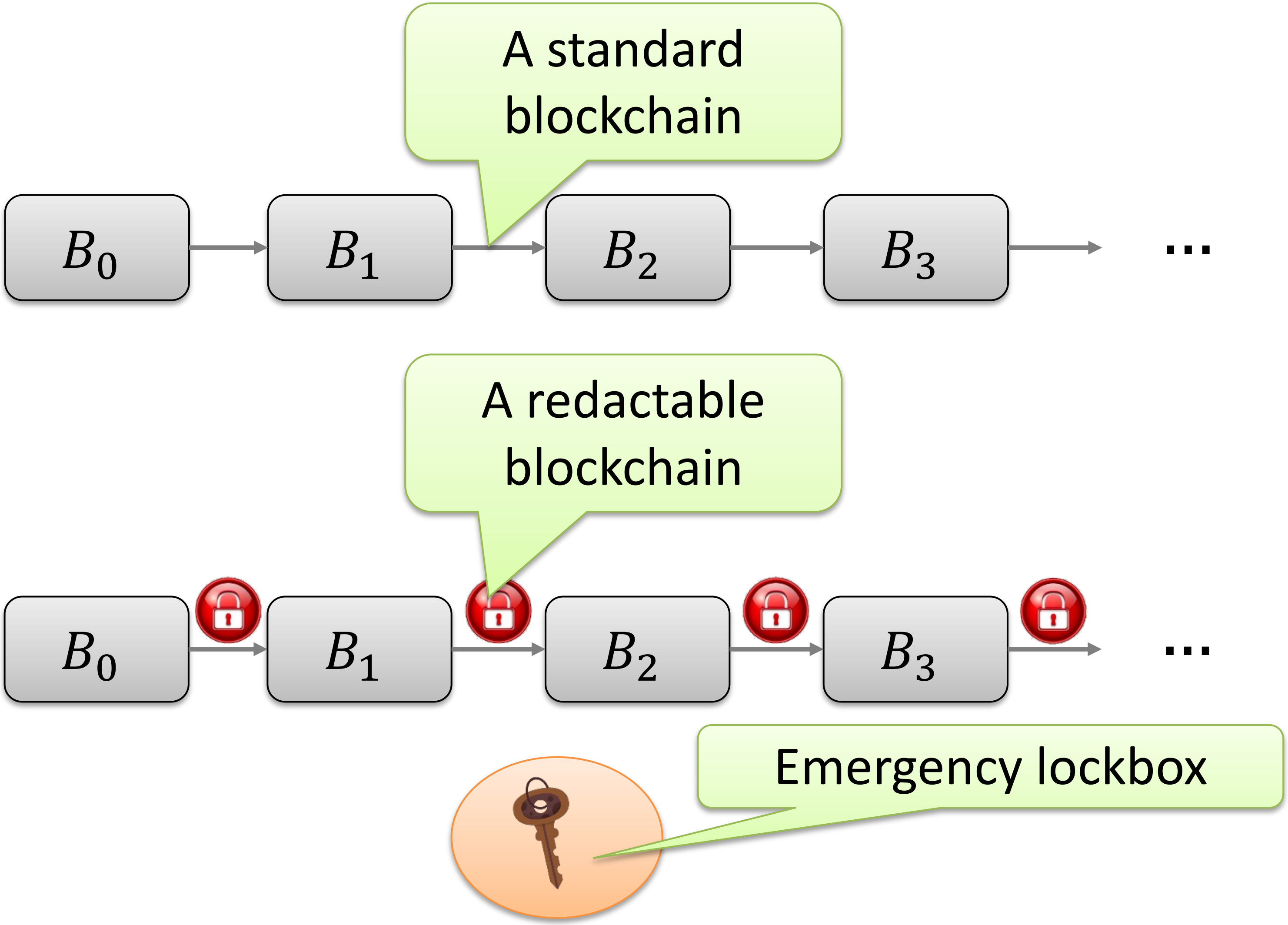
Recent Developments (2/3)

- The European Union Agency for Network and Information Security (ENISA) Report:
  - "Define what to be **kept confidential** in order to remain compliant with regulatory requirements"
  - "Identify or develop standard methods for **removing data** from a ledger"

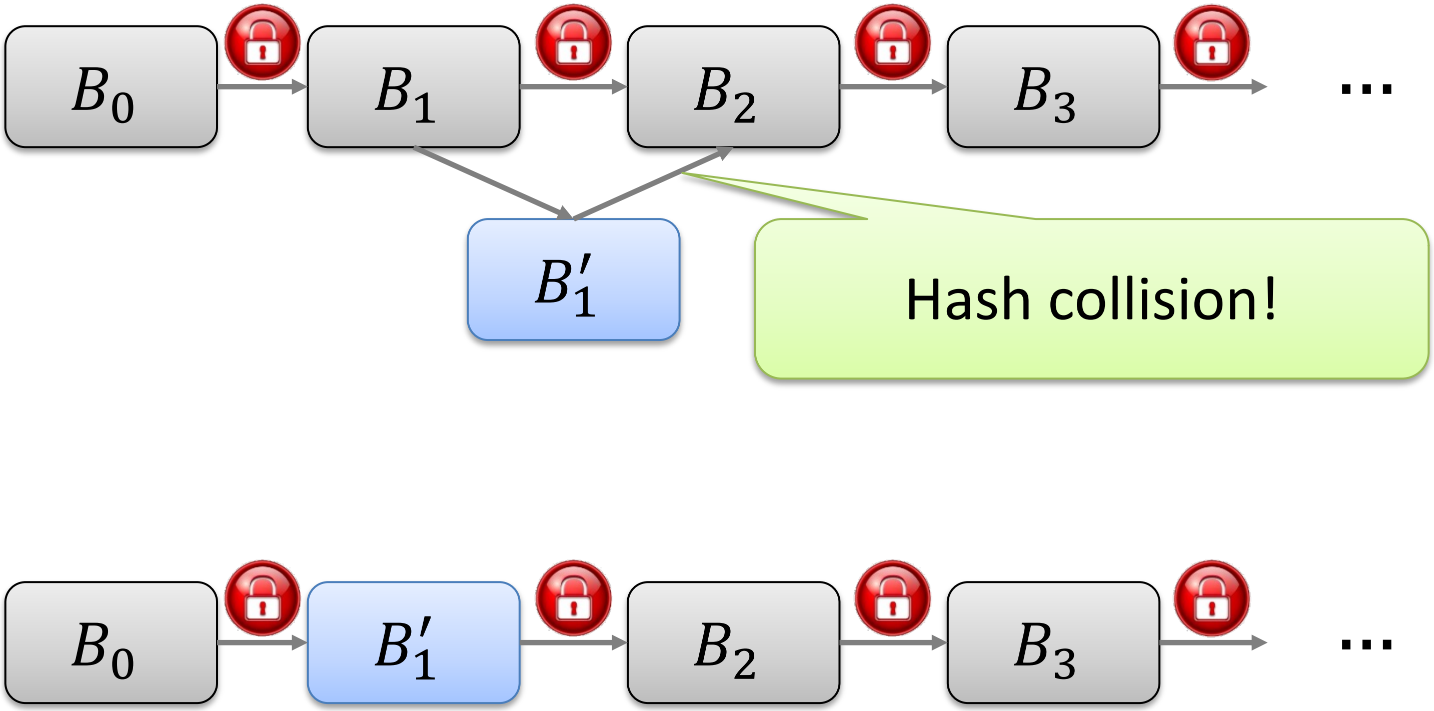
Recent Developments (3/3)

- The European Securities and Markets Authority (ESMA) Report:
  - "The DLT that was originally designed for Bitcoin created **immutable** records, meaning that transactions once validated cannot be modified, cancelled or revoked"
  - "While this immutability had clear benefits in a **permissionless** DLT framework, it appears **ill-suited** to securities markets, e.g., operational errors may necessitate the cancellation of some transactions"

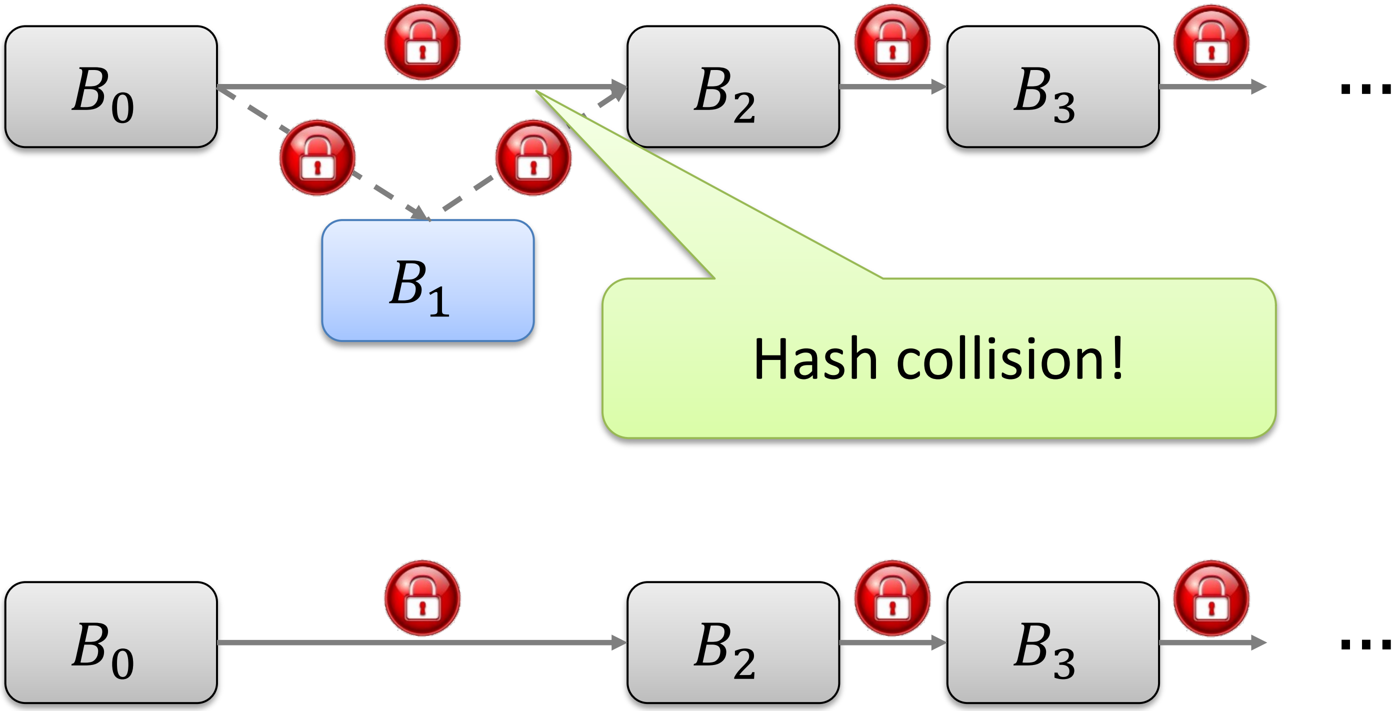
An Emergency Lockbox



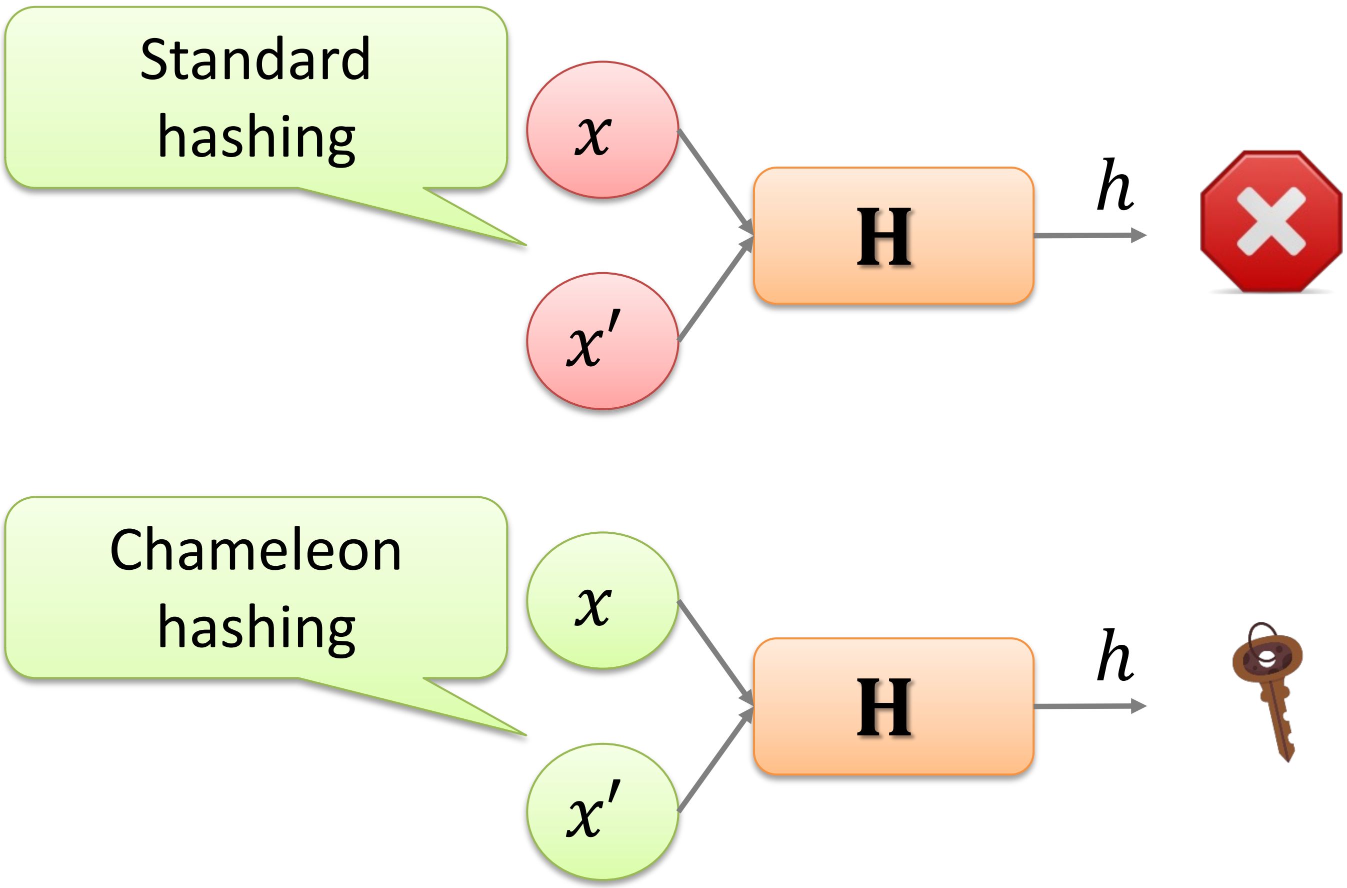
Edit a Block



Remove a Block



Chameleon Hashing

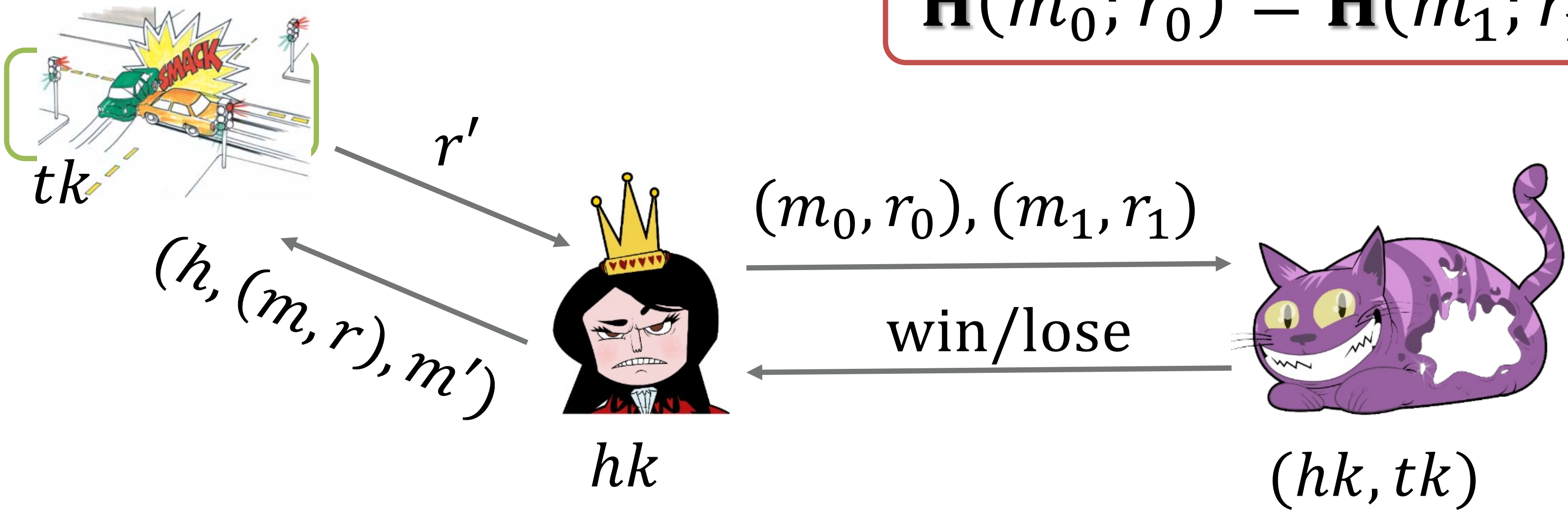


Simple Construction (Inadequate)

- Let $\mathbb{G}$ be a cyclic group of order q with generator g
  - E.g., $\mathbb{G}$ is the subgroup of quadratic residues of $\mathbb{Z}_p^*$
- Hash key and **trapdoor**: $hk = g^a$ and $tk = a$
- Hash computation: $h = g^m \cdot hk^r$ for random $r \in \mathbb{Z}_q$
- Hash collision: Given m, r, m', solve for $ar + m = ar' + m' \bmod q$
  - After few collisions **the trapdoor is exposed**!

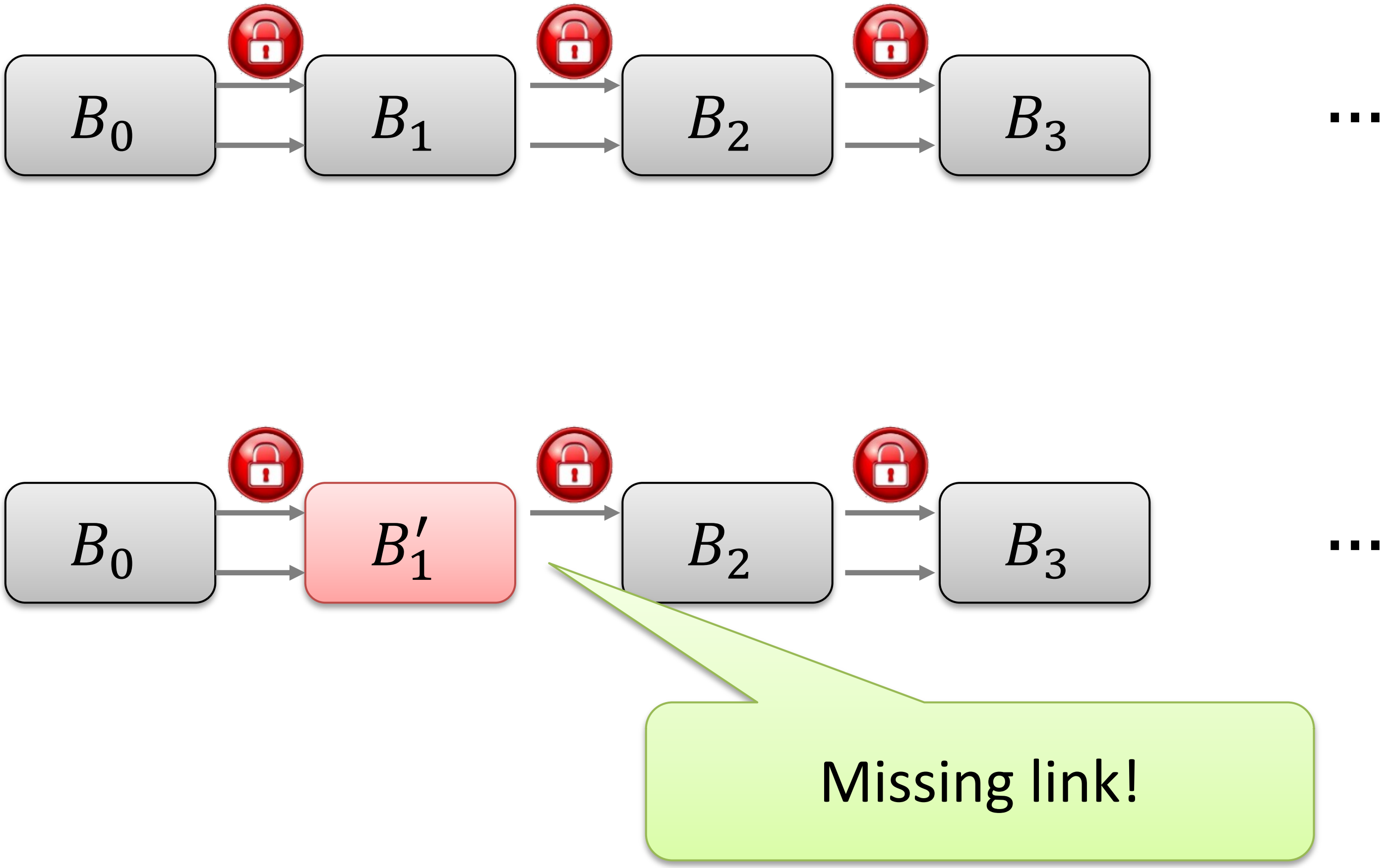
Enhanced Collision Resistance

$$\mathbf{H}(m_0; r_0) = \mathbf{H}(m_1; r_1)$$



- Hard finding collisions even with access to **collision oracle**
  - Collision should be fresh
- Randomness plays the role of **"check value"**

Leaving an Immutable Scar



Concluding Remarks (1/2)

- Geared for **"permissioned" systems**, not for open, decentralized cryptocurrency systems
- Database or spreadsheet?
  - A redactable blockchain is decentralized and **immutable** as all other blockchains
  - There is no centralized server and bad actors **won't be able** to make changes
- Only trusted administrators acting on agreed **rules of governance** can edit, rewrite or remove blocks without breaking the chain

Concluding Remarks (2/2)

- The key can be divided in shares
  - Must be protected as the keys of CAs
  - **None** of the authorities knows the trapdoor
  - When needed collisions can be **computed** via a secure distributed protocol (MPC)
- Amending by appending is often pointless
- Storing just the hash does not help since the hash provides a **"proof of existence"**

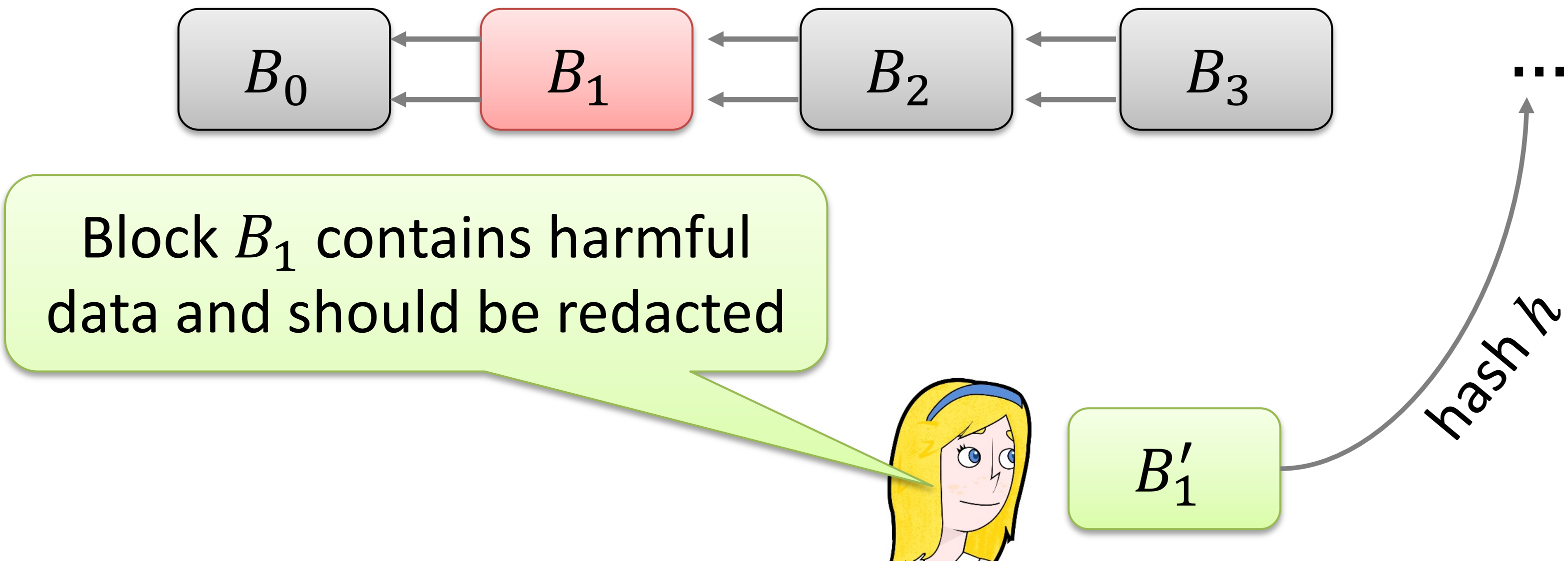
Summary

- Technology developed and patented with Accenture
- The blockchain remains **decentralized and immutable**
  - But a "plan b" is supported if things go wrong
- The invention **preserves** blockchain's benefits, while making it viable for **enterprise use**
- Disruptive, breaking a taboo
  - NYT, FT, Forbes, Reuters, Fortune, MIT Tech Review

Redaction in the Permissionless Setting

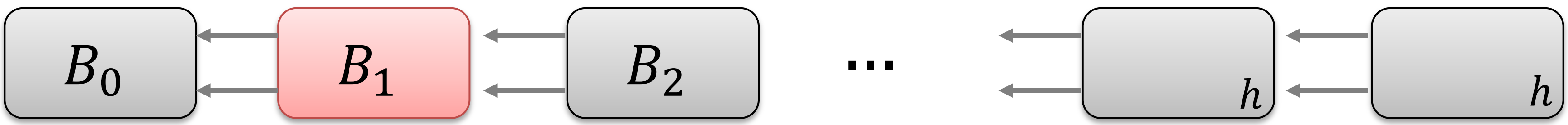
- The previous solution is clearly **impractical** in the **permissionless** setting
- We now give a more **practical** solution
  - No additional trust assumption
  - Consensus on what needs to be redacted
  - Publicly verifiable and accountable
- D. Deuber et al. Redactable Blockchain in the Permissionless Setting. IEEE S&P 2019

Redaction Request



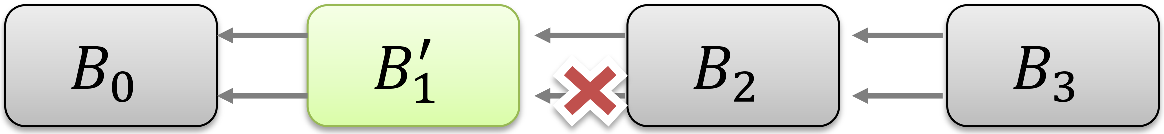
- Modify the block structure
  - **Two** links instead of one (**old** link, **new** link)
- The new block is also sent to a **candidate pool**

Voting



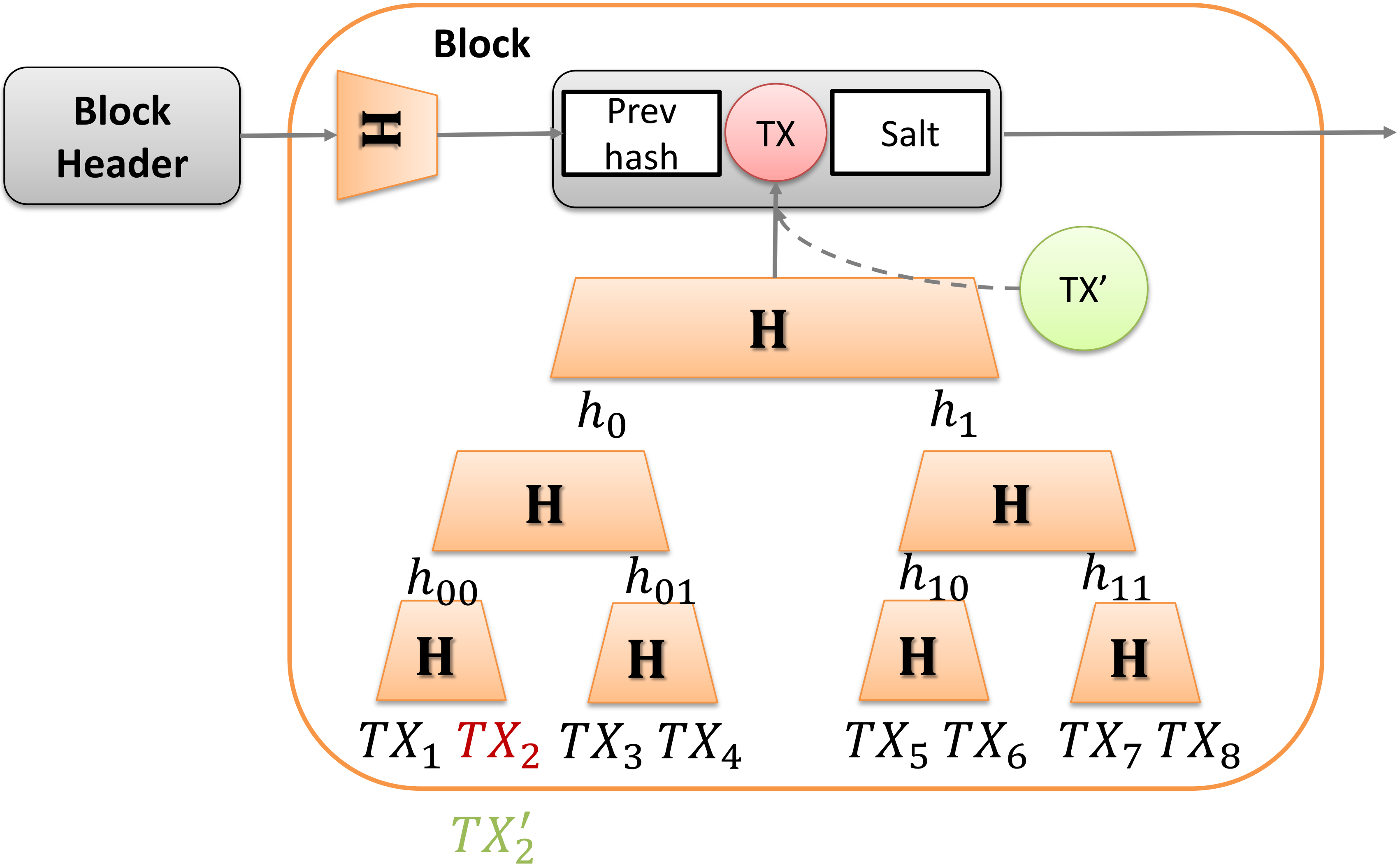
- Miners retrieve proposed blocks
- As they know the hash h, each miner can **vote** by including h in newly minted blocks
- Voting phase spans an epoch
  - 1024 blocks in Bitcoin (2 weeks)
- **Policy**: Say if 50% of the blocks voted, the redaction is approved

Validating Blocks

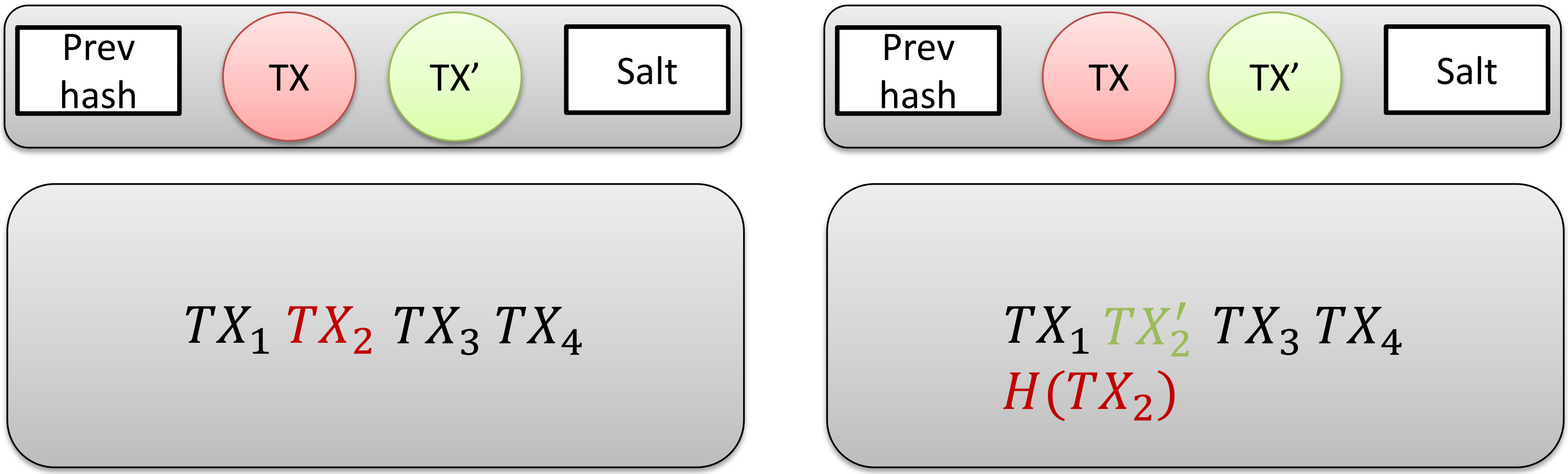


- **Standard** blocks
  - Check PoW, PoS, etc.
  - Check validity of data and links (**old/new**)
- **Redacted** blocks
  - Check PoW, PoS, etc. (w.r.t. **old** link)
  - Check **new** link **broken**, **old** link **good**
  - Check the redaction was approved

Integration in Bitcoin



Integration in Bitcoin



- Old link is $H(prev\_hash, TX, TY, salt)$
  - TY is from the **previous block** header
- New link is $H(prev\_hash, TX', TY, salt)$