AARHUS UNIVERSITY

Secure Computation
EWSCS, Lecture 2

Claudio Orlandi

What we learned yesterday

- Security is defined **comparing the real world** (the protocol) with the **ideal world** (the functionality)
  - **To prove passive security:** show that everything that the adversary learns during the protocol can be simulated having access to input/output only.
  - **To prove active security:** above + show that every "attack" in the real world can be "translated" to an attack to the ideal world (i.e., extracting an input from the adversary).
- **OTTT/GMW:** preview of simple and efficient **passive** secure protocol in the preprocessing model.

Today: protocols in the preprocessing model

- **Truth-table based**
  - OTTT with statistical security vs. active adversaries (computing with MACs)
  - OTTT with perfect security vs. active adversaries
- **Circuit based**
  - Beaver's circuit rerandomization
  - Active security using MACs (TinyOT/BeDOZa)
  - Shorter MACs for multiparty (SPDZ)

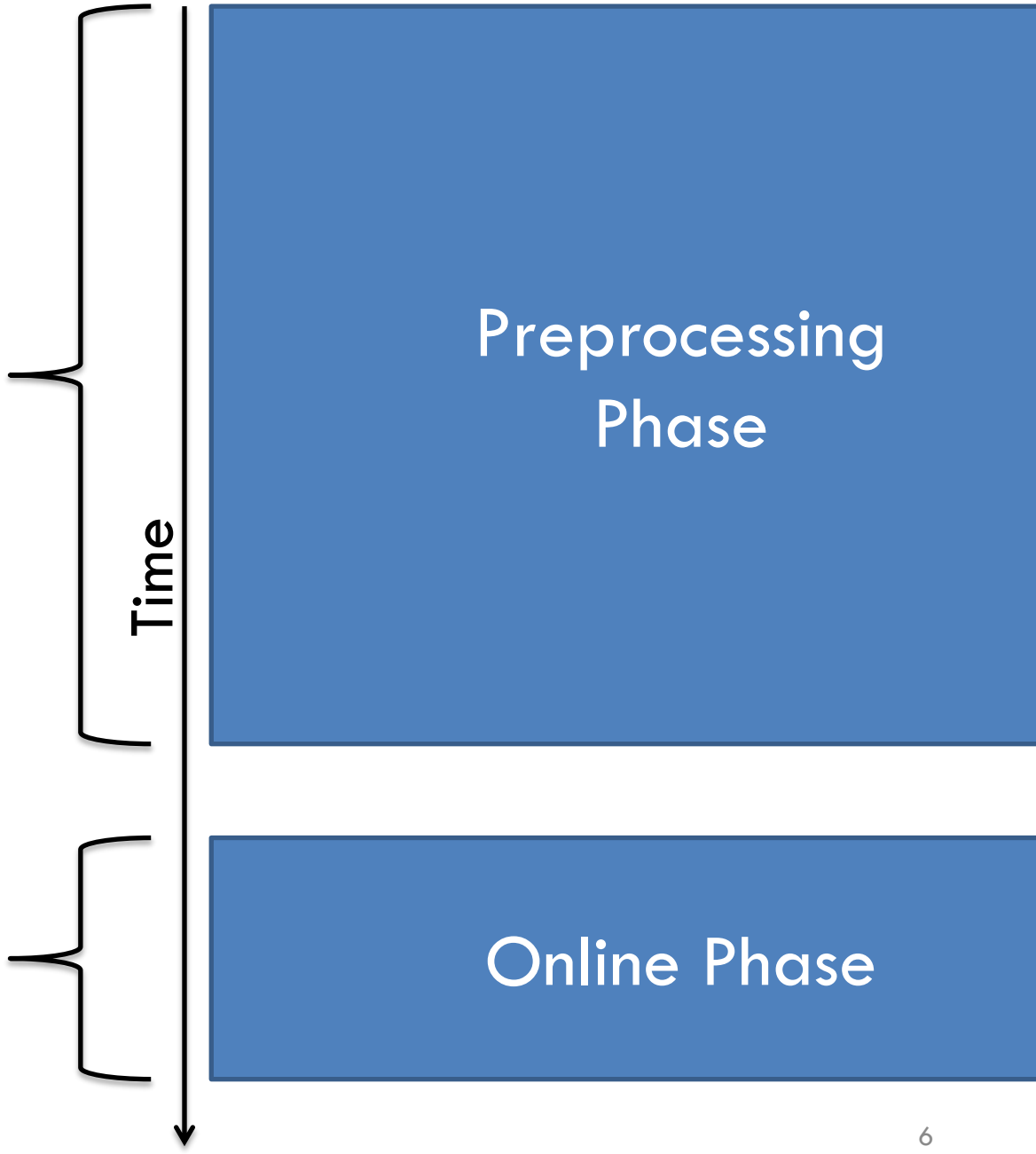
Protocols in the preprocessing model

Two kind of MPC protocols

- Information theoretic protocols
  - High efficiency (only field operations) ☺
  - Provably requires honest majority ☹

- Computationally secure protocols
  - Computationally heavy ☹
  - You only need to trust yourself ☺

Slower, use heavy cryptographic tools



Fast, use only field operations

The preprocessing model

$(r_A, r_B) \leftarrow D$

r_A

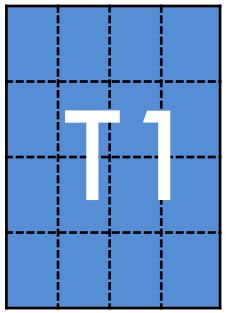
r_B

x

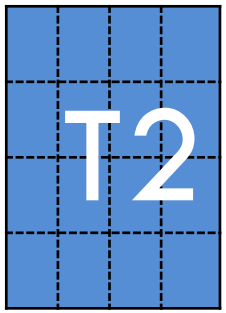
y

f(x,y)

"The simplest 2PC protocol ever" OTTT (Online phase)



, r



, s

$u = x + r$

$v = y + s$

T2[u,v]

output f(x,y) = T1[u,v] + T2[u,v]

Already secure vs. malicious Alice
Bob can send the wrong output share

How to force Bob to send the right value?

- **Problem:** Bob can send the wrong shares
- **Solution:** use information theoretic MACs of the form $M=aT+b \bmod p$ *with* $(a,b) \leftarrow Zp$

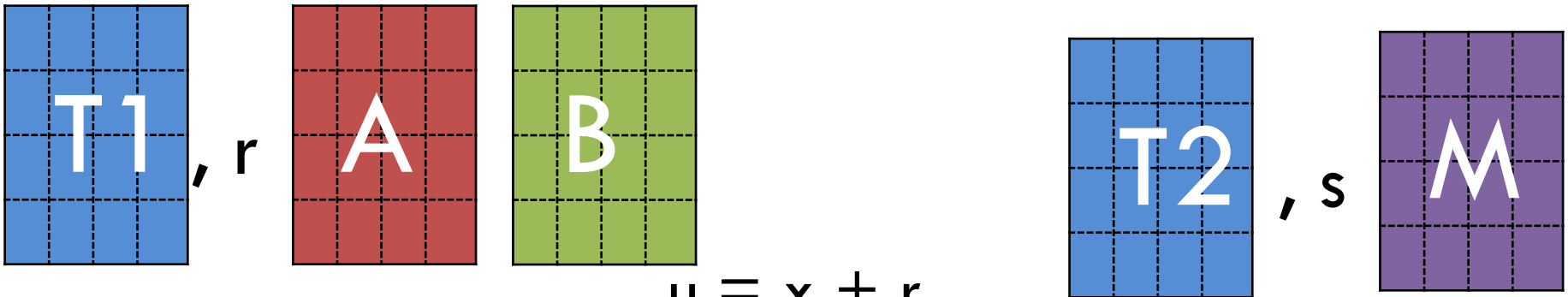
(a,b)

(M,T)

(M',T')

Abort if $M' \neq aT'+b \bmod p$

OTTT+MAC



output f(x,y) = T1[u,v] + T2[u,v]

If M[u,v]=A[u,v]*T2[u,v]+B[u,v]

Else Abort

Statistical security vs. malicious Bob w.p. 1-1/p

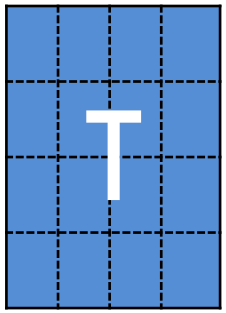
Exercise: Can we do the same with fewer MACs/keys?

OTTT with Perfect Security!

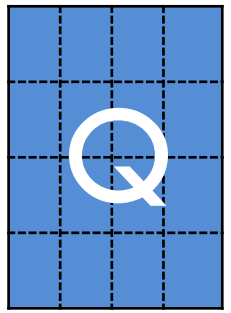
"The simplest 2PC protocol ever" OTTT (Preprocessing phase)

1) Write the truth table of the function F you want to compute
2) Pick random shift r for x
3) *Pick random* permutations $\{Q_i: Y \rightarrow Y \mid i \in X\}$ for y
   //Q is a random $|X|*|Y|$ matrix where each number compare only once in each row
4) $f(x,y)$ is stored in $T[x, Q_u(y)]$
   //T is the truth table of f with each row permuted independetly according to Q
5) A is given r,T and B is given $\{Q_i \mid i \in X\}$.

"The simplest 2PC protocol ever" OTTT (Online phase)



, r



$u = x + r$

$v = Q_u(y)$

output f(x,y) = T[x,v]
(or $f(x,y_0)$ if v is invalid)

1-1 mapping between messages and inputs → perfect security

Perfect Security

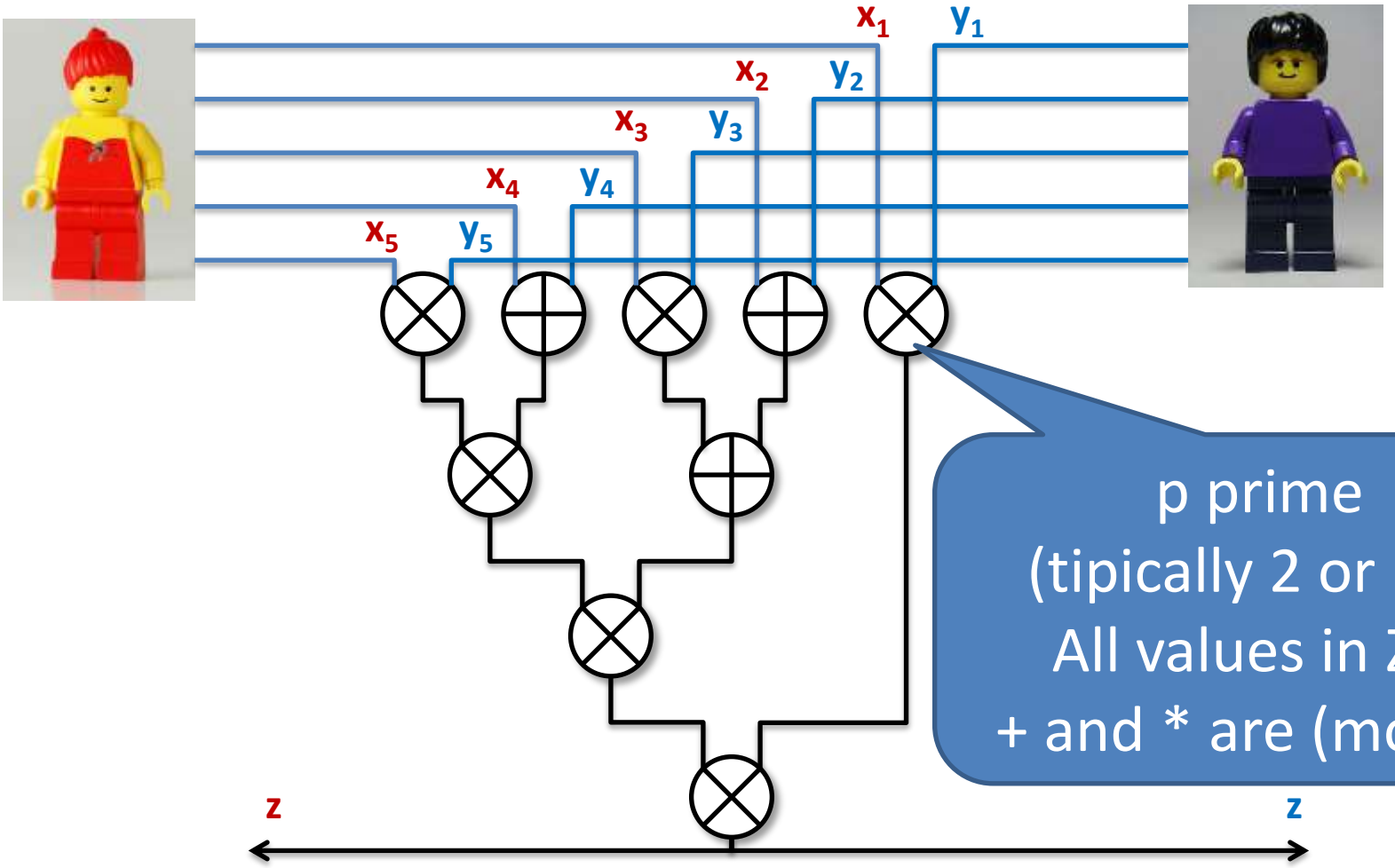
– B only sees collection of random permutations $\{Q_i\}$ and a random value u=x+r. **Most importantly, every possible choice for *v* uniquely determines an input *y* for *B*.**

–A gets truth table *T*, where each row is separately permuted, and a random column index *v*

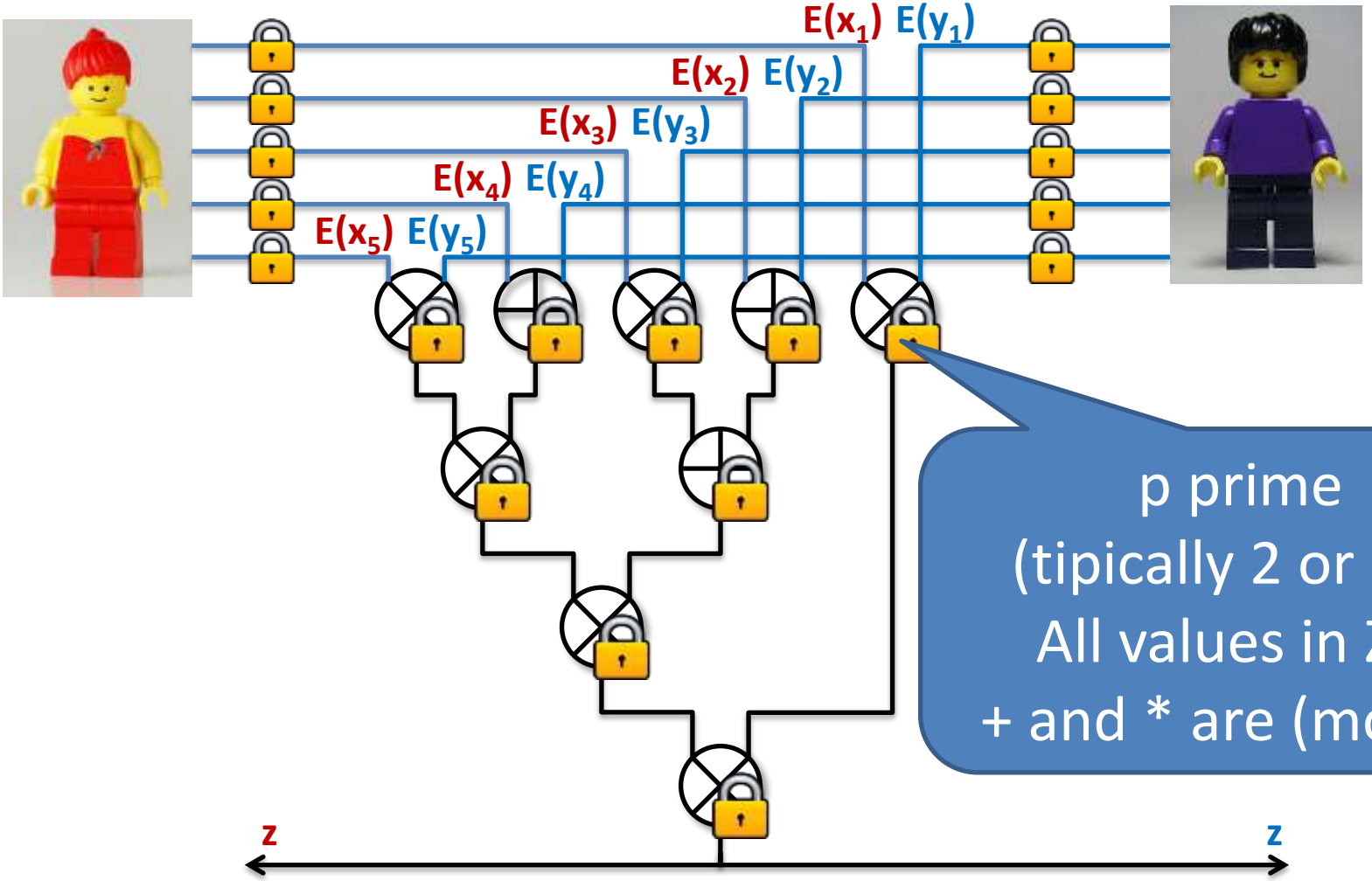
//Note: we need a different permutation for every x, otherwise Alice could compute the function on multiple x!

COMPUTING WITH CIRCUITS

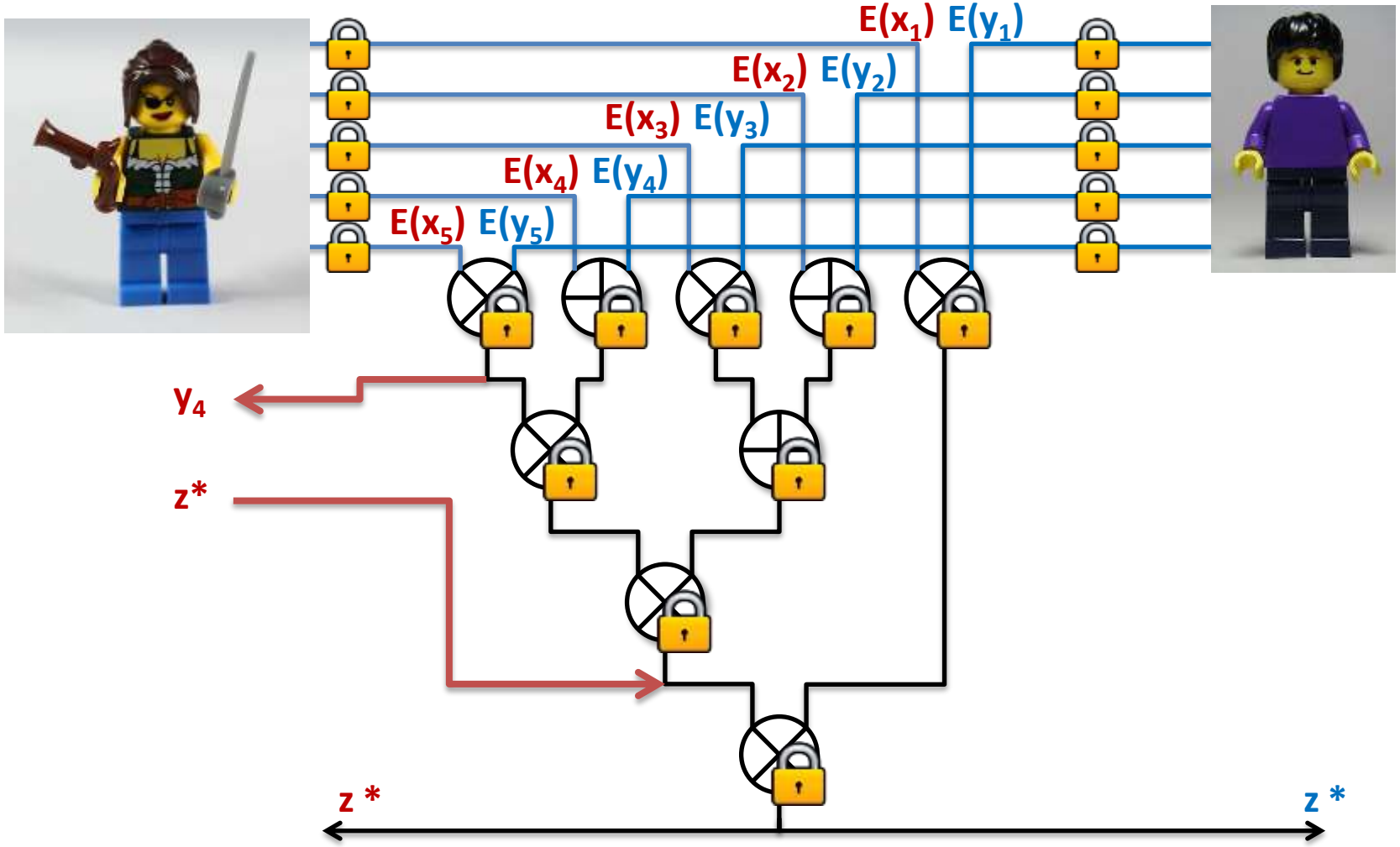
Computation



Secure Computation



Secure Computation



Circuit Evaluation (Online phase)

1) **Input gate**: Alice has **input x**, chooses random x_B and send it to Bob, set
   $x_A = x - x_B \ (mod\ p)$
   (symmetric for Bob)

We write for short:

[x]←Input(A,x)

Meaning Alice has x_A, Bob has x_B

s.t. $x = x_A + x_B \ (mod\ p)$ *//I will drop the mod p from now on*

Alice only sends a random bit! ”Clearly” secure

Circuit Evaluation (Online phase)

2) Output gate: Alice should learn a value [z]

Bob sends z_B

Alice outputs $z=z_A+z_B$

$z \leftarrow$ Output(A,[z]) or

$z \leftarrow$ Output([z]) when both A,B get output

Alice should learn z anyway!

”Clearly” secure (for passive)

Circuit Evaluation (Online phase)

2) Addition: to compute [z]=[x+y]

- Alice computes $z_A = x_A + y_A$
- Bob computes $z_B = x_B + y_B$

- We write [z] = [x] + [y]

No interaction! ”Clearly” secure

As expensive as a local addition!

Circuit Evaluation (Online phase)

3) Multiplication?

How to compute [z]=[xy] ?

Alice, Bob should compute

$$z_A + z_B = (x_A+x_B)(y_A+y_B)$$

$$= x_Ay_A + x_By_A + x_Ay_B + x_By_B$$

How do we compute this?

Alice can compute this

Bob can compute this

Beaver rerandomization technique

Circuit Evaluation (Online phase)

3) Multiplication?

How to compute [z]=[xy] ?

Suppose someone gives us a random multiplication triple

[a],[b],[c=ab]

Circuit Evaluation (Online phase)

3) Multiplication

Output of prepr. *[a],[b],[c]*

Input *[x],[y]*

1. e = Open([x+a])
2. d = Open([y+b])
3. [xy] = [c]+e[y] + d[x] – ed

e,d are “one-time-pad” encryptions of x and y using a and b

Recap

- The problem of **securely computing any function** is equivalent to securely computing **random multiplication triples**!

- If no ***semi-trusted party is available***, we can use **cryptographic assumption** (later today)

What about *active* adversaries?

Remember

- **Input:** A sends B a random value
  //"cheating" = changing input
- **Addition:** only local operations
- **Output:** **parties can reveal wrong shares!**
- **Multiplication:** **2 Open** + local operation

- **Solution:** compute on **authenticated values**!

Authenticated Shares

- **Passive share:** [x] means
  - Alice has x_A, Bob has x_B,

$$x_A + x_B = x$$

- **Authenticated sharing:** [[x]] means
  - Alice has x_A, Bob has x_B,
  - Bob has a MAC key (Δ_B, K_B), Alice has a MAC M_A

$$M_A = \Delta_B \, x_A + K_B$$

  - (Symmetric for Bob)

Authenticated Shares

- **Is the representation [[x]] still linear?**
  Yes, if Δ_A, Δ_B are "global" **keys**

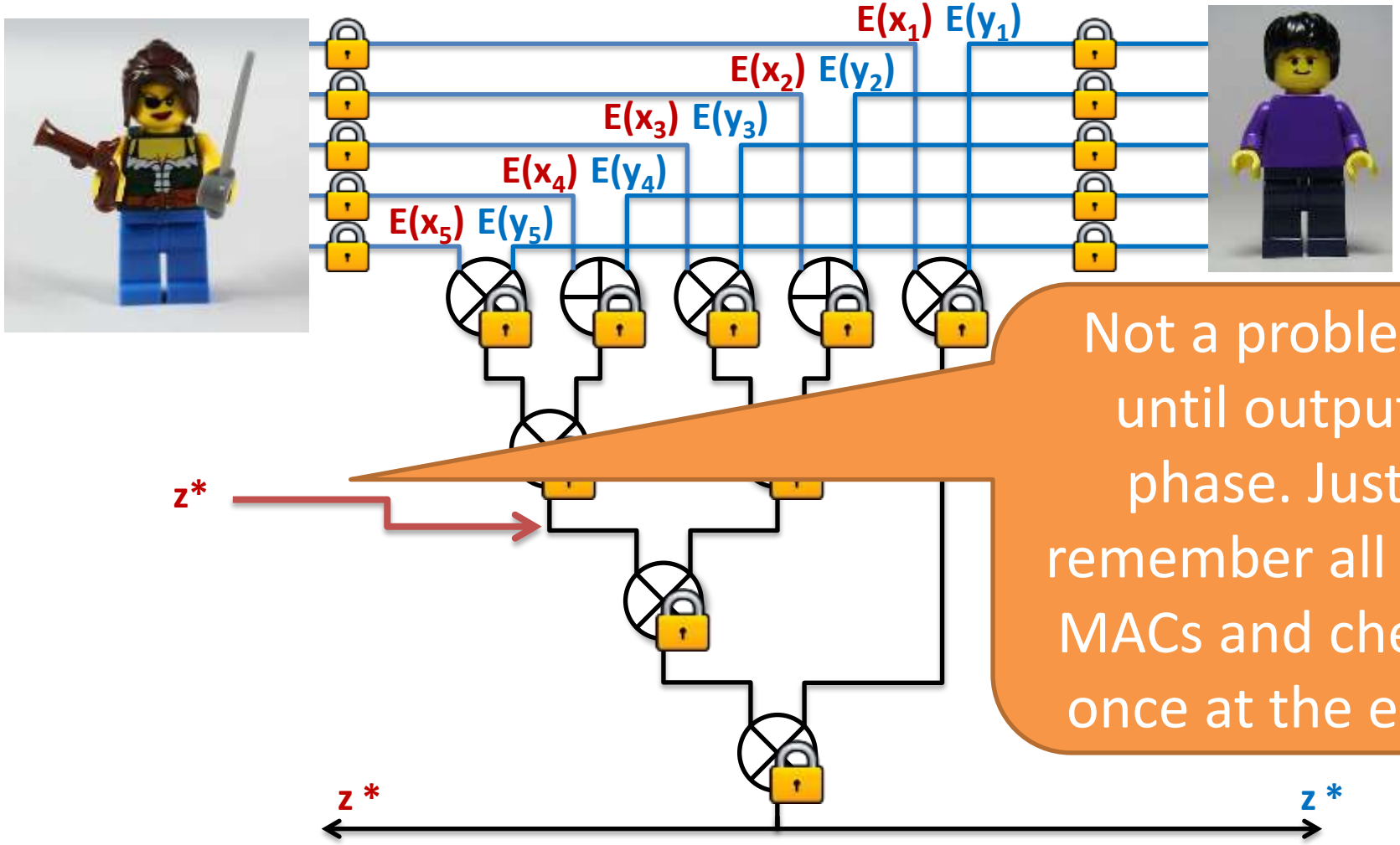
$$[[x1]] = ([x1], (\Delta_A, K1_A, M1_A), (\Delta_B, K1_B, M1_B))$$

$$[[x2]] = ([x2], (\Delta_A, K2_A, M2_A), (\Delta_B, K2_B, M2_B))$$

$$[[x1+x2]] = ([x1+x2], (\Delta_A, K1_A+K2_A, M1_A + M2_A), (\Delta_B, K1_B+K2_B, M1_B + M2_B))$$

Still n equations and n+1 unknown!

Idea: "lazy verification of MACs"



What about multiparty?

- **The size** of [[x]] representation **grows as the square of the number of parties** (every party needs to remember one key/MAC for each other party)

- In SPDZ, a **different authenticated representation** [(x)] is introduced, that **grows linearly** in the number of parties.

SPDZ representation

- $[(x)] = ([x], [M], [\Delta])$ with $M = \Delta x$

  (Δ is global)

- Lazy verification of MACs.
- At the end, everyone opens Δ,M1,M2…, x1,x2,… (simultaneous exchange) and the MACs are verified.

"Lazy verification" is crucial, after Δ is revealed it's easy to forge MACs!

Recap

- **Truth table 2PC**
  - Using MACs to achieve active security.
  - Simpler protocol with perfect, active security
- **Circuit based 2PC**
  - Secure computation **is about generating random multiplications.**
  - Compute on authenticated data for active security.