INDOCRYPT 2016 Tutorial

Efficient Secure Two-Party Computation

Claudio Orlandi, Aarhus University

Plan for the next 3 hours…

- **Part 1: Secure Computation with a Trusted Dealer**
  - Warmup: One-Time Truth Tables
  - Evaluating Circuits with Beaver's trick
  - MAC-then-Compute for Active Security
- **Part 2: Oblivious Transfer**
  - OT: Definitions and Applications
  - Passive Secure OT Extension
  - OT Protocols from DDH (Naor-Pinkas/PVW)
- **Part 3: Garbled Circuits**
  - GC: Definitions and Applications
  - Garbling gate-by-gate: Basic and optimizations
  - Active security 101: simple-cut-and choose, dual-execution

Want more?

- **Cryptographic Computing – Foundations**
  - http://orlandi.dk/crycom
  - Programming & Theory Exercises
  - Will be happy to answer questions by mail!

Online Poker

2♠, 5♠ ,2♥,5 ♥,J♦

Q♠,Q♣,7♣,3♥,2♦

10 ♠,9♣, 8♣,7♦,6♦

3♠, 4♠,7♥,Q ♦,10♦

Poker with Pirates

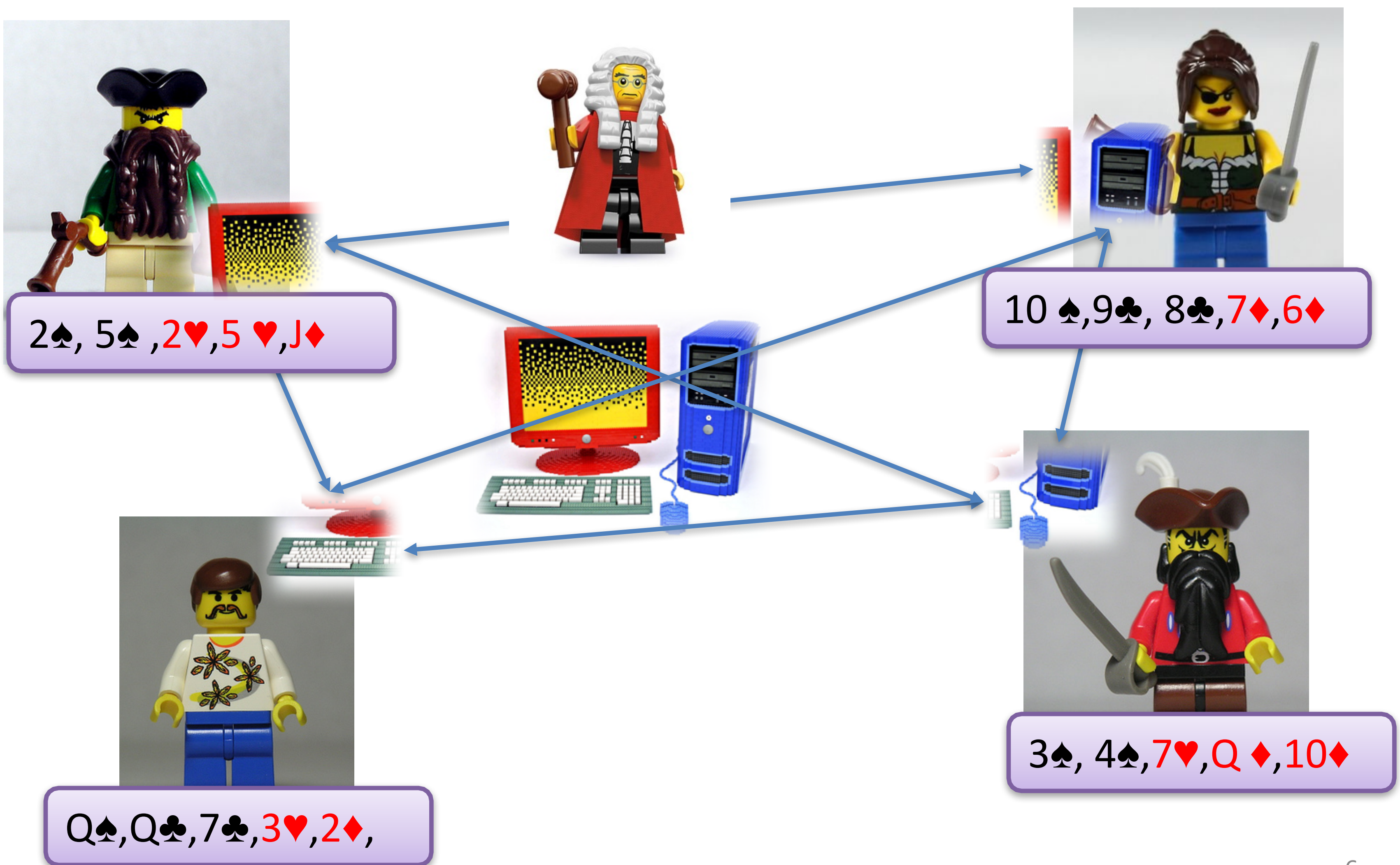
2♠, 5♠ ,2♥,5 ♥,J♦

Q♠,Q♣,7♣,3♥,2♦,

10 ♠,9♣, 8♣,7♦,6♦

A♠,A♣,A♥,A♦,K♦

Secure Computation



Hospitals and Insurances

Syge mister millioner af kroner

Af CHARLOTTE BEDER
Offentliggjort 19.02.09 kl. 08:39

Danskerne går årligt glip af 80 mio. kr., fordi de ikke aner, at de er forsikret ved kritisk sygdom.

Brystundersøgelse. Foto: Colourbox

Relaterede artikler

- Nyt system sikrer syge 80 mio. kr.
- Forsikringsklager i bund
- Boom i sundhedsforsikringer

Hundredvis af alvorligt syge danskere går hvert år glip af millioner af kroner, fordi de ikke har overblik over deres forsikringsdækning.

Derfor kontakter de ikke deres pensions- eller forsikringsselskab, når de bliver ramt af kræft, blodpropper eller anden kritisk sygdom. Og så får de aldrig den check på typisk mellem 50.000 og 200.000 kr., som de har ret til, lyder det fra forsikrings- og pensionsbranchen.

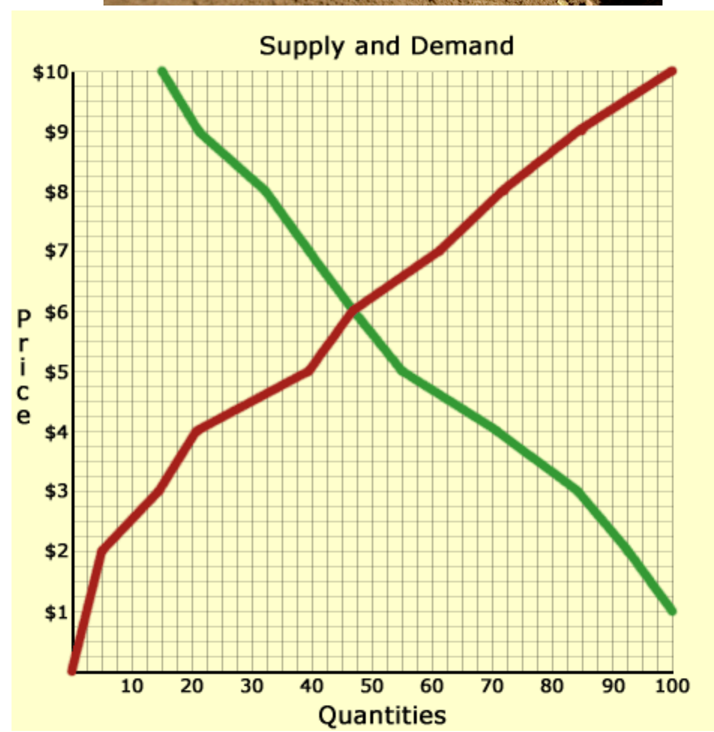
»Forudsætningen er, at systemet skrues sammen på en måde, så selskaberne ikke får andre oplysninger om kunderne, end de bør få. For det enkelte individ må ikke miste kontrollen over egne helbredsoplysninger,« siger jurist Lars Kofod.

- **Problem:** Sick people forget to claim their insurance money
- **Solution:** Insurances and hospitals could periodically compare their data to find and help these people
- **Privacy Issue:** insurance and medical records are sensitive data! No other information than what is strictly necessary must be disclosed!

MPC Goes Live (2008)

Bogetoft et al.
"Multiparty Computation Goes Live"

- January 2008
- **Problem**: determine market price of sugar beets contracts
- 1200 farmers
- Computation: 30 minutes
- *Weak security* ☹
  - *Passive adversary*
  - *Honest majority*



Sharemind

- **Benchmarking**
  - ICT Companies,
  - Public sector
- Satellite collisions
- …

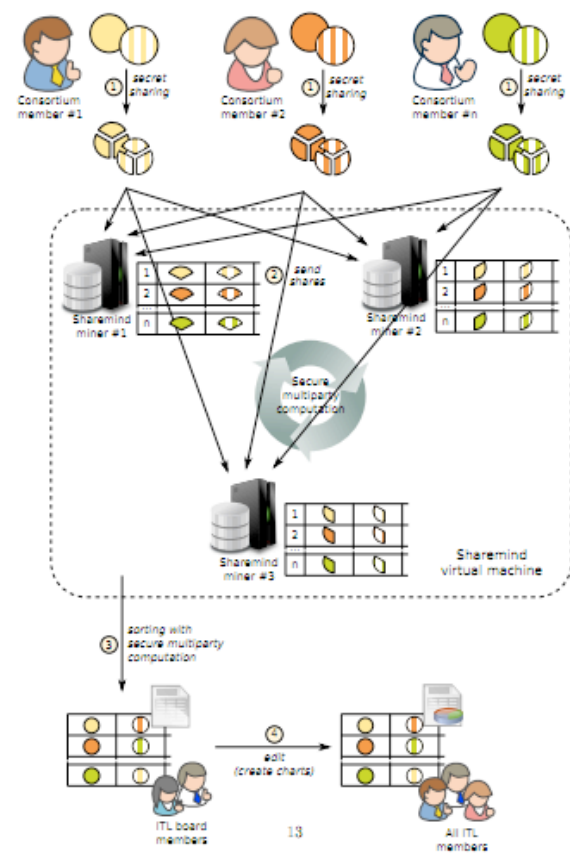
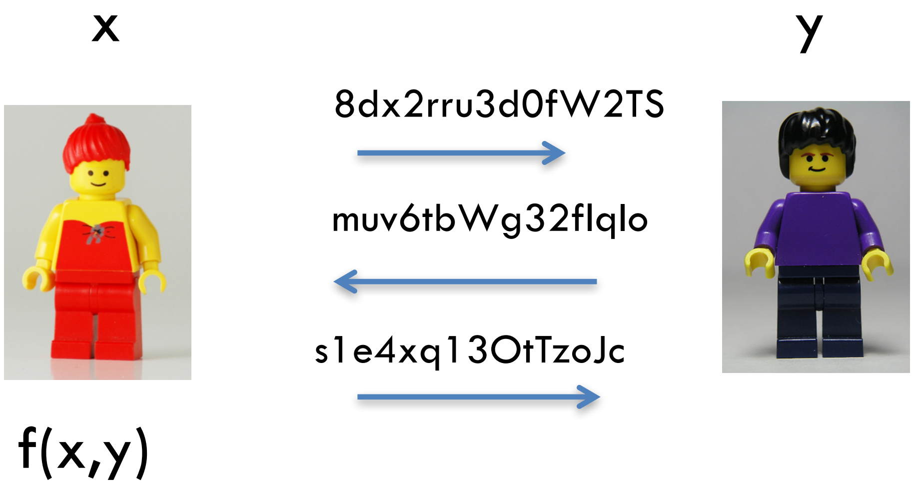


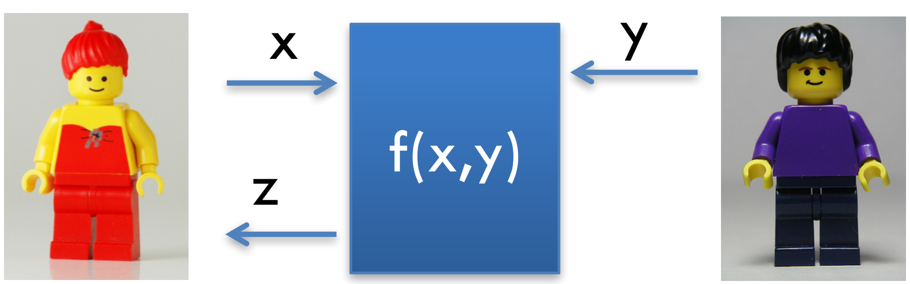
Figure 3: Data flow and visibility in the improved solution using the SHAREMIND framework.

MPC in PRACTICE

- ***Partisia***: Secure auctions
- ***Dyadic Security***: Server breach mitigation
- ***Sharemind***: Benchmarking, satellite collision
- ***SAP***: Private smart-metering
- ***IBM***: Secure cloud computing
- ***Google?***: (lookinf forward to rwc2017)

Secure Computation





- *Privacy*
- *Correctness*
- *Input independence*
- …

What kind of *Secure* Computation?

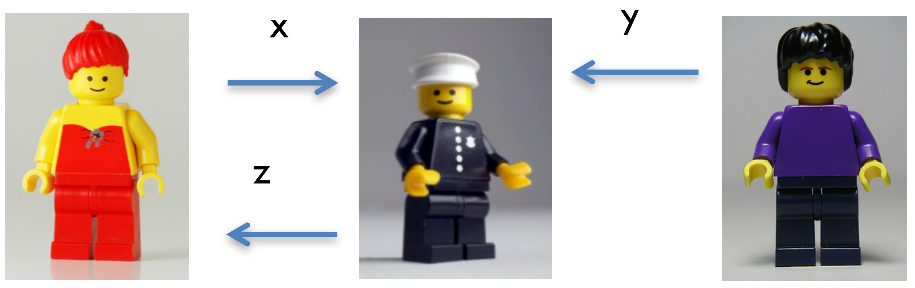
- ***Dishonest majority***
  - The adversary can corrupt up to n-1 participants (n=2).

- ***Static Corruptions***
  - The adversary chooses which party is corrupted before the protocol starts.

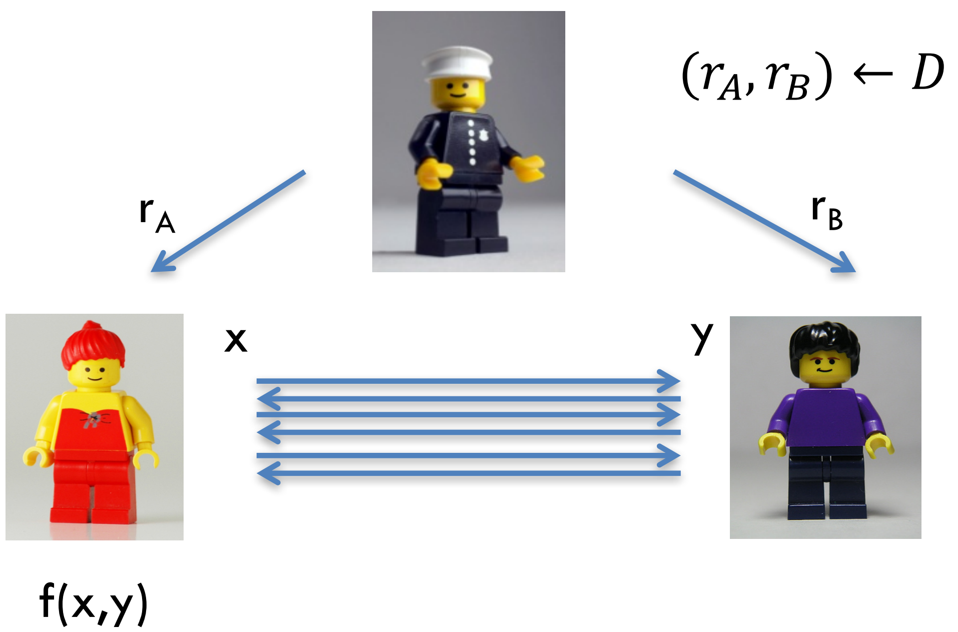
- ***Passive & Active Corruptions***
  - Adversary follows the protocol vs.
    (aka *semi-honest, honest-but-curious*)
  - Adversary can behave arbitrarily
    (aka *malicious, byzantine*)

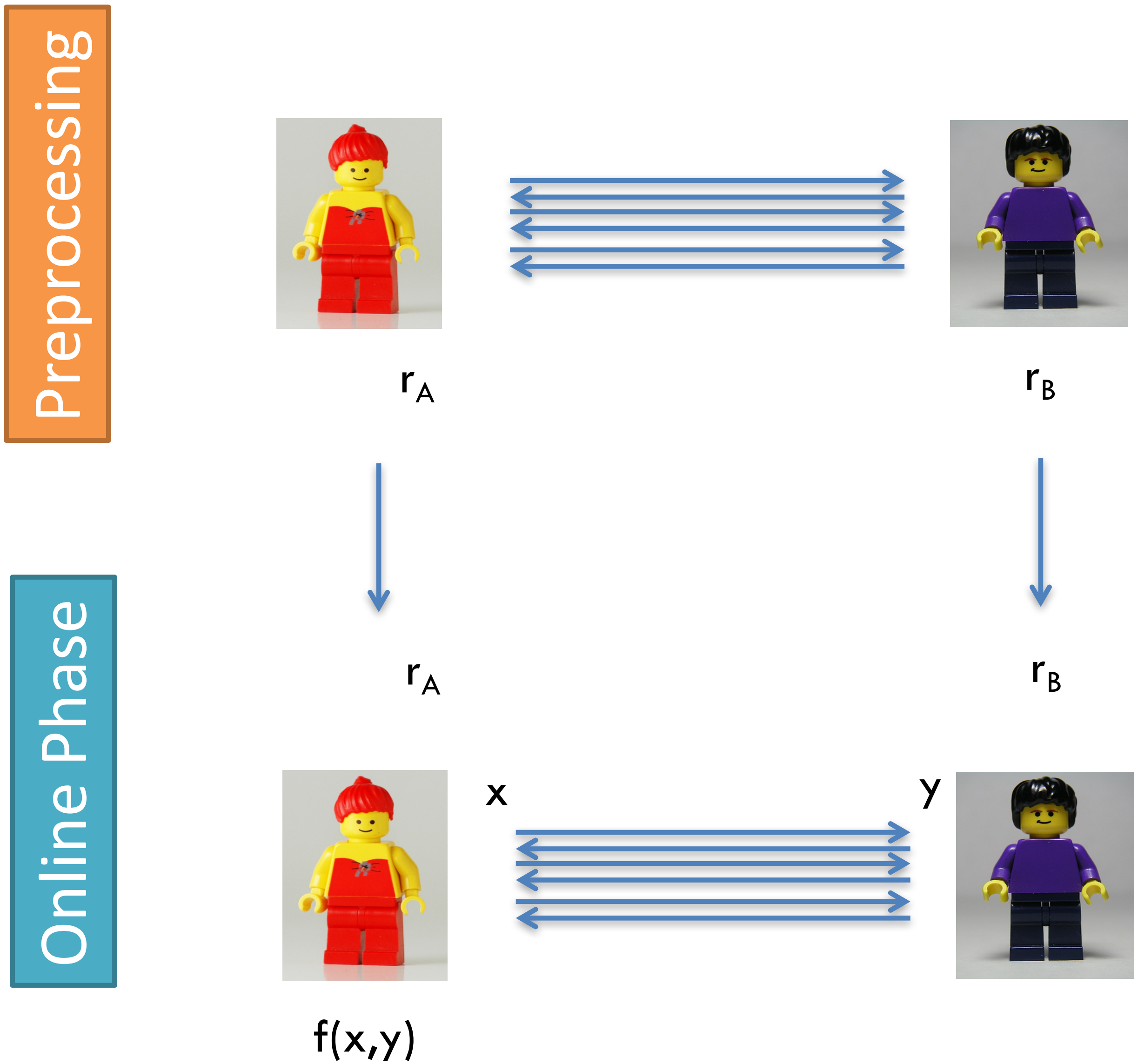
- ***No guarantees of fairness or termination***
  - Security with abort

Trusted Party



Trusted Dealer





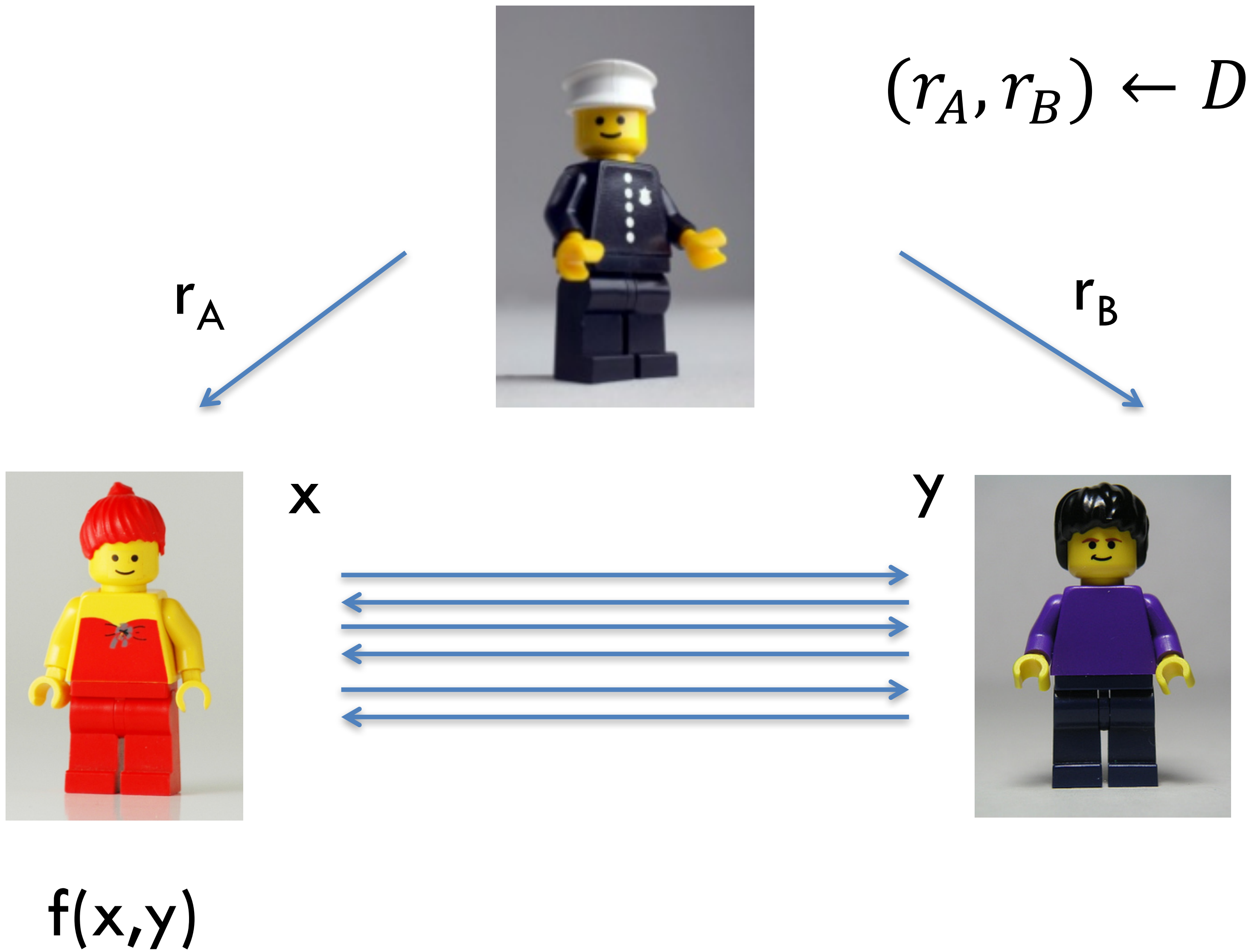
- Independent of ***x,y***
- Tipically only depends on ***size of f***
- Uses public key crypto technology ***(slower)***

- Uses only information theoretic tools ***(order of magn. faster)***

Part 1: Secure Computation with a Trusted Dealer

- **Warmup: One-Time Truth Tables**
- Evaluating Circuits with Beaver's trick
- MAC-then-Compute for Active Security

“The simplest 2PC protocol ever”



“The simplest 2PC protocol ever” OTTT (Preprocessing phase)

1) Write the truth table of the function F you want to compute

y

x		0	1	2	3
	0	3	2	2	2
	1	3	0	0	4
	2	1	0	0	4
	3	1	1	4	4

“The simplest 2PC protocol ever” OTTT (Preprocessing phase)

2) Pick random (r, s), rotate rows and columns

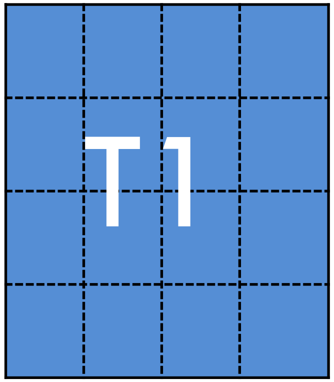
s=3

r=1

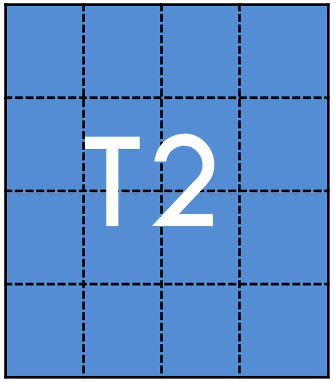
	0	1	2	3
0	1	4	4	1
1	2	2	2	3
2	0	0	4	3
3	0	0	4	1

“The simplest 2PC protocol ever” OTTT (Preprocessing phase)

3) Secret share the truth table i.e.,

Pick 

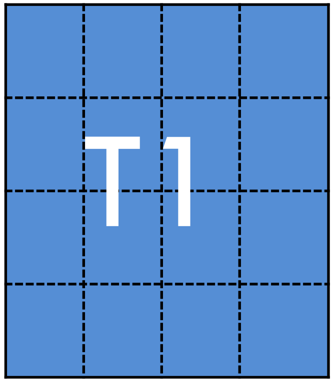
at random, and let



=

1	4	4	1
2	2	2	3
0	0	4	3
0	0	4	1

\-



“The simplest

(c

“Privacy”:
inputs masked w/uniform
random values

T1 , r

T2 , s

$u = x + r$ →

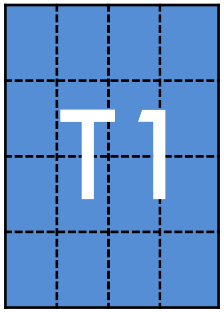
$v = y + s$ ←

T2[u,v] ←

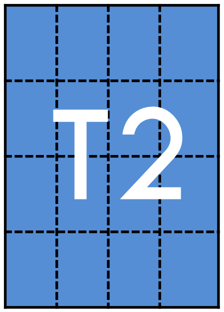
Correctness:
by construction

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

What about active security?



, r



, s

$u = x + r$ →

$v = y + s + e1$ ←

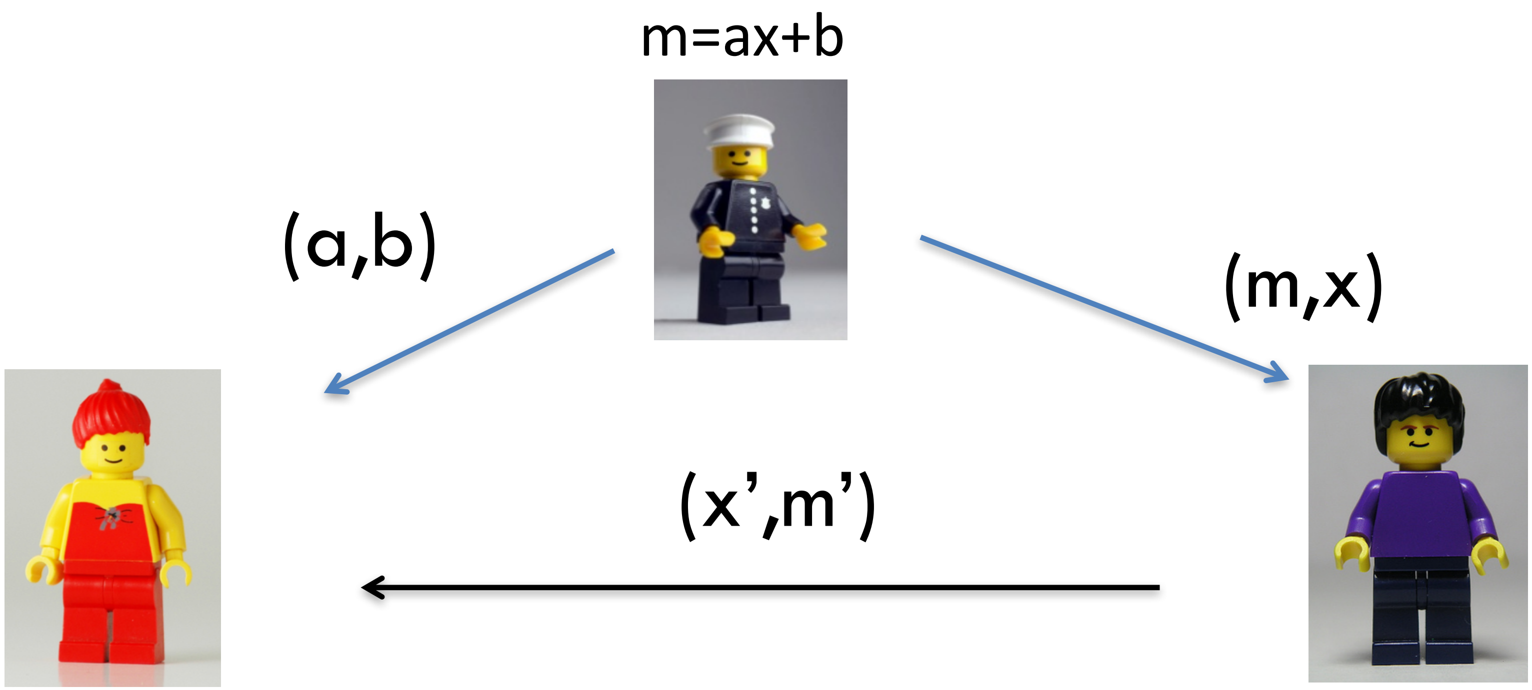
$T2[u,v] + e2$ ←

Is this cheating?

- v = y + s + e1 = (y+e1) + s = y' + s
  - Input substitution, not really cheating a (see formal definition)
- M2[u,v] + e2
  - Changes output to z' = f(x,y) + e2
  - Example: f(x,y)=1 iff x=y *(e.g. pwd check)*
  - e2=1 the output is 1 whp *(login without pwd!)*
    - *Clearly breach of security!*

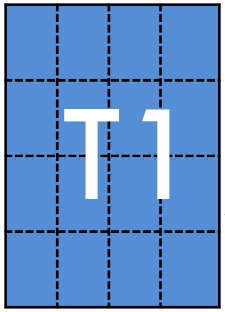
Force Bob to send the right value

- **Problem:** Bob can send the wrong shares
- **Solution:** use MACs
  - e.g. $m = ax+b$ *with* $(a,b) \leftarrow F$

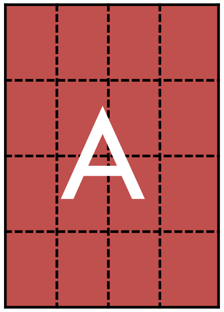


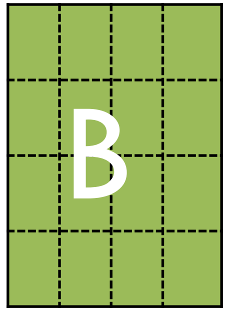
Abort if $m' \neq ax'+b$

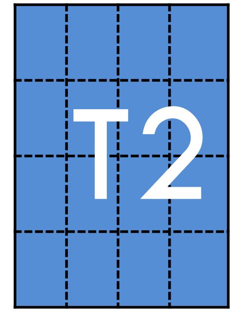
OTTT+MAC



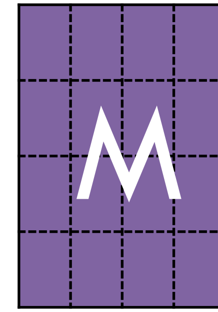
, r







, s



u = x + r →

v = y + s ←

T2[u,v], M[u,v] ←

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

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

else

abort

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

“The simplest 2PC protocol ever” OTTT

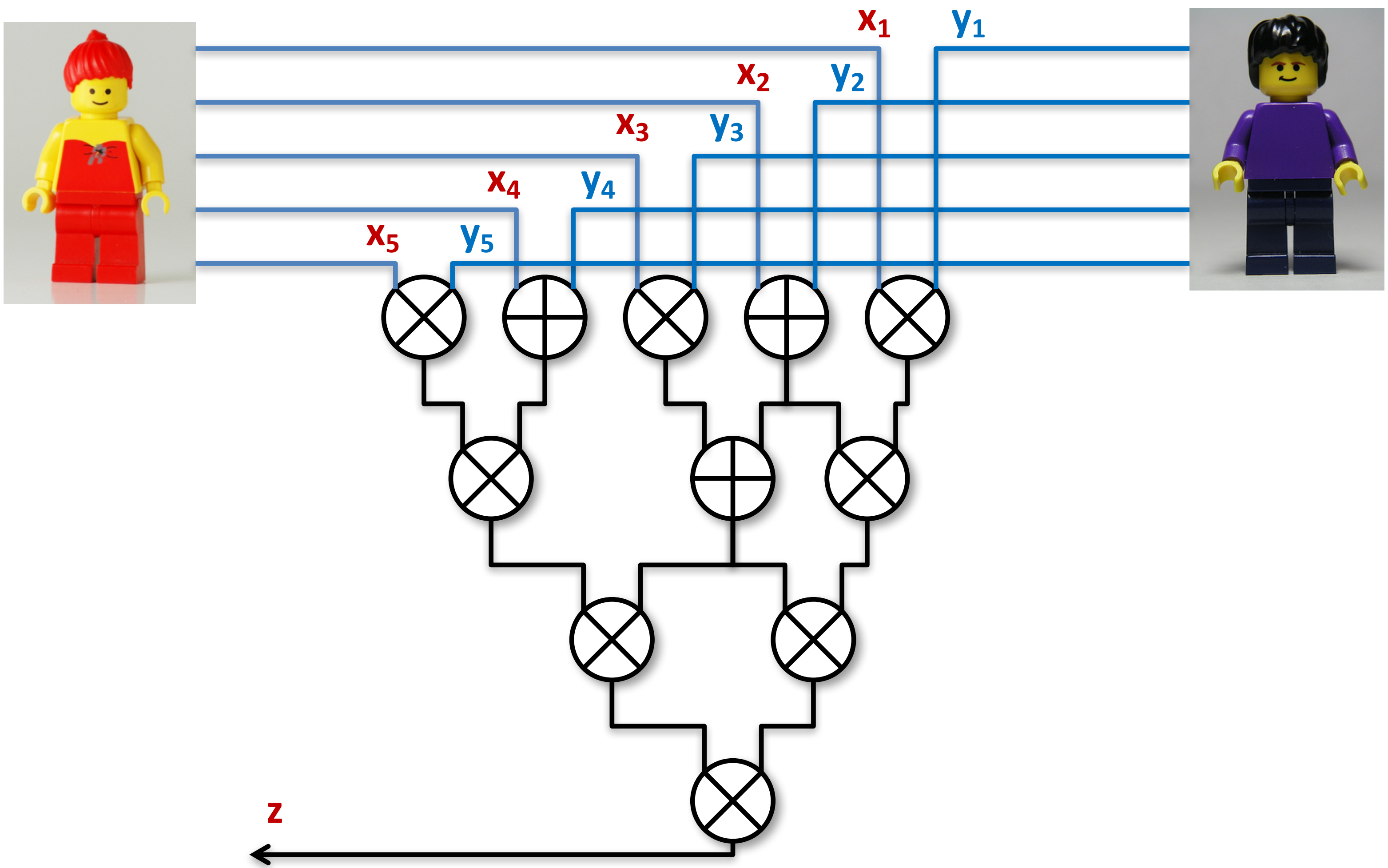
- **Optimal communication complexity ☺**
- **Storage exponential in input size ☹**

➔Represent function using circuit instead of truth table!

Part 1: Secure Computation with a Trusted Dealer

- Warmup: One-Time Truth Tables
- **Evaluating Circuits with Beaver’s trick**
- MAC-then-Compute for Active Security

Circuit based computation



Invariant

- For each ***wire x*** in the circuit we have
  - [x] := (x_A, x_B) // read *"x in a box"*
  - Where Alice holds x_A
  - Bob holds x_B
  - Such that $x_A+x_B=x$

- Notation overload:
  - x is both the r-value and the l-value of x
  - use n(x) for name of x and v(x) for value of x when in doubt.
  - Then [n(x)] = (x_A,x_B) such that x_A+x_B=v(x)

Circuit Evaluation (Online phase)

1) [x] ← Input(A,x) :

- chooses random x_B and send it to Bob
- set $x_A = x + x_B$ // symmetric for Bob

Alice only sends a random bit! "Clearly" secure

2) z ← Open(A,[z]): // z ← Open([z]) if both get output

- Bob sends z_B
- Alice outputs $z = z_A + z_B$ // symmetric for Bob

Alice should learn z anyway! "Clearly" secure

Circuit Evaluation (Online phase)

2) [z]← Add([x],[y]) // at the end 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

"for free" : only a local addition!

Circuit Evaluation (Online phase)

2a) [z]← Mul(a,[x]) // at the end z=a*x

- Alice computes $z_A = a*x_A$
- Bob computes $z_B = a*x_B$

2c) [z]← Add(a,[x]) // at the end z=a+x

- Alice computes $z_A = a+x_A$
- Bob computes $z_B = x_B$

Circuit Evaluation (Online phase)

3) Multiplication?

How to compute [z]=[xy] ?

How do we compute this?

Alice, Bob should compute

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

$$= x_A y_A + x_B y_A + x_A y_B + x_B y_B$$

Alice can compute this

Bob can compute this

Circuit Evaluation (Online phase)

3) [z]←Mul([x],[y]):

1. Get *[a],[b],[c]* with *c=ab* from trusted dealer

2. *e=Open([a]+[x])*
3. *d=Open([b]+[y])*

Is this secure?
e,d are "one-time-pad" encryptions of x and y using a and b

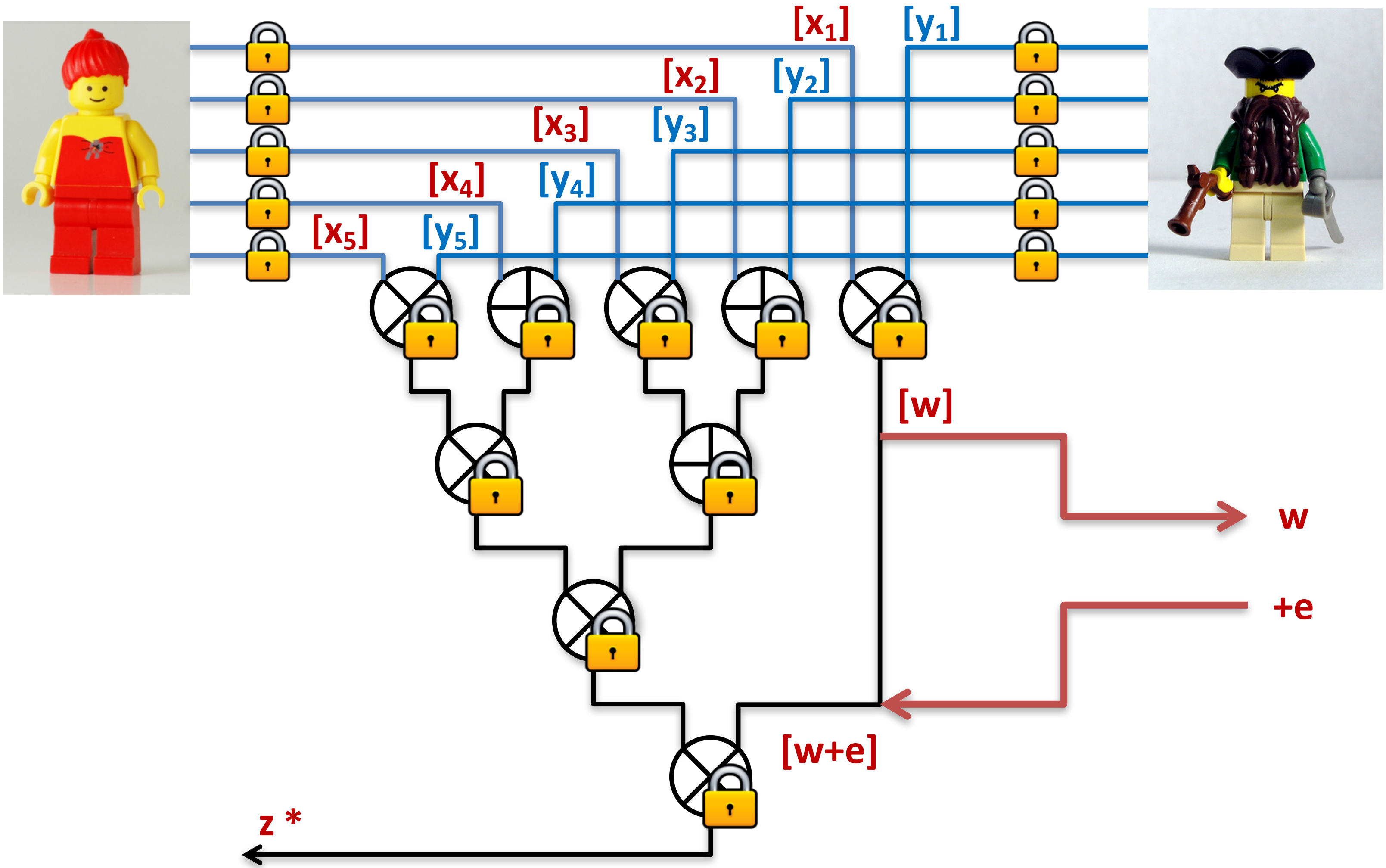
4. Compute $[z] = [c] + e[y] + d[x] - ed$

$$ab + (ay+xy) + (bx+xy) - (ab+ay+bx+xy)$$

Part 1: Secure Computation with a Trusted Dealer

- Warmup: One-Time Truth Tables
- Evaluating Circuits with Beaver's trick
- **MAC-then-Compute for Active Security**

Secure Computation



Active Security?

- **“Privacy?”**
  - even a malicious Bob does not learn anything ☺
- **“Correctness?”**
  - a corrupted Bob can change his share during any “Open” (both final result or during multiplication) leading the final output to be incorrect ☹

Problem

2) z ← Open(A,[z]):

- Bob sends z_B +e
- Alice outputs $z=z_A+z_B$ +e // error change output distribution in way that cannot be simulated by input substition

Solution: add MACs

2) z ← Open(A,[z]):

- Bob sends $z_{B,}$ m_B
- Alice outputs
  - $z = z_A + z_B$ if $m_B = z_B \Delta_A + k_A$
  - "abort" otherwise

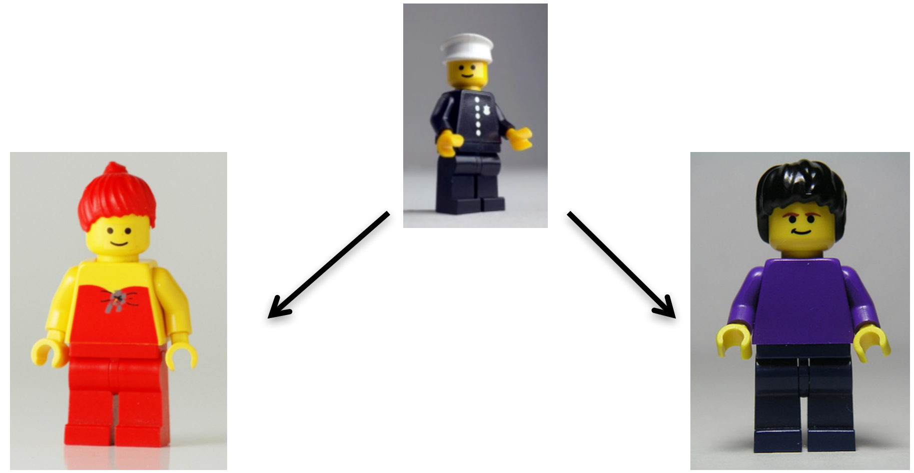
- **Solution:** Enhance representation [x]
  - $[x] = (\ (x_A, k_A, m_A)\ ,\ (x_B, k_B, m_B)\ )$ s.t.
  - $m_B = x_B \Delta_A + k_A$ (symmetric for m_A)
  - $\Delta_{A,} \Delta_B$ is the same for all wires.

Linear representation

- Given
  - $[x] = ((x_A, k_{Ax}, m_{Ax}) , (y_B, k_{Bx}, m_{Bx}))$
  - $[y] = ((y_A, k_{Ay}, m_{Ay}) , (y_B, k_{By}, m_{By}))$
  - Compute $[z] = ($
    $(z_A = x_A + y_A \;,\; k_{Az} = k_{Ax} + k_{Ay} \;,\; m_{Az} = m_{Ax} + m_{Ay})$,
    $(z_B = x_B + y_B \;,\; k_{Bz} = k_{Bx} + k_{By} \;,\; m_{Bz} = m_{Bx} + m_{By})$,)

- And [z] is in the right format since…
  $m_{Bz} = (m_{Bz} + m_{By}) = (k_{Ax} + x_B\Delta_A) + (k_{Ay} + y_B\Delta_A)$
  $= (k_{Ax} + k_{Ay}) + (x_B + y_B)\Delta_A = k_{Az} + z_B\Delta_A$

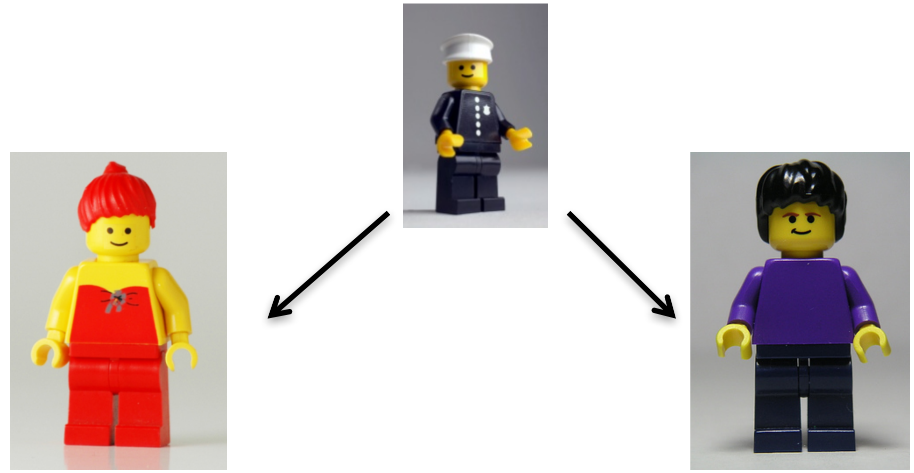
Recap



1. **Output Gates:**
   – Exchange shares and MACs
   – Abort if MAC does not verify
2. **Input Gates**:
   – Get a random [r] from ***trusted dealer***
   – r ← Open(A,[r])
   – Alice sends Bob *d=x-r,*
   – Compute [x]=[r]+d

Allows simulator to extract x* = r+d*

Recap



1. Addition Gates:

– Use linearity of representation to compute

$$[z]=[x]+[y]$$

2. Multiplication gates:

– Get a random triple [a][b][c] with c=ab from 

– $e \leftarrow \mathrm{Open}([a]+[x])$, $d \leftarrow \mathrm{Open}([b]+[y])$

– Compute $[z] = [c] + a[y] + b[x] - ed$

Final remarks

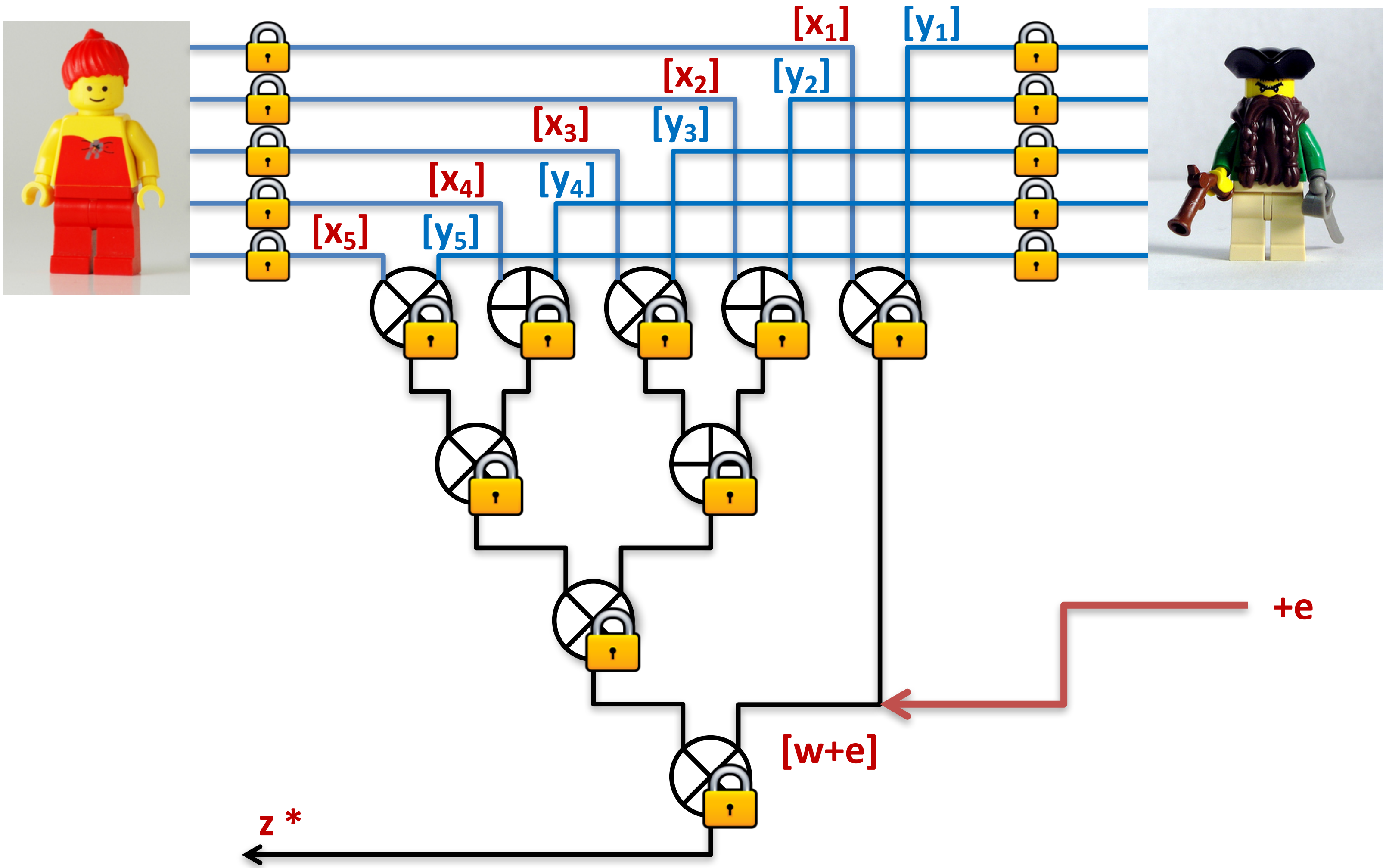
- Size of MACs
- Lazy MAC checks

Size of MACs

1. Each party must store a mac/key pair ***for each other party***
   - quadratic complexity! ☹
   - SPDZ for linear complexity.

2. MAC is only as hard as guessing key!
   k MACs in parallel give security $\boldsymbol{1/|F|^k}$
   - In *TinyOT* $\mathbf{F=Z_2}$, then MACs/Keys are *k-bit strings*
   - *MiniMACs* for constant overhead

Lazy MAC Check

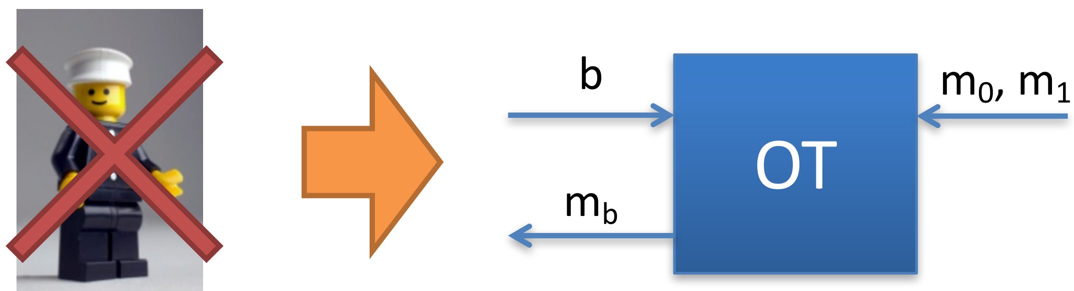


Recap of Part 1

- Two protocols *"in the trusted dealer model"*
  - **One Time-Truth Table**
    - **Storage** exp(input size) ☹
    - **Communication** O(input size) ☺
    - 1 **round** ☺
  - **(SPDZ)/BeDOZa/TinyOT online phase**
    - **Storage** linear #number of AND gates
    - **Communication** linear #number of AND gates
    - **#rounds** = depth of the circuit
  - ...and add enough **MACs** to get **active security**

Recap of Part 1

- To do secure computation is enough to precompute enough **random multiplications**!



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

Plan for the next 3 hours…

- **Part 1: Secure Computation with a Trusted Dealer**
  - Warmup: One-Time Truth Tables
  - Evaluating Circuits with Beaver’s trick
  - MAC-then-Compute for Active Security
- **Part 2: Oblivious Transfer**
  - OT: Definitions and Applications
  - Passive Secure OT Extension
  - OT Protocols from DDH (Naor-Pinkas/PVW)
- **Part 3: Garbled Circuits**
  - GC: Definitions and Applications
  - Garbling gate-by-gate: Basic and optimizations
  - Active security 101: simple-cut-and choose, dual-execution

Circuit Evaluation (Online phase)

3) Multiplication?

How to compute [z]=[xy] ?

How do we compute this?

Alice, Bob should compute

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

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

Alice can compute this

Bob can compute this

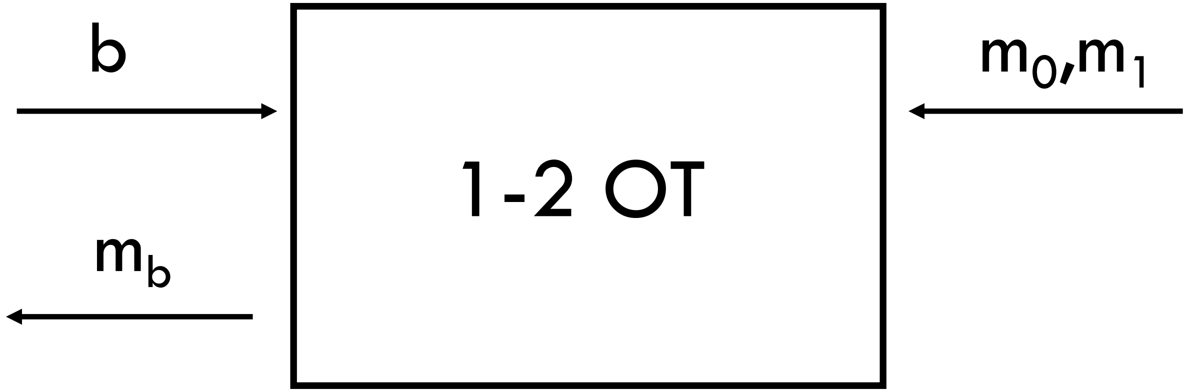
Part 2: Oblivious Transfer

- **OT: Definition, Applications (Gilboa's protocol)**
- OT Protocols from DDH (Naor-Pinkas/PVW)
- Passive Secure OT Extension

1-2 OT

Receiver

Sender

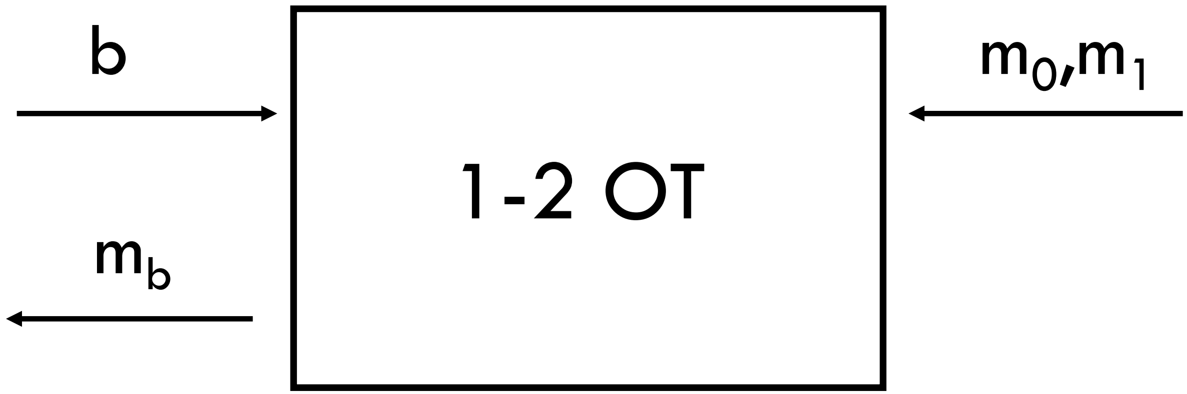


- Receiver does not learn m_{1-b}
- Sender does not learn b

Receiver

1-2 OT

Sender

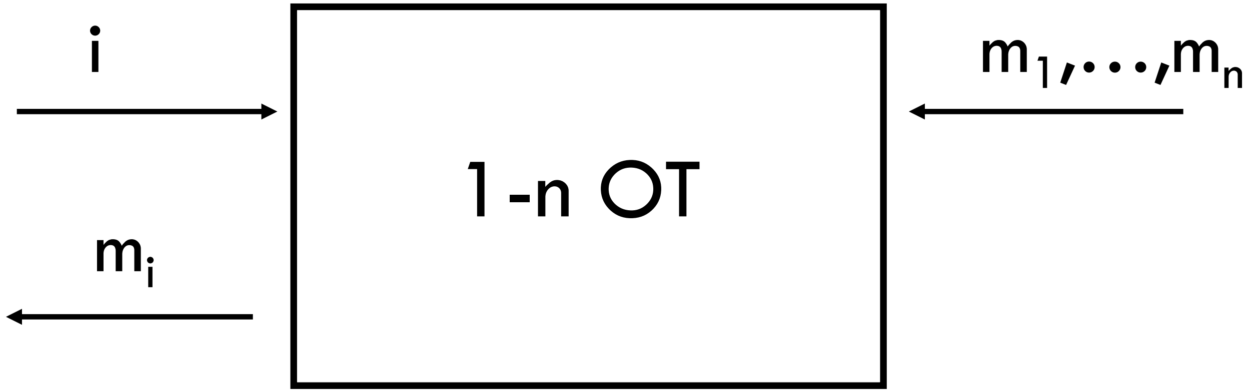


- $m_b = (1\text{-}b)\ m_0 + b\ m_1$
- $m_b = m_0 + b\ (m_1 - m_0)$

1-n OT

Receiver

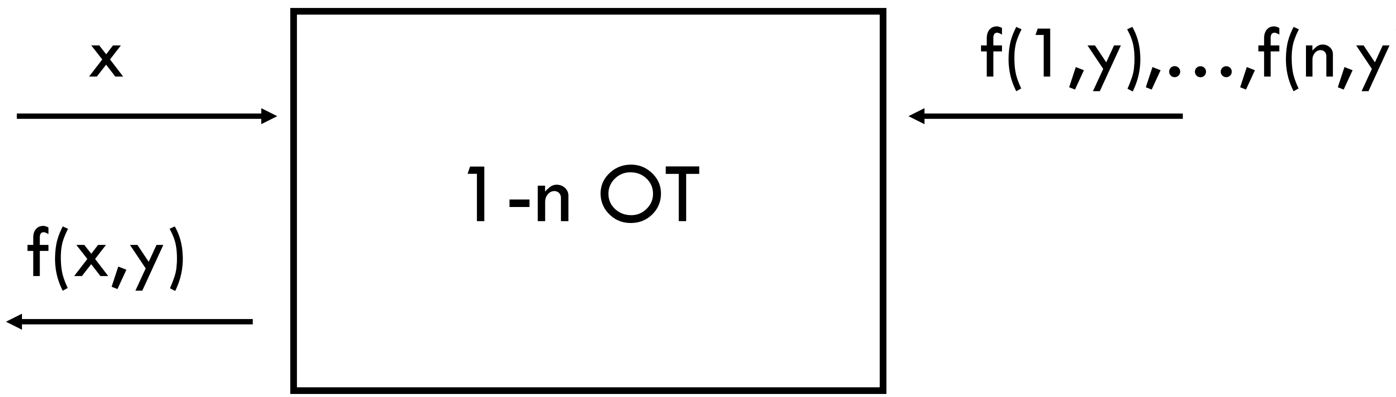
Sender



2PC via 1-n OT

Receiver

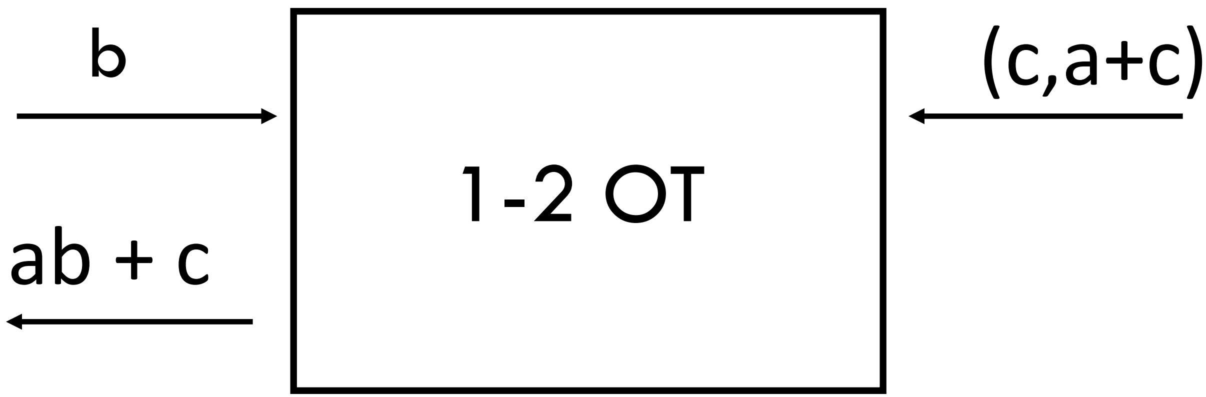
Sender



Oblivious Transfer = bit multiplication


Receiver


Sender



GILBOA’S PROTOCOL

n OTs = Arith. Multiplication

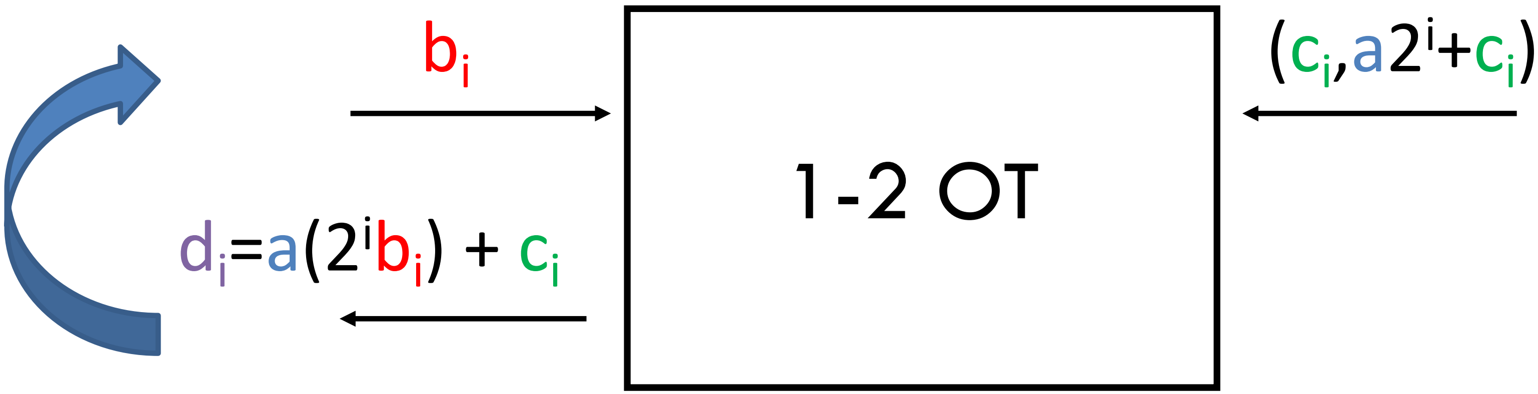
Receiver

$b=(b_0,b_1,\ldots,b_{n-1})$

Sender

a (n bit number)

$c_0+\ldots+c_{n-1}=c$



$$d_0+\ldots+d_{n-1}=a(b_0+2b_1+\ldots+2^{n-1}b_{n-1})+(c_0+\ldots+c_{n-1})=ab+c$$

Part 2: Oblivious Transfer

- OT definition, applications (Gilboa's protocol)
- **OT Protocols from DDH (Naor-Pinkas/PVW)**
- Passive Secure OT Extension (IKNP03)

Passive Secure OT

Receiver(b) Sender(m_0,m_1)

Receiver privacy:
Real pk ≈ “random” pk

$pk_b \leftarrow G(sk)$

$pk_{1\text{-}b} \leftarrow Rand()$

(pk_0,pk_1) →

$c_0=E(pk_0,m_0)$, $c_1=E(pk_1,m_1)$ ←

$m_b = D(sk,c_b)$

Sender privacy:
encryption is secure
(Alice does not have sk)

Passive Secure OT

Malicious

Receiver(b)

Sender(m_0,m_1)

$pk_0 \leftarrow G(sk_0)$
$pk_1 \leftarrow G(sk_1)$

(pk_0,pk_1) →

← c_0=E(pk_0,m_0), c_1=E(pk_1,m_1)

$m_0 \leftarrow D(sk_0, c_0)$
$m_1 \leftarrow D(sk_1, c_1)$

Active Secure OT

Receiver(b) | Sender(m_0,m_1)

crs

mpk ← f(crs,sk,b)

mpk

$(pk_0,pk_1)=G(mpk,crs)$

$c_0=E(pk_0,m_0)$, $c_1=E(pk_1,m_1)$

$m_b = D(sk,c_b)$

Keys are correlated,
Receiver cannot learn
the sk for both

Naor-Pinkas OT

(a la Chou-Orlandi)

Receiver(b)

Sender(m_0,m_1)

crs = h (single group element)

mpk = $g^{sk}h^b$

From
$pk_0=g^{sk0}$
$pk_1=g^{sk1}$

$h=pk_0/pk_1$

→
$h=g^{sk0-sk1}$

mpk

pk_0=mpk
pk_1=mpk/h

$c_0=E(pk_0,m_0)$, $c_1=E(pk_1,m_1)$

Encryption
is ElGamal

Part 2: Oblivious Transfer

- OT definition, applications (Gilboa’s protocol)
- OT Protocols from DDH (Naor-Pinkas/PVW)
- **Passive Secure OT Extension (IKNP03)**

Efficiency

- ***Problem:*** OT requires public key primitives, inherently efficient

The Crypto Toolbox

Weaker assumption

Stronger assumption

OTP >> SKE >> PKE >> FHE >> Obfuscation

More efficient

Less efficient

Efficiency

- ***Problem:*** OT requires public key primitives, inherently efficient

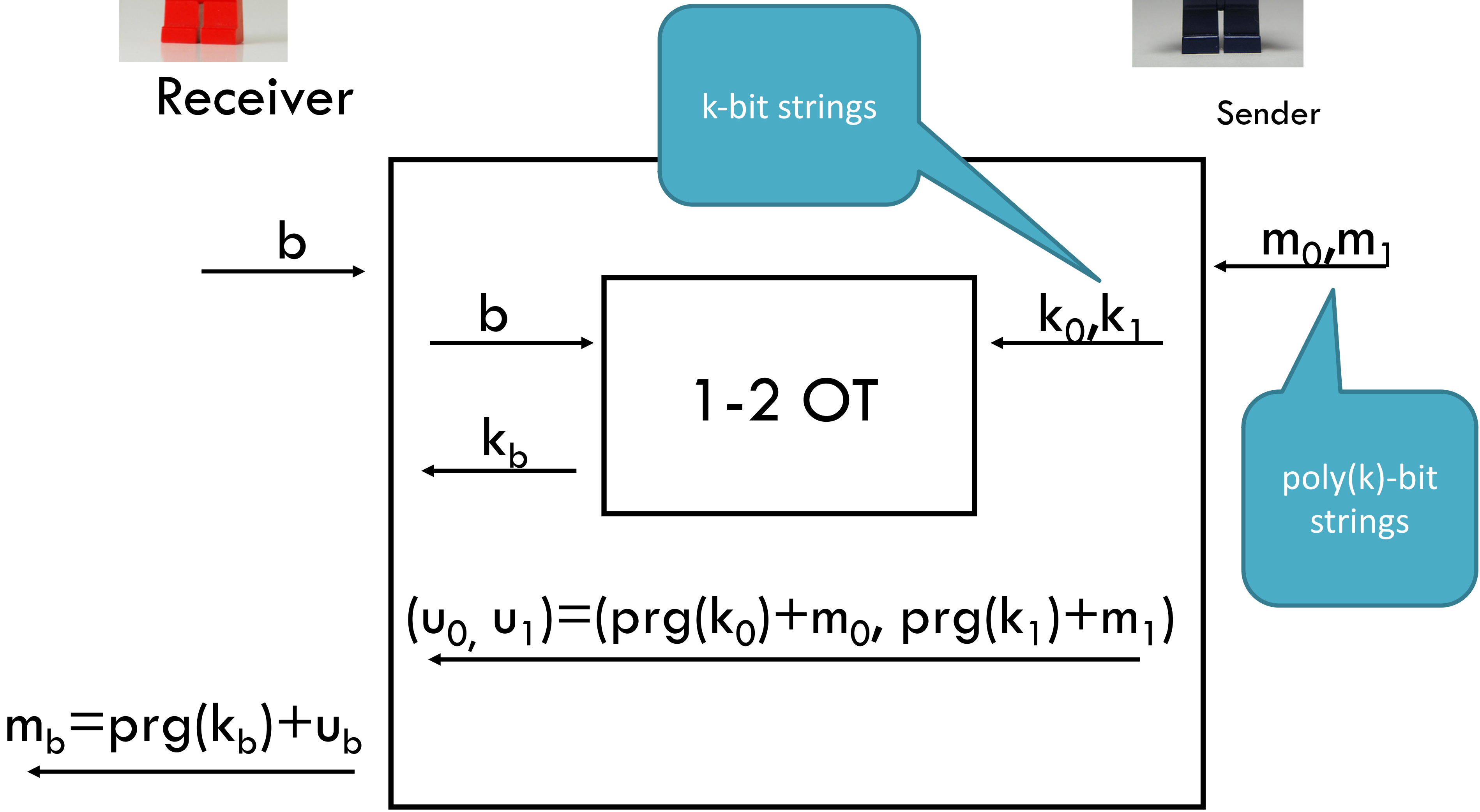
- ***Solution:*** OT extension
  - Like hybrid encryption!
  - Start with few (expensive) OT based on PKE
  - Get many (inexpensive) OT using only SKE

WARMUP: USEFUL OT PROPERTIES

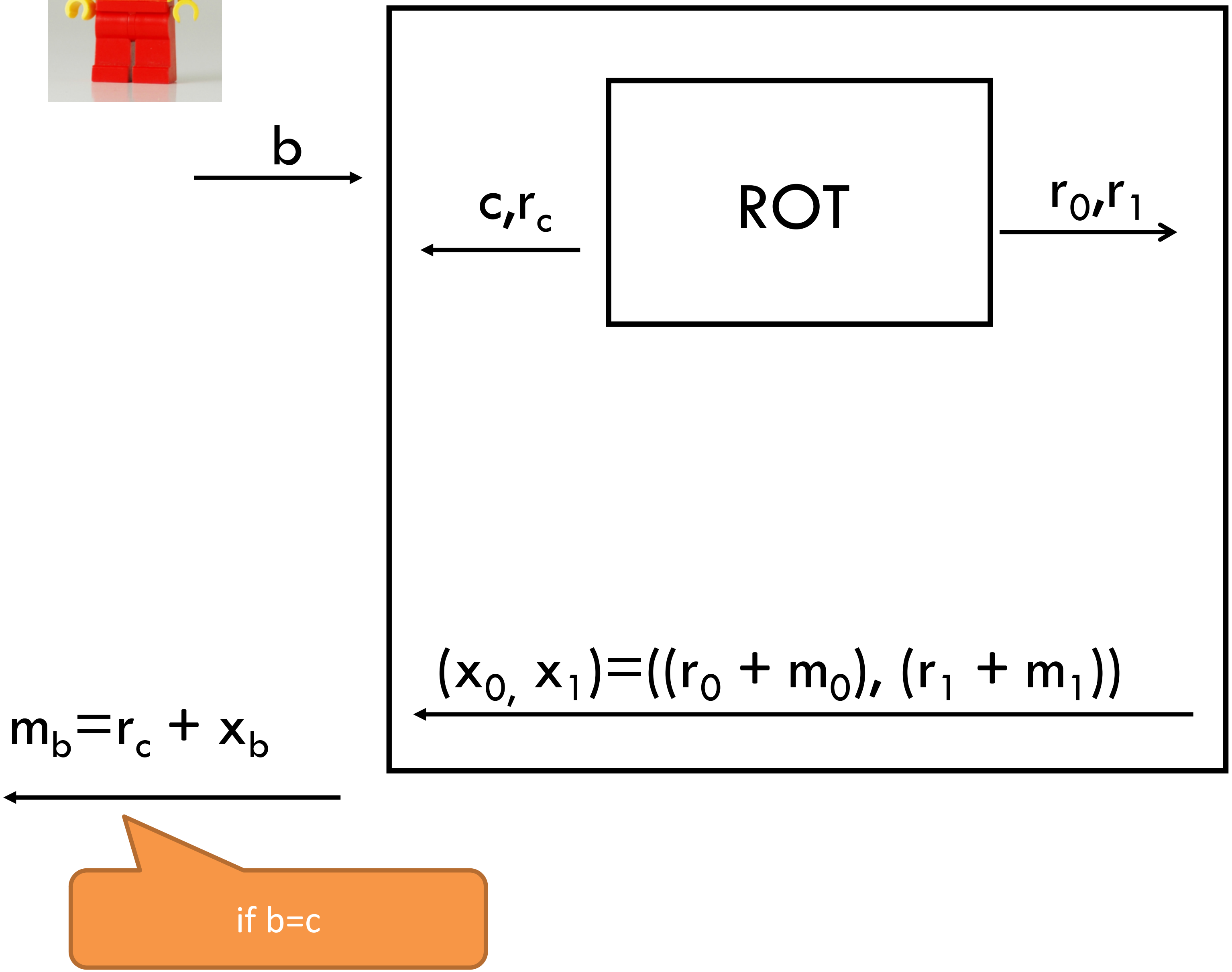
Short OT → Long OT

Receiver

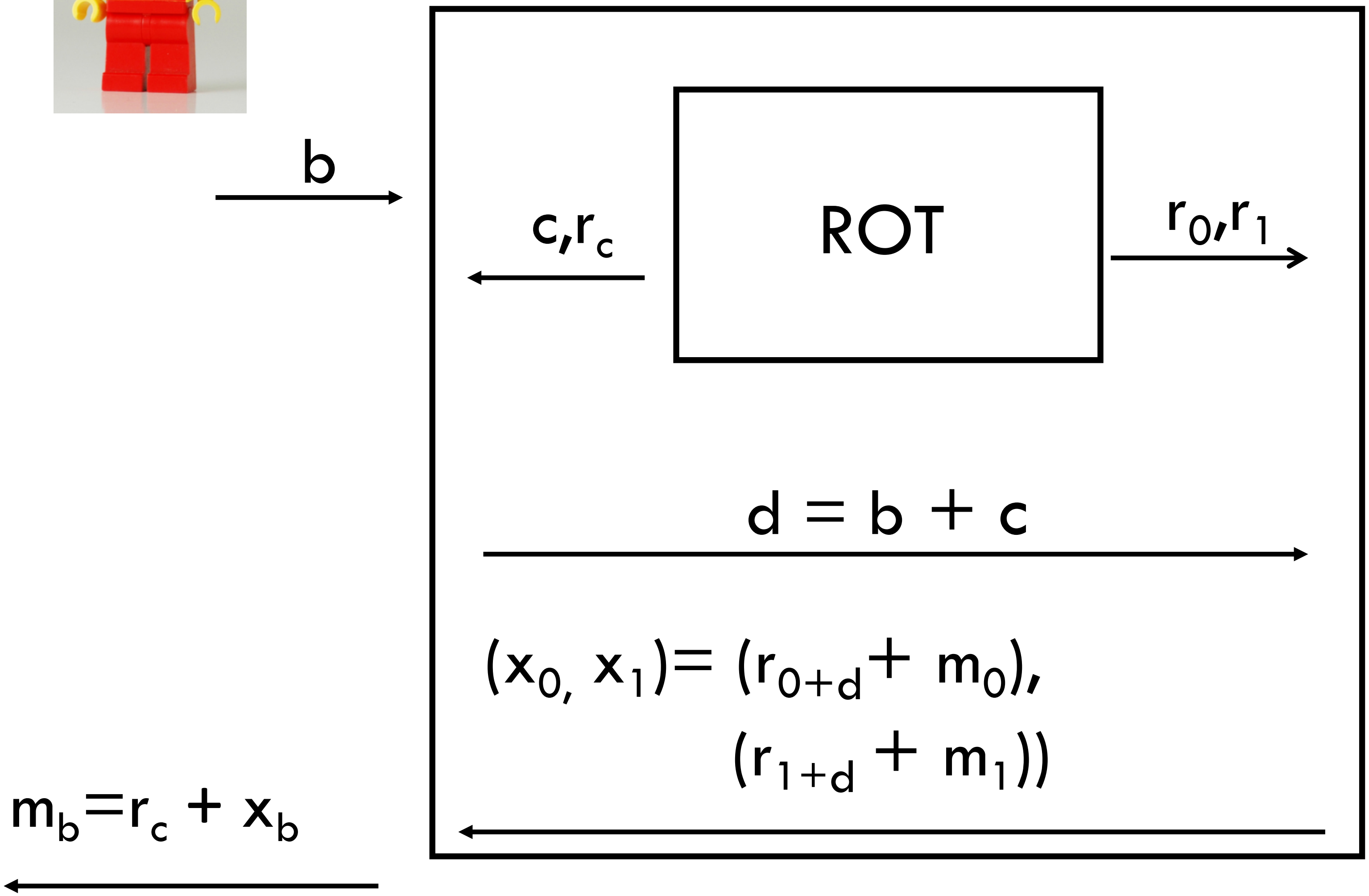
Sender



Random OT = OT

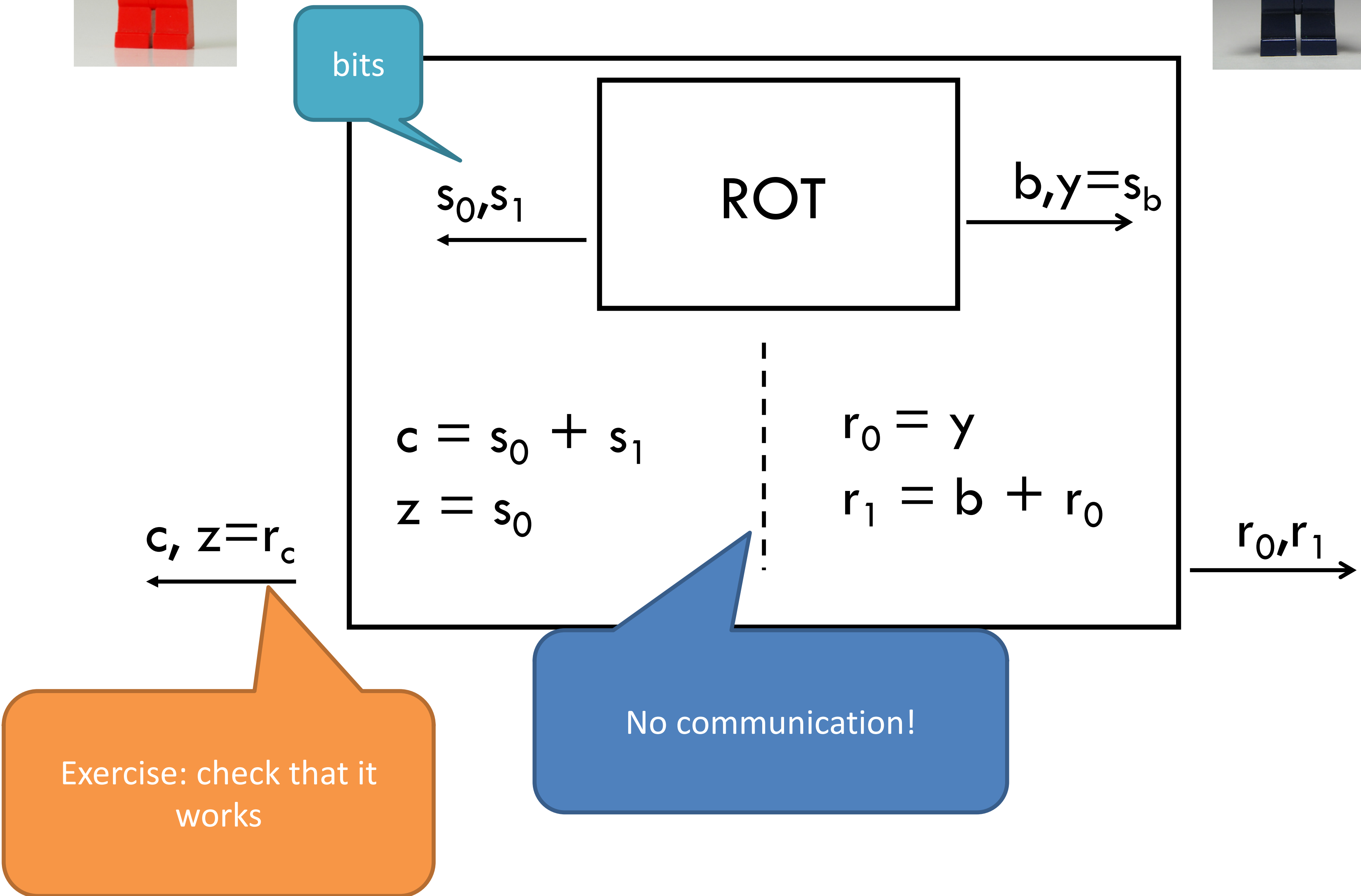


Random OT = OT



Exercise: check that it works!

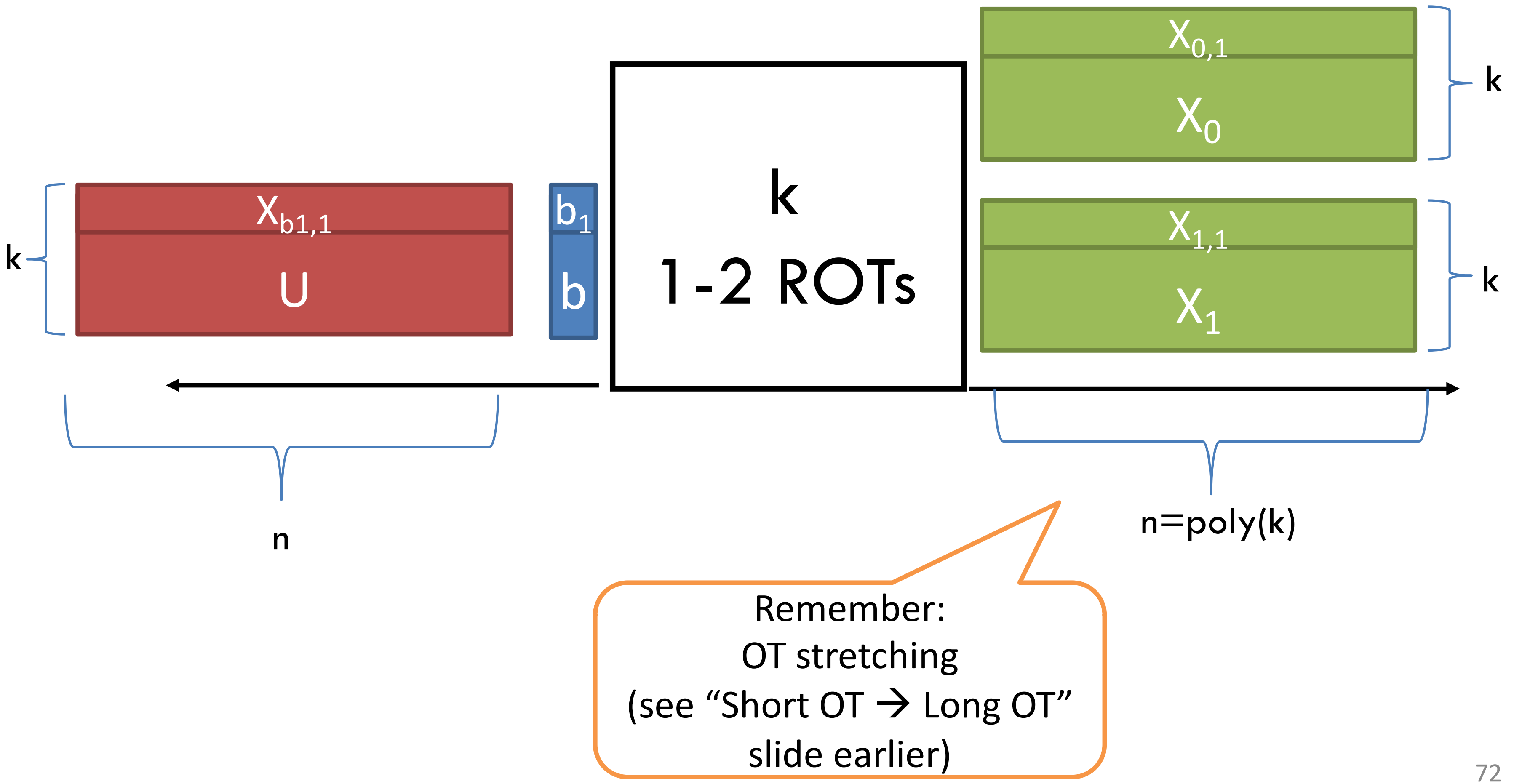
(R)OT is symmetric



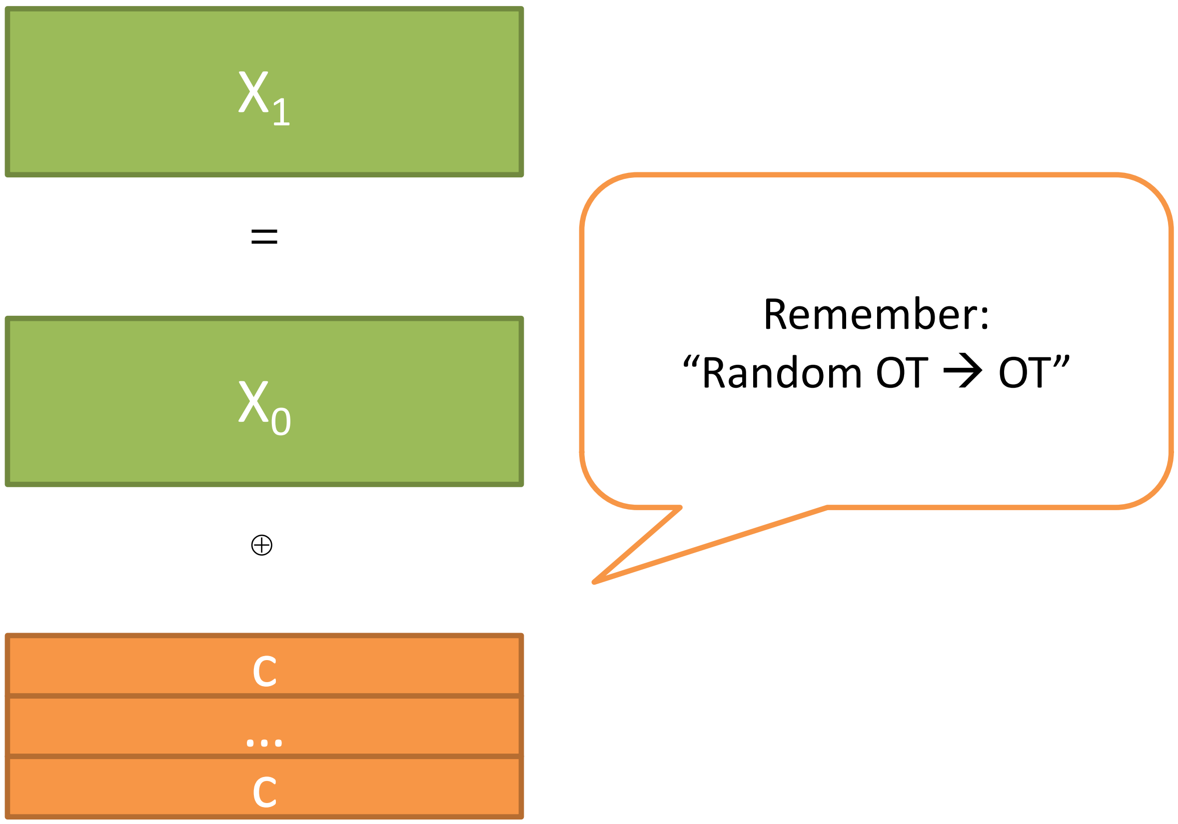
OT Extension

- OT pro(v/b)ably requires public-key primitivies
  - OT extension ≈ hybrid encryption
  - **Start from k “real” OTs**
  - **Turn them into poly(k) OTs using only few symmetric primitives per OT**

OT Extension, Pictorially

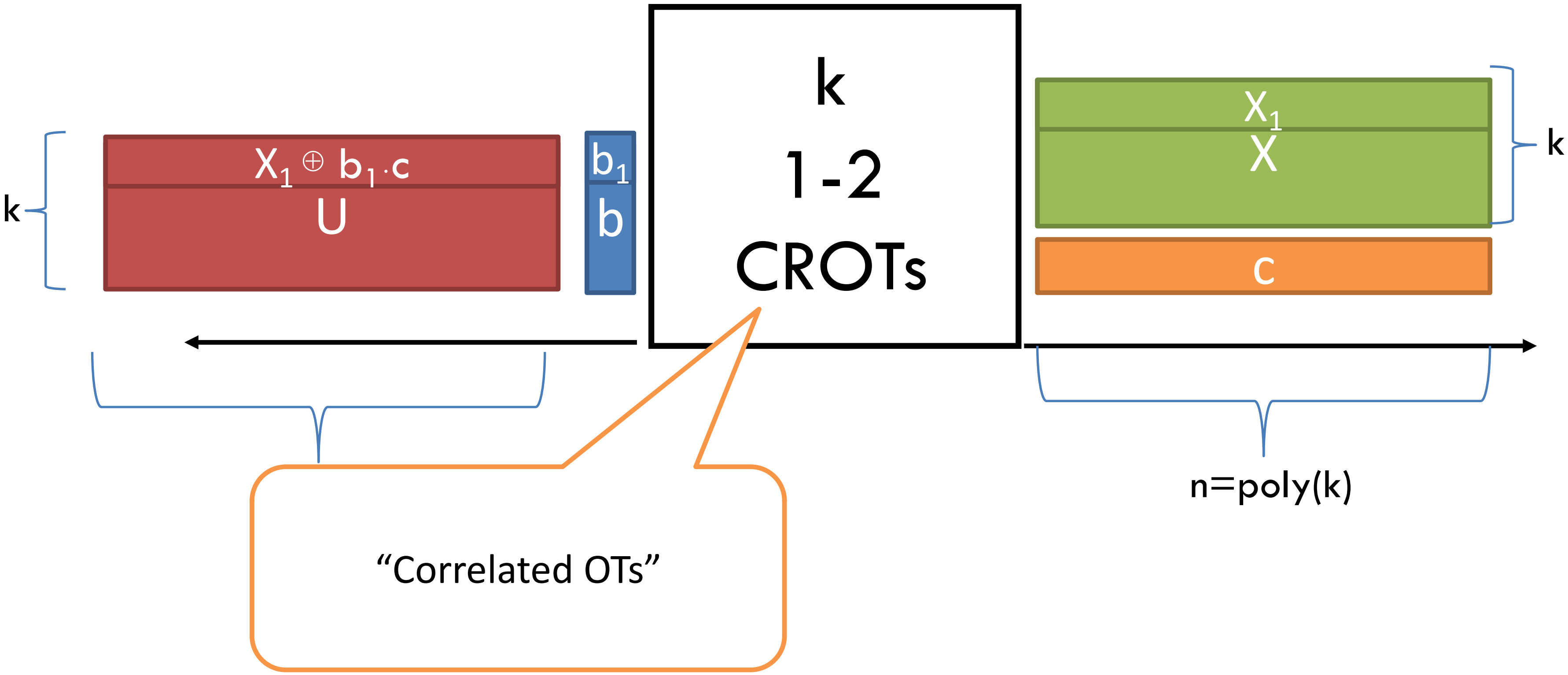


Condition for OT extension

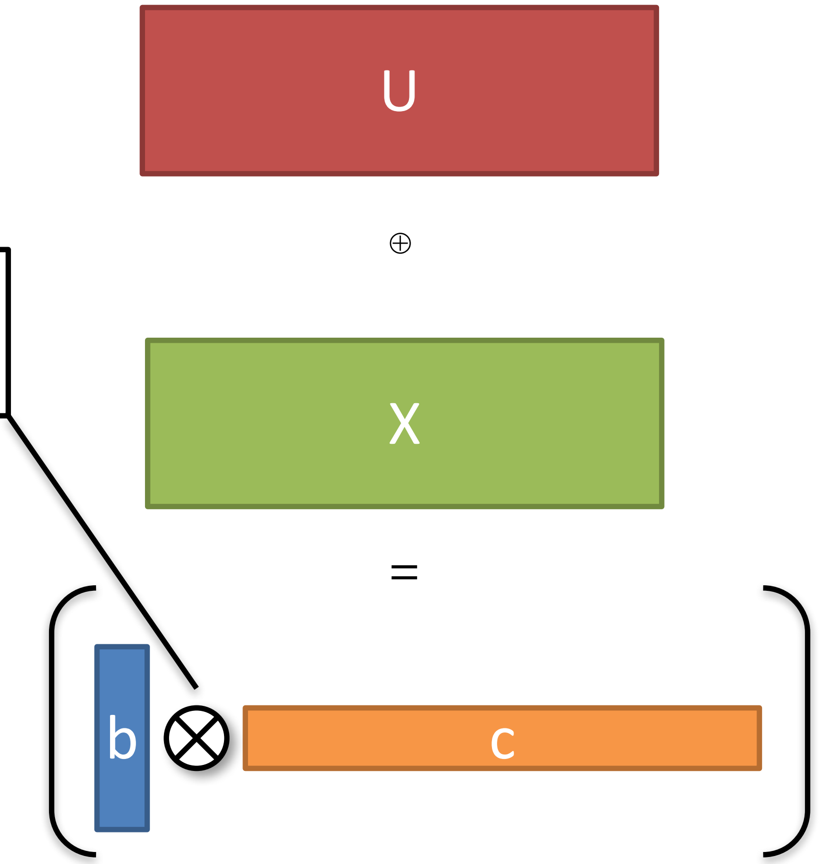


Problem for active security!

OT Extension, Pictorially

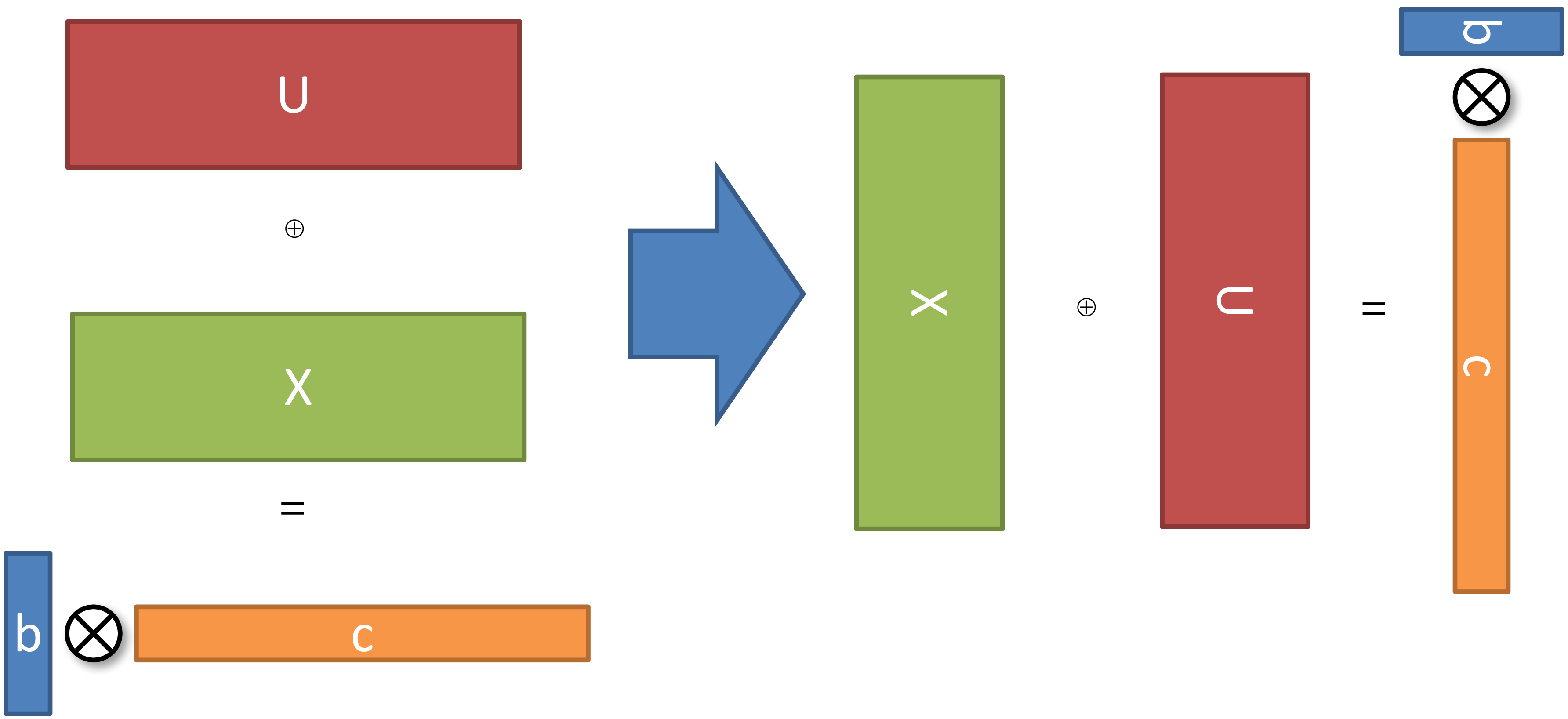


OT Extension, Pictorially

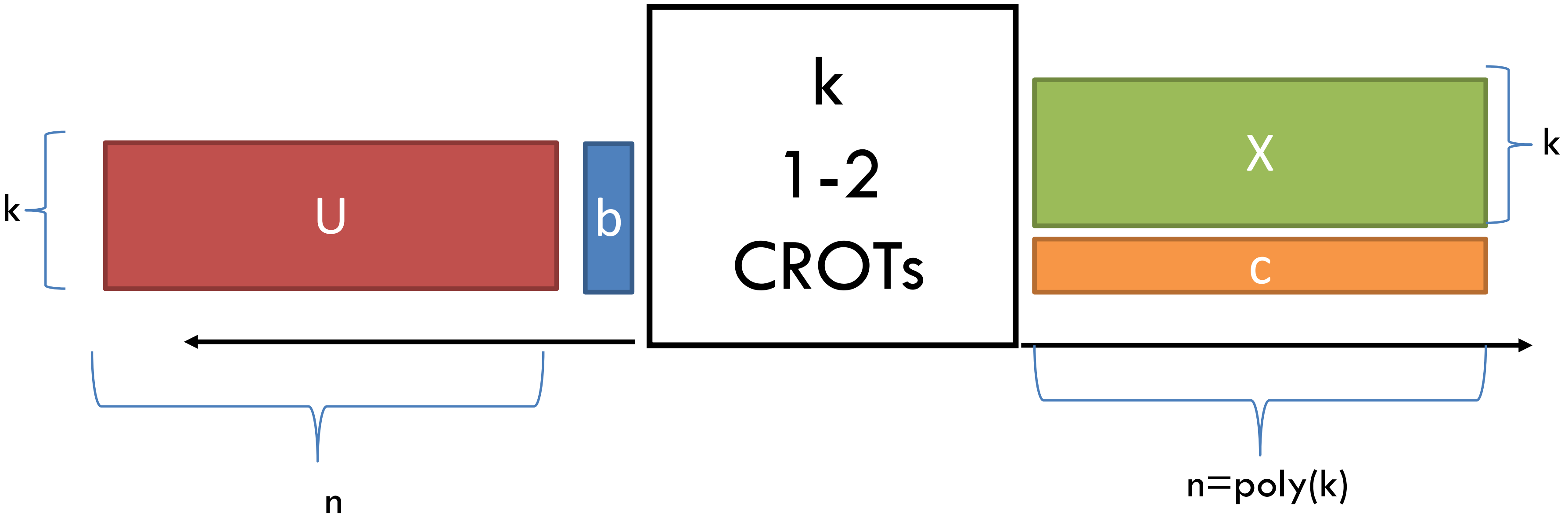


$$(b \otimes c)_{ij} = b_i \cdot c_j$$

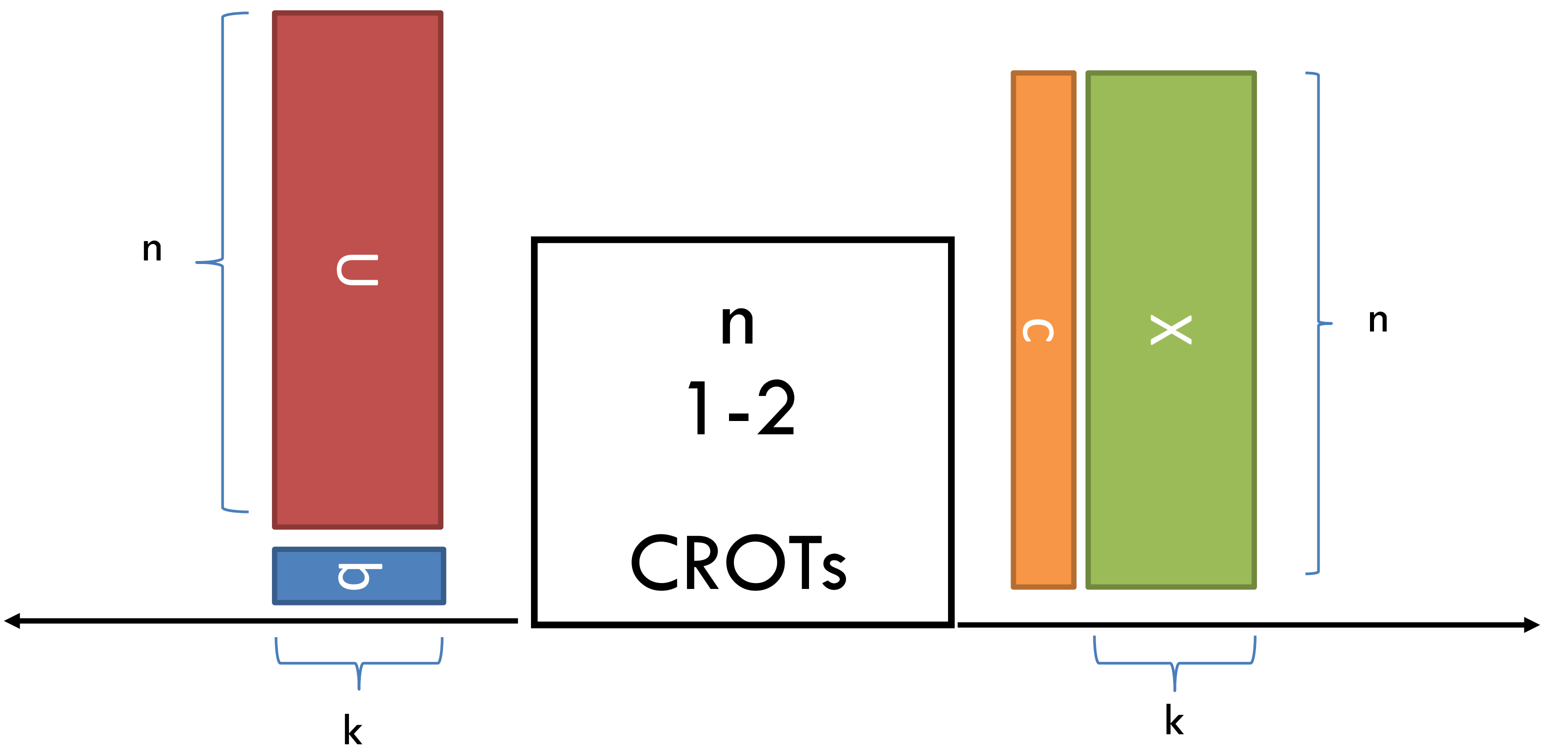
OT Extension, Turn your head!



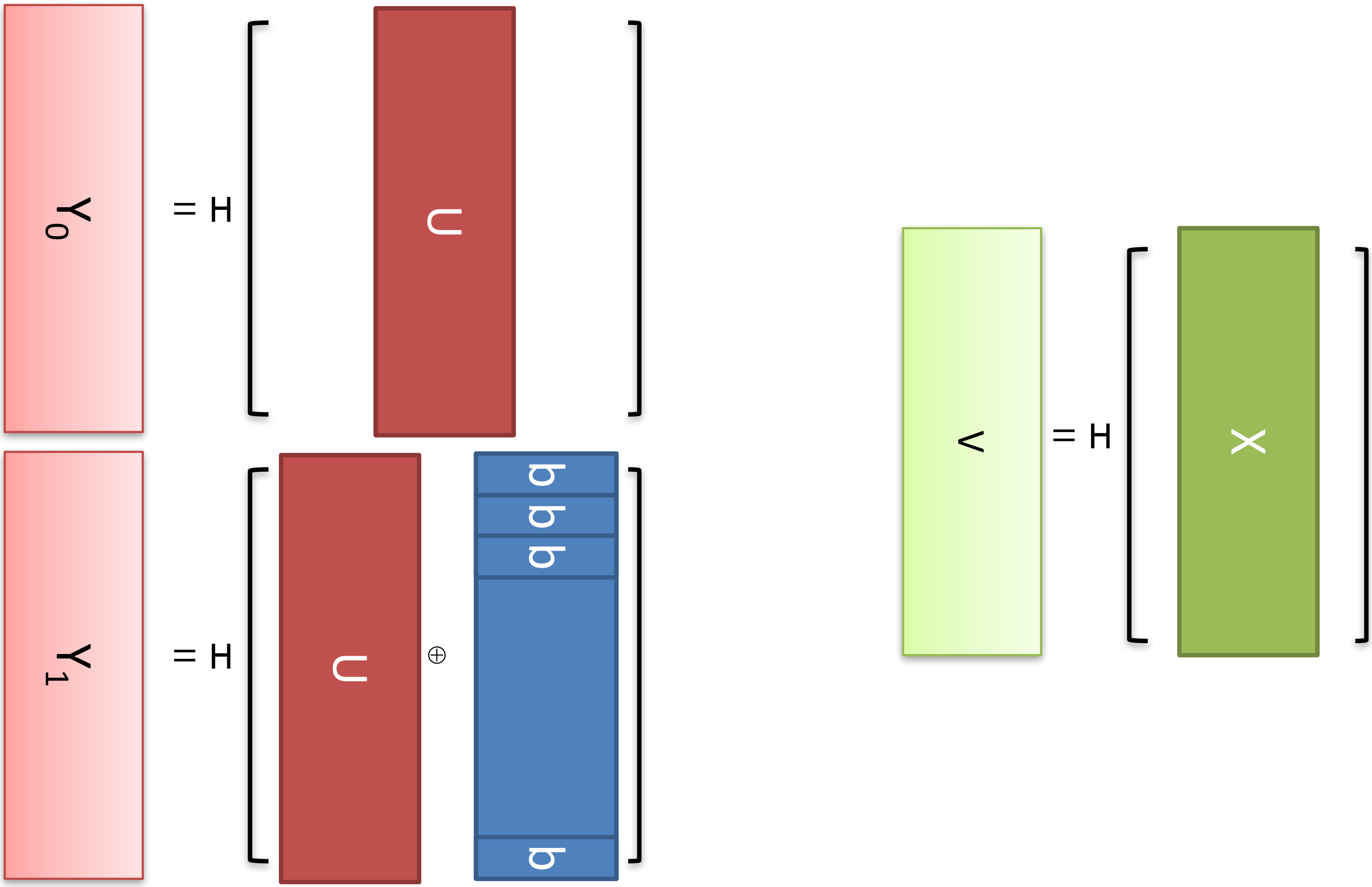
OT Extension, Pictorially



OT Extension, Pictorially



Break the correlation!

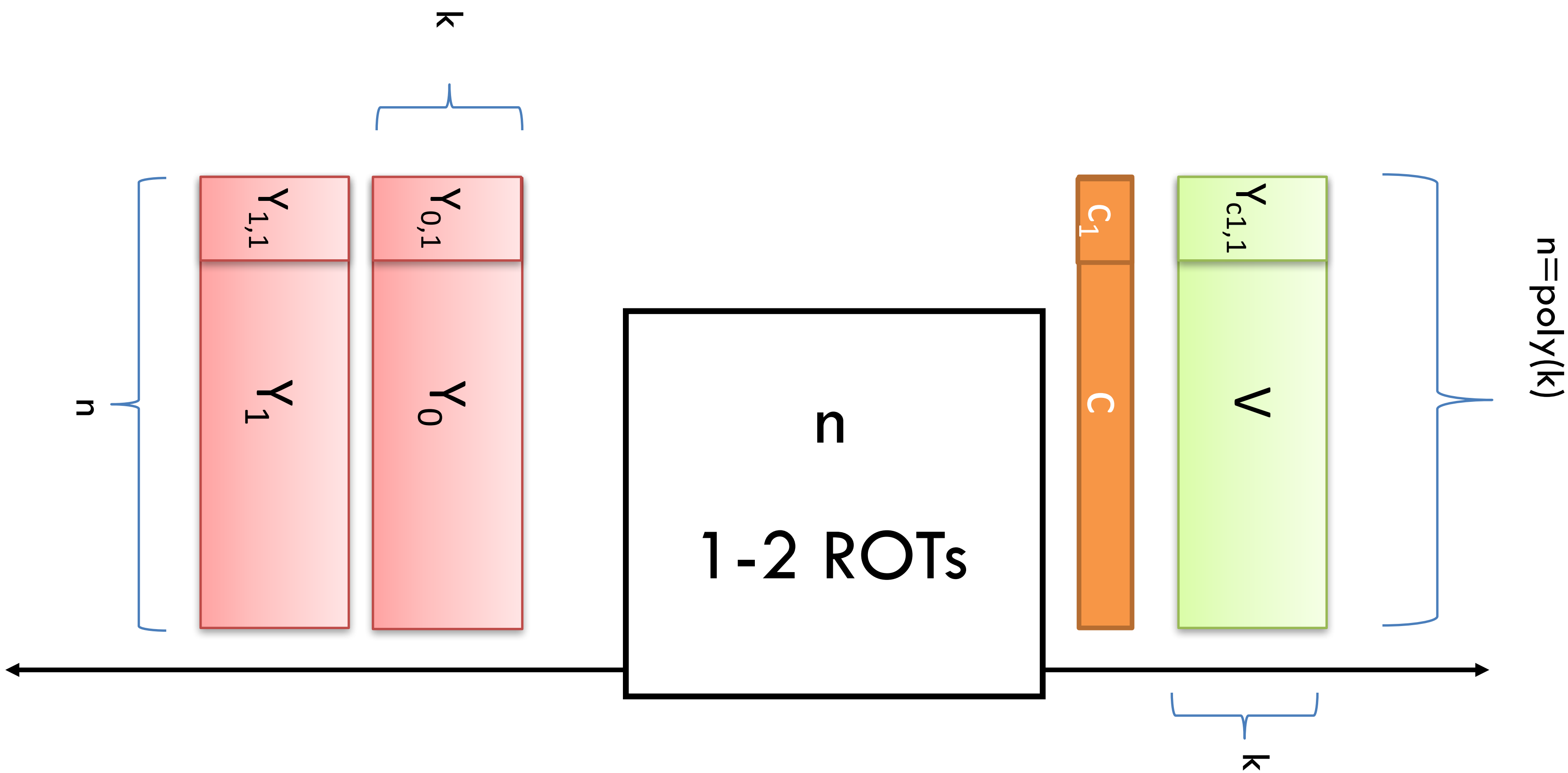


Breaking the correlation

- Using a **correlation robust hash function** H s.t.
  1. $\{a_0, \ldots, a_n, H(a_0 + r), \ldots, H(a_n + r)\}$ // (a_i's, r random)
  2. $\{a_0, \ldots, a_n, b_0, \ldots, b_n\}$ // (a_i's,b_i's random)

  are ***computationally indistinguishable***

OT Extension, Pictorially

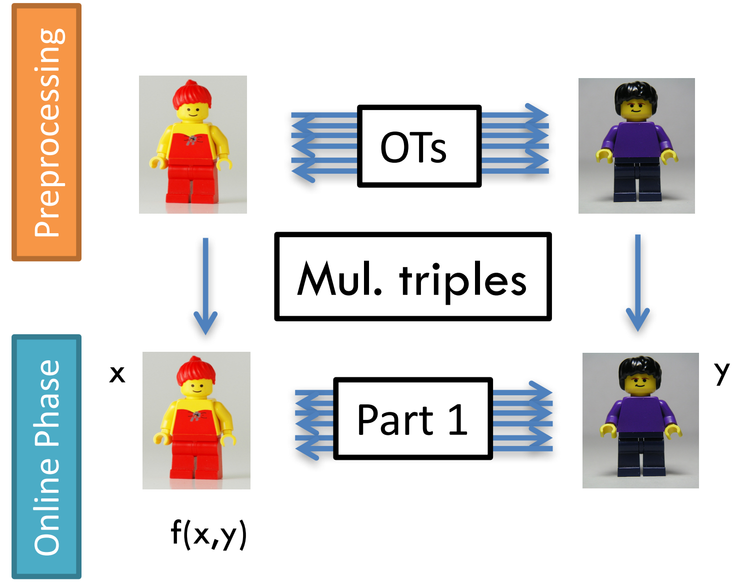


Recap

0. Strech **k OTs** from *k- to poly(k)=n-bitlong strings*
1. Send correction for each pair of messages x^i_0, x^i_1 s.t., $\mathbf{x^i_0 \oplus x^i_1 = c}$
2. **Turn your head** (S/R swap roles)
3. The bits of **c** are the new **choice bits**
4. Break the correlation: $\mathbf{y^j_0 = H(u^j),\ y^j_1 = H(u^j \oplus b)}$

- **Not secure against active adversaries**

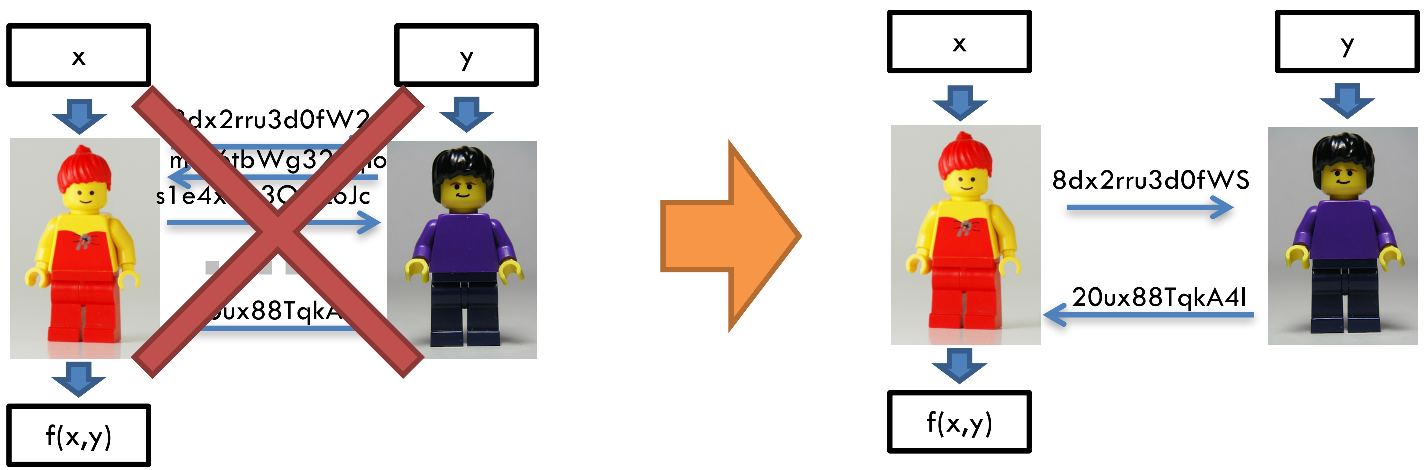
Recap of Part 2



- OT: building block for 2PC
  - **Requires PKE** ☹
  - **OT Extension (using only SKE)** ☺
  - Can be combined with protocols from part 1 for 2PC without a trusted dealer (using computational assumptions) ☺
  - **#rounds** = depth of the circuit 😐

Coming up next…

- OT + Garbled Circuits → **Constant round 2PC!**



…aka layman fully-homomorphic encryption

Plan for the next 3 hours…

- **Part 1: Secure Computation with a Trusted Dealer**
  - Warmup: One-Time Truth Tables
  - Evaluating Circuits with Beaver's trick
  - MAC-then-Compute for Active Security
- **Part 2: Oblivious Transfer**
  - OT: Definitions and Applications
  - Passive Secure OT Extension
  - OT Protocols from DDH (Naor-Pinkas/PVW)
- **Part 3: Garbled Circuits**
  - GC: Definitions and Applications
  - Garbling gate-by-gate: Basic and optimizations
  - Active security 101: simple-cut-and choose, dual-execution

Part 3: Garbled Circuits

- **GC: Definitions and Applications**
- Garbling gate-by-gate: Basic and optimizations
- Active security 101: simple-cut-and choose, dual-execution

Garbled Circuit

Cryptographic primitive that allows to evaluate

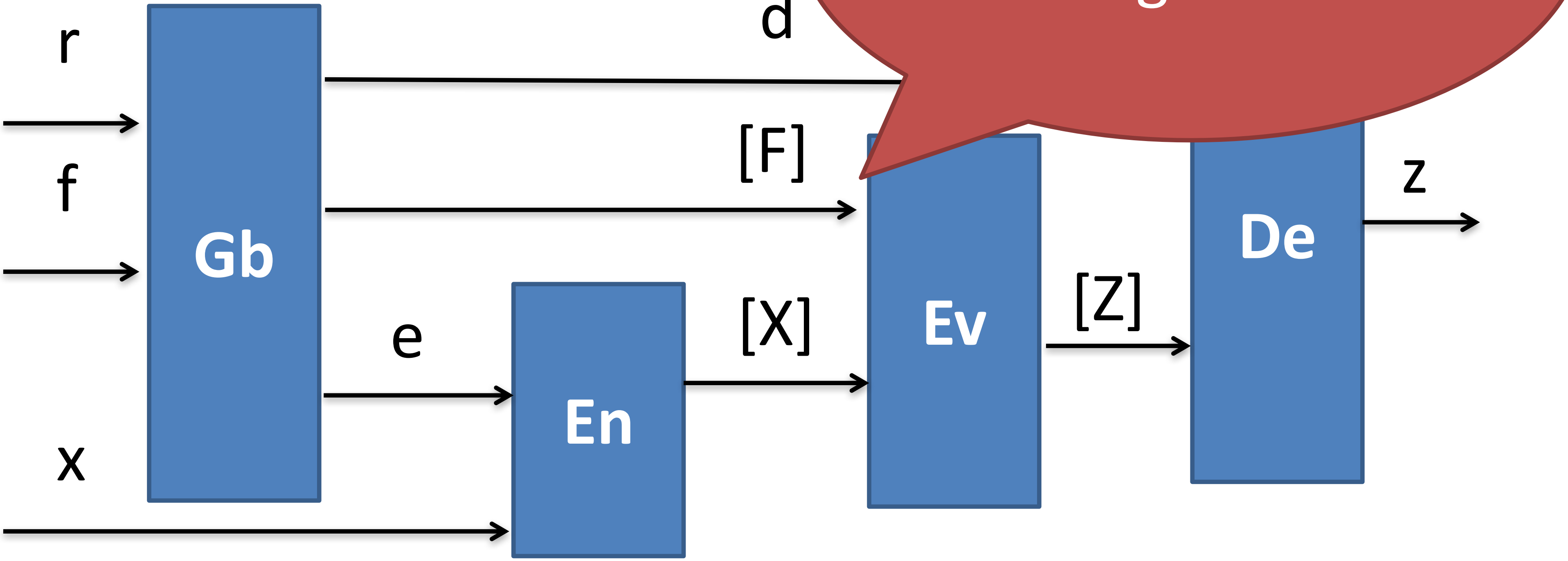
encrypted functions

on

encrypted inputs

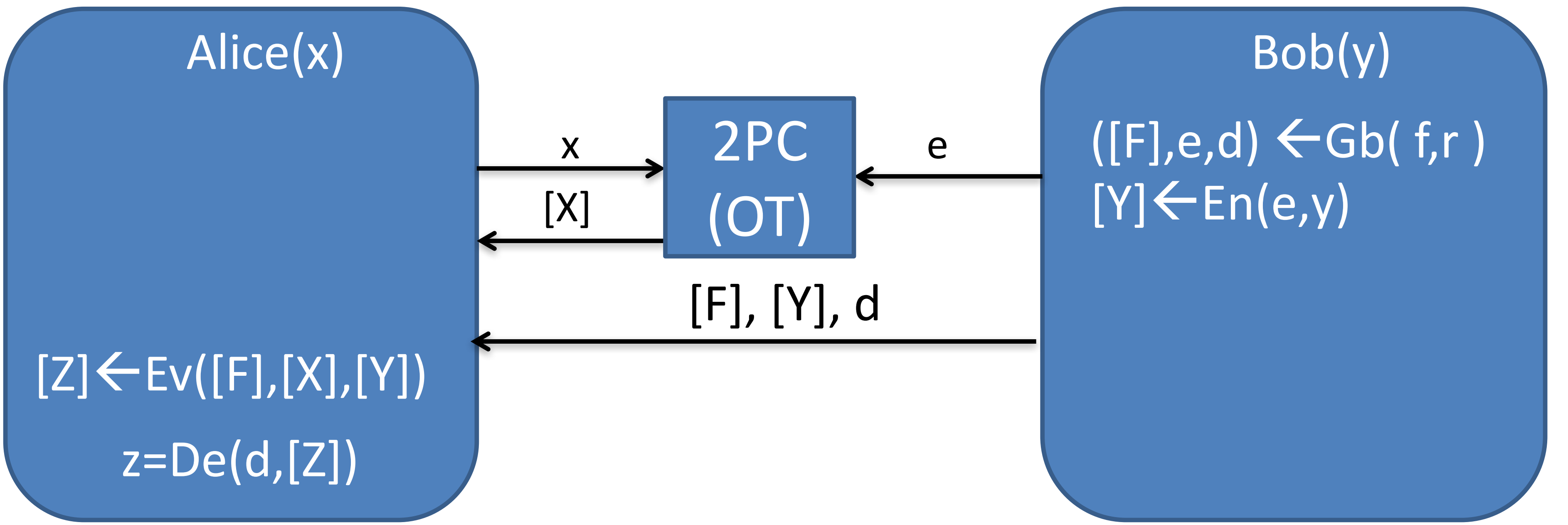
Garbled C

Values *in a box* are "garbled"

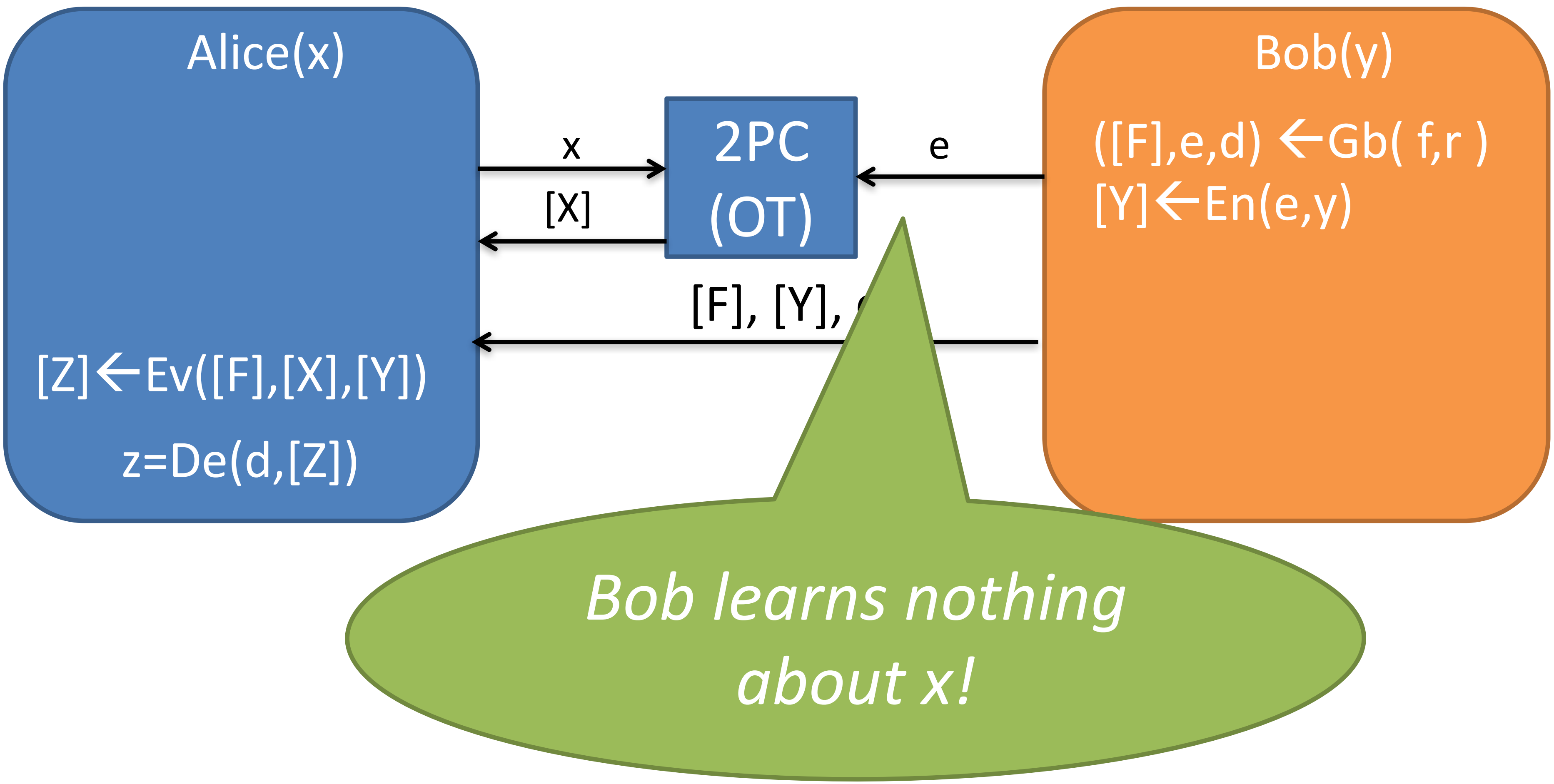


Correct if z=f(x)

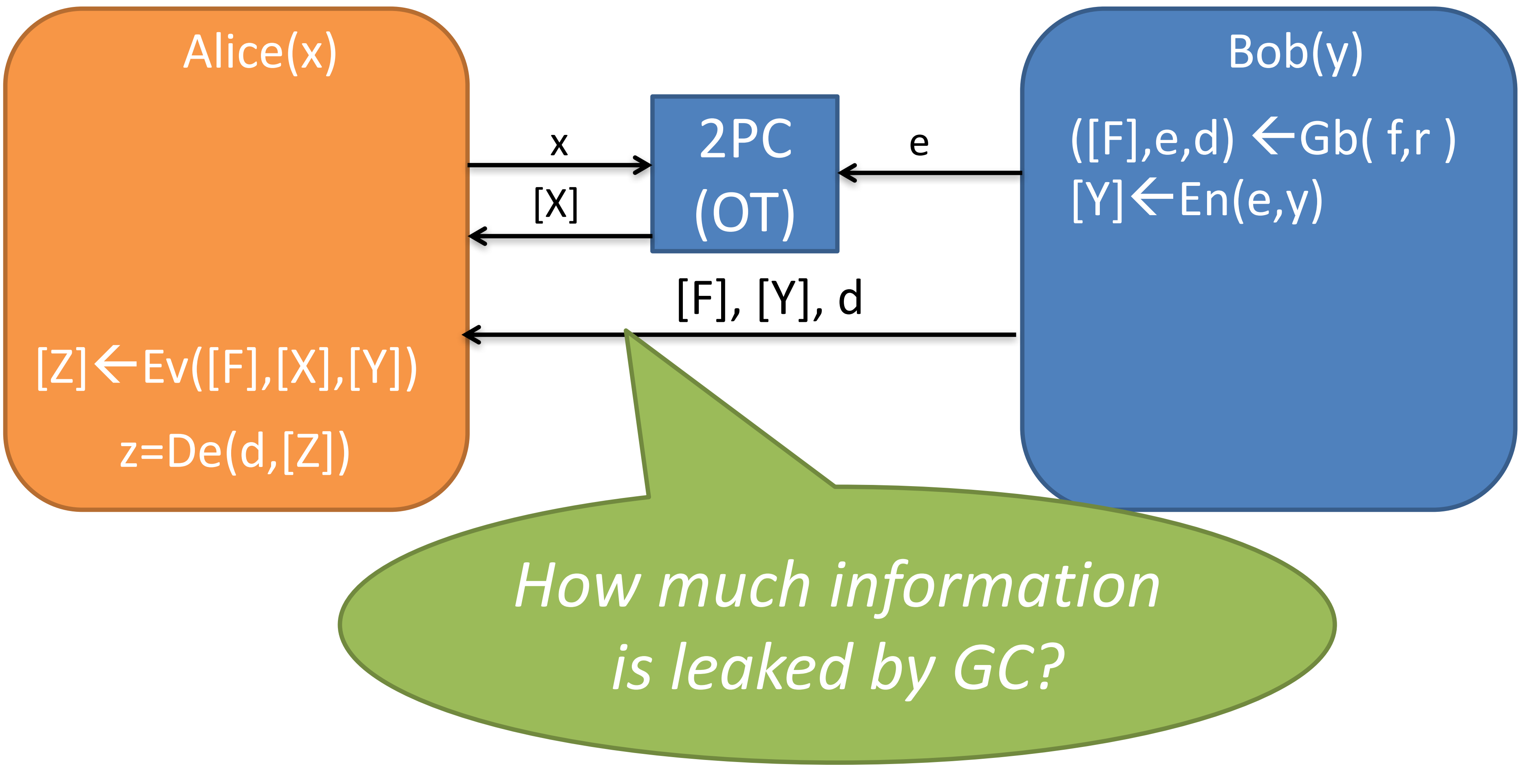
Passive Constant Round 2PC (Yao)



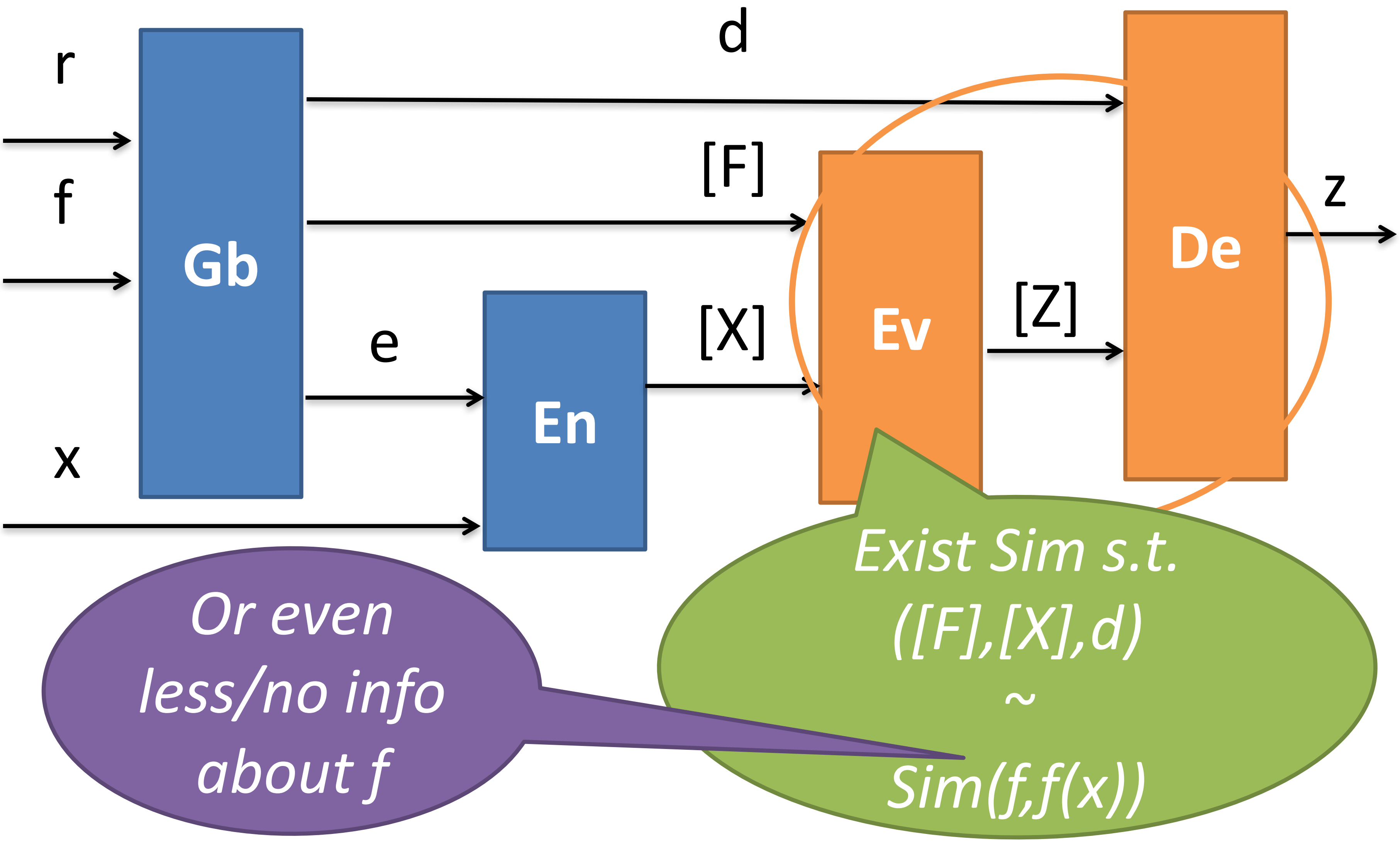
Passive Constant Round 2PC (Yao)



Passive Constant Round 2PC (Yao)



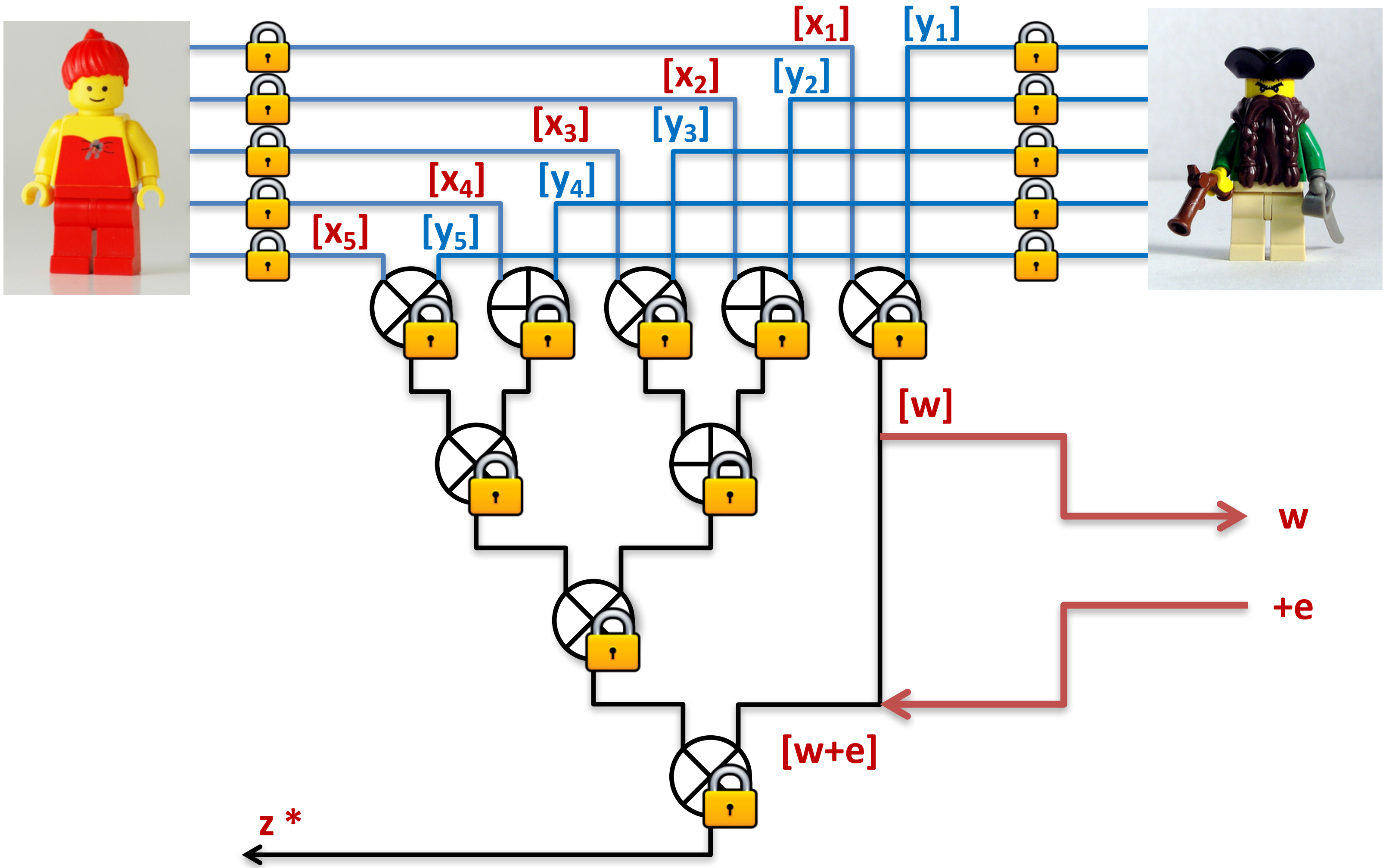
Garbled Circuits: Privacy



Part 3: Garbled Circuits

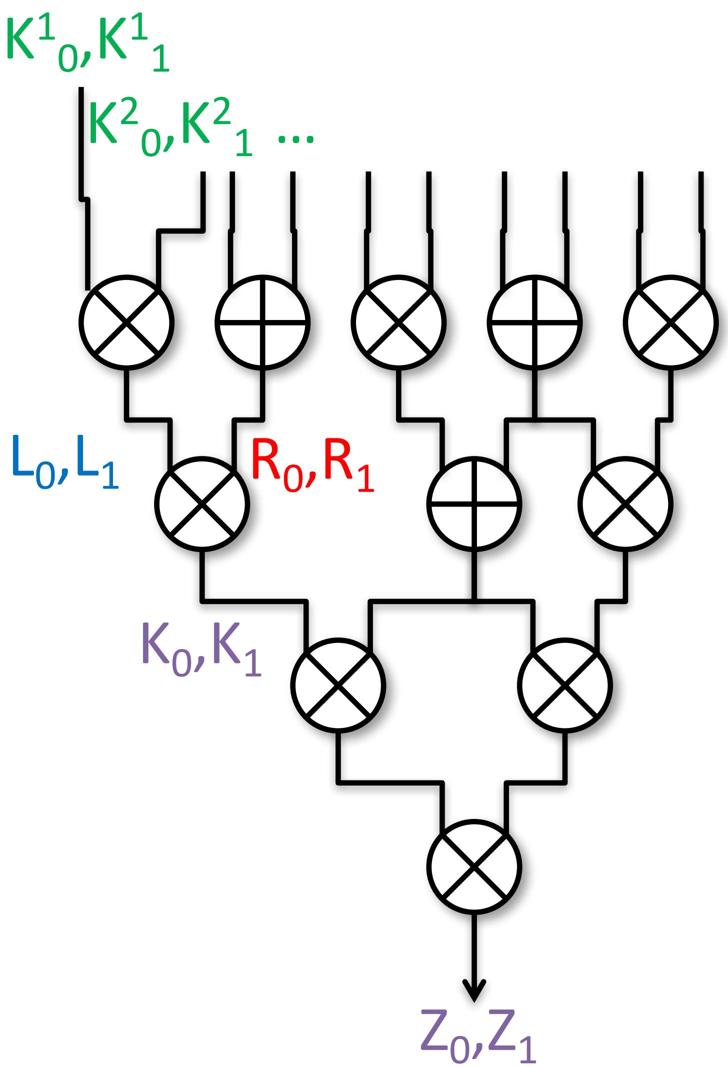
- Definitions and Applications
- **Garbling gate-by-gate: Basic and optimizations**
- Active security 101: simple-cut-and choose, dual-execution

Garbling: Gate-by-gate



PROJECTIVE SCHEMES:
CIRCUIT BASED GARBLING/EVALUATIONS

Garbling a Circuit : ([F],e,d)← Gb(f)



- Choose 2 random keys K^i_0,K^i_1 for each wire in the circuit
  - *Input, internal and, output wires*
- For each gate g compute
  - gg ← $Gb(g,L_0,L_1,R_0,R_1,K_0,K_1)$
- Output
  - $e=(K^i_0,K^i_1)$ for all input wires
  - $d=(Z_0,Z_1)$
  - $[F]=(gg^i)$ for all gates i

Encoding and Decoding

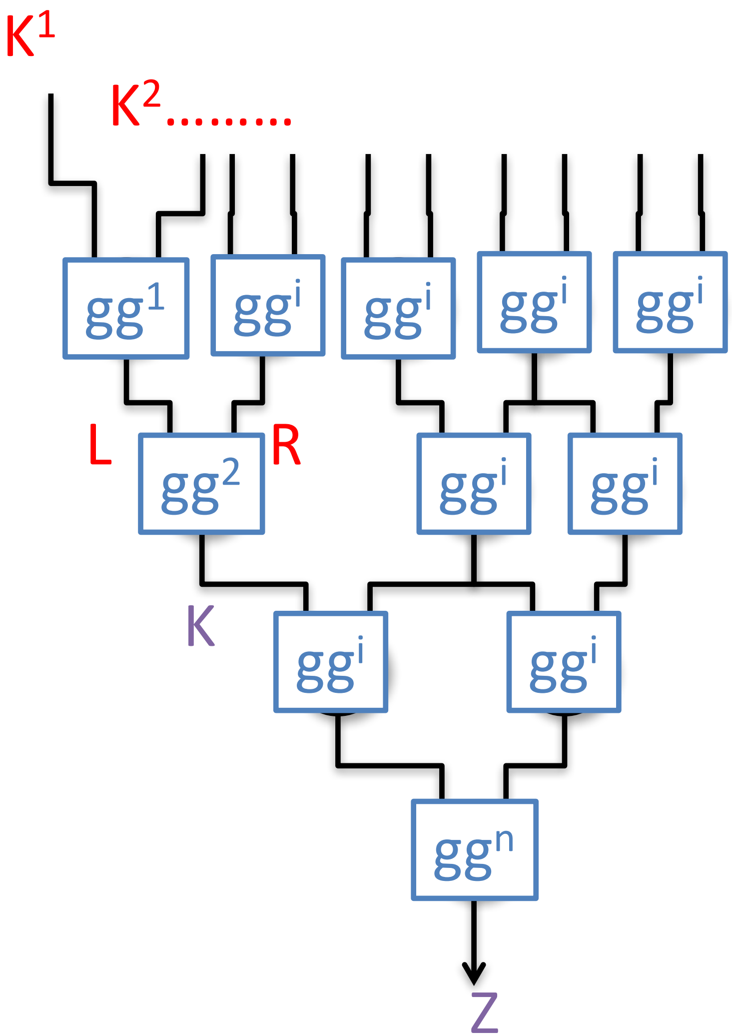
$[X] = En(e,x)$

- $e=\{ K^i_0, K^i_1\}$
- $x= \{ x_1,\ldots,x_n \}$
- $[X]=\{K^1_{x1},\ldots,K^n_{xn}\}$

$z=De(d,[Z])$

- $d = \{ Z_0,Z_1 \}$
- $[Z] = \{ K \}$
- $z=$
  - 0 if $K=Z_0$,
  - 1 if $K=Z_1$,
  - "abort" else

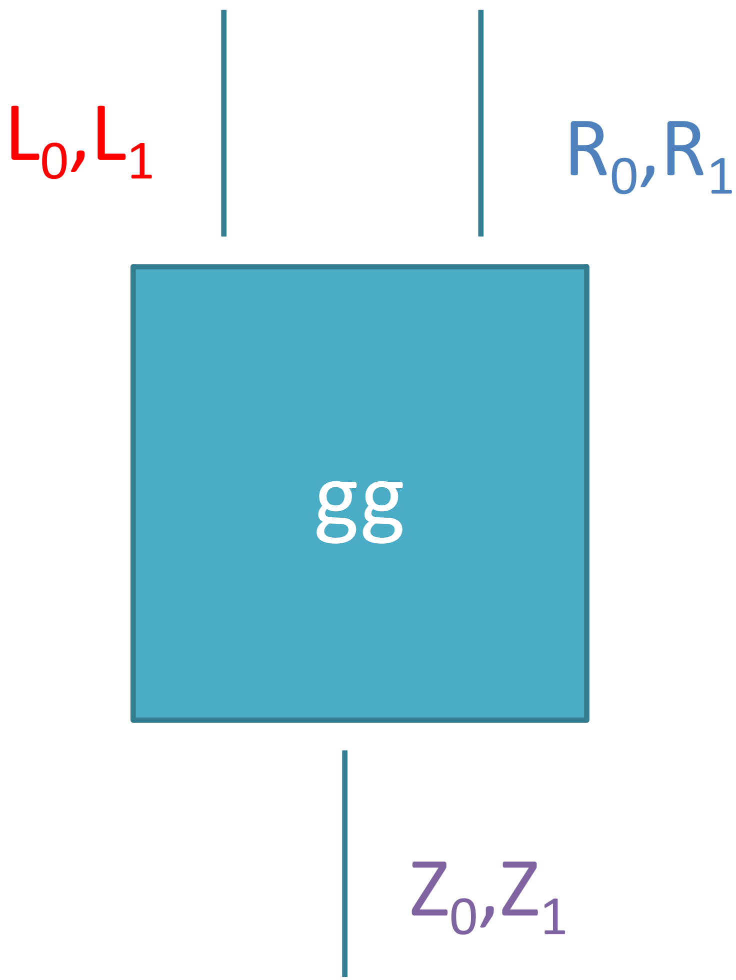
Evaluating a GC : [Z] ← Ev([F],[X])



- Parse [X]={K^1,…,K^n}
- Parse [F]={gg^i}
- For each gate i compute
  - K ← Ev(gg^i,L,R)
- Output
  - Z

INDIVIDUAL GATES GARBLING/EVALUATION

Notation



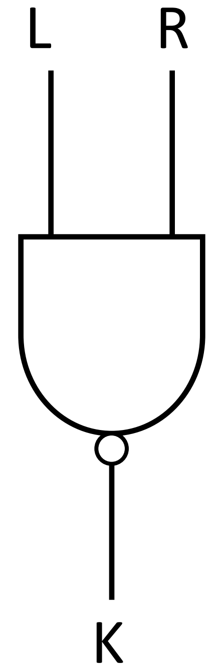
- A garbled gate is a gadget that given two inputs keys gives you the right output key *(and nothing else)*

- gg ← $Gb(g, L_0, L_1, R_0, R_1, Z_0, Z_1)$
- $Z_{g(a,b)}$ ← $Ev(gg, L_a, R_b)$

- //and not $Z_{1-g(a,b)}$

Yao Gate Garbling (1)

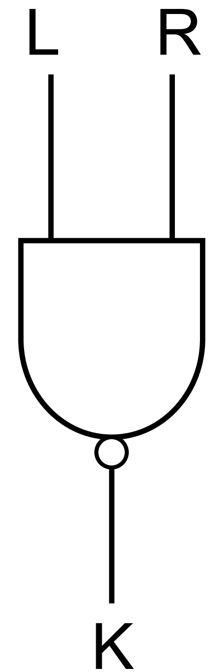
L	R	K
0	0	1
0	1	1
1	0	1
1	1	0



- NAND gate

Yao Gate Garbling (2)

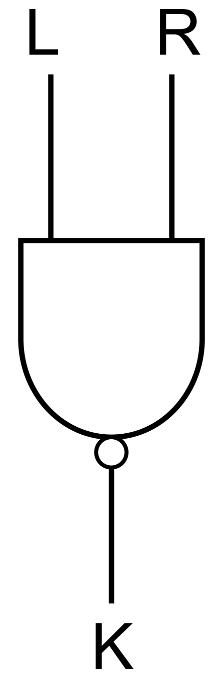
L	R	K
L_0	R_0	K_1
L_0	R_1	K_1
L_1	R_0	K_1
L_1	R_1	K_0



- Choose labels (e.g., 128 bits strings) for every value on every wire

Yao Gate Garbling (3)

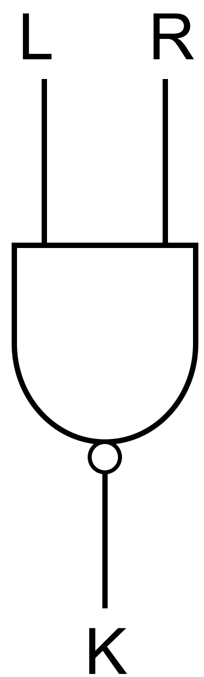
C
$C_1 = H(L_0,R_0) \oplus K_1$
$C_2 = H(L_0,R_1) \oplus K_1$
$C_3 = H(L_1,R_0) \oplus K_1$
$C_4 = H(L_1,R_1) \oplus K_0$



- Encrypt the output key with the input keys

Yao Gate Garbling (4)

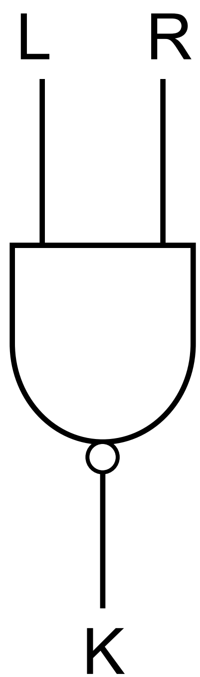
C
$C_1 = H(L_0,R_0) \oplus (K_1,0^k)$
$C_2 = H(L_0,R_1) \oplus (K_1,0^k)$
$C_3 = H(L_1,R_0) \oplus (K_1,0^k)$
$C_4 = H(L_1,R_1) \oplus (K_0,0^k)$



- Add redundancy (later used to check if decryption is successful)

Yao Gate Garbling (5)

C
$C_1 = H(L_0,R_0) \oplus (K_1,0^k)$
$C_2 = H(L_0,R_1) \oplus (K_1,0^k)$
$C_3 = H(L_1,R_0) \oplus (K_1,0^k)$
$C_4 = H(L_1,R_1) \oplus (K_0,0^k)$



$C'_1,C'_2,C'_3,C'_4 = \text{perm}(C_1,C_2,C_3,C_4)$

- Permute the order of the ciphertexts (to hide information about inputs/outputs)

Yao Gate Evaluation (1)

Eval(gg, L_a, R_b) //not a,b

- For i=1..4
  - $(K,t)=C'_i \oplus H(L_a,R_b)$
  - If $t=0^k$ output K

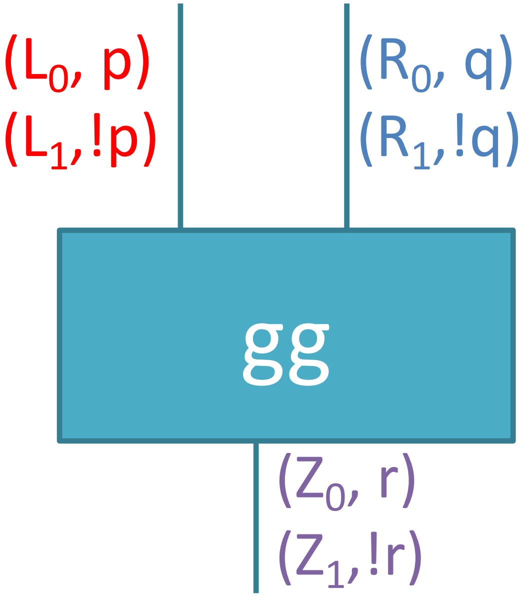
gg (permuted)
$C_1 = H(L_0,R_0) \oplus (K_1,0^k)$
$C_2 = H(L_0,R_1) \oplus (K_1,0^k)$
$C_3 = H(L_1,R_0) \oplus (K_1,0^k)$
$C_4 = H(L_1,R_1) \oplus (K_0,0^k)$

- Output is correct:
  - $t=0^k$ only for right row
- Evaluator learns nothing else:
  - Encryption + permutation

GARBLING OPTIMIZATIONS: POINT-AND-PERMUTE

Point-and-permute

- Problem: Evaluator needs to try to decrypt all 4 rows
- Solution: add permutation bits to keys

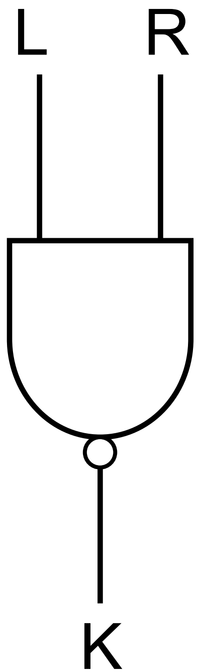


$gg \leftarrow Gb(g, L_0, L_1, p, R_0, R_1, q, Z_0, Z_1, r)$

$(Z_{g(a,b)}, r \oplus g(a,b)) \leftarrow Ev(gg, L_a, a \oplus p, R_b, b \oplus q)$

Point-and-permute Garbling (4)

C
$C_1 = H(L_0,R_0) \oplus (K_{g(0,0)}, r \oplus g(0,0))$
$C_2 = H(L_0,R_1) \oplus (K_{g(0,1)}, r \oplus g(0,1))$
$C_3 = H(L_1,R_0) \oplus (K_{g(1,0)}, r \oplus g(1,0))$
$C_4 = H(L_1,R_1) \oplus (K_{g(1,1)}, r \oplus g(1,1))$



- Remove redundancy
- Add random permutation bit

Point-and-permute Garbling (5)

C
$C_1 = H(L_p, R_q) \oplus (K_{g(p,q)}, r \oplus g(p,q))$
$C_2 = H(L_p, R_{!q}) \oplus (K_{g(p,!q)}, r \oplus g(p,!q))$
$C_3 = H(L_{!p}, R_q) \oplus (K_{g(!p,q)}, r \oplus g(!p,q))$
$C_4 = H(L_{!p}, R_{!q}) \oplus (K_{g(!p,!q)}, r \oplus g(!p,!q))$

- Permute rows using p,q

Point-and-permute Evaluation

Eval(gg, L, u, R, v) //not a,b

- $(K,r)=C'_{2u+v} \oplus H(L,R)$

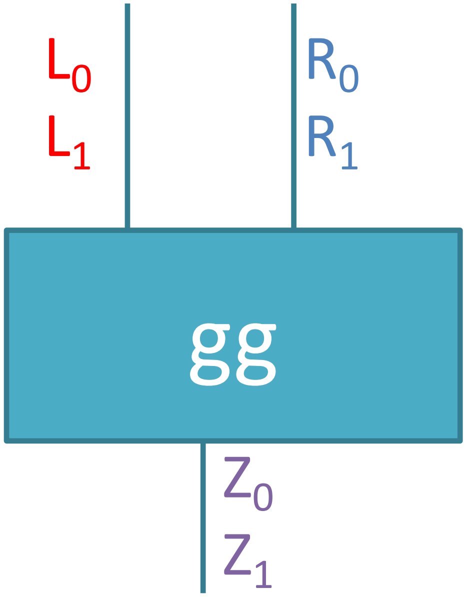
C
$C_1 = H(L_p,R_q) \oplus (K_{g(p,q)},\ r \oplus g(p,q))$
$C_2 = H(L_p,R_{!q}) \oplus (K_{g(p,!q)},\ r \oplus g(p,!q))$
$C_3 = H(L_{!p},R_q) \oplus (K_{g(!p,q)},\ r \oplus g(!p,q))$
$C_4 = H(L_{!p},R_{!q}) \oplus (K_{g(!p,!q)},\ r \oplus g(!p,!q))$

- Output is correct:
  - (Check permutation)
- Privacy:
  - $u=p \oplus a$, $v=q \oplus b$
  - p,q are "one time pads" for a,b

GARBLING OPTIMIZATIONS:
SIMPLE GARBLED ROW REDUCTION

Point-and-permute

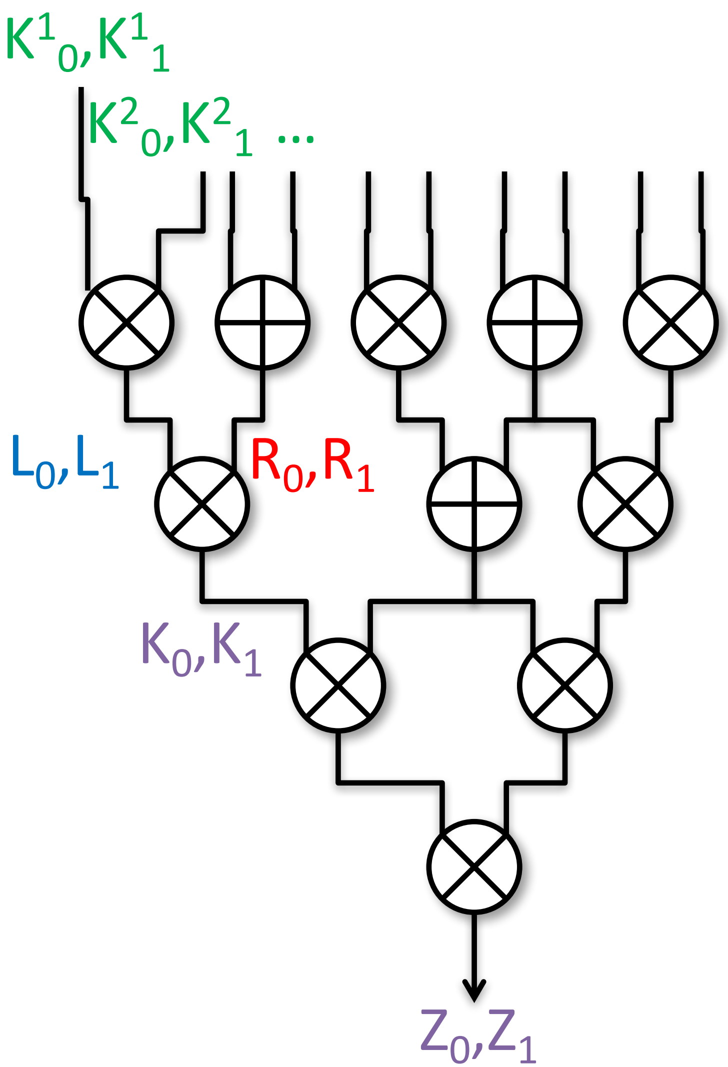
- Problem: each gg is 4 ciphertexts
- Solution: define output key pseudorandomly as functions of input keys, reduce comm. complexity



$(gg, Z_0, Z_1) \leftarrow Gb(g, L_0, L_1, R_0, R_1)$

$(Z_{g(a,b)}) \leftarrow Ev(gg, L_a, R_b)$

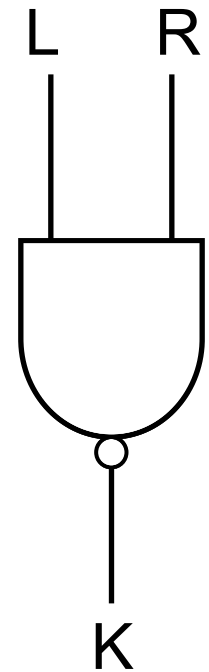
Garbling a Circuit : ([F],e,d)← Gb(f)



- Choose 2 random keys K^i_0,K^i_1 for each wire in the circuit
  - *Input wire only!*
- For each gate g compute
  - $(gg,K_0,K_1) \leftarrow Gb(g,L_0,L_1,R_0,R_1)$
- Output
  - $e=(K^i_0,K^i_1)$ for all input wires
  - $d=(Z_0,Z_1)$
  - $[F]=(gg^i)$ for all gates i

Yao Gate Garbling (3)

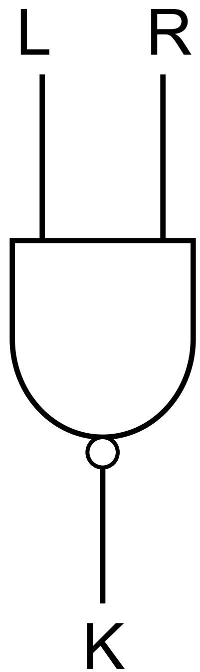
C
$C_1 = H(L_0,R_0) \oplus K_1$
$C_2 = H(L_0,R_1) \oplus K_1$
$C_3 = H(L_1,R_0) \oplus K_1$
$C_4 = H(L_1,R_1) \oplus K_0$



- Encrypt the output key with the input keys

Garbled Row Reduction Garbling

C
$K_1 = H(L_0,R_0)$ $(C_1=0^k)$
$C_2 = H(L_0,R_1) \oplus K_1$
$C_3 = H(L_1,R_0) \oplus K_1$
$C_4 = H(L_1,R_1) \oplus K_0$

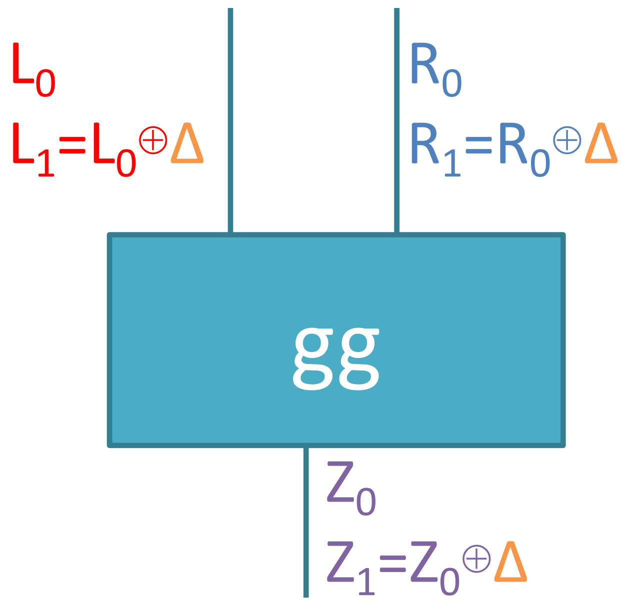


- Define output keys as function of input keys
  - (compatible with p&p)
  - Can reduce 2 rows, but 1 is compatible with Free-XOR (coming up!)

GARBLING OPTIMIZATIONS:
FREE XOR

Free-XOR

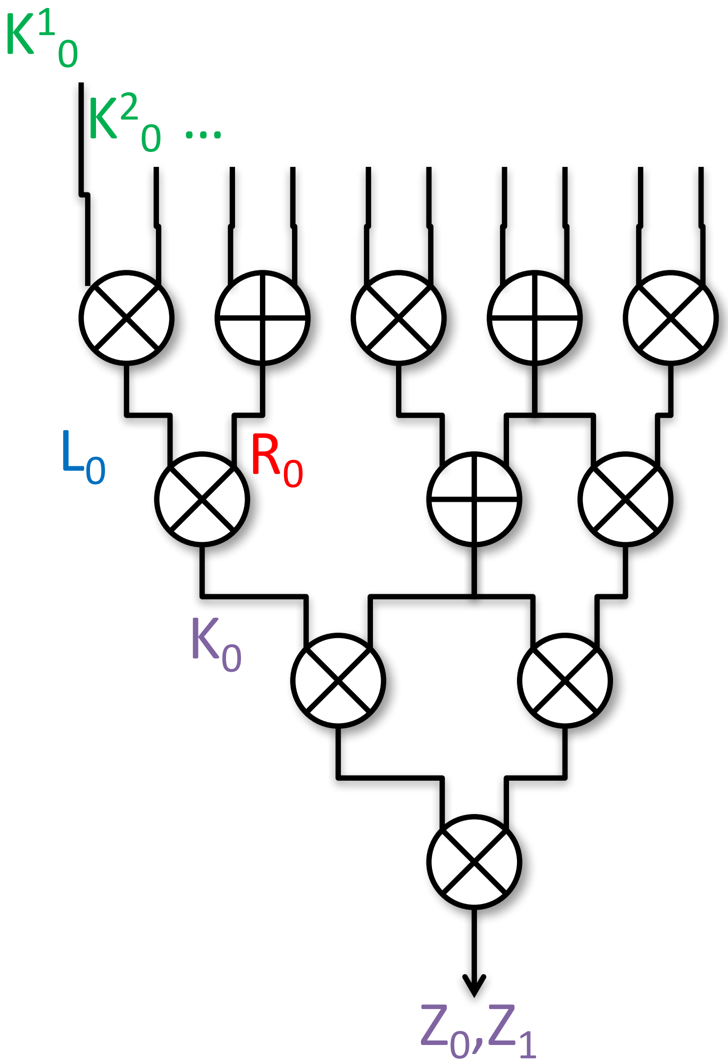
- Problem: in BeDOZa linear gates are for free. What about GC?
- Solution: introduce correlation between keys, make XOR computation "free"



$$(gg, Z_0) \leftarrow Gb(g, L_0, R_0, \Delta)$$

$$(Z_{g(a,b)}) \leftarrow Ev(gg, L_a, R_b)$$

Garbling a Circuit : ([F],e,d)← Gb(f)



- Choose 1 random key K^i_0 for each input wire in the circuit
  - *And global difference Δ*
- For each gate g compute
  - $(gg,K_0) \leftarrow Gb(g,L_0,R_0,\Delta)$
- Output
  - $e=(K^i_0,K^i_1)$ for all input wires
  - $d=(Z_0,Z_1)$
  - $[F]=(gg^i)$ for all gates i

Garbling non-linear gates

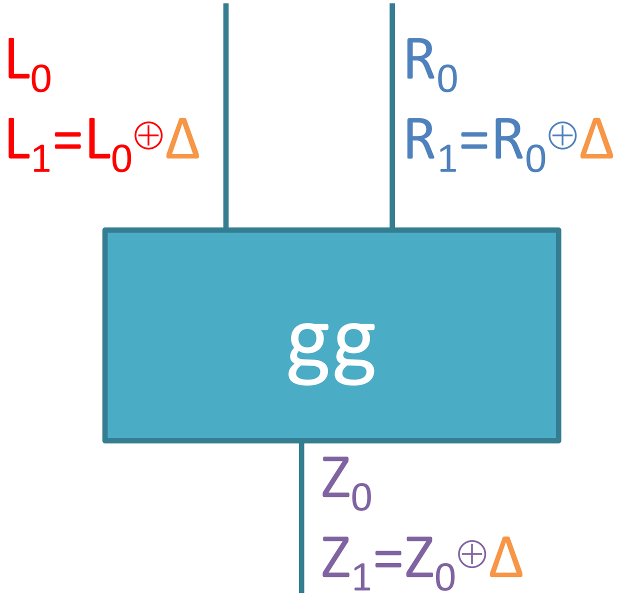
- Like before, but requires “circular security assumption”
  – (Compatible with GRR and p&P)
- Example for AND gate
  – Evaluator sees

$$L_0,\ R_0,\ K_0,$$

$$H(L_0 \oplus \Delta, R_0 \oplus \Delta) \oplus K_0 \oplus \Delta$$

  – And should not be able to compute Δ !

Garbling/Evaluating XOR Gates



$(gg,,Z_0) \leftarrow Gb(g,L_0,R_0,\Delta)$

$(Z_{g(a,b)}) \leftarrow Ev(gg,L_a,R_b)$

$Gb(XOR,L_0,R_0,\Delta)$

- Output $Z_0=L_0 \oplus R_0$
- (gg is empty)

$Ev(XOR,L_a,R_b,\Delta)$

- Output $Z_{a\oplus b}=L_a \oplus R_b$

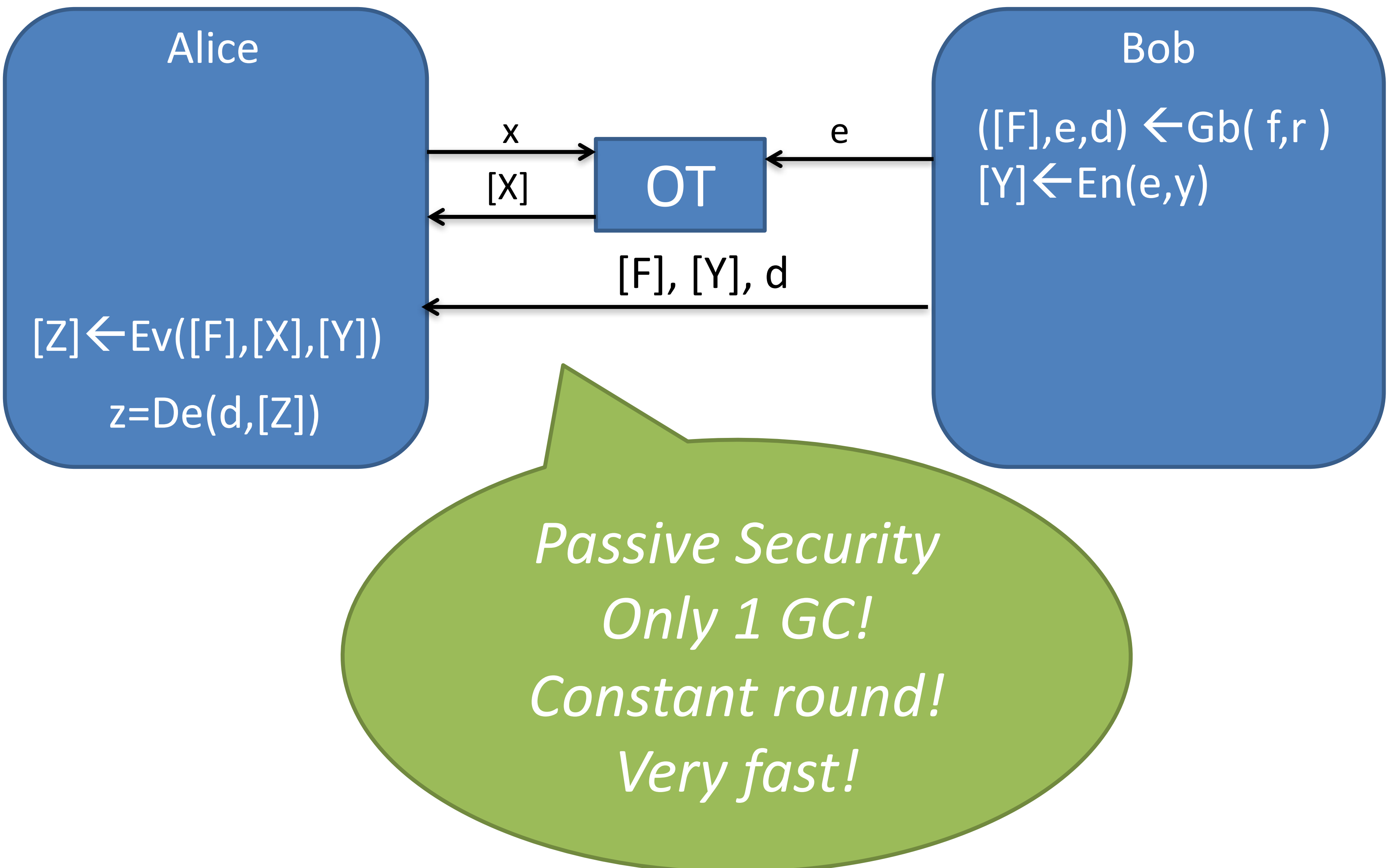
$$L_a \oplus R_b = L_0 \oplus a\Delta \oplus R_0 \oplus b\Delta = Z_0 \oplus (a\oplus b)\Delta = Z_{a\oplus b}$$

Part 3: Garbled Circuits

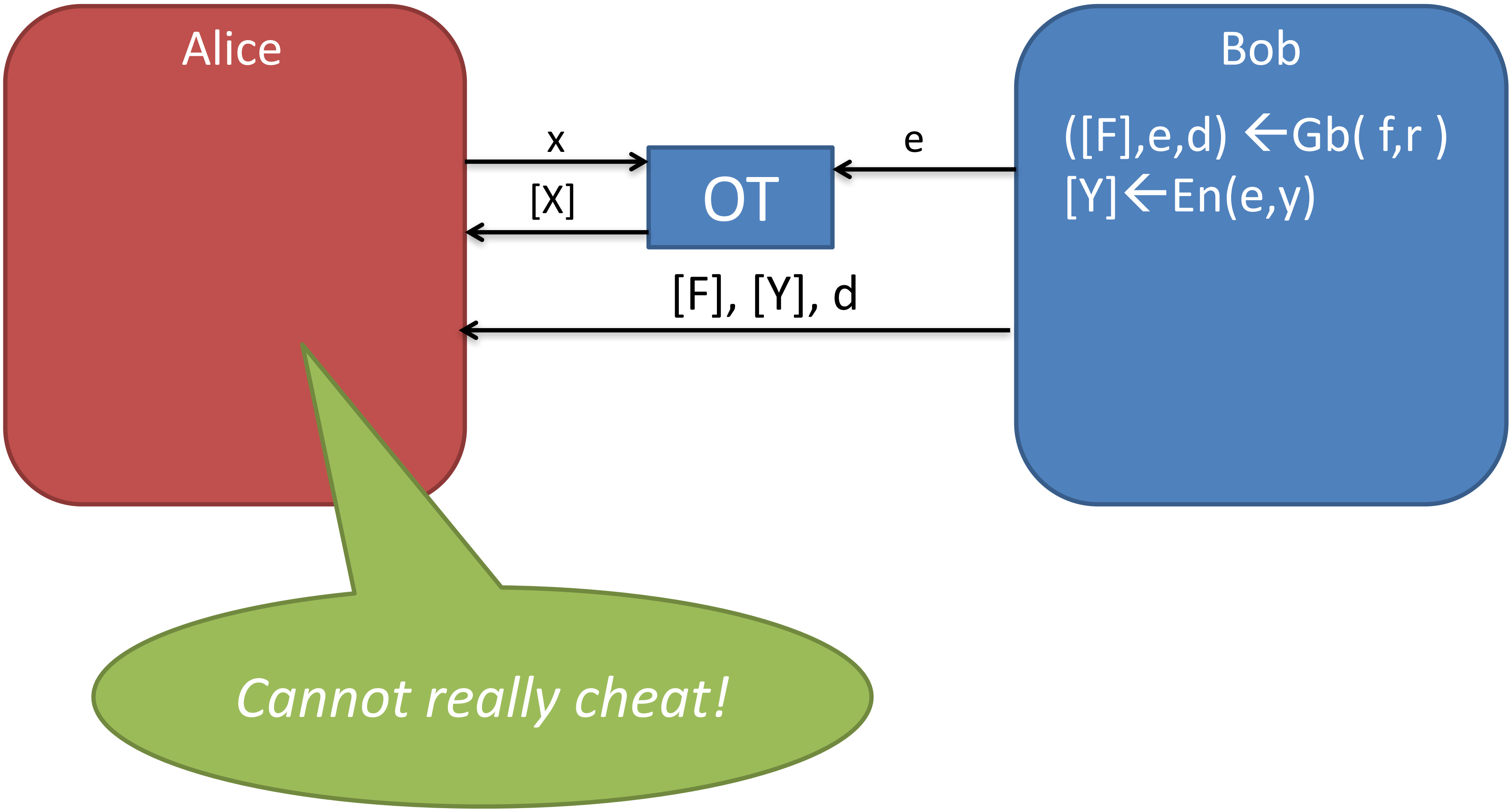
- Definitions and Applications
- Garbling gate-by-gate: Basic and optimizations
- **Active security 101: simple-cut-and choose, dual-execution**

ACTIVE ATTACKS VS YAO

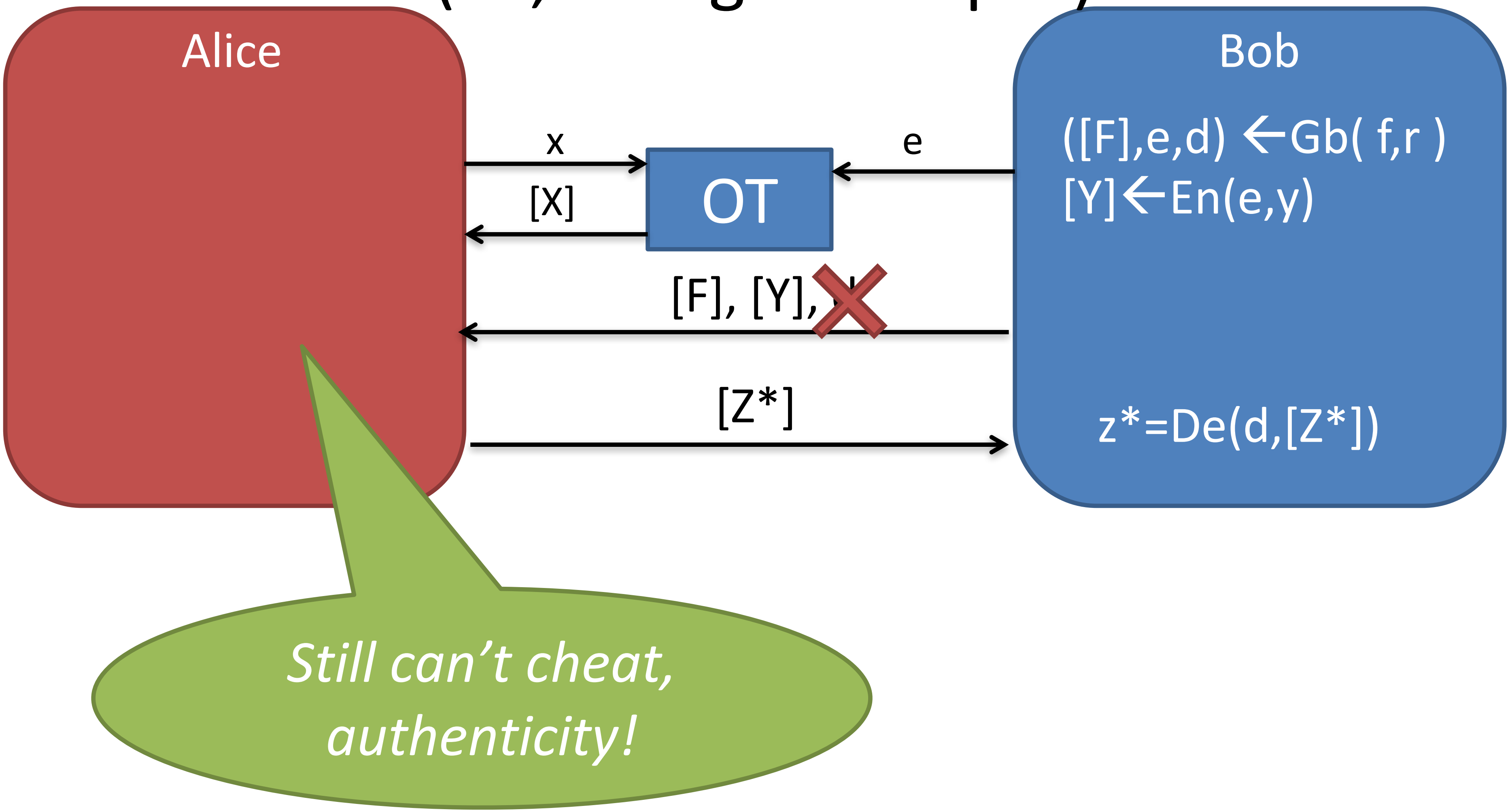
Yao´s protocol



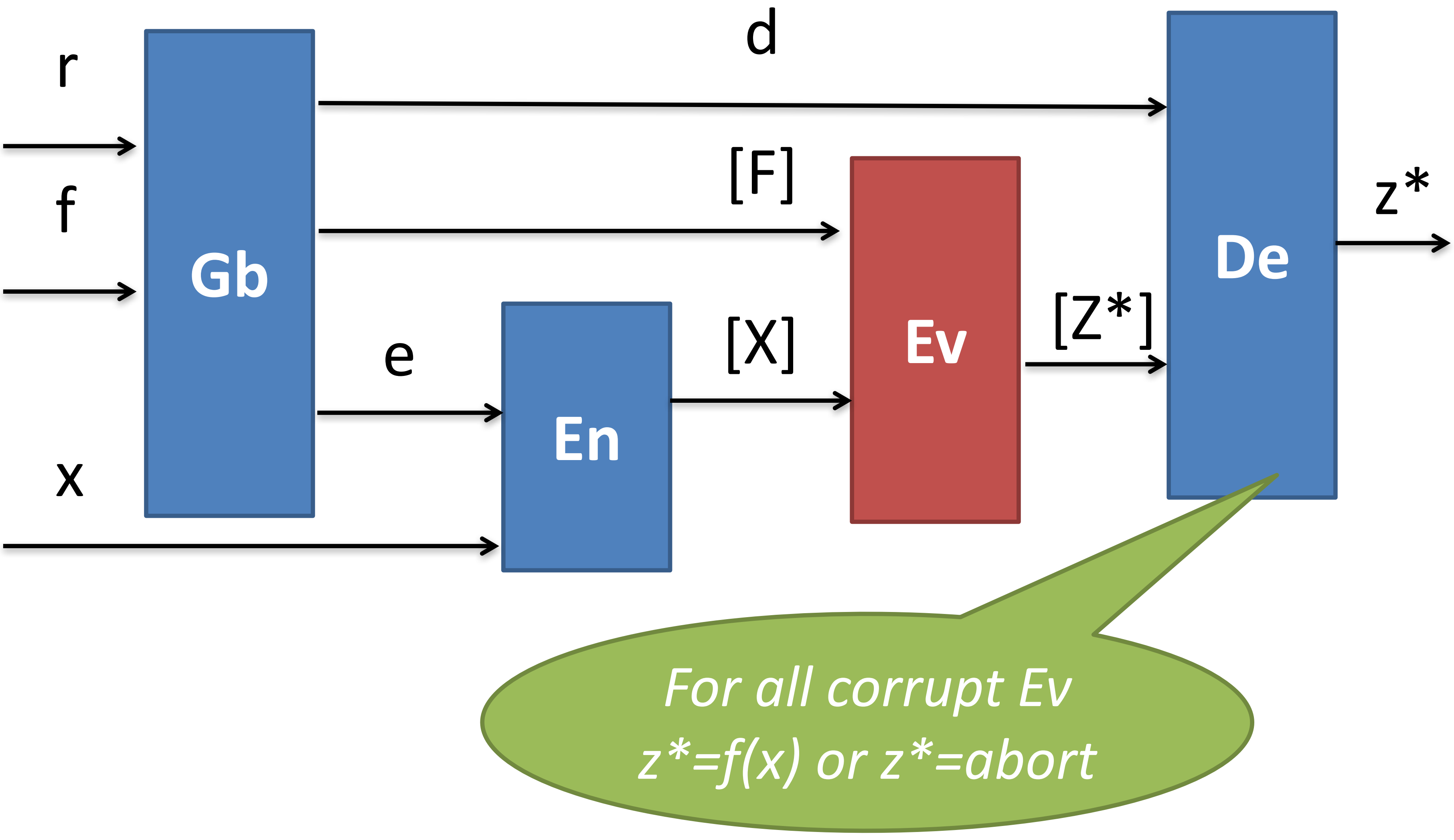
Active security of Yao



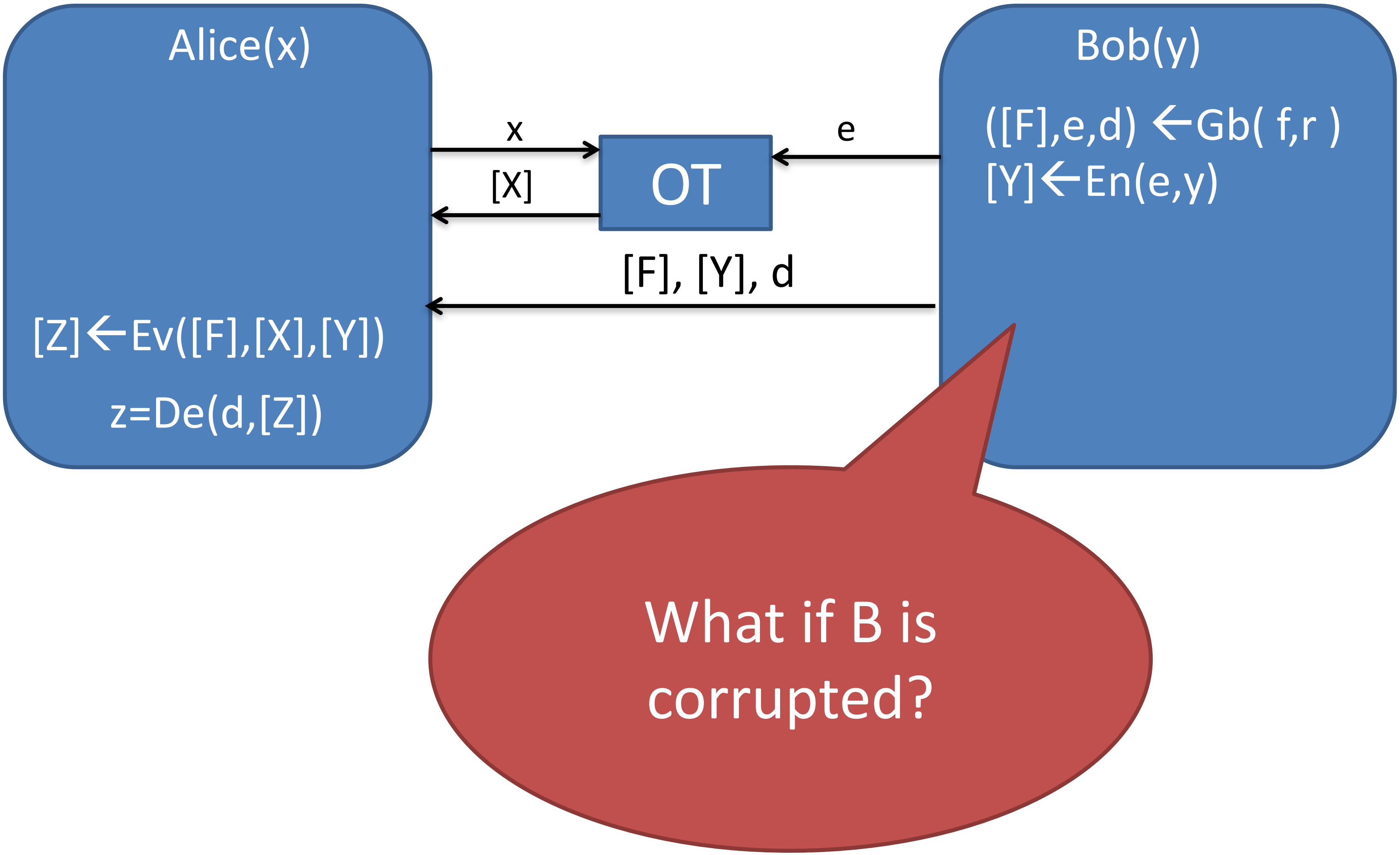
Active security of Yao (v2, Bob gets output)



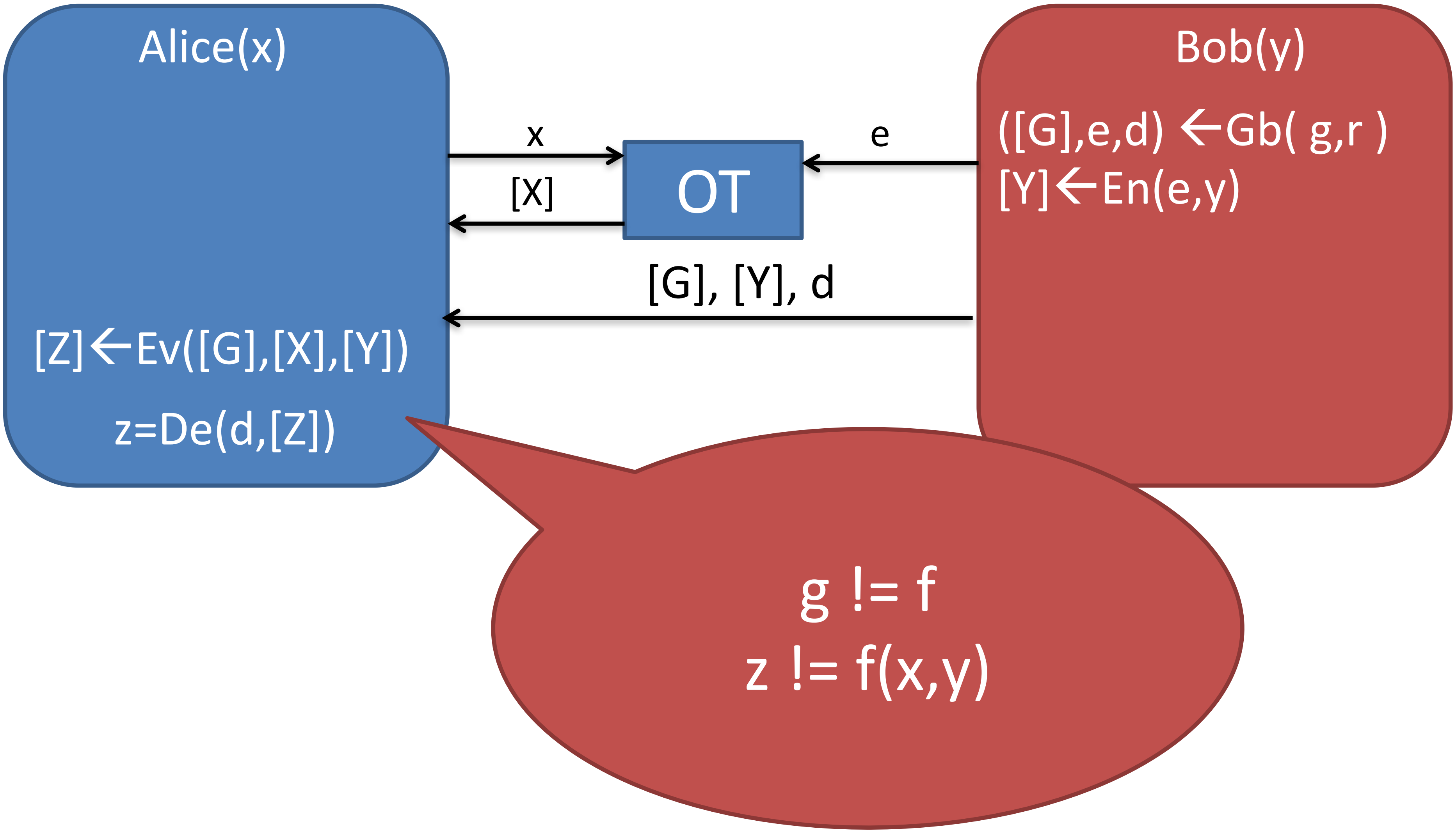
Garbled Circuits: Authenticity



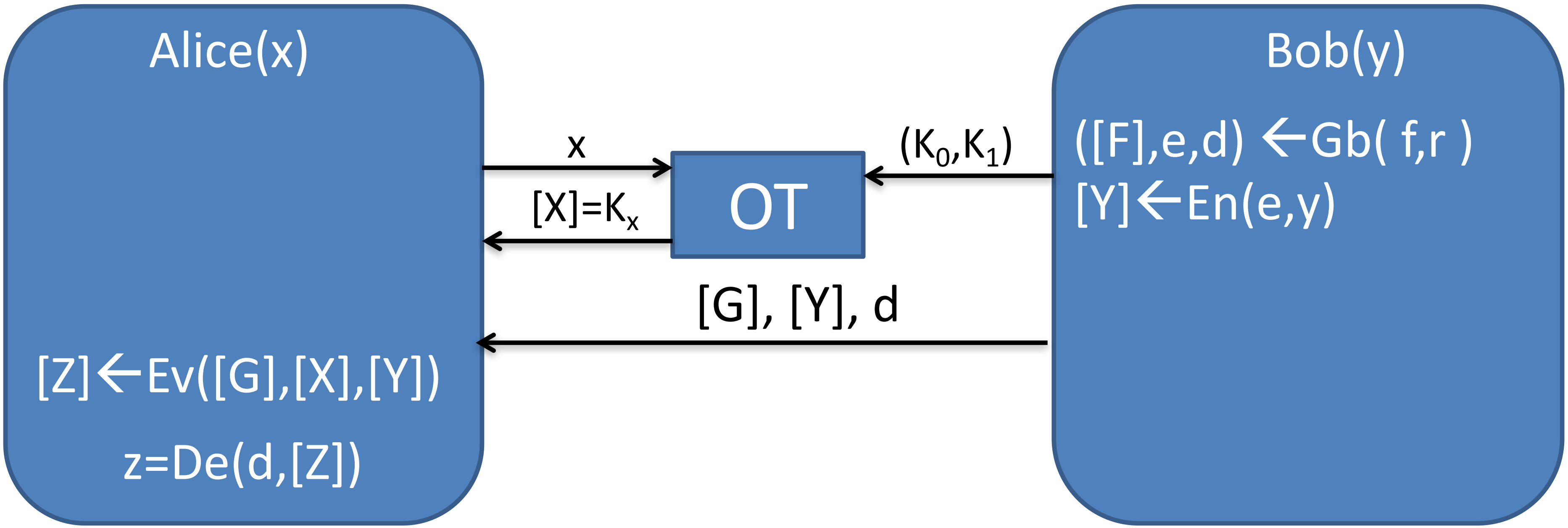
Active security of Yao



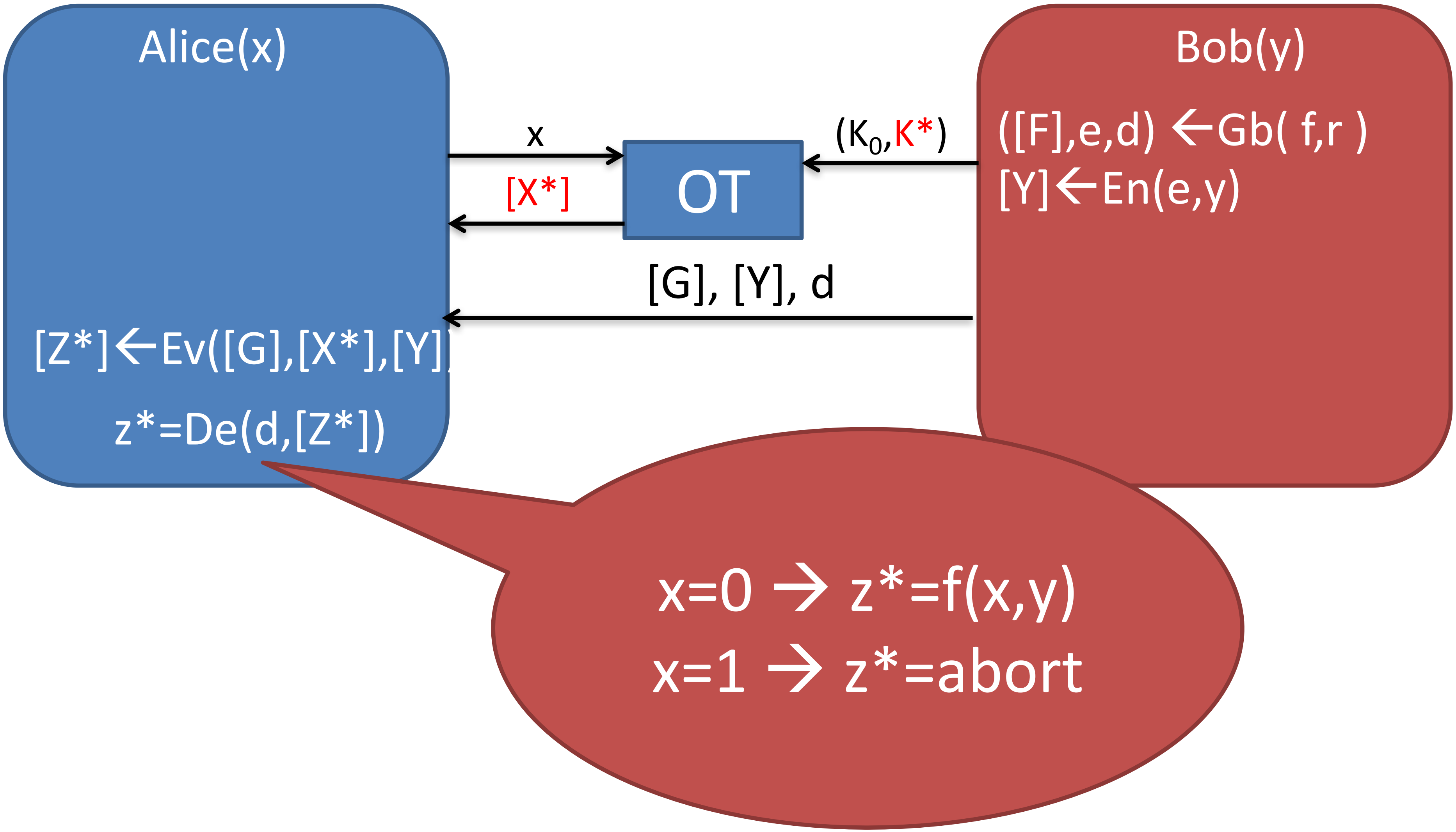
Insecurity 1 (wrong f)



Insecurity 2 (selective failure)



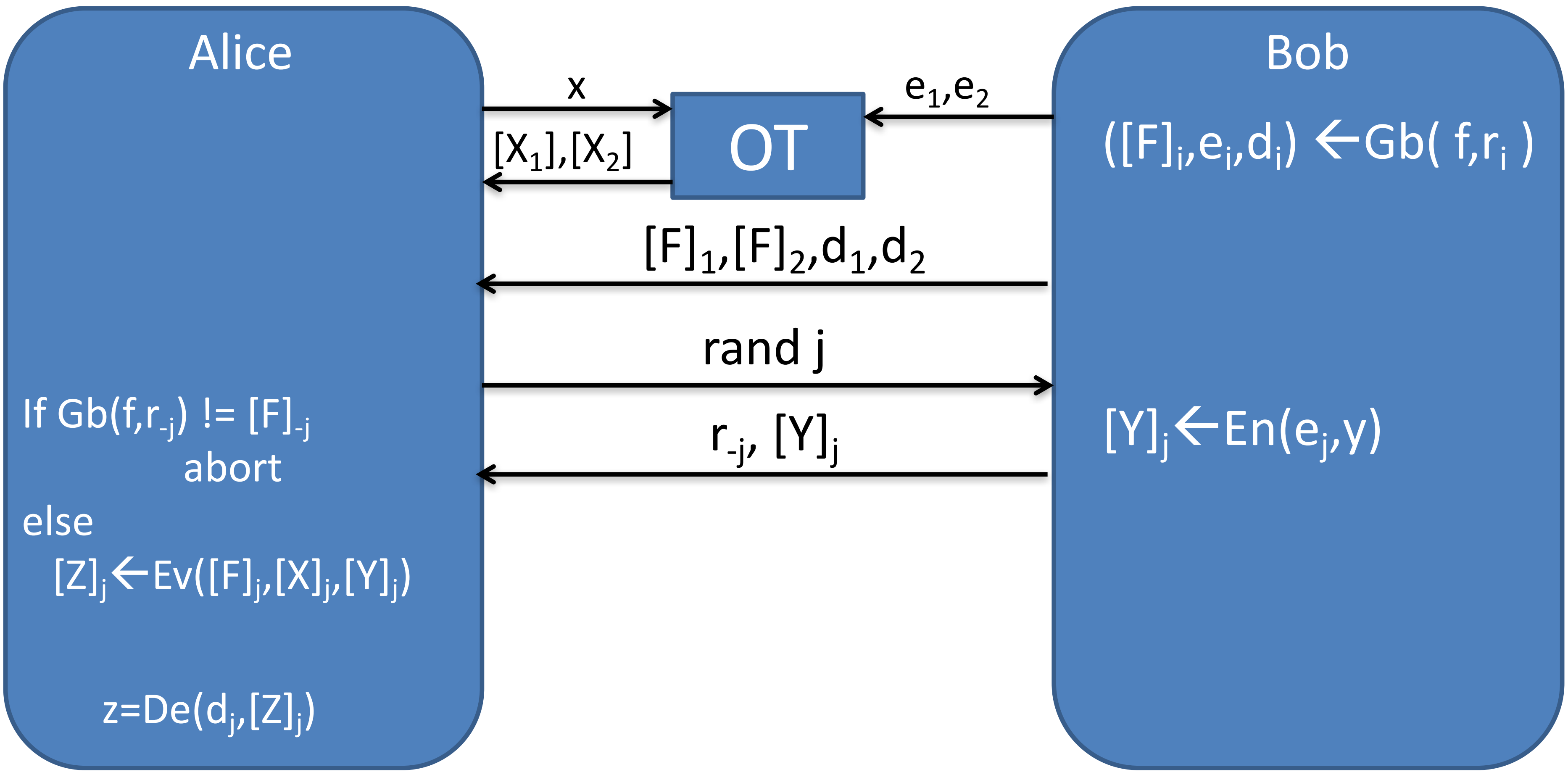
Insecurity 2 (selective failure)



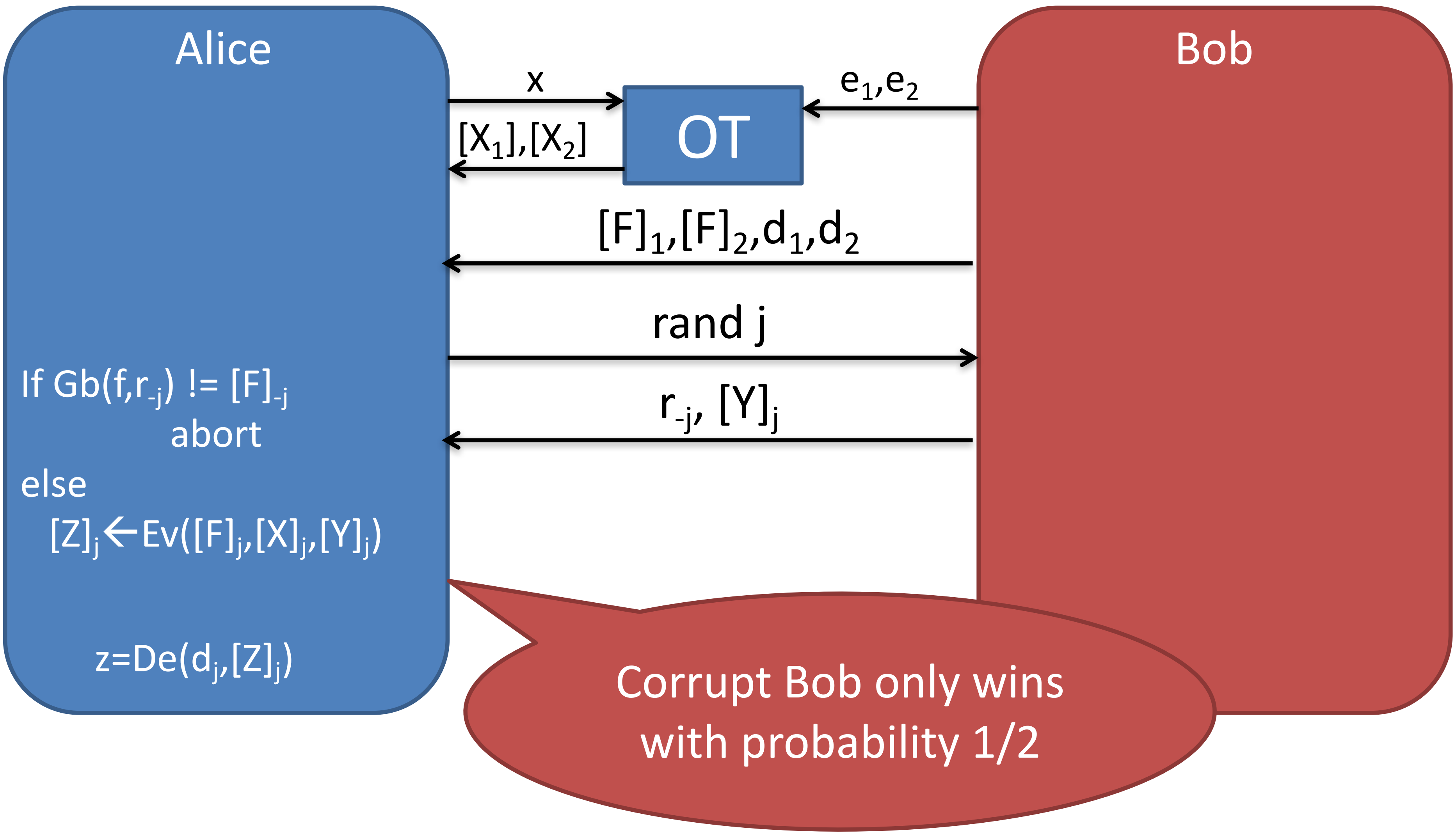
SIMPLE TRICKS FOR ACTIVE SECURITY

Cut-And-Choose

2PC, simple cut-and-choose



2PC, simple cut-and-choose



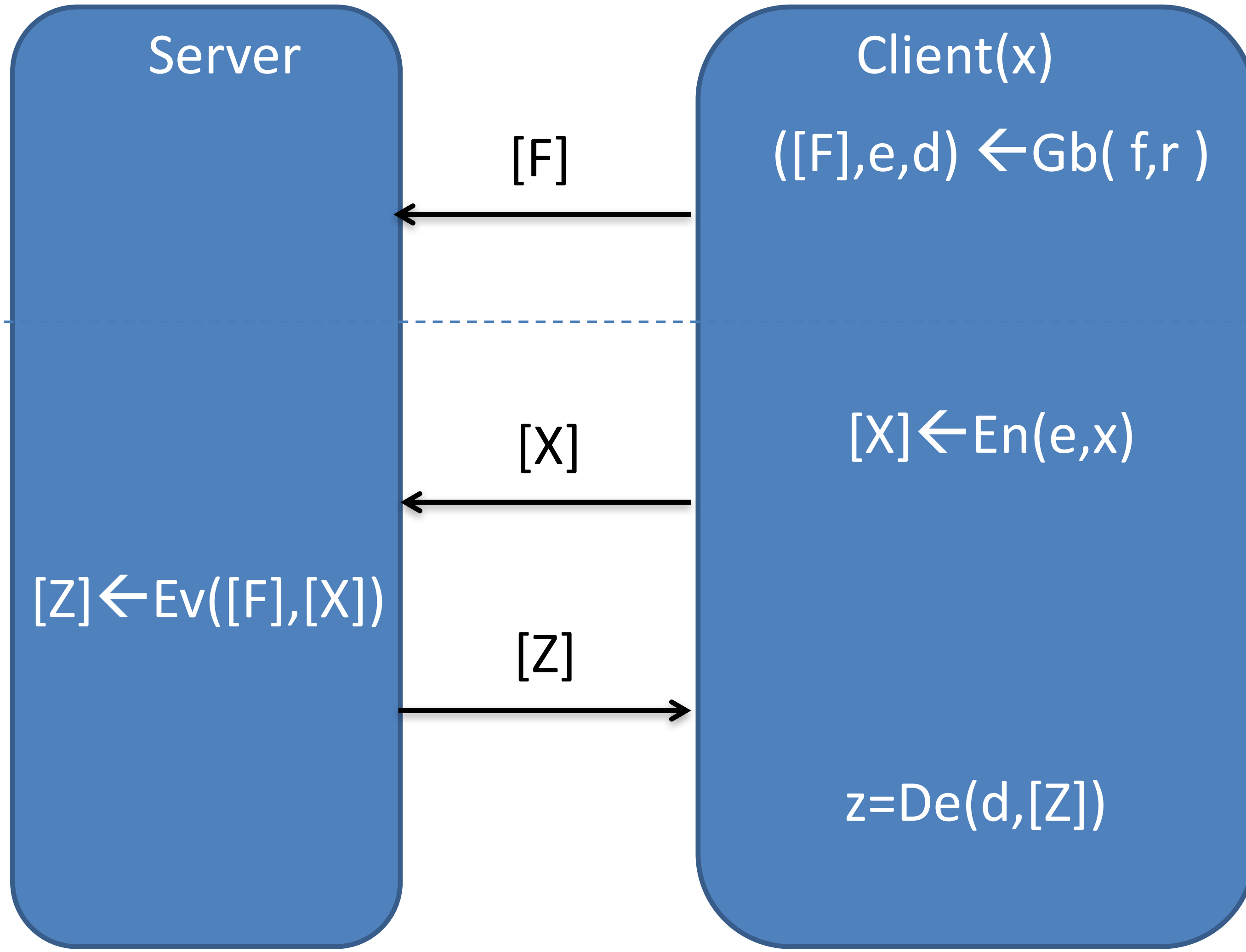
2PC, cut-and-choose

- Simple cut-and-choose
  - Garble k, check k-1, evaluate 1.
  - Security 1-1/k

- “Real cut-and-choose”
  - Use O(k) circuits, get security 2^{-k}
  - Requires more complex techniques

Delegation via GC

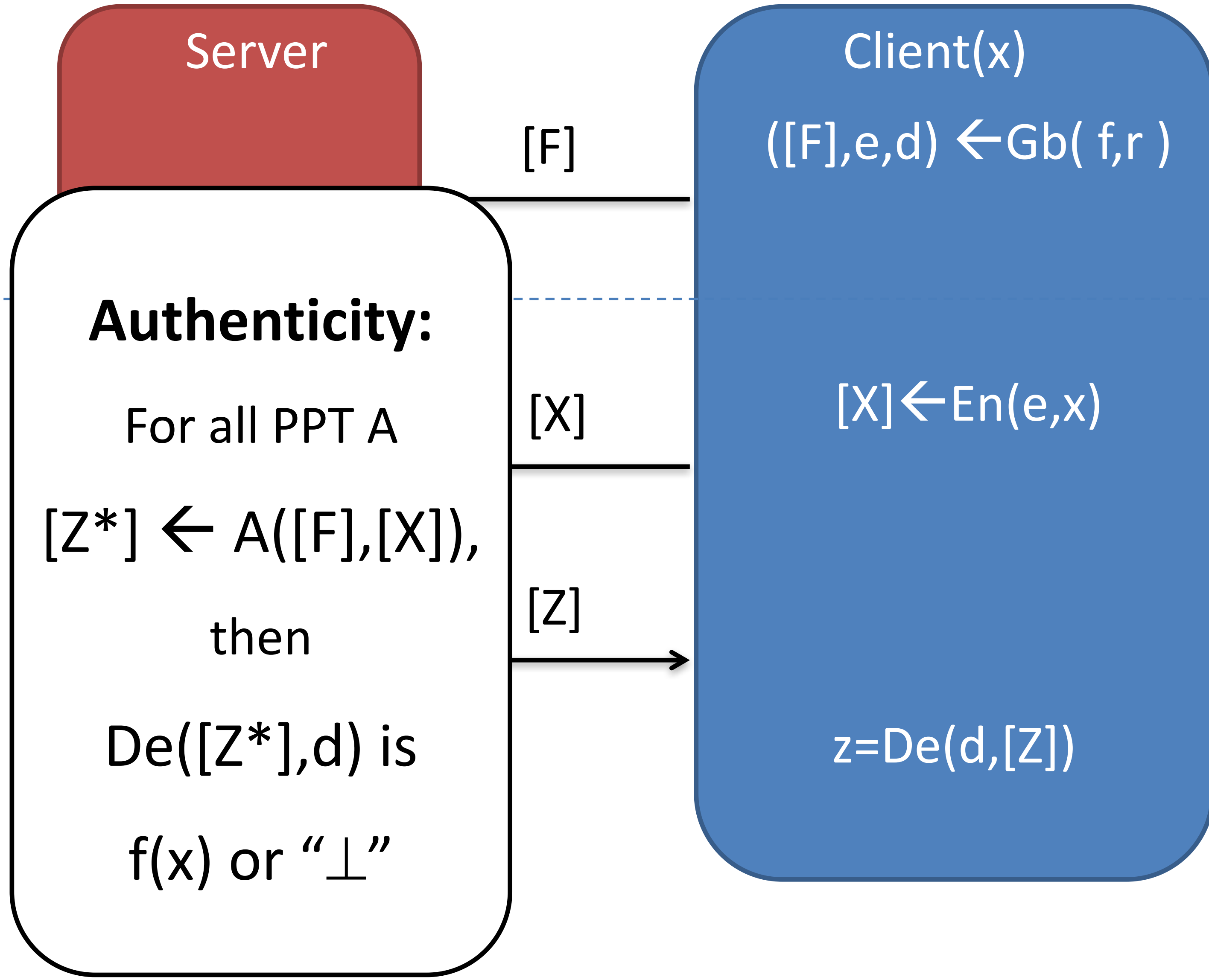
Application 1: Delegation via GC



Preprocessing:
The client work is
proportional
to |f|

Online:
The client can
Delegate comp.
The work of the client
Is independent of
|f|

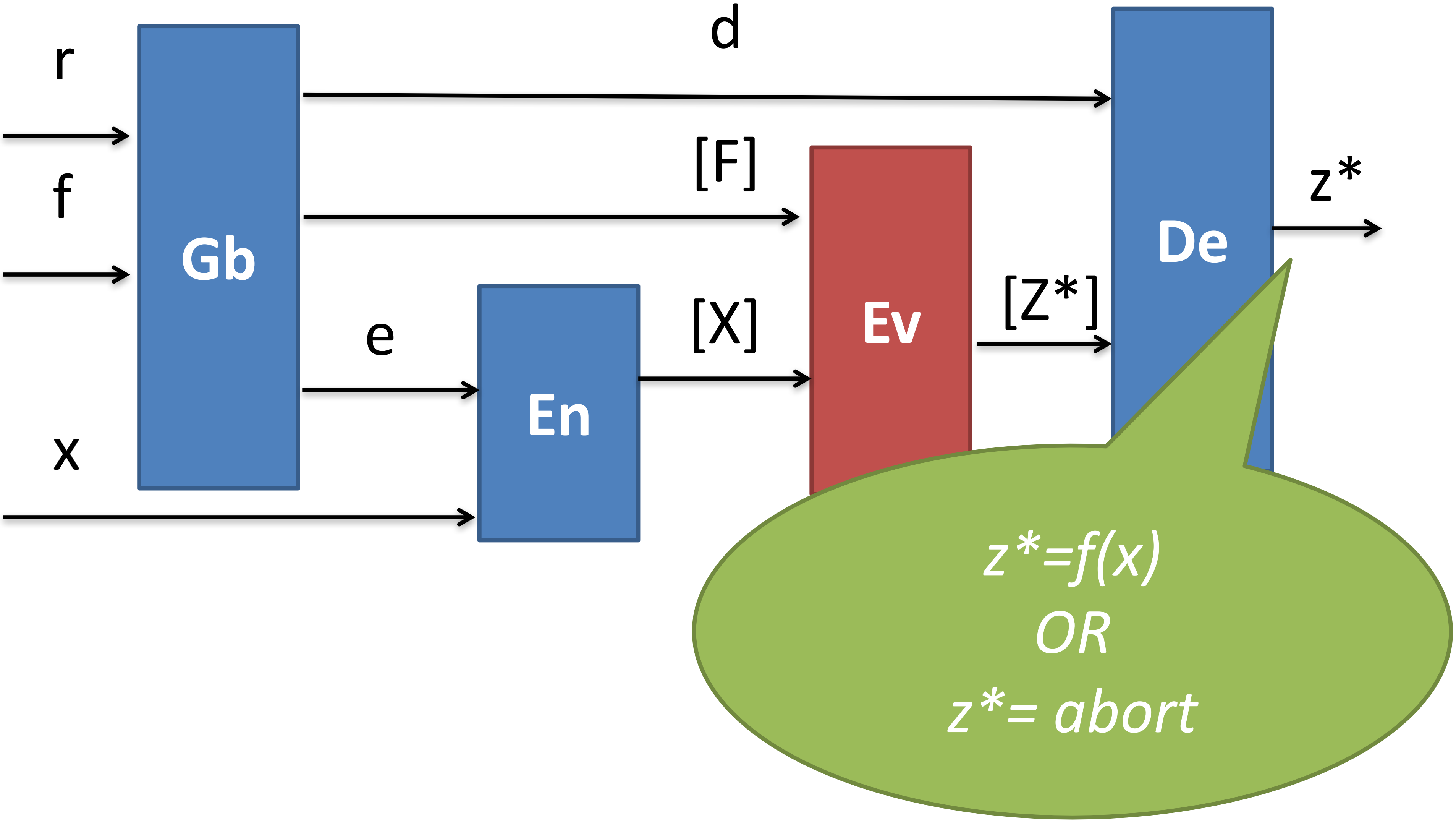
Application 1: Delegation via GC



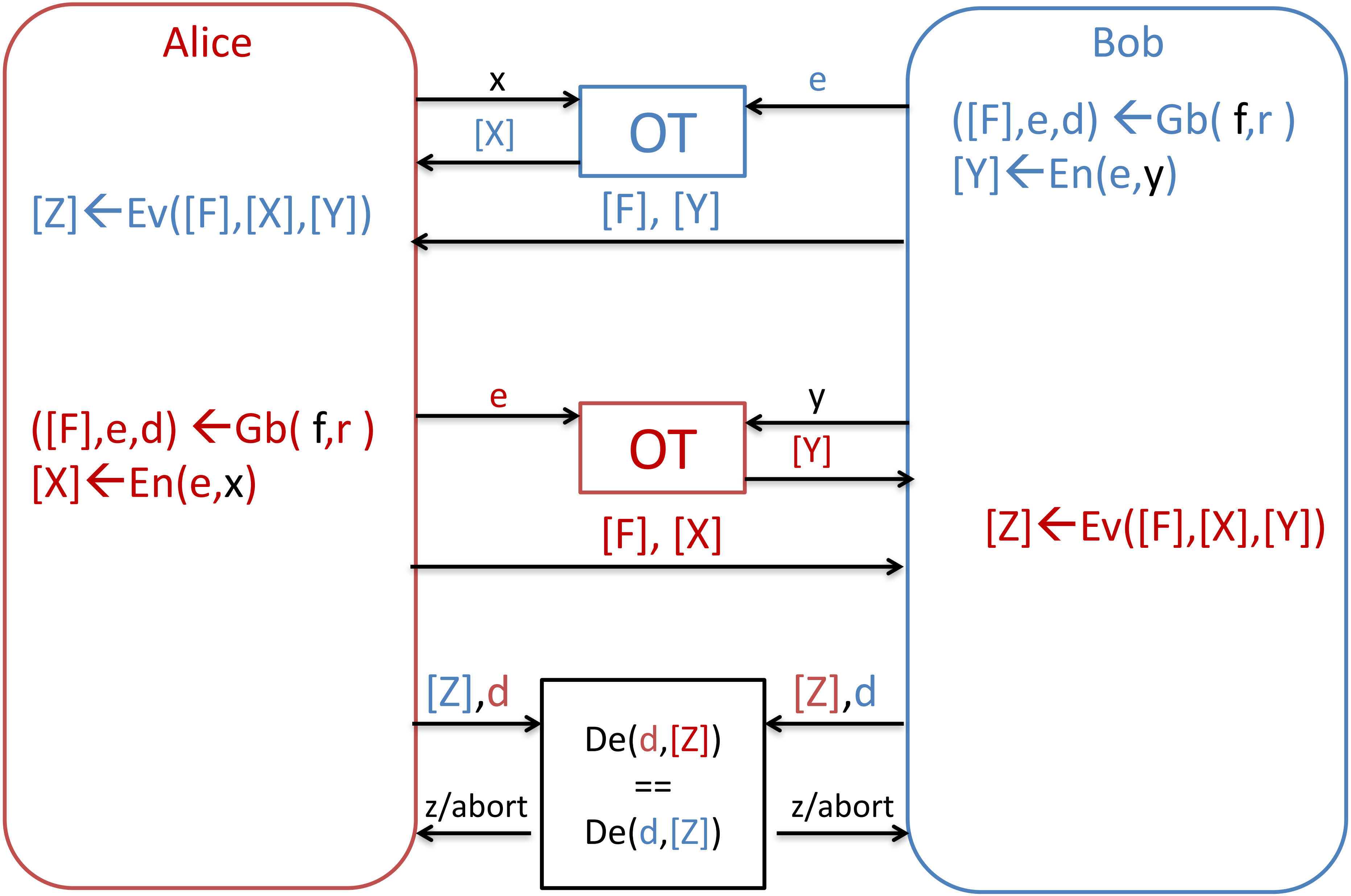
Preprocessing:
The client work is proportional to |f|

Online:
The client can Delegate comp.
The work of the client Is independent of |f|

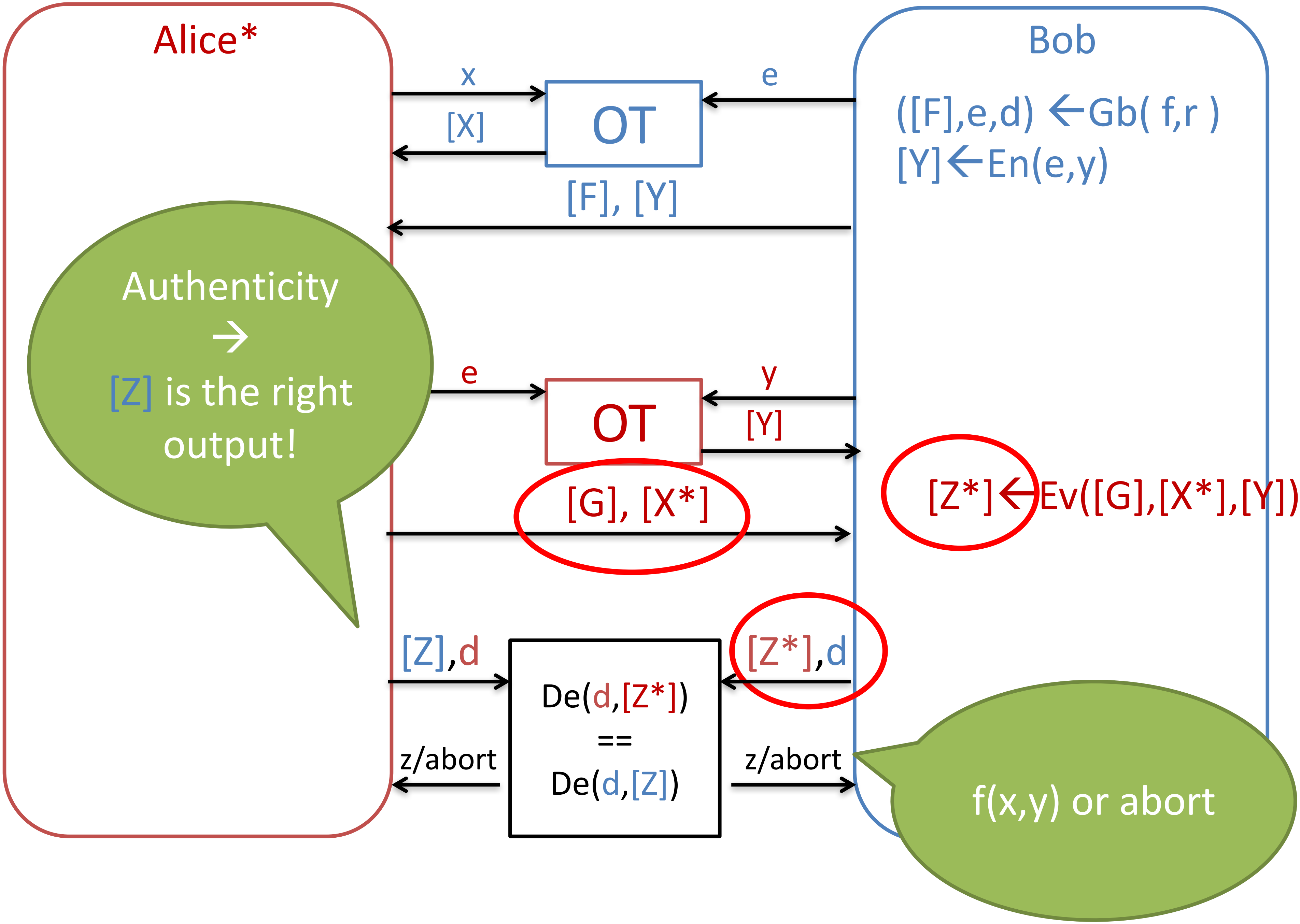
Garbled Circuits: Authenticity



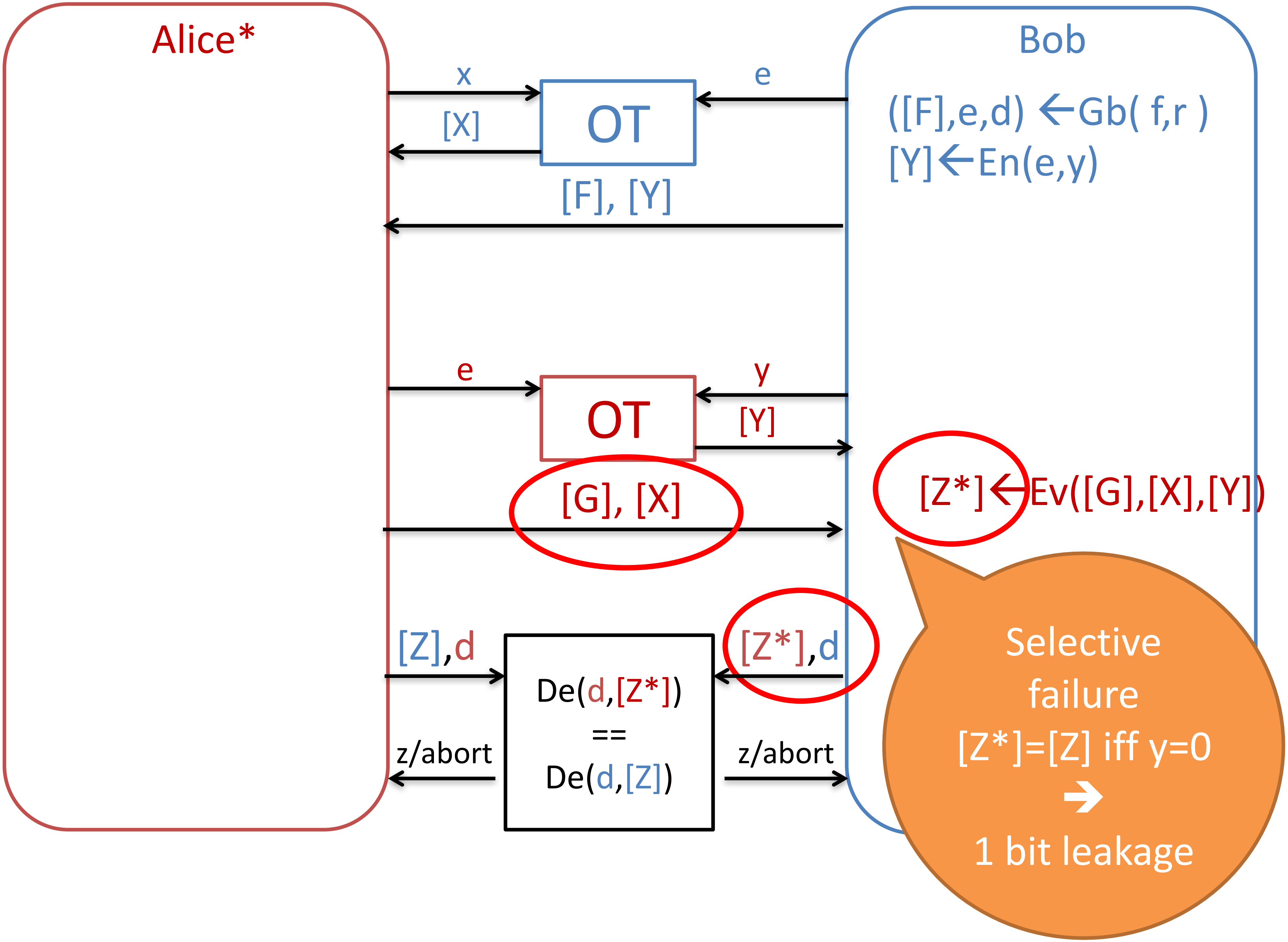
Dual Execution



Alice
Bob
x
e
OT
[X]
([F],e,d) ←Gb(f,r)
[Y]←En(e,y)
[Z]←Ev([F],[X],[Y])
[F], [Y]
e
y
OT
[Y]
([F],e,d) ←Gb(f,r)
[X]←En(e,x)
[Z]←Ev([F],[X],[Y])
[F], [X]
[Z],d
[Z],d
De(d,[Z])
==
De(d,[Z])
z/abort
z/abort



Alice*
Bob
x
e
OT
[X]
([F],e,d) ←Gb(f,r)
[Y]←En(e,y)
[F], [Y]
Authenticity
→
[Z] is the right output!
e
y
OT
[Y]
[Z*]←Ev([G],[X*],[Y])
[G], [X*]
[Z],d
[Z*],d
De(d,[Z*])
==
De(d,[Z])
z/abort
z/abort
f(x,y) or abort



Alice*
Bob
x
e
OT
[X]
([F],e,d) ←Gb(f,r)
[Y]←En(e,y)
[F], [Y]
e
y
OT
[Y]
[G], [X]
[Z*]←Ev([G],[X],[Y])
[Z],d
[Z*],d
De(d,[Z*])
==
De(d,[Z])
z/abort
z/abort
Selective
failure
[Z*]=[Z] iff y=0
➔
1 bit leakage

Recap: Garbled Circuits

- Garbled circuits: allow to evaluate *encrypted functions* on *encrypted inputs*
  - With properties like *privacy*, *authenticity*, etc.
- Applications: constant-round 2PC
- Different techniques for garbling gates
  - Efficiency vs. Assumptions
- Active security
  - How to check that the right function is garbled?
  - Cut-and-choose and other tricks…

Thanks!

Want more?

- **Cryptographic Computing – Foundations**
  - http://orlandi.dk/crycom
  - Programming & Theory Exercises
  - Will be happy to answer questions by mail!