~~Multi~~ 2-Party Computation
Part 3

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

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

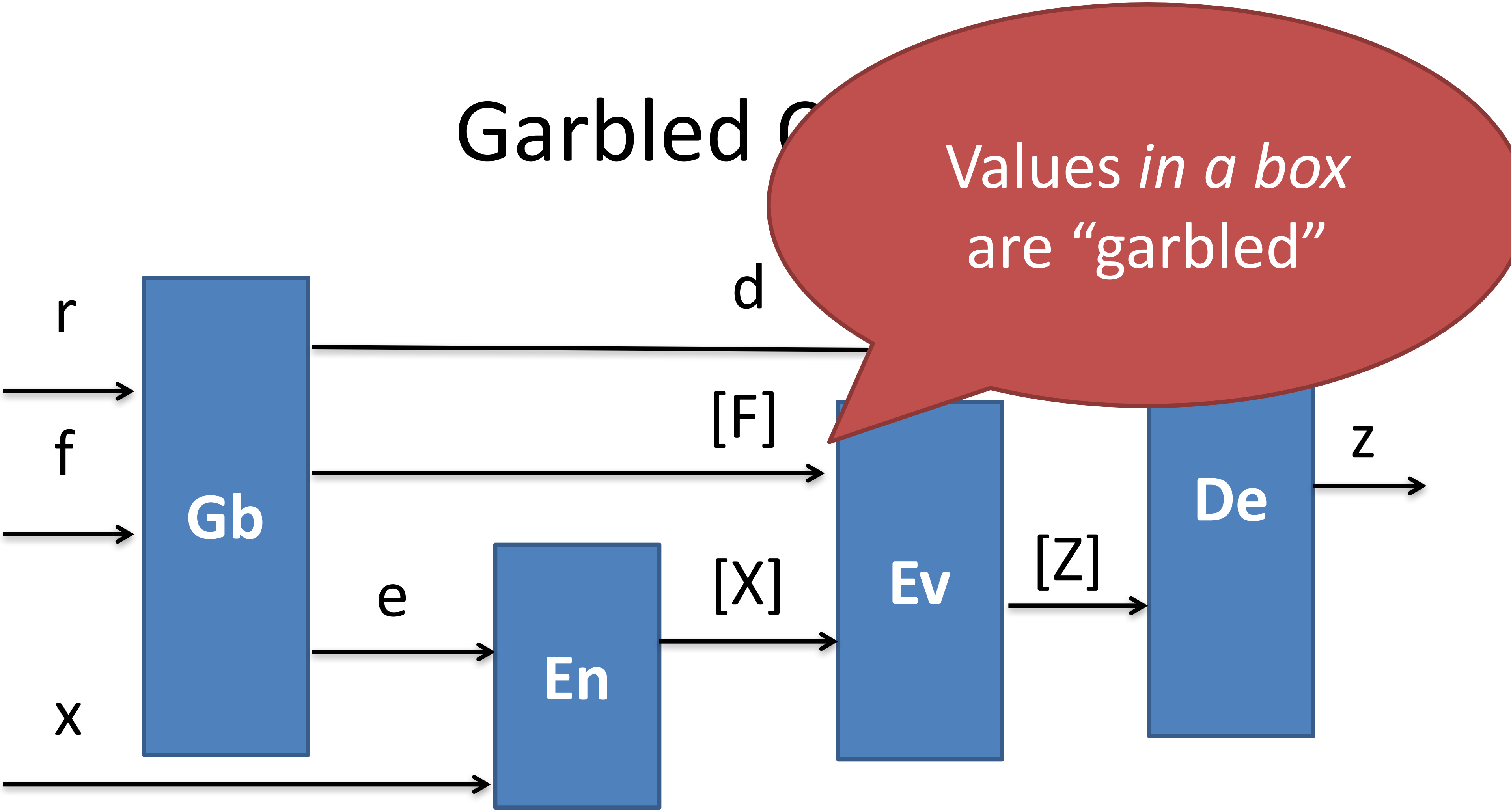
Cryptographic primitive that allows to evaluate

encrypted functions

on

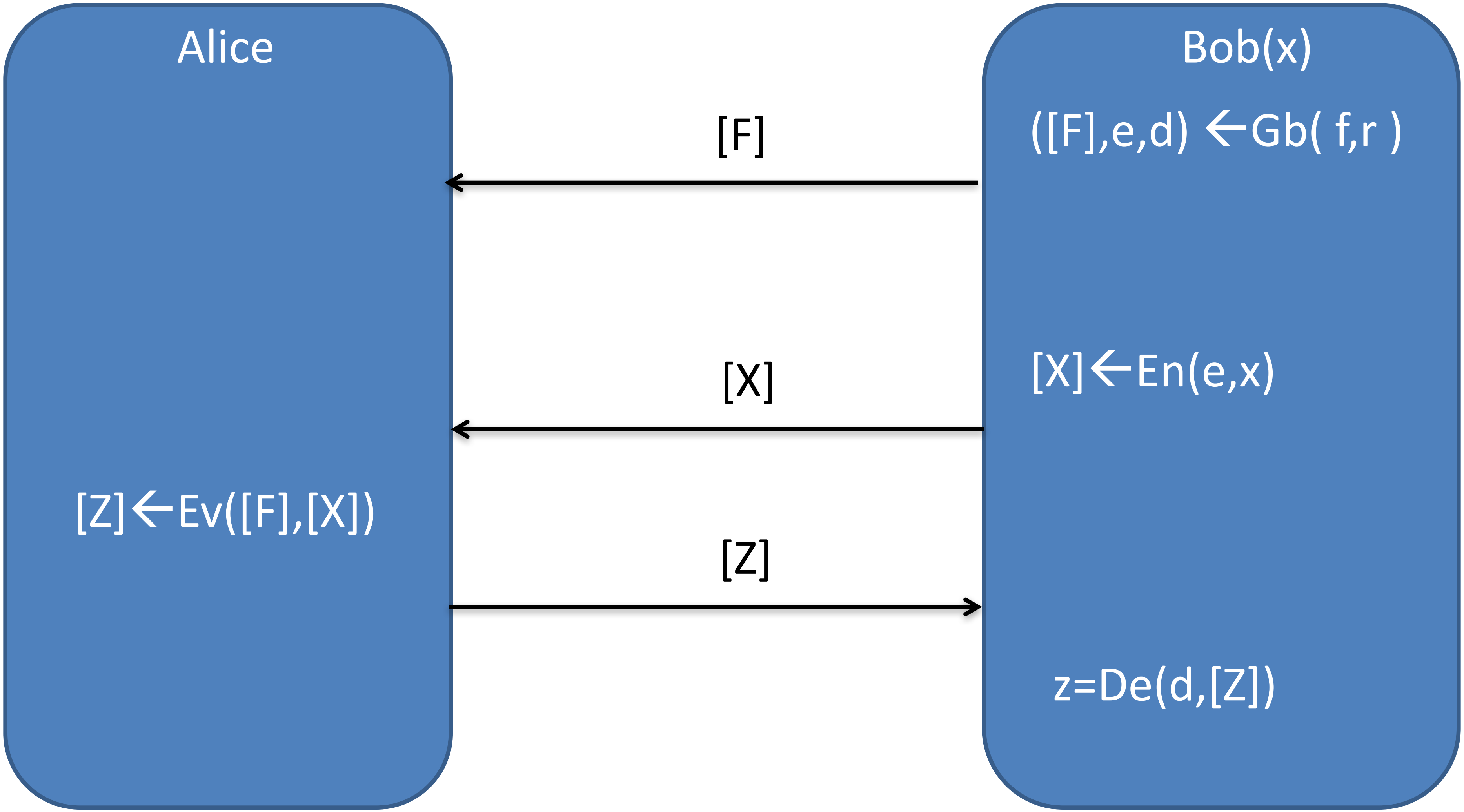
encrypted inputs

Garbled C

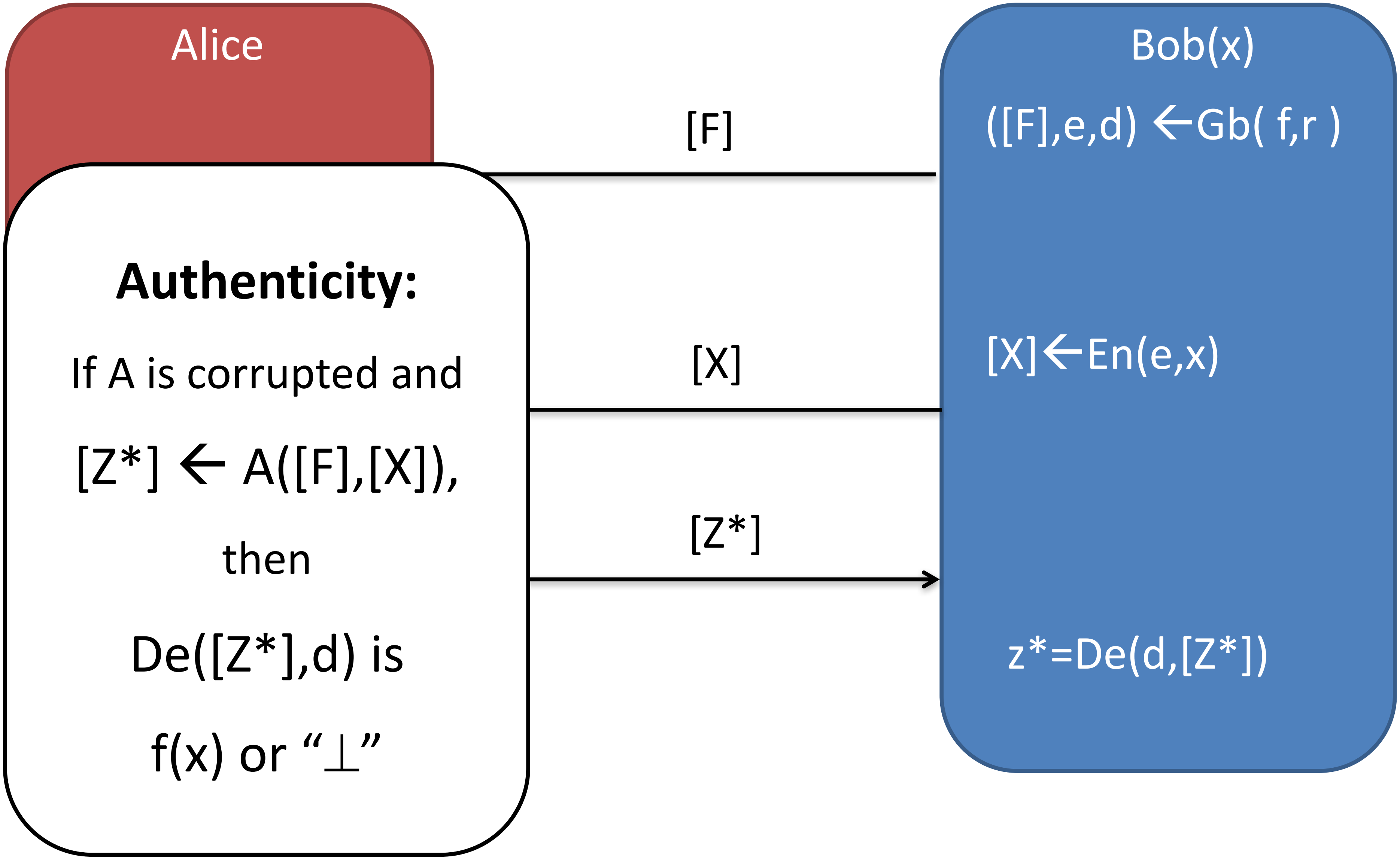


Correct if z=f(x)

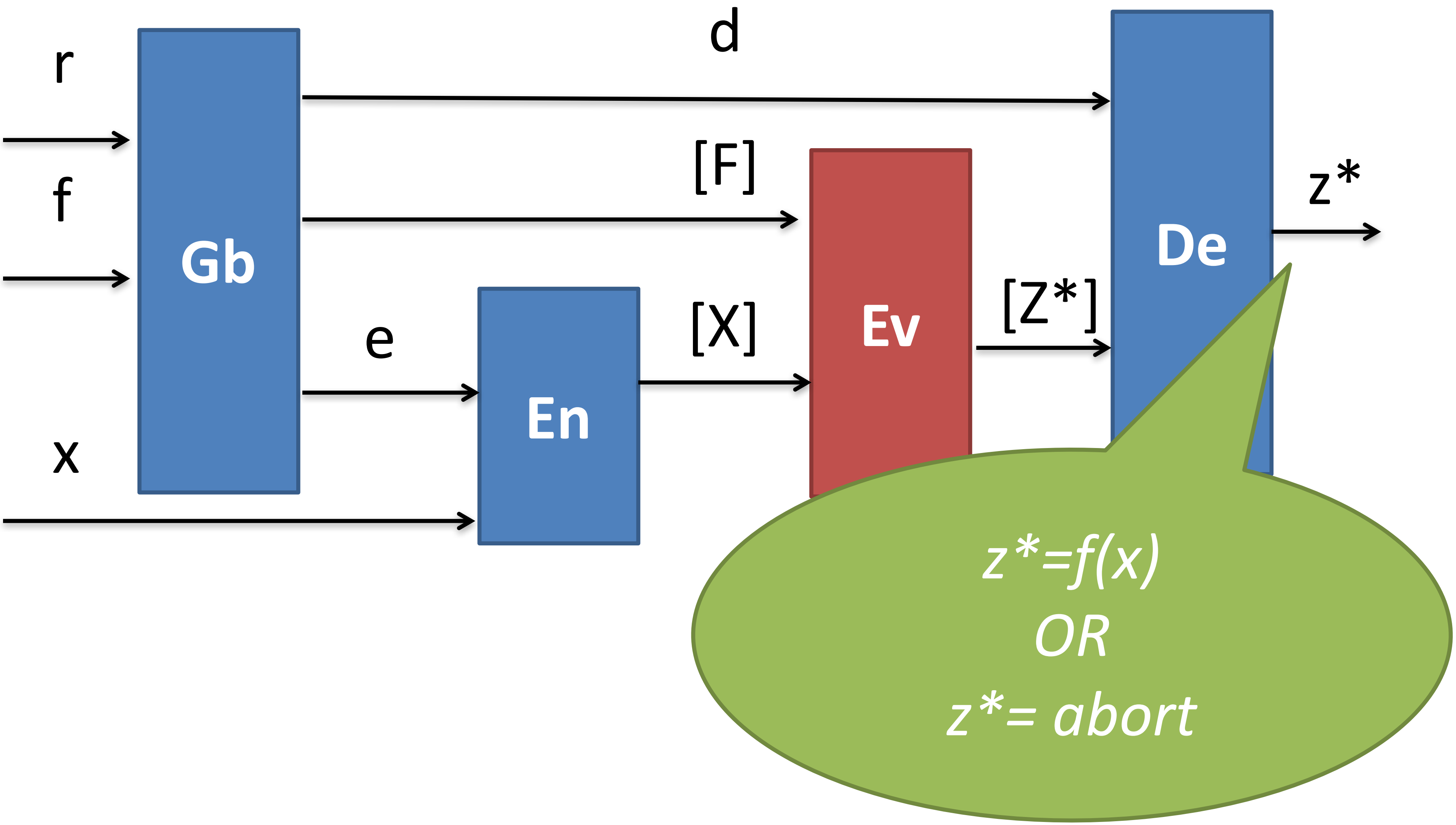
Application 1: Delegation via GC



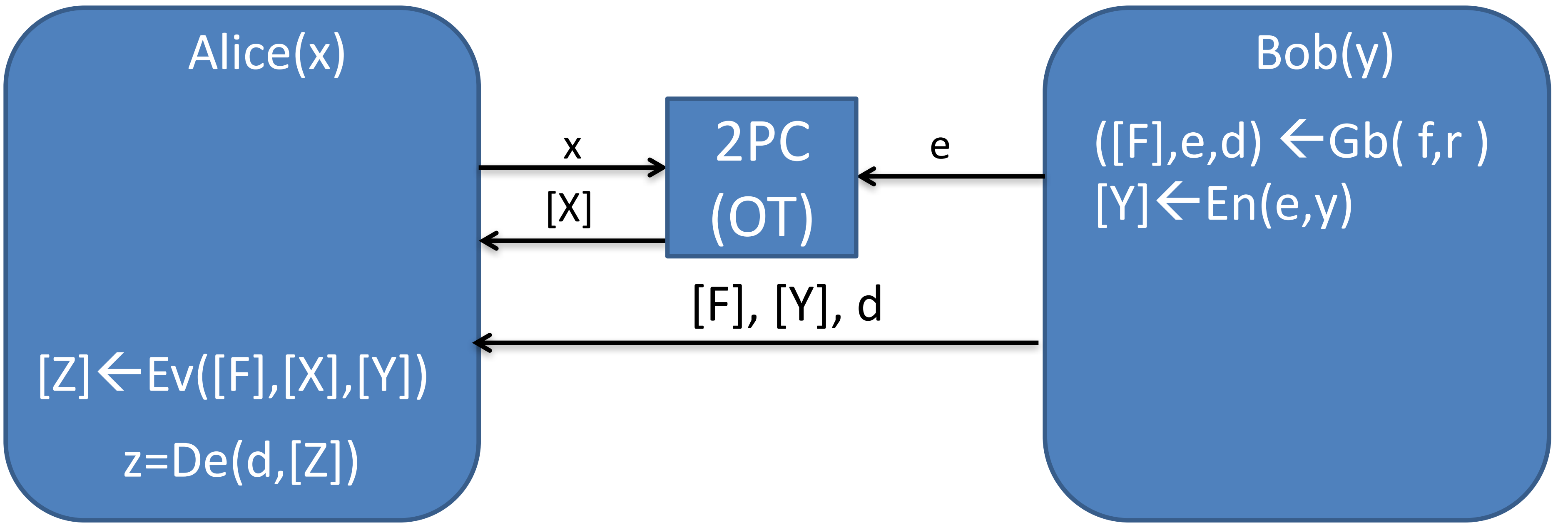
Application 1: Delegation via GC



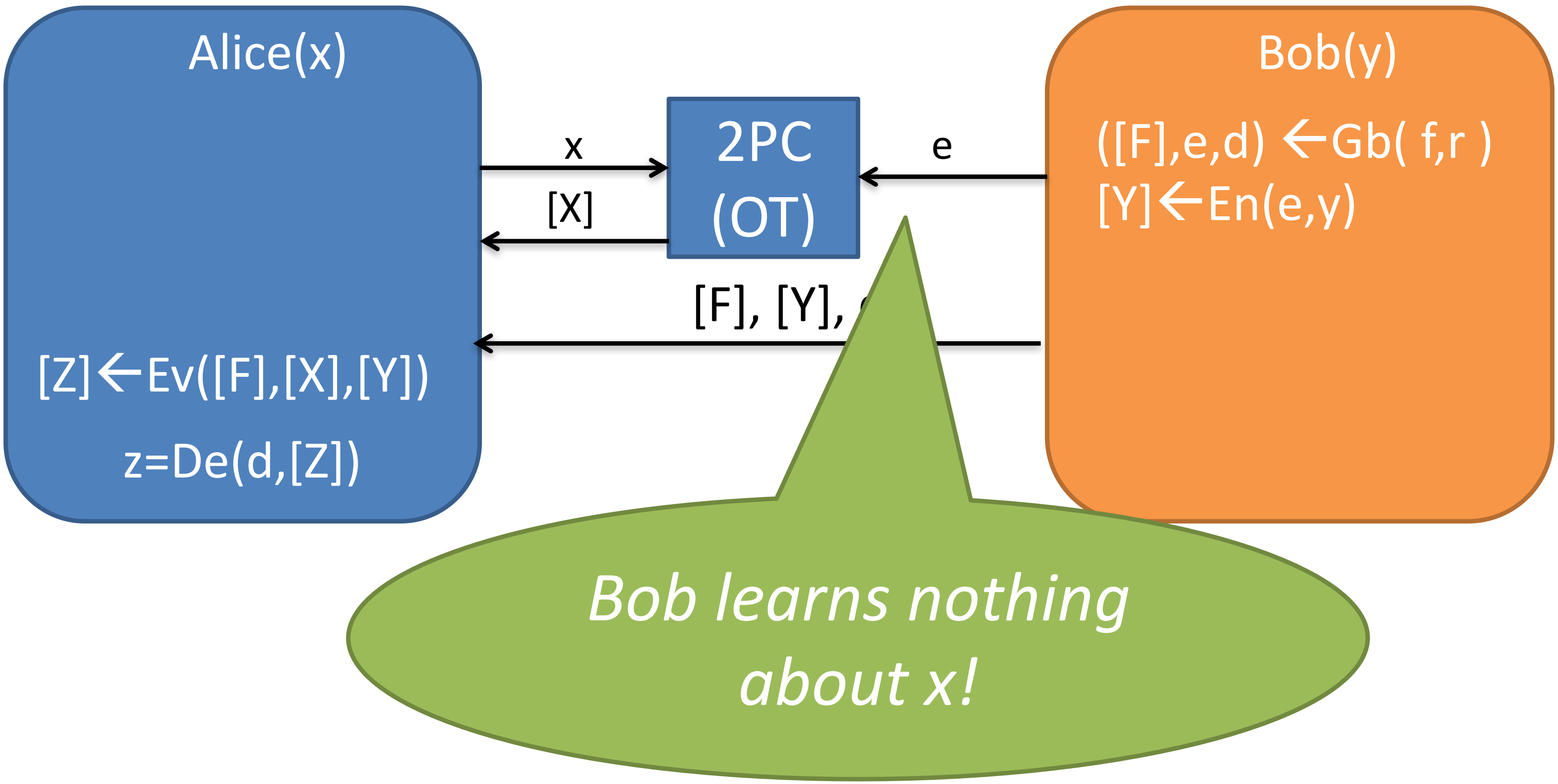
Garbled Circuits: Authenticity



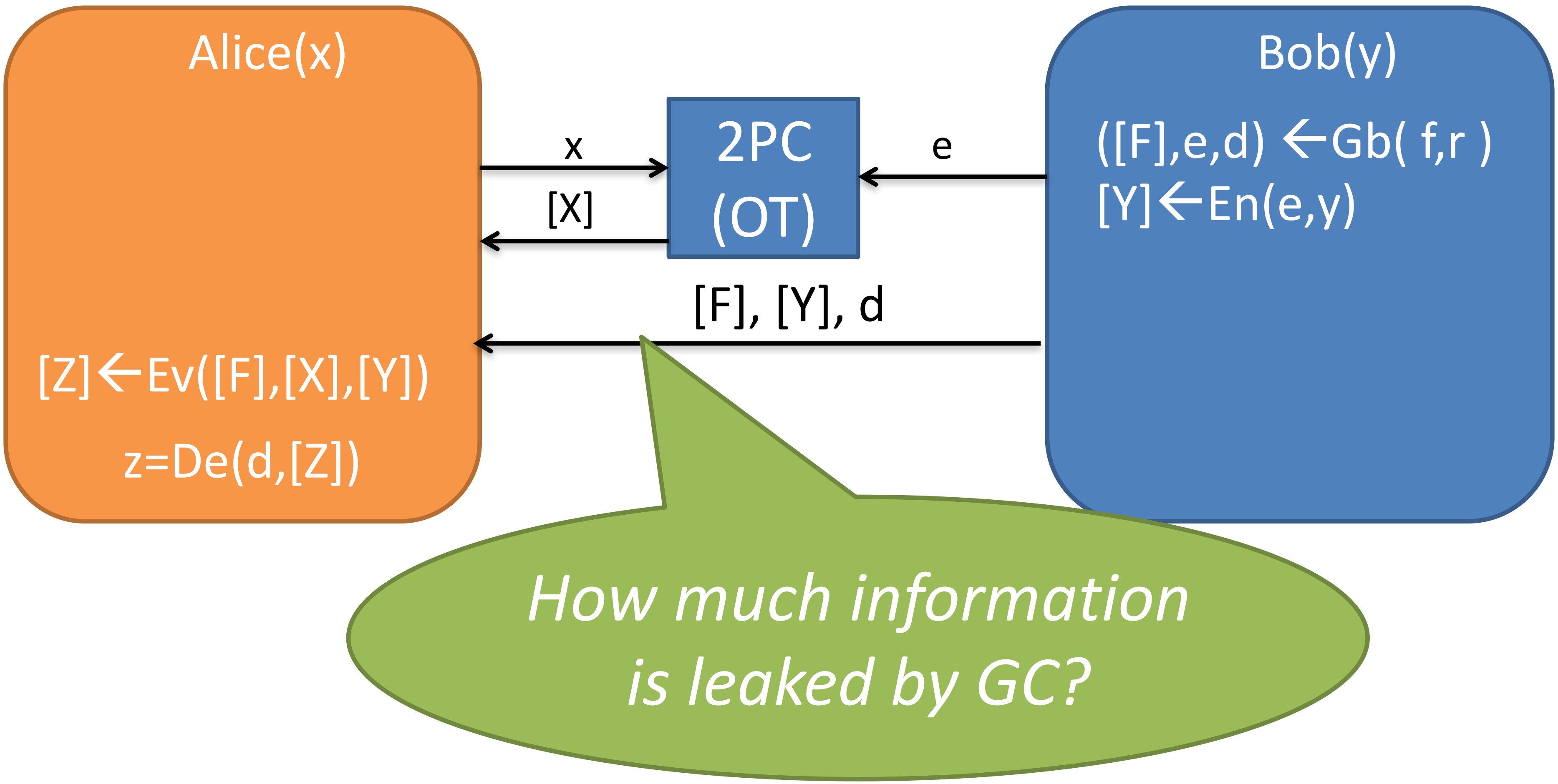
Application 2:
Passive Constant Round 2PC (Yao)



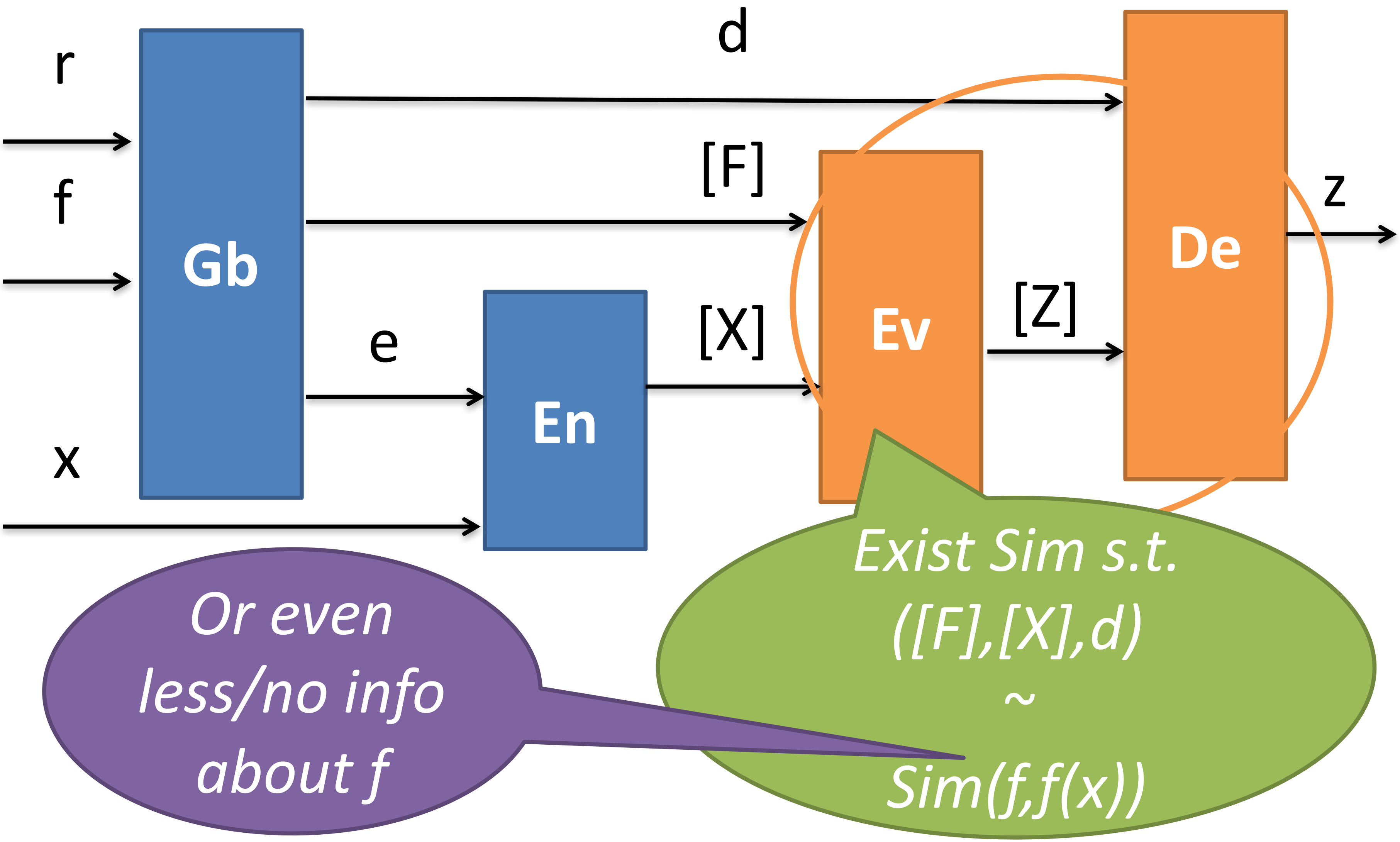
Application 2:
Passive Constant Round 2PC (Yao)



Application 2:
Passive Constant Round 2PC (Yao)



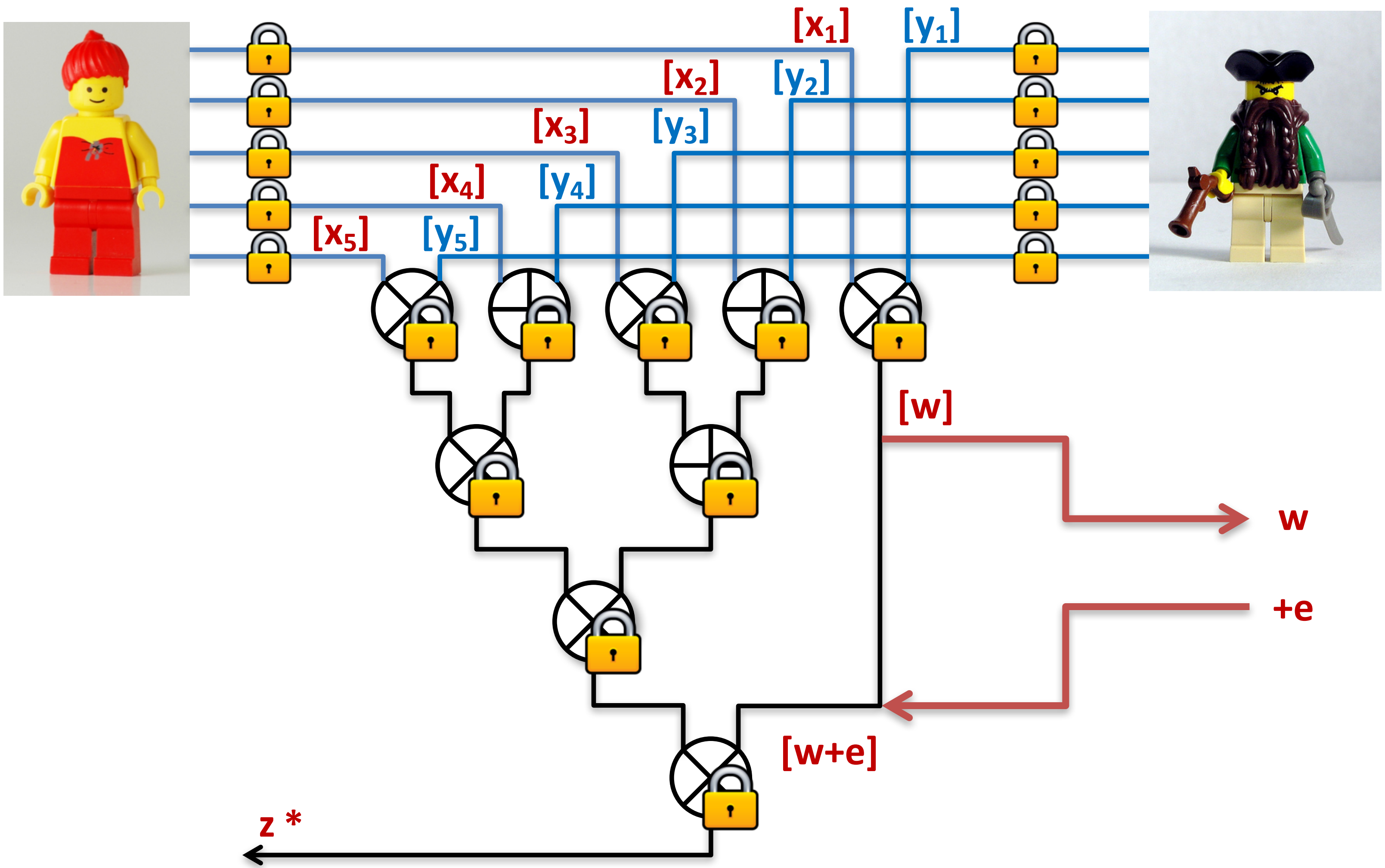
Garbled Circuits: Privacy



Part 3: Garbled Circuits

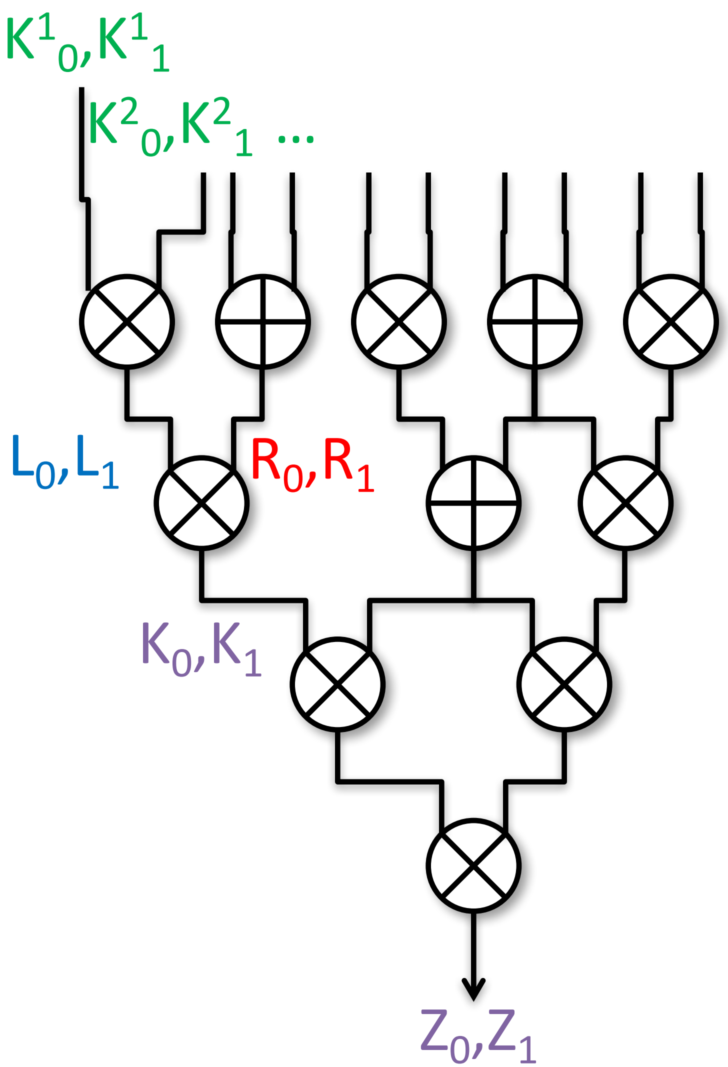
- Definitions and Applications
- **Garbling gate-by-gate: Basic and optimizations**
- Active security 101: simple-cut-and choose, dual-execution

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

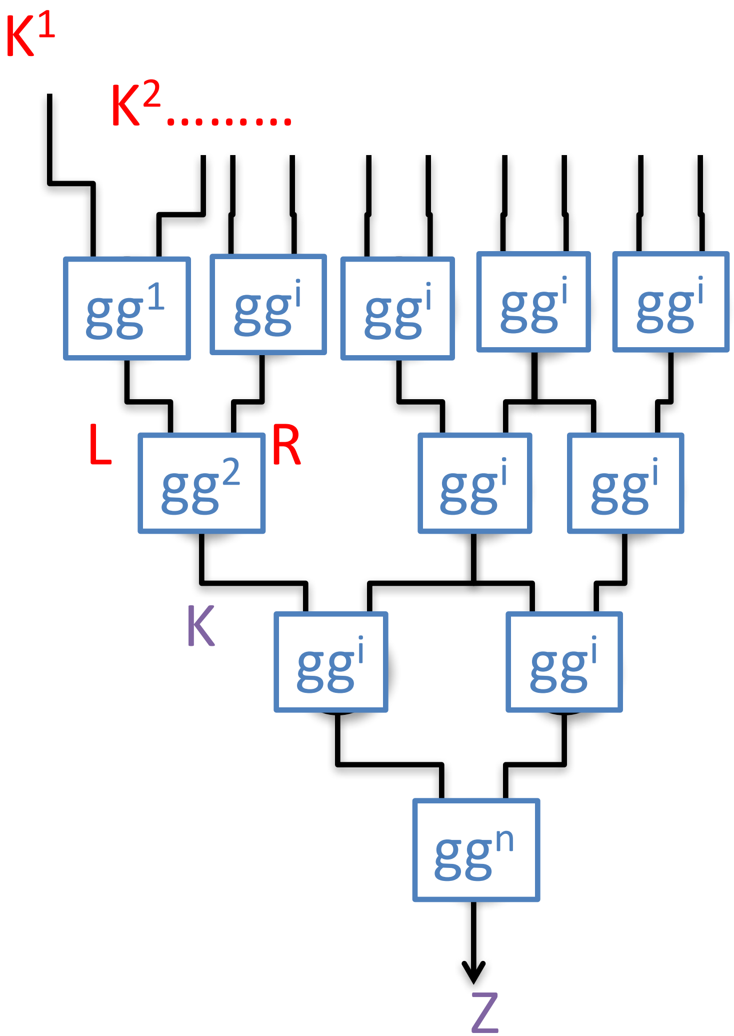
$[X] = En(e, x)$

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

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

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

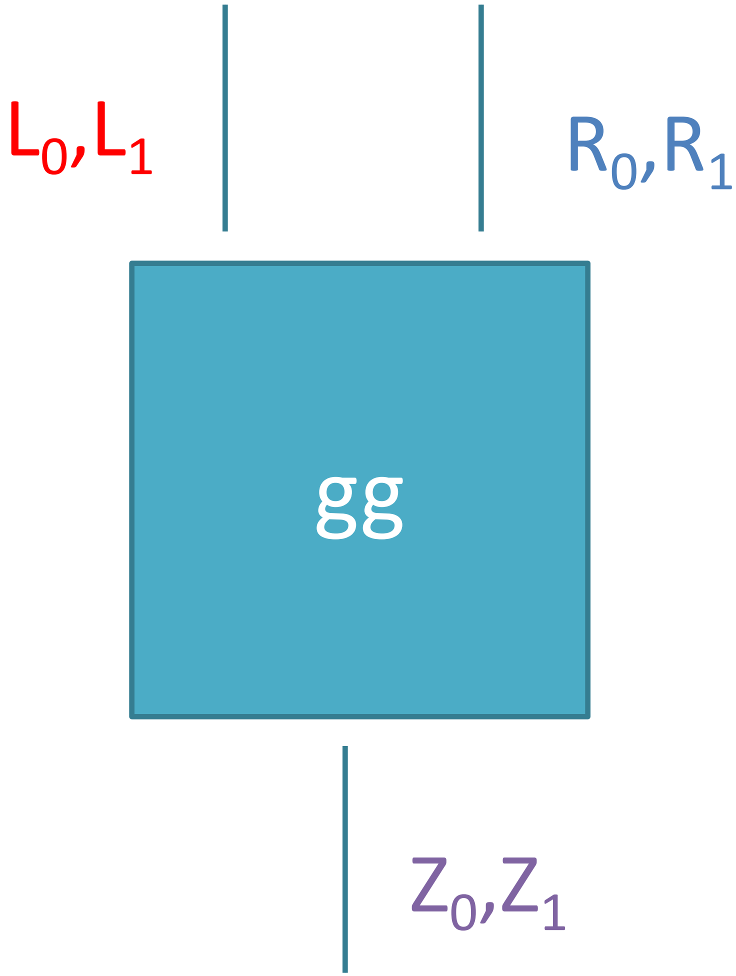
Evaluating a GC : [Z] ← Ev([F],[X])



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

INDIVIDUAL GATES GARBLING/EVALUATION

Notation



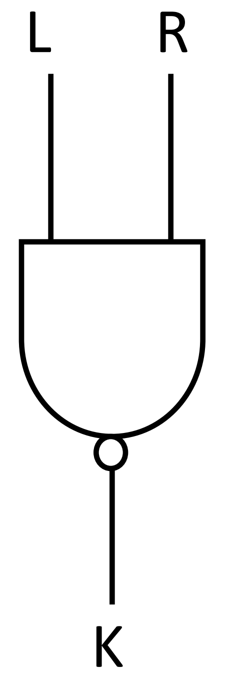
- A garbled gate is a gadget that given two inputs keys gives you the right output key *(and nothing else)*

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

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

Yao Gate Garbling (1)

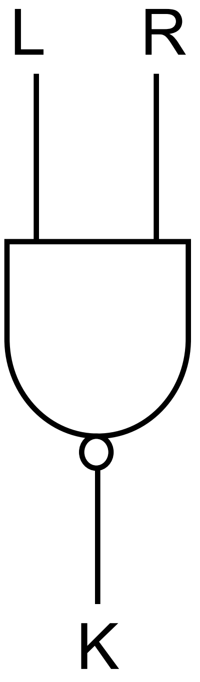
L	R	K
0	0	1
0	1	1
1	0	1
1	1	0



- NAND gate

Yao Gate Garbling (2)

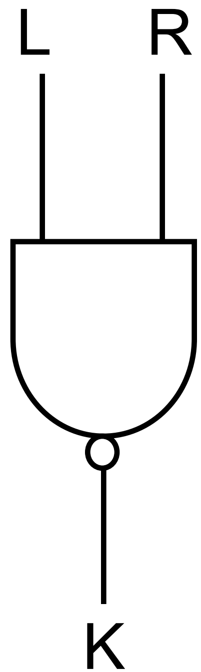
L	R	K
L_0	R_0	K_1
L_0	R_1	K_1
L_1	R_0	K_1
L_1	R_1	K_0



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

Yao Gate Garbling (3)

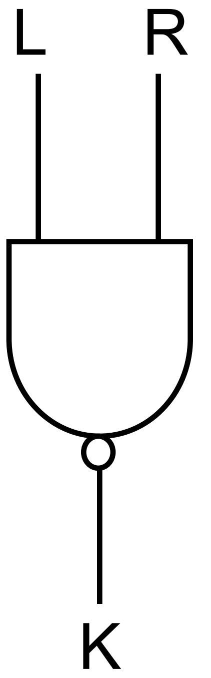
C
$C_1 = H(L_0,R_0) \oplus K_1$
$C_2 = H(L_0,R_1) \oplus K_1$
$C_3 = H(L_1,R_0) \oplus K_1$
$C_4 = H(L_1,R_1) \oplus K_0$



- Encrypt the output key with the input keys

Yao Gate Garbling (4)

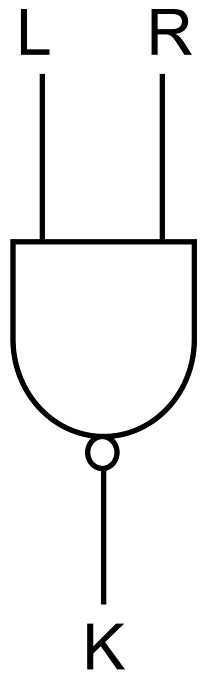
C
$C_1 = H(L_0,R_0) \oplus (K_1,0^k)$
$C_2 = H(L_0,R_1) \oplus (K_1,0^k)$
$C_3 = H(L_1,R_0) \oplus (K_1,0^k)$
$C_4 = H(L_1,R_1) \oplus (K_0,0^k)$



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

Yao Gate Garbling (5)

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



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

- 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

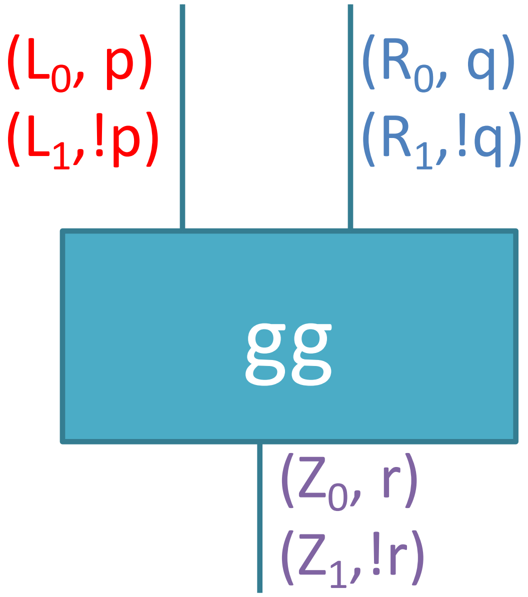
- Output is correct:
  - $t=0^k$ only for right row
- Evaluator learns nothing else:
  - Encryption + permutation

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

GARBLING OPTIMIZATIONS: POINT-AND-PERMUTE

Point-and-permute

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

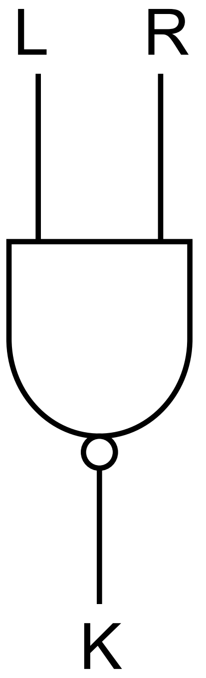


$gg \leftarrow Gb(g, L_0, L_1, p, R_0, R_1, q, Z_0, Z_1, r)$

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

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

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

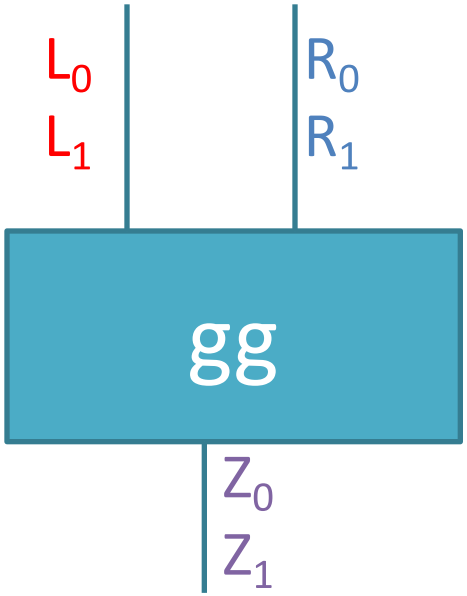
C
$C_1 = H(L_p,R_q) \oplus (K_{g(p,q)},\ r \oplus g(p,q))$
$C_2 = H(L_p,R_{!q}) \oplus (K_{g(p,!q)},\ r \oplus g(p,!q))$
$C_3 = H(L_{!p},R_q) \oplus (K_{g(!p,q)},\ r \oplus g(!p,q))$
$C_4 = H(L_{!p},R_{!q}) \oplus (K_{g(!p,!q)},\ r \oplus g(!p,!q))$

- 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

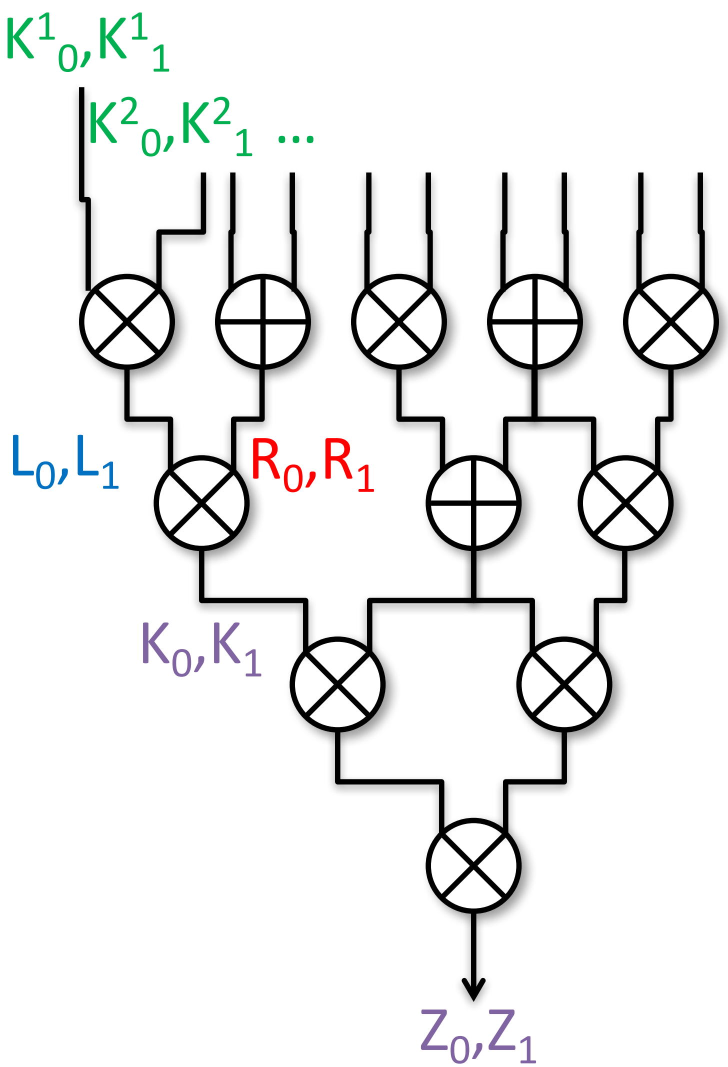
- Problem: each gg is 4 ciphertexts
- Solution: define output key pseudorandomly as functions of input keys, reduce comm. complexity



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

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

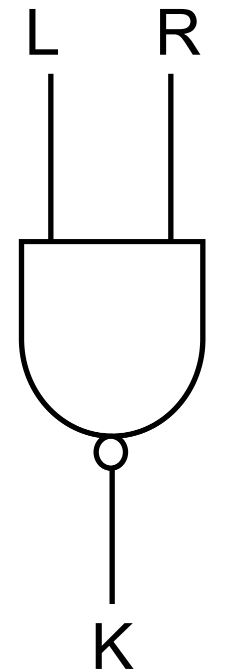
Garbling a Circuit : ([F],e,d)← Gb(f)



- 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)

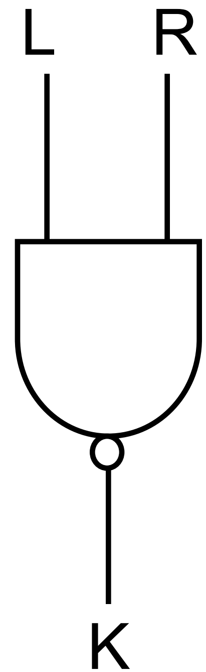
C
$C_1 = H(L_0,R_0) \oplus K_1$
$C_2 = H(L_0,R_1) \oplus K_1$
$C_3 = H(L_1,R_0) \oplus K_1$
$C_4 = H(L_1,R_1) \oplus K_0$



- Encrypt the output key with the input keys

Garbled Row Reduction Garbling

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

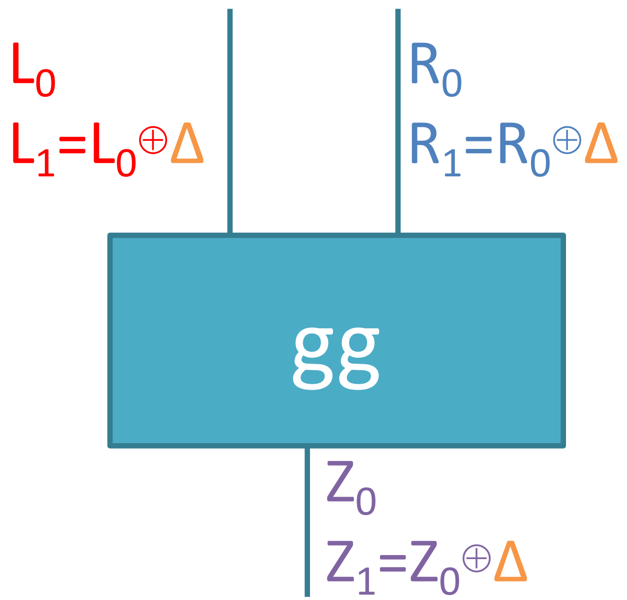


- Define output keys as function of input keys
  - (compatible with p&p)
  - Can reduce 2 rows, but 1 is compatible with Free-XOR (coming up!)

GARBLING OPTIMIZATIONS:
FREE XOR

Free-XOR

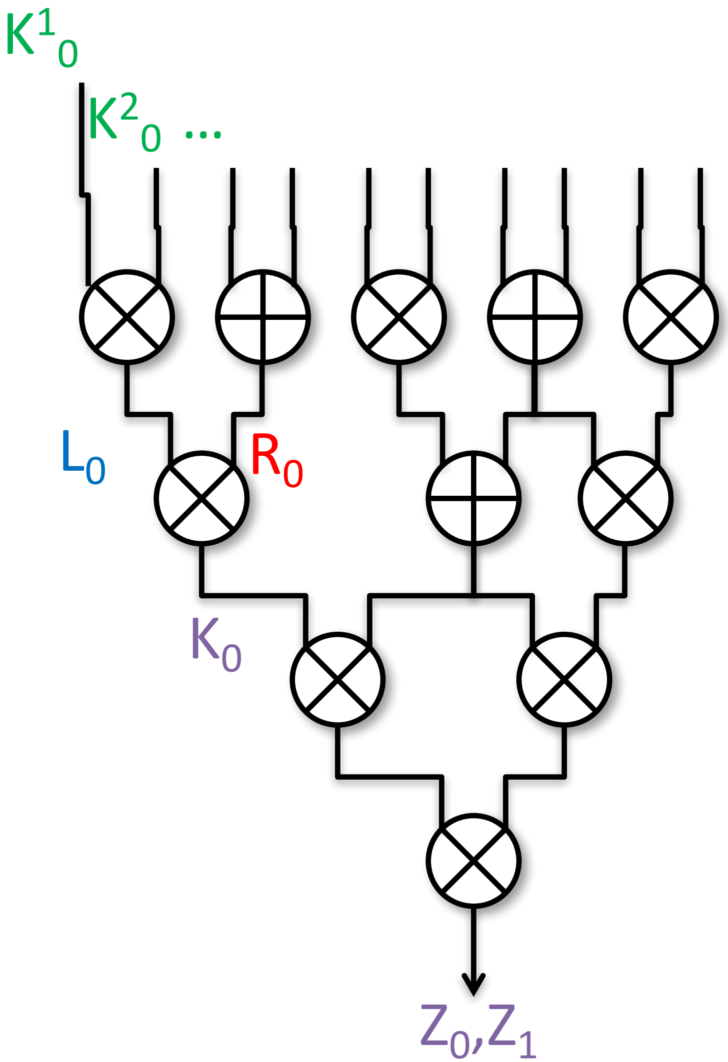
- Problem: in BeDOZa linear gates are for free. What about GC?
- Solution: introduce correlation between keys, make XOR computation "free"



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

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

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

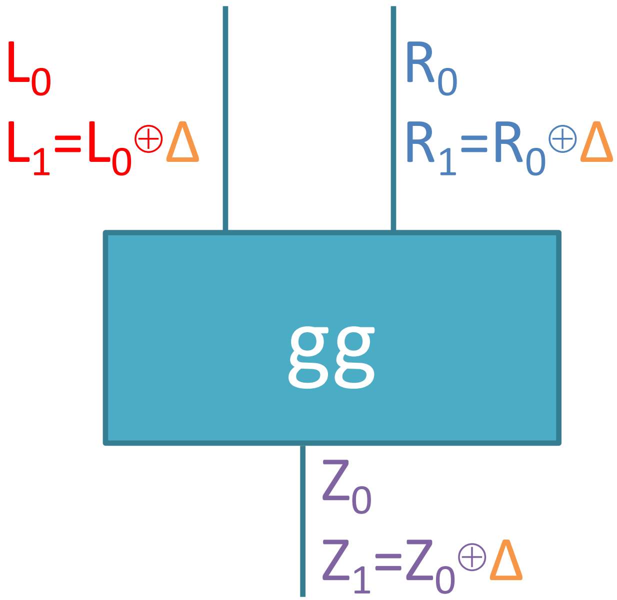
- Like before, but requires "circular security assumption"
  - (Compatible with GRR and p&P)
- Example for AND gate
  - Evaluator sees

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

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

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

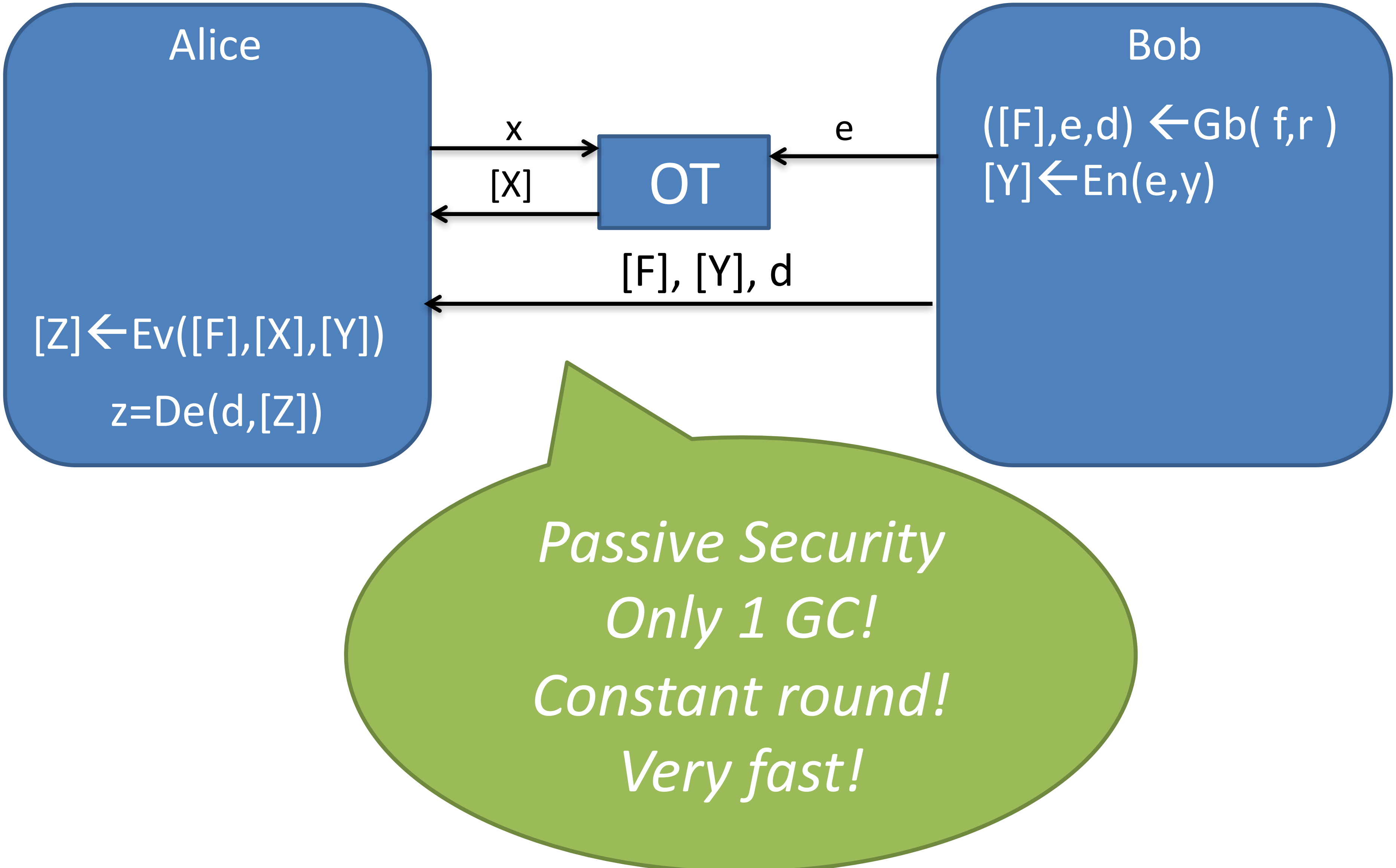
$$L_a\oplus R_b = L_0 \oplus a\Delta \oplus R_0 \oplus b\Delta = Z_0 \oplus (a\oplus b)\Delta = Z_{a\oplus b}$$

Part 3: Garbled Circuits

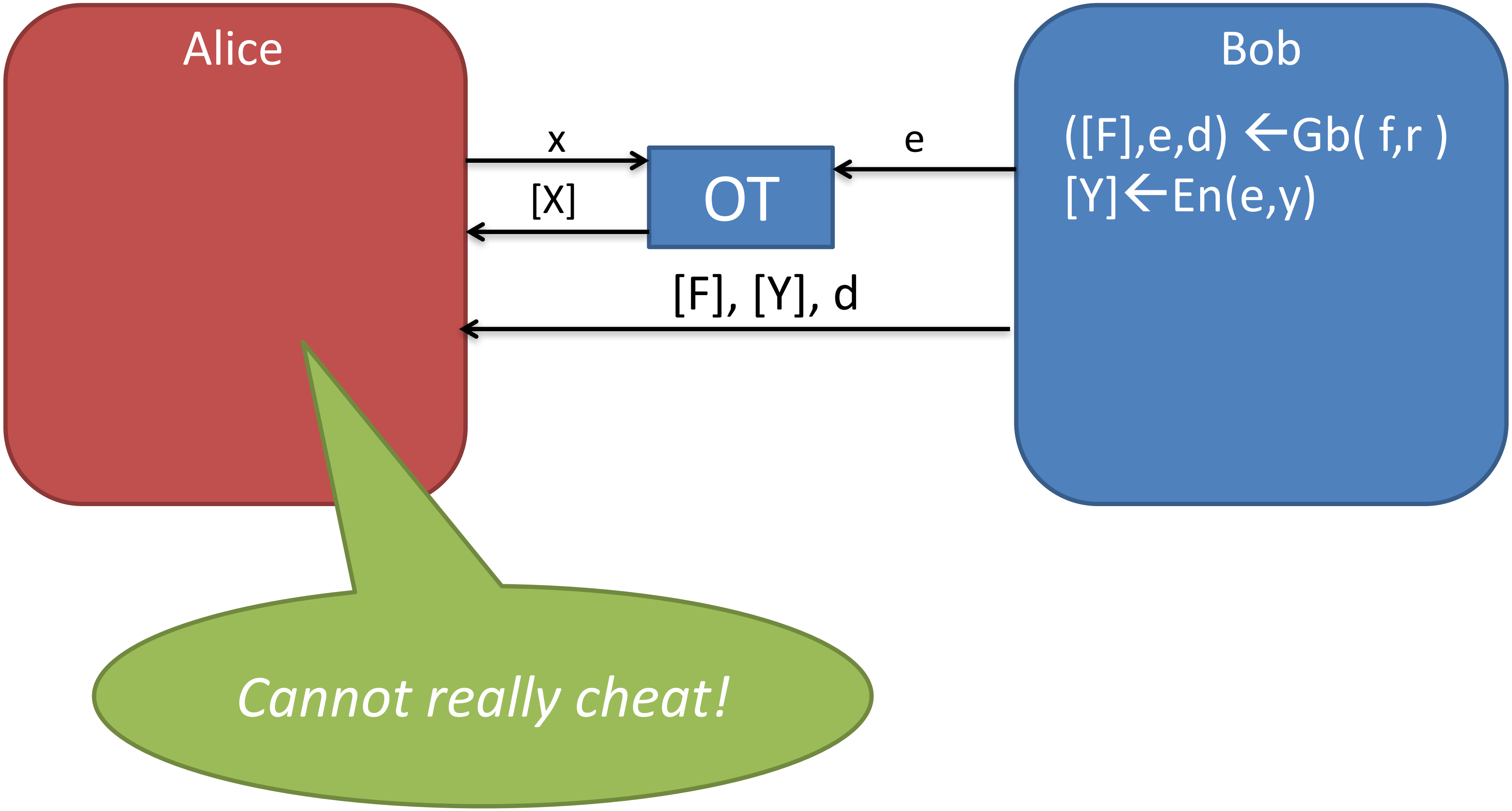
- Definitions and Applications
- Garbling gate-by-gate: Basic and optimizations
- **Active security 101: simple-cut-and choose, dual-execution**

ACTIVE ATTACKS VS YAO

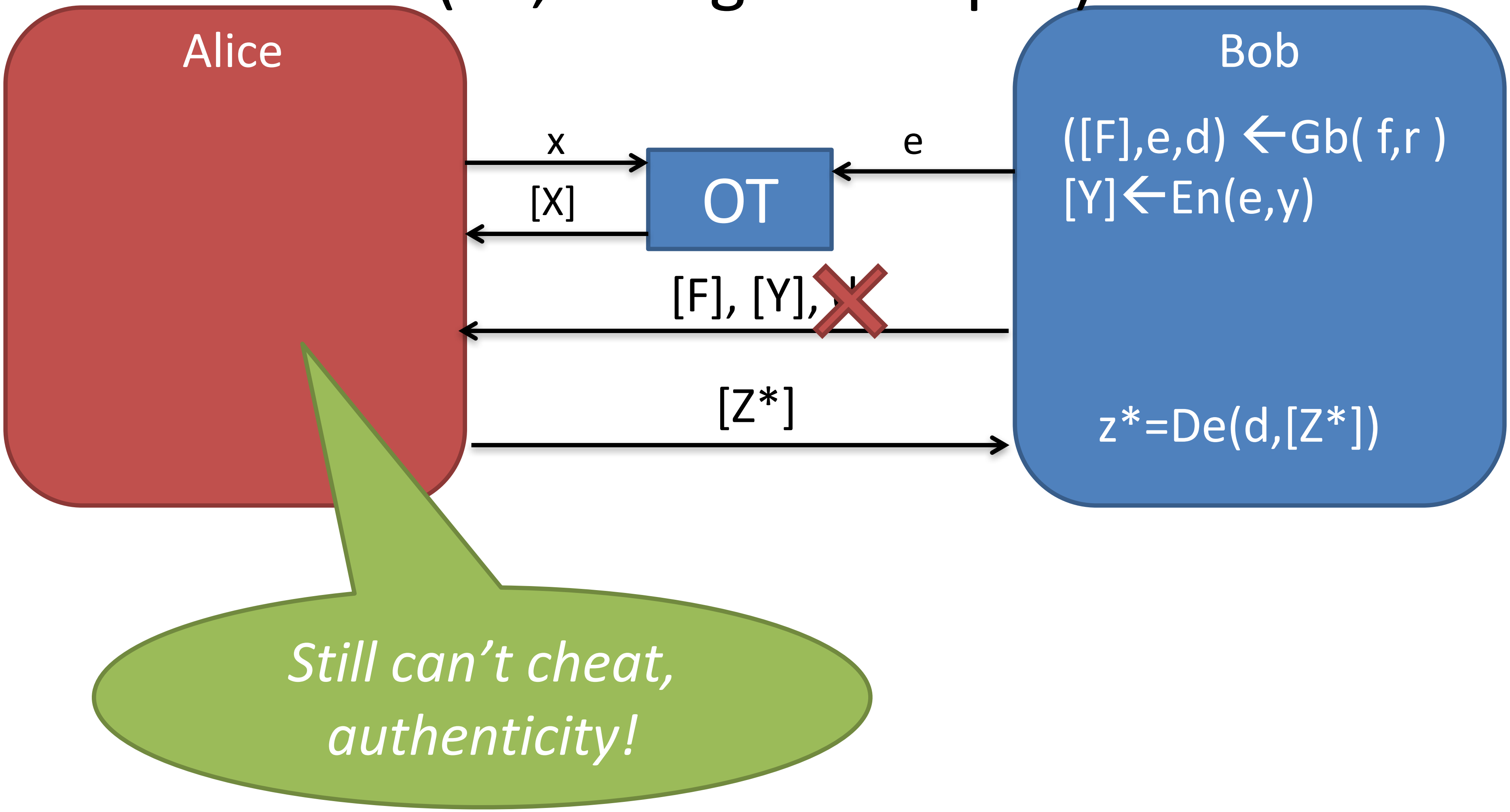
Yao´s protocol



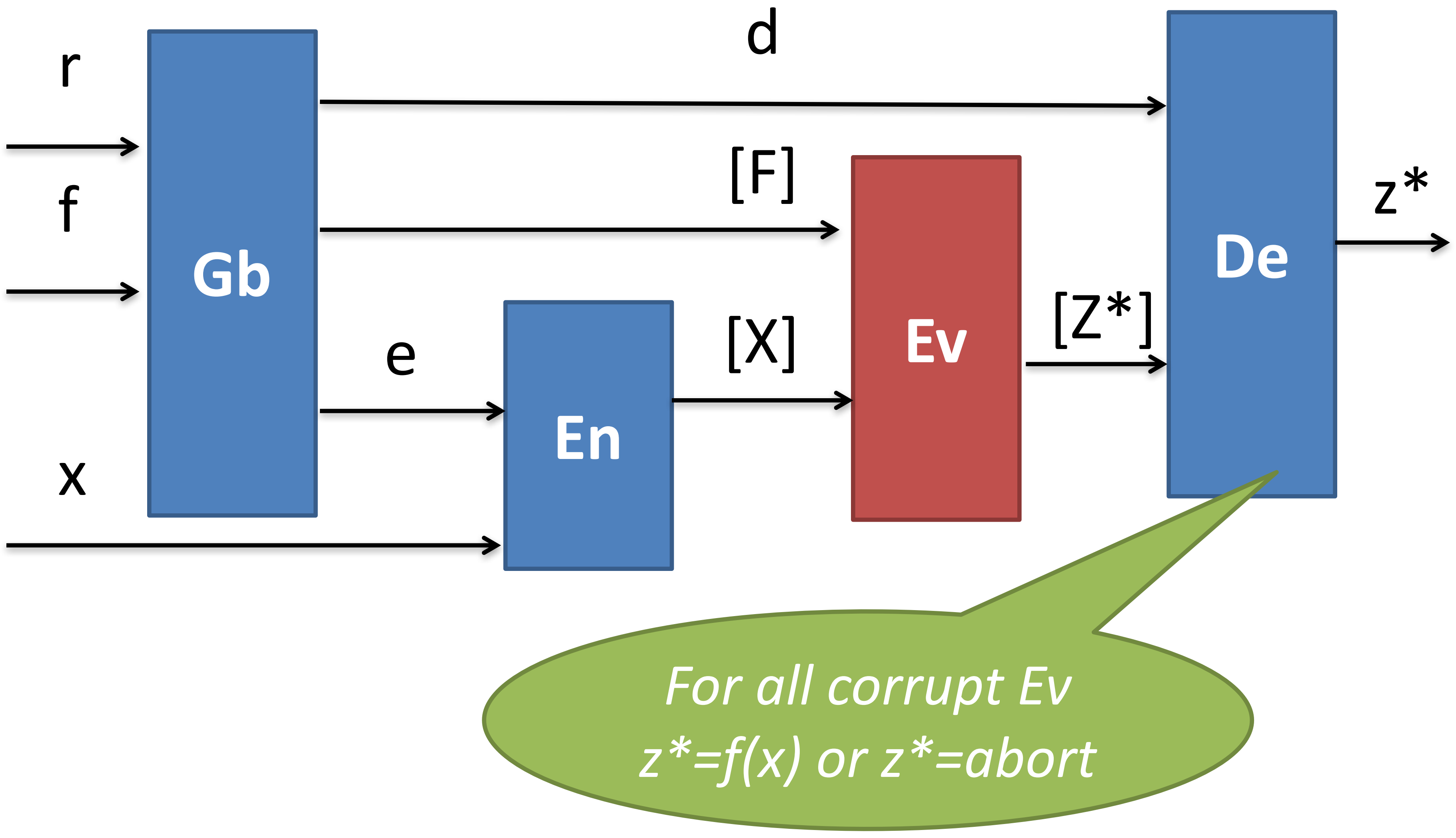
Active security of Yao



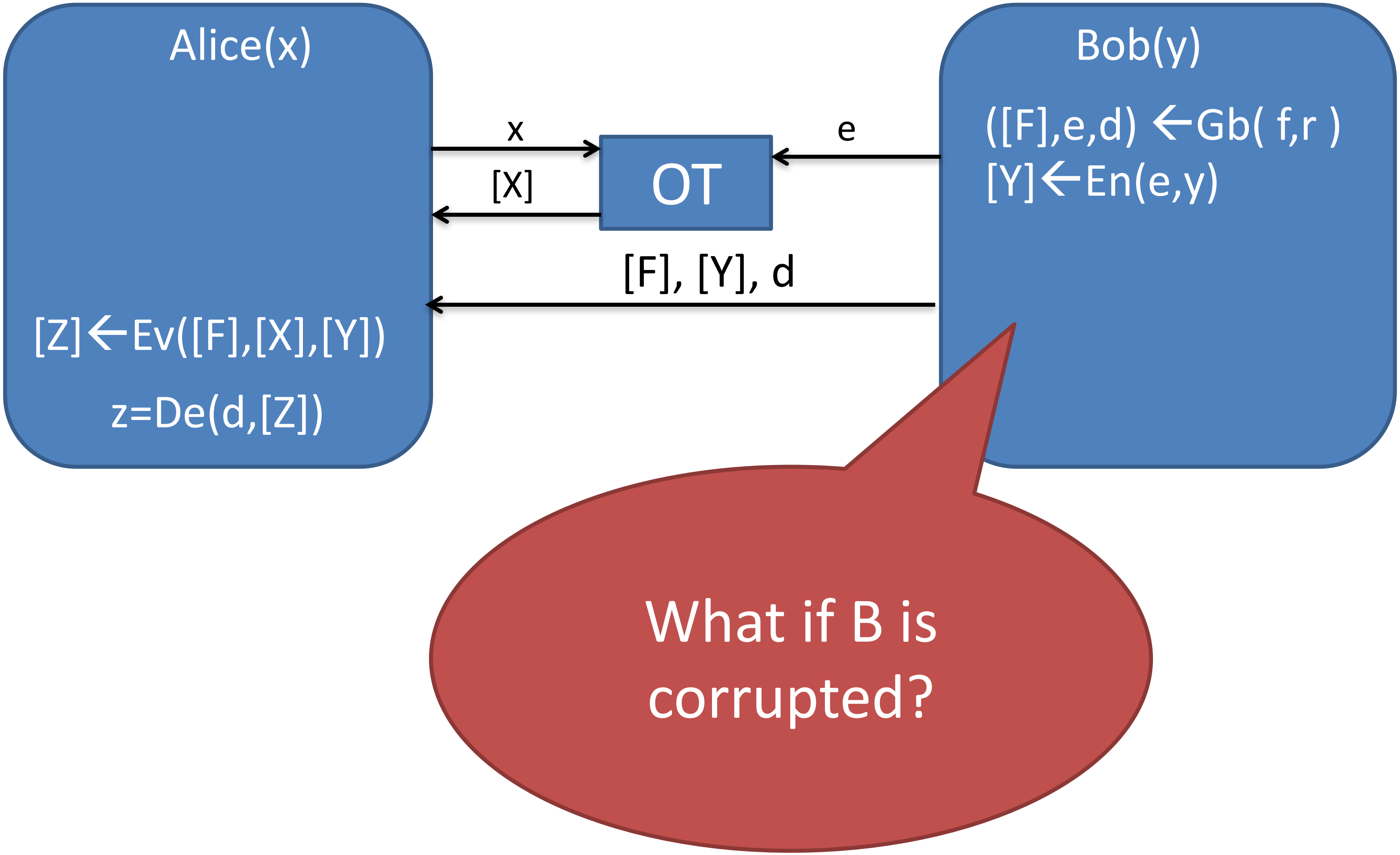
Active security of Yao
(v2, Bob gets output)



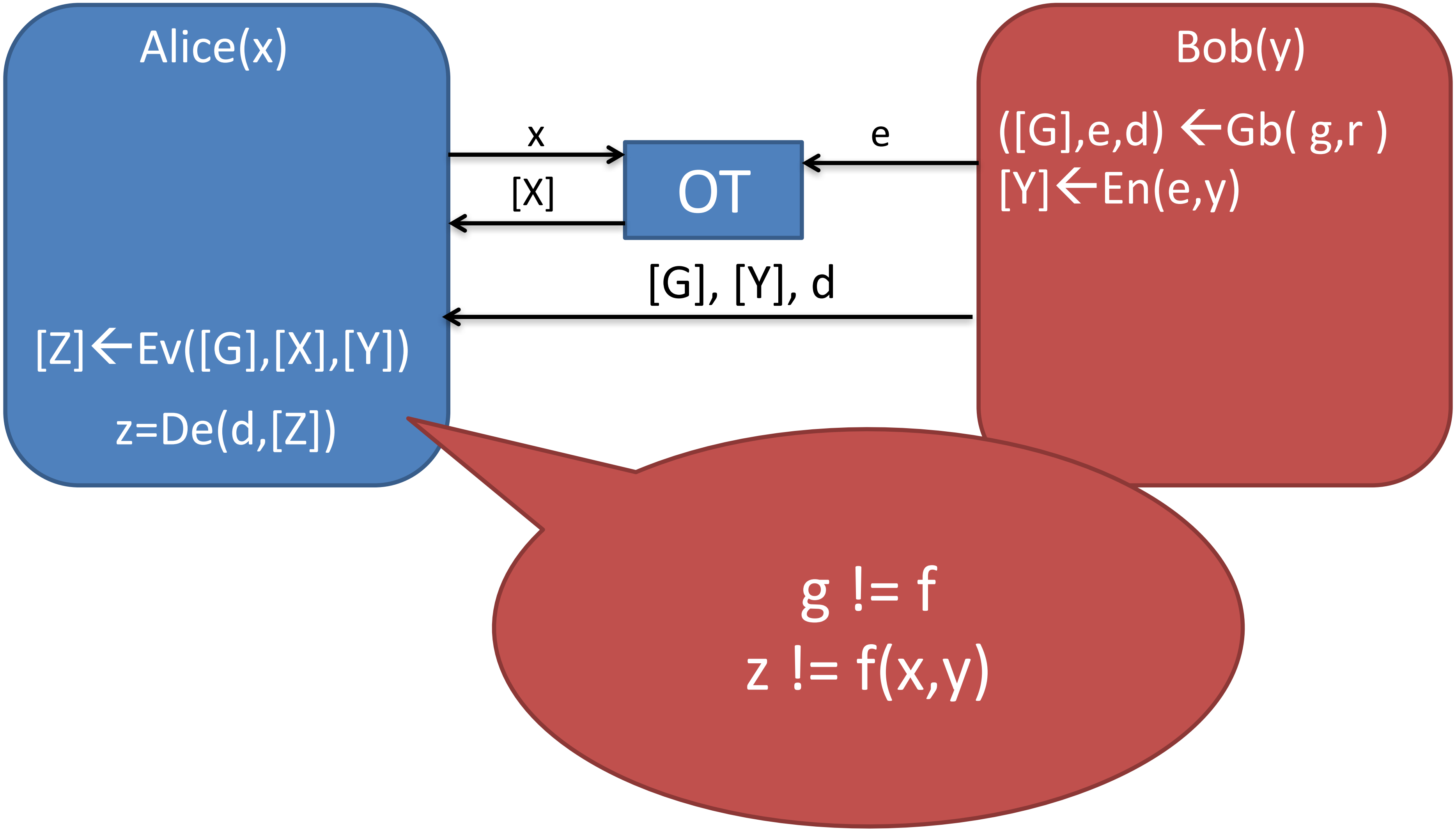
Garbled Circuits: Authenticity



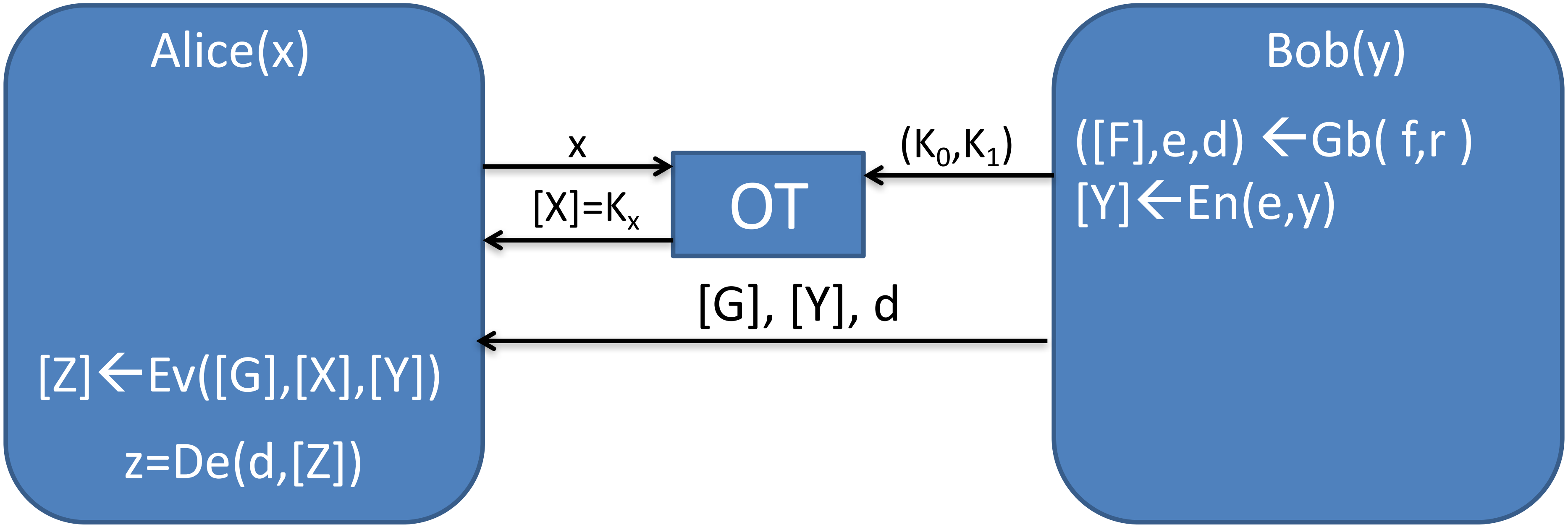
Active security of Yao



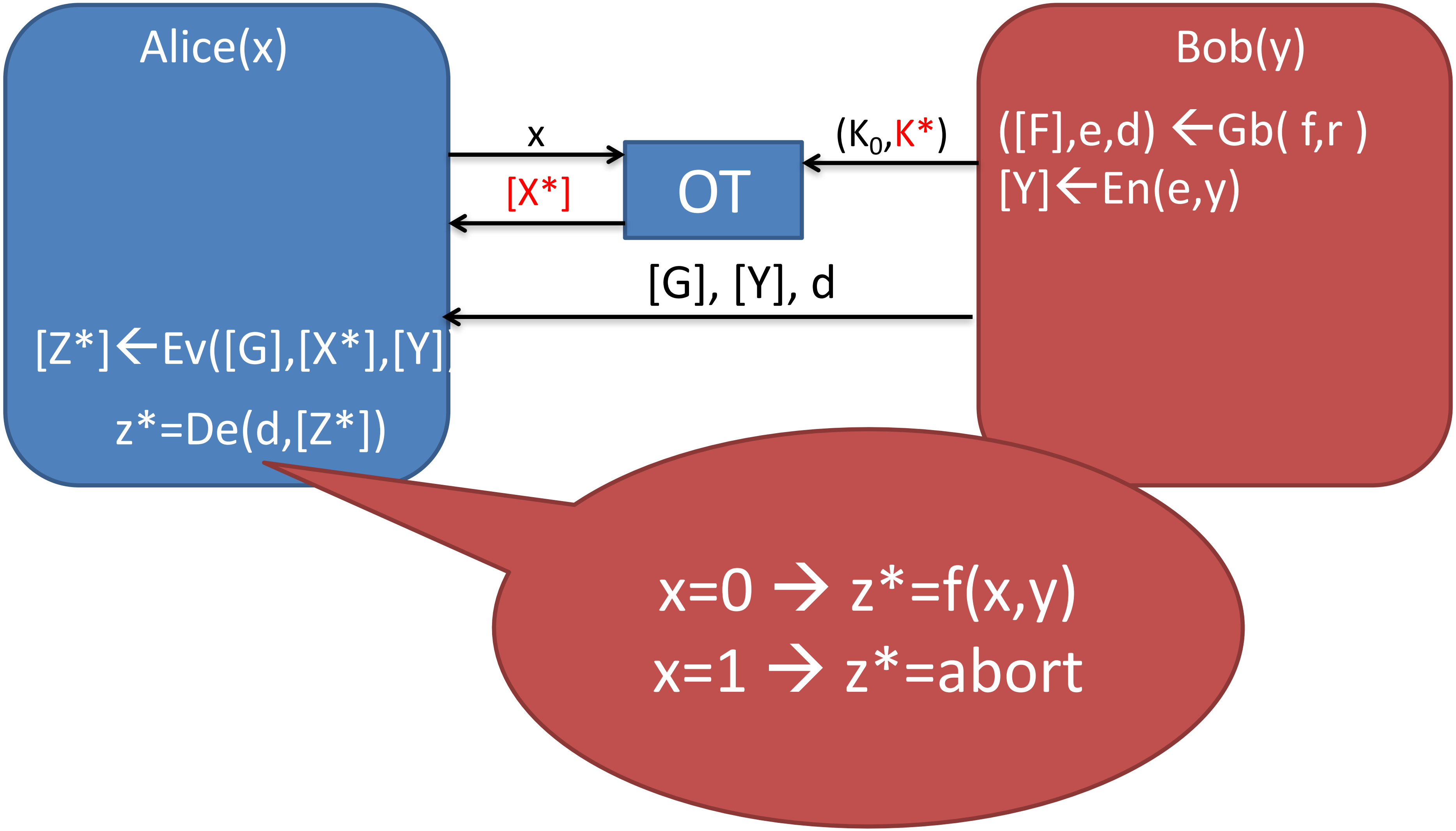
Insecurity 1 (wrong f)



Insecurity 2 (selective failure)

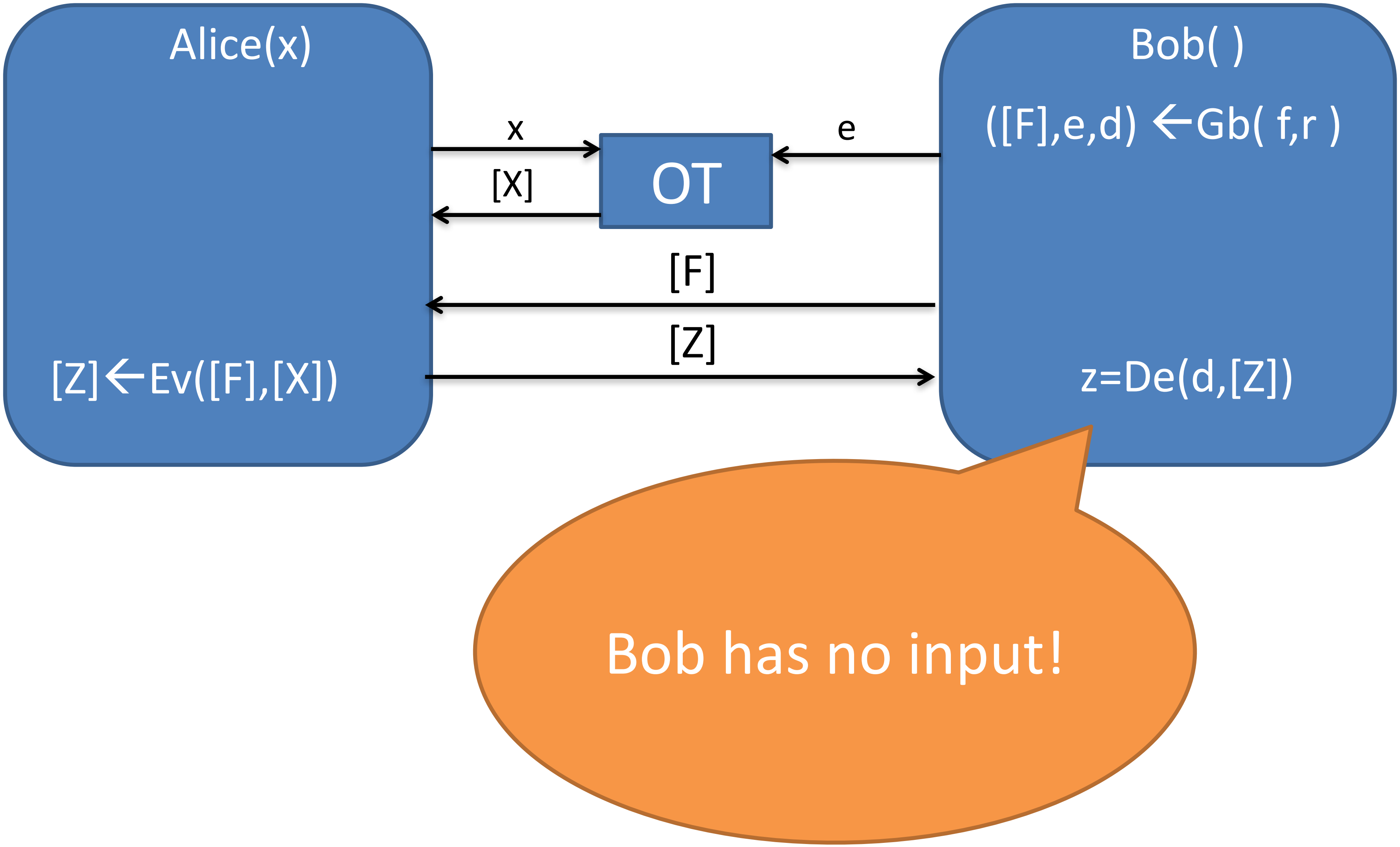


Insecurity 2 (selective failure)

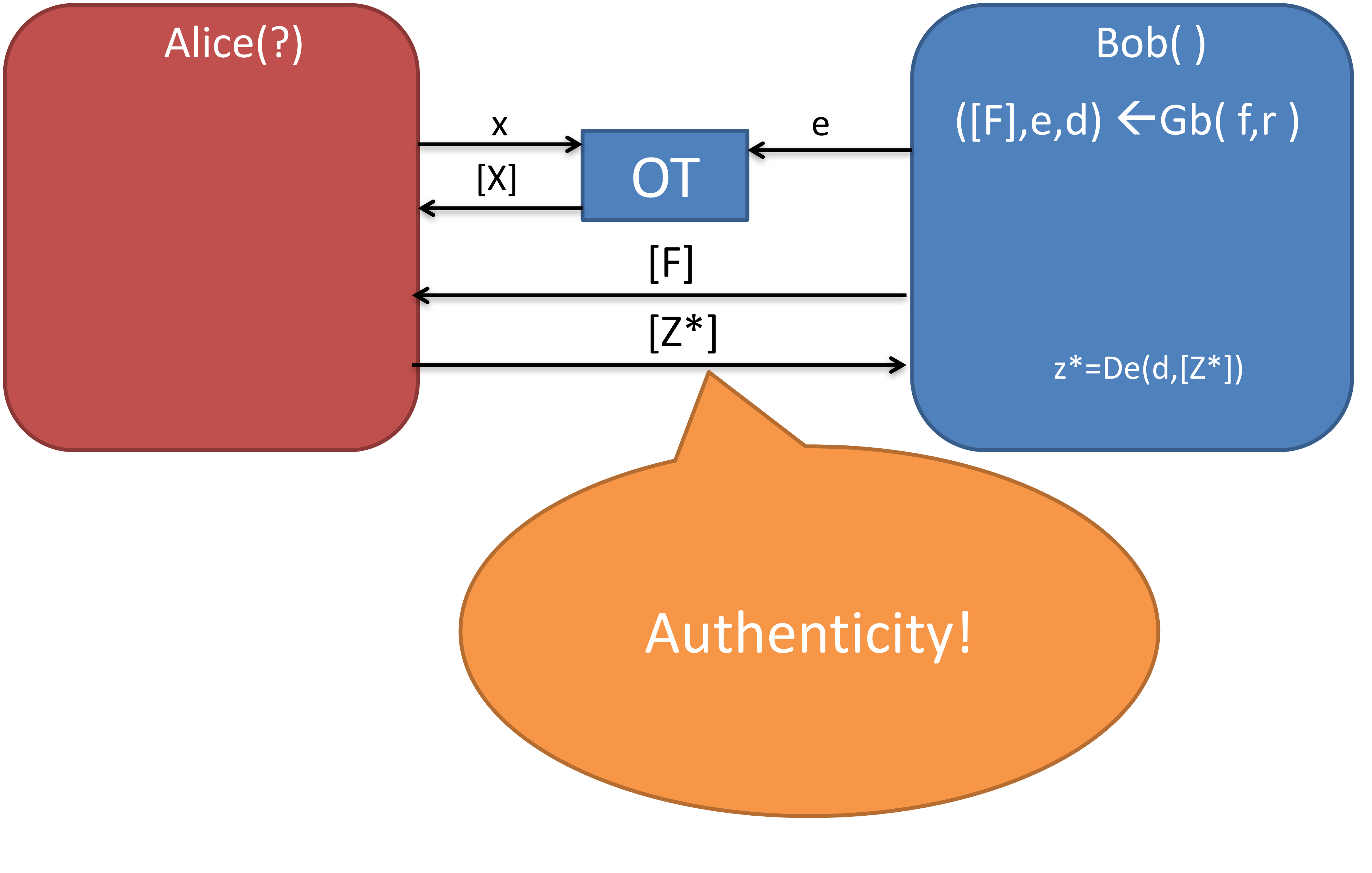


SIMPLE TRICKS FOR ACTIVE SECURITY

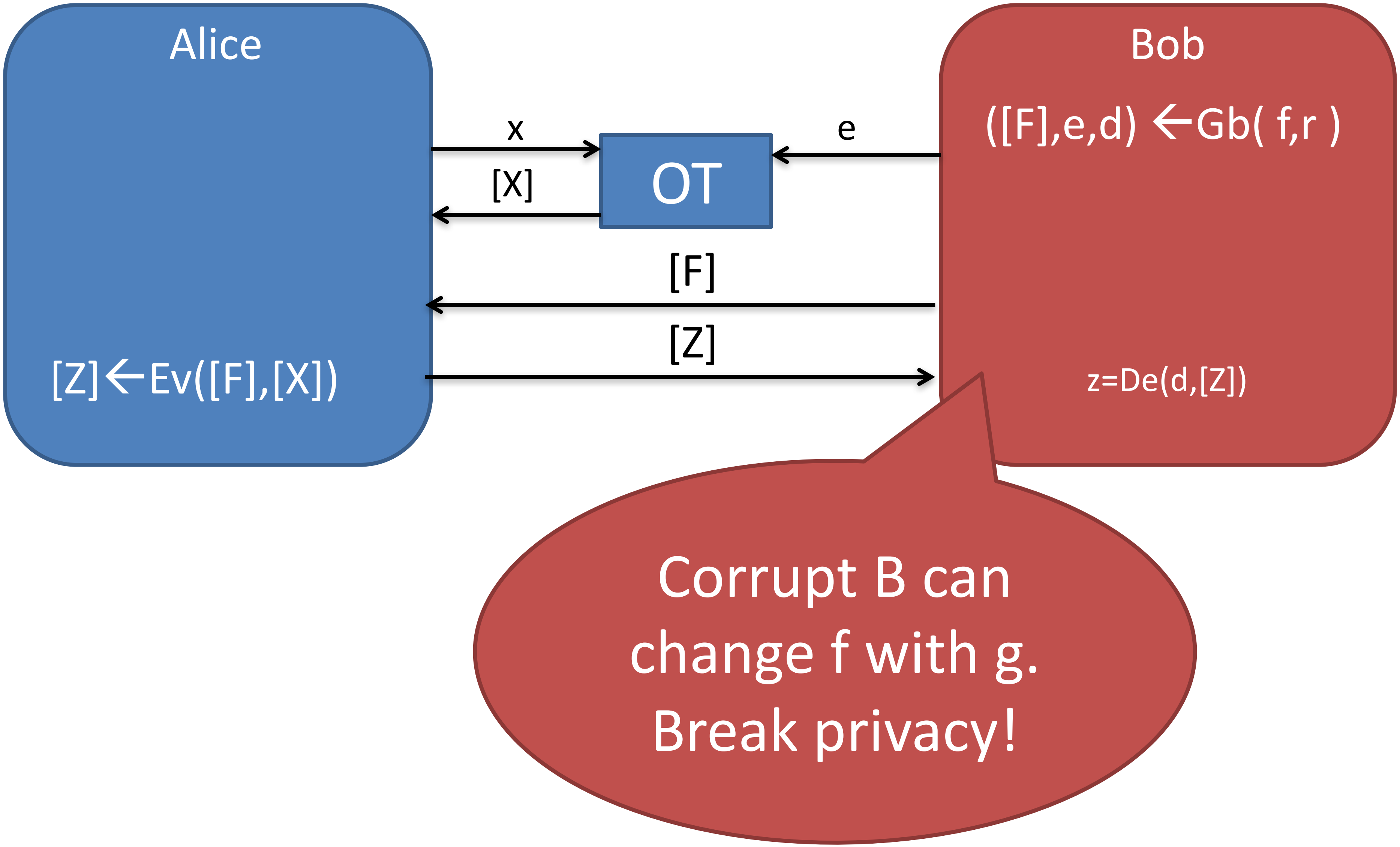
ZKGC (Alice proves f(x)=z)



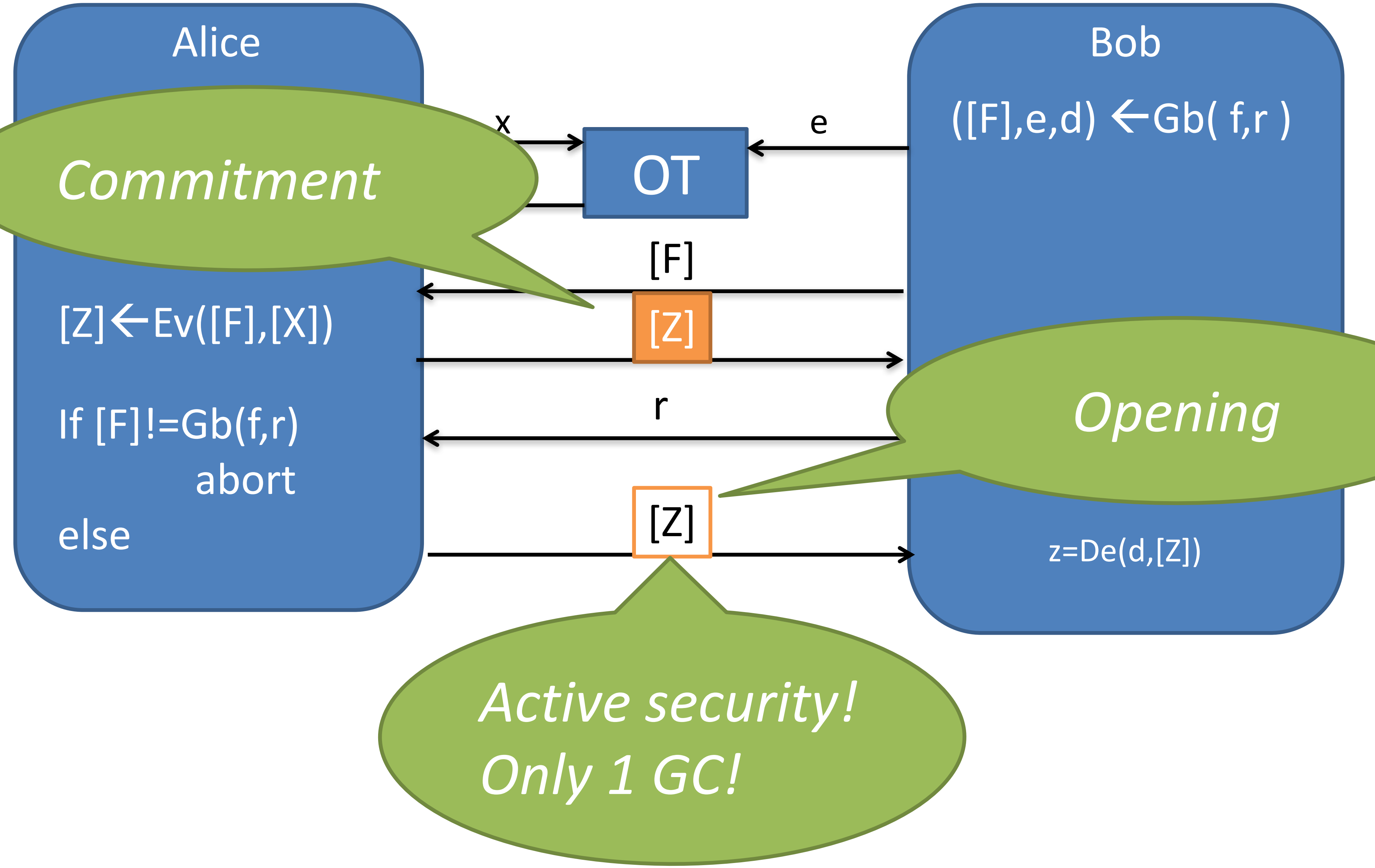
ZKGC (Alice proves f(x)=z)



ZKGC (Alice proves f(x)=z)

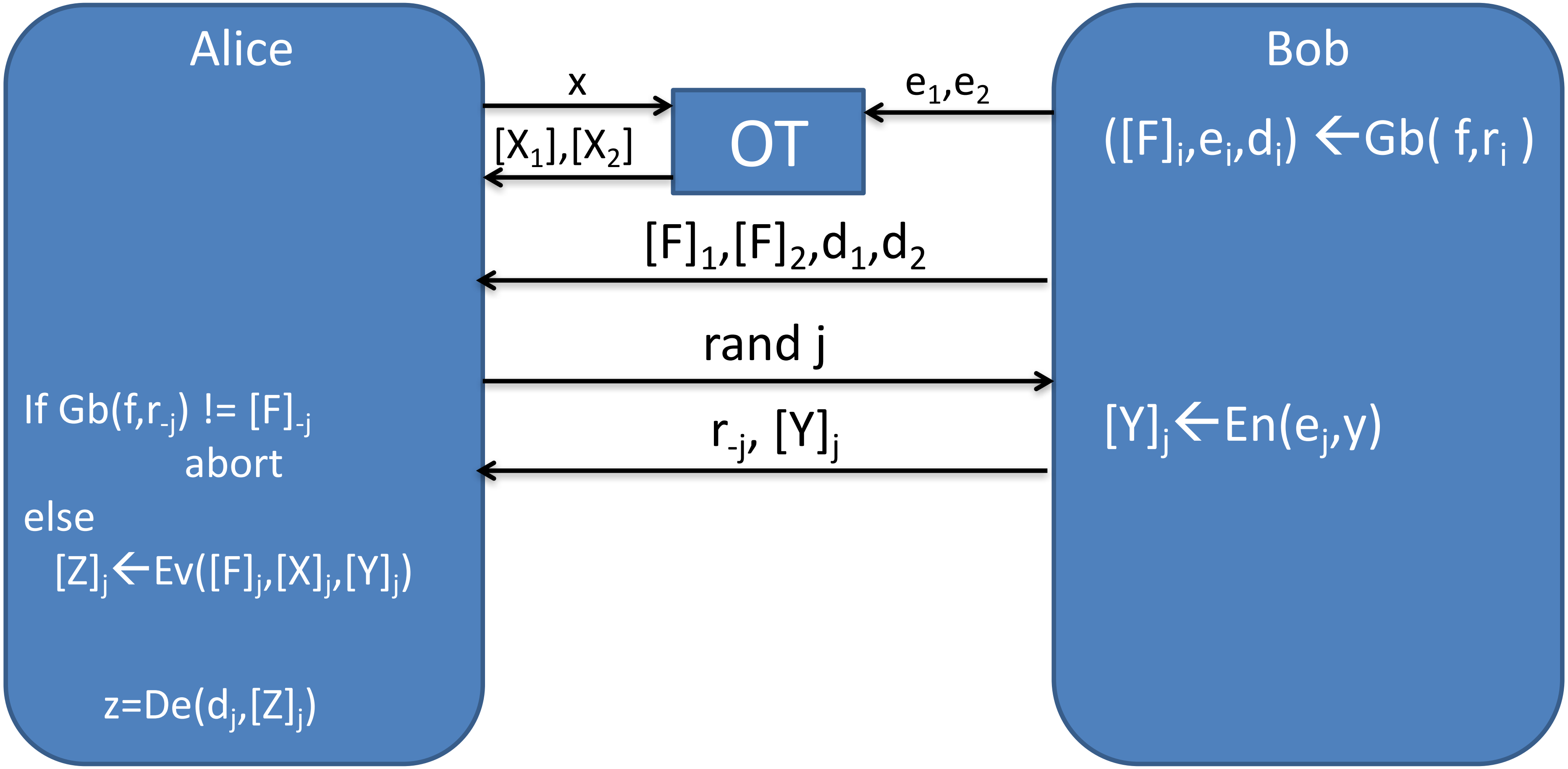


ZKGC (Alice proves f(x)=z)

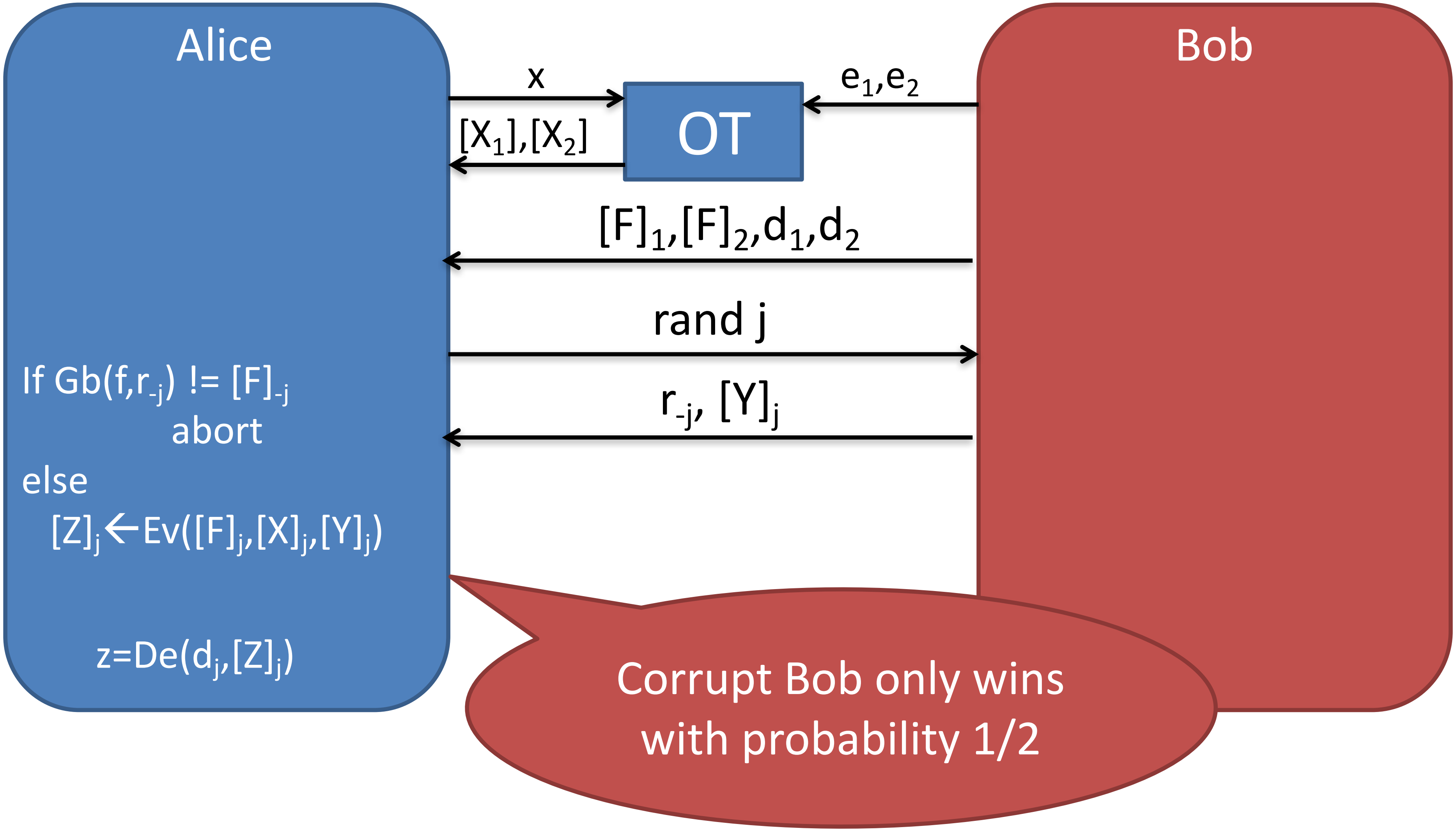


Cut-And-Choose

2PC, simple cut-and-choose



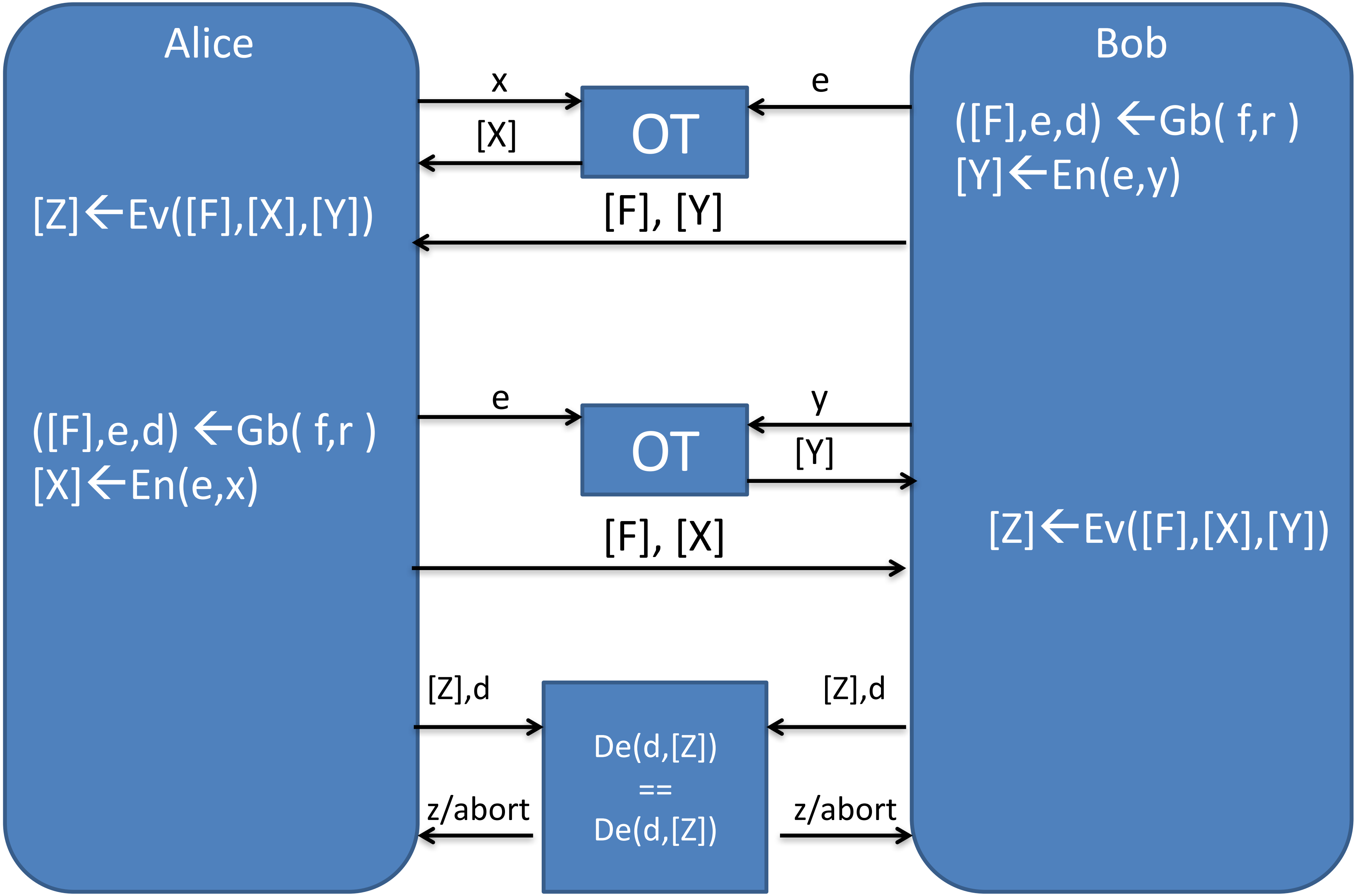
2PC, simple cut-and-choose

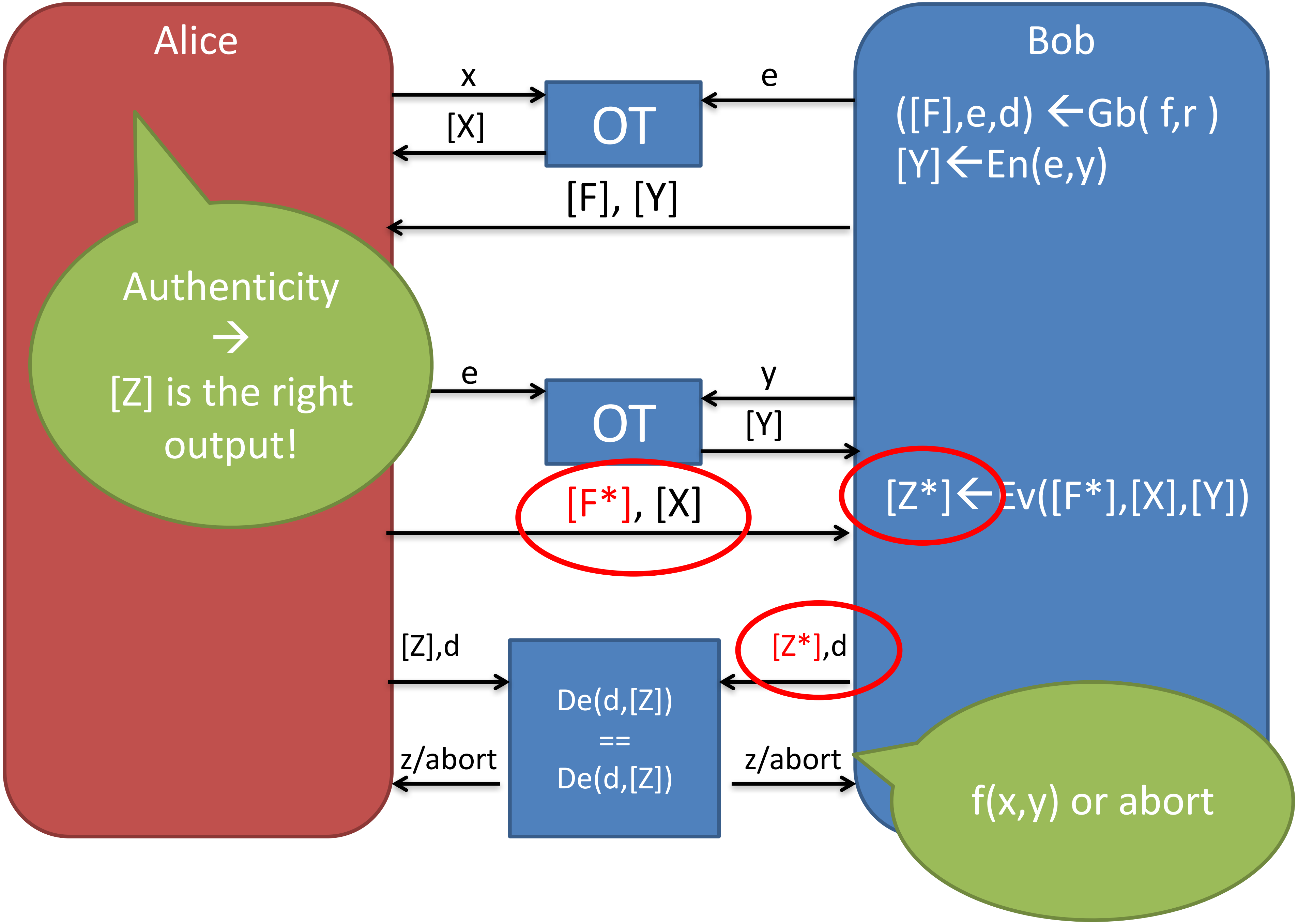


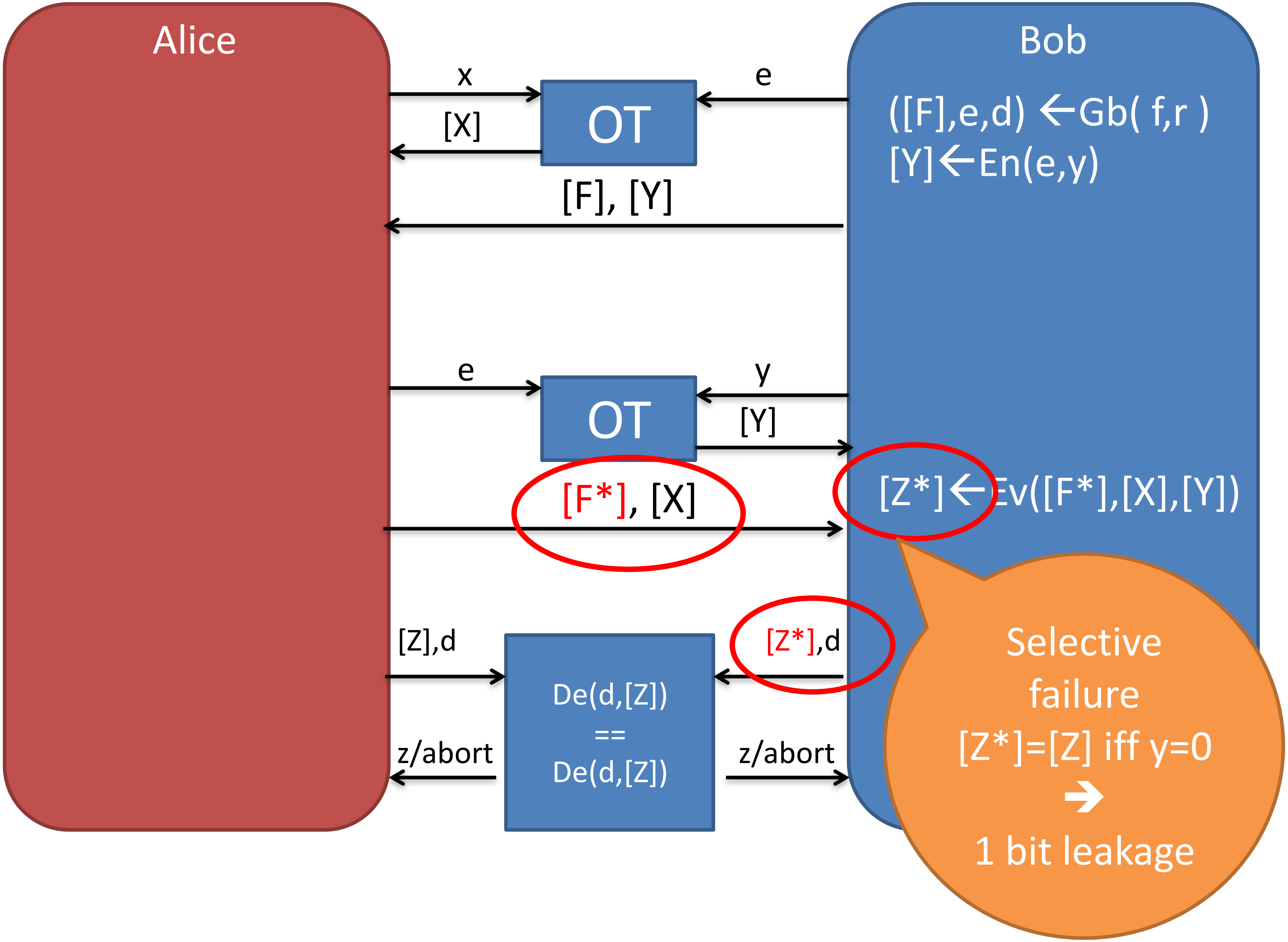
2PC, cut-and-choose

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

Dual Execution

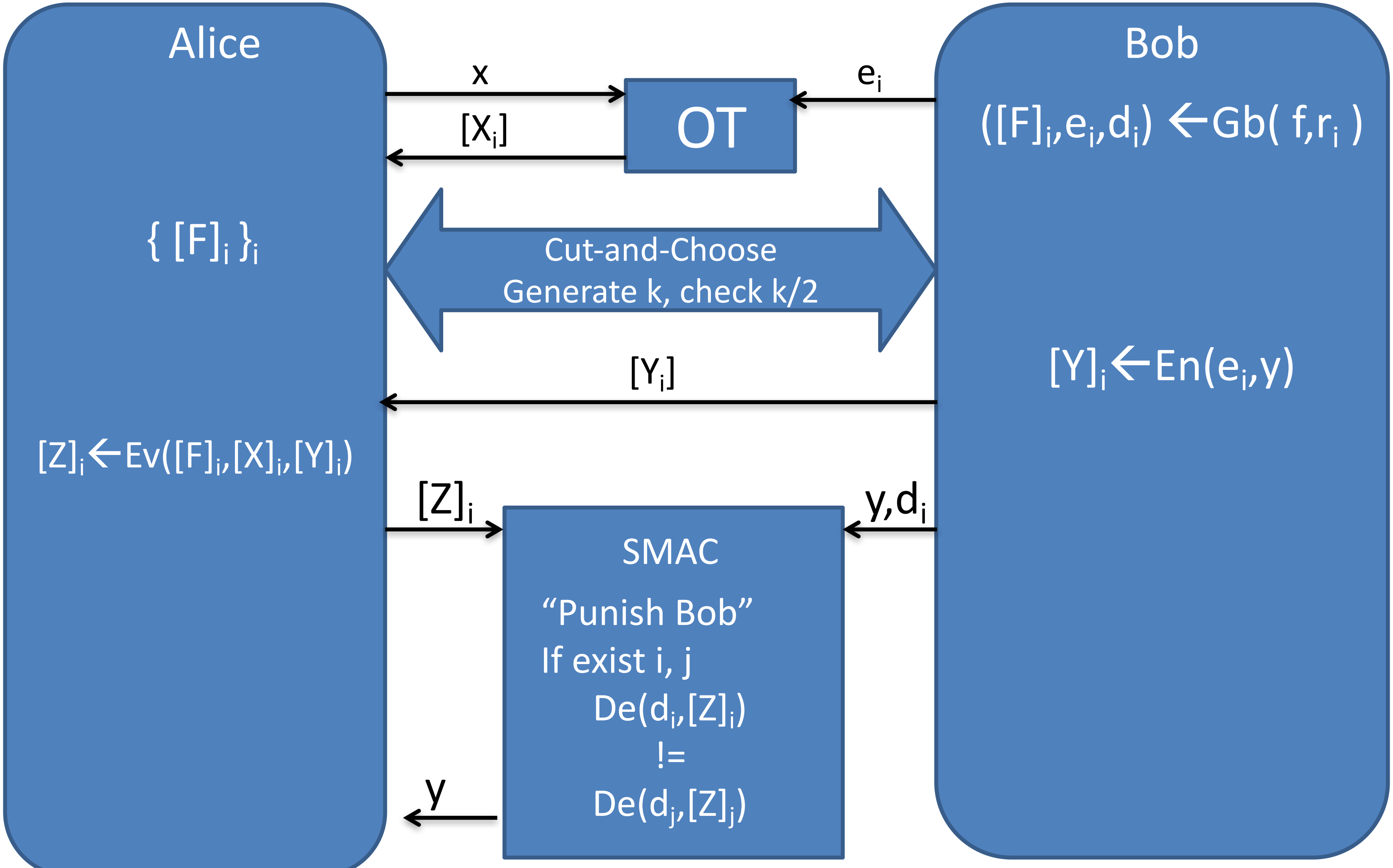

Alice
Bob
x
e
OT
[X]
([F],e,d) ←Gb(f,r)
[Y]←En(e,y)
[Z]←Ev([F],[X],[Y])
[F], [Y]
e
y
OT
[Y]
([F],e,d) ←Gb(f,r)
[X]←En(e,x)
[Z]←Ev([F],[X],[Y])
[F], [X]
[Z],d
[Z],d
De(d,[Z])
==
De(d,[Z])
z/abort
z/abort


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

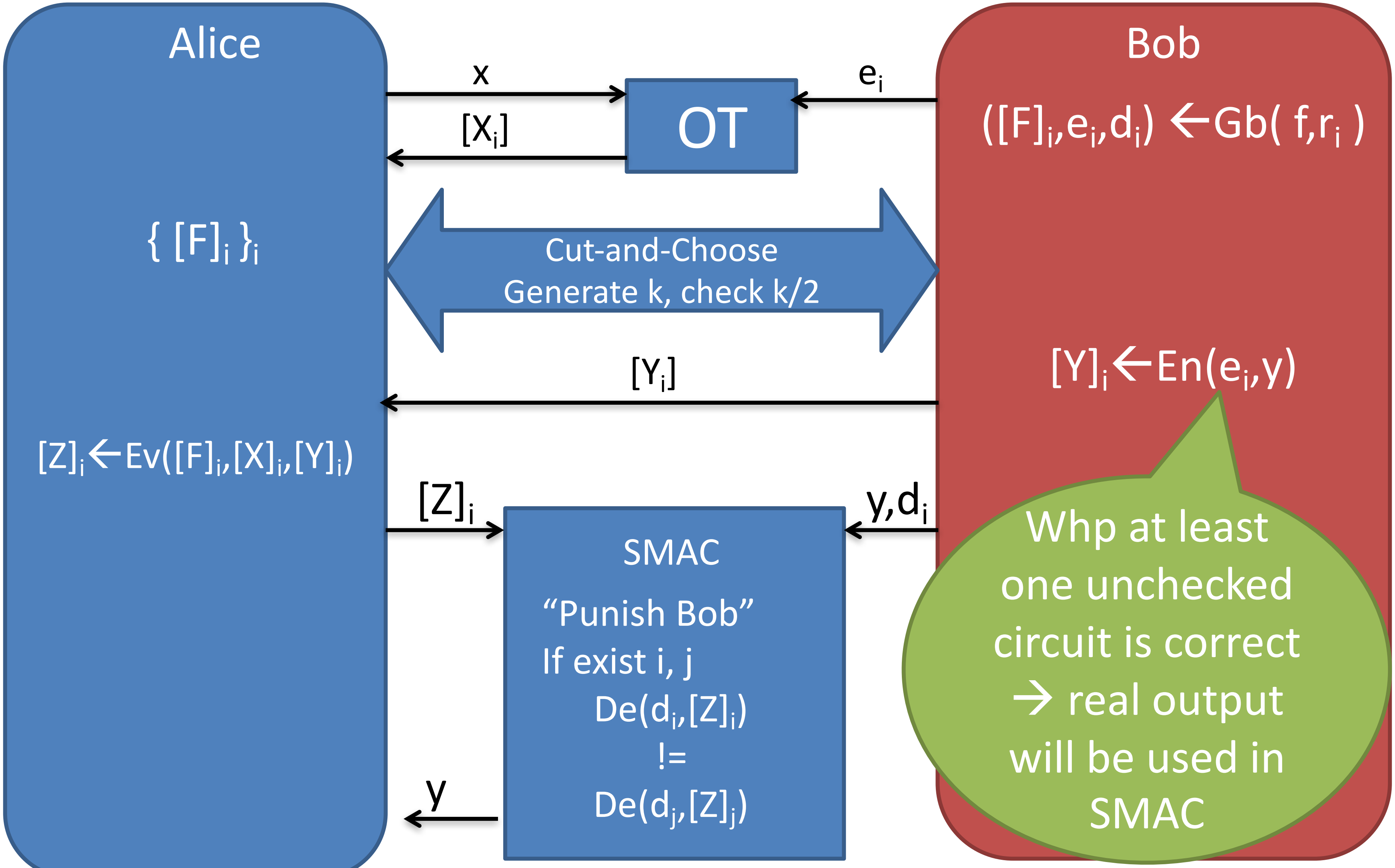

Alice
Bob
x
[X]
OT
e
([F],e,d) ←Gb(f,r)
[Y]←En(e,y)
[F], [Y]
e
OT
y
[Y]
[F*], [X]
[Z*]←Ev([F*],[X],[Y])
[Z],d
De(d,[Z])
==
De(d,[Z])
[Z*],d
z/abort
z/abort
Selective
failure
[Z*]=[Z] iff y=0
→
1 bit leakage

Forge And Lose

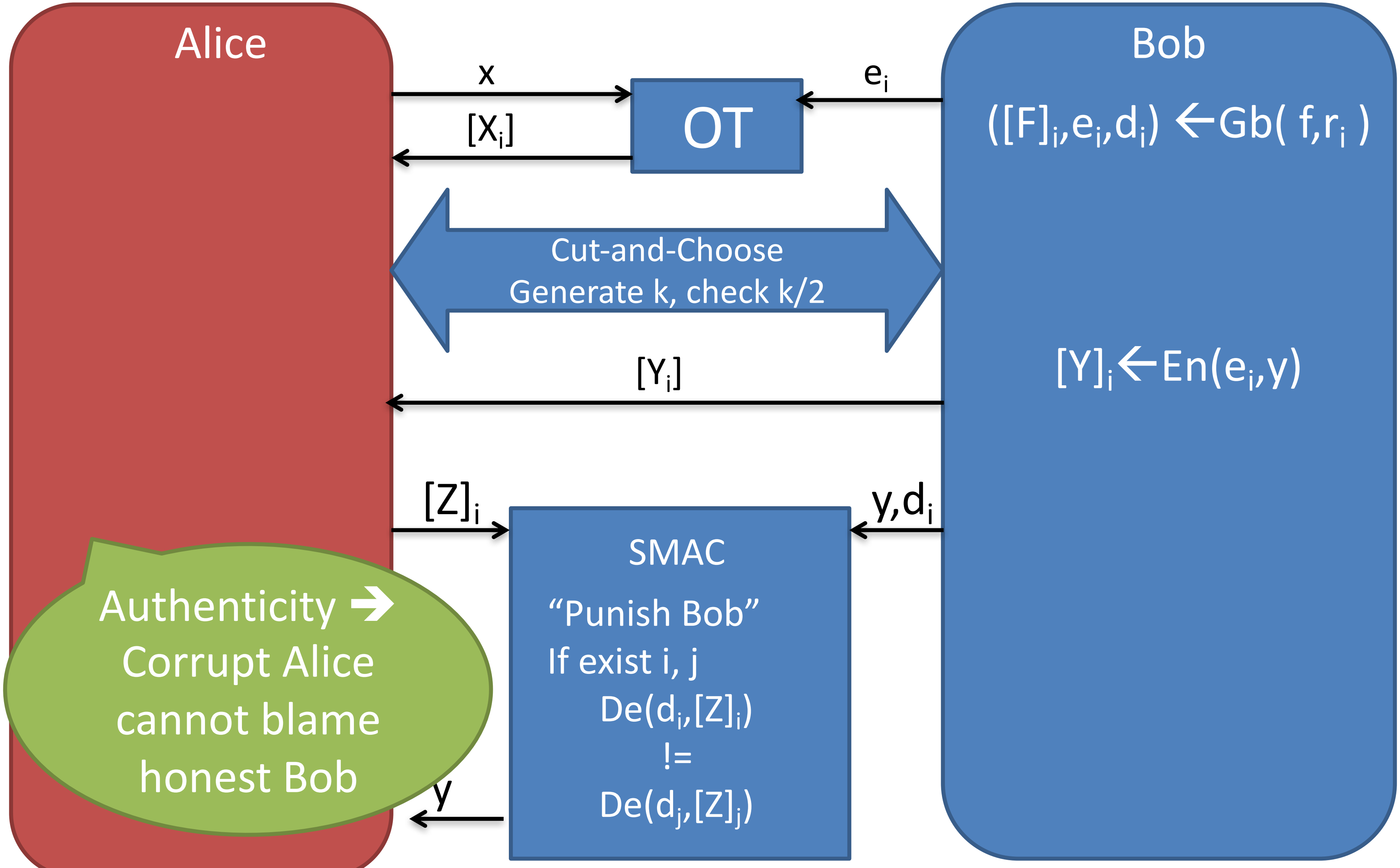
2PC, forge-and-lose (idea)



2PC, forge-and-lose (idea)



2PC, forge-and-lose (idea)



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!

…also the reason why I cannot stay here longer ☹

- **These slides** (+ references & pointers)
  - http://orlandi.dk/ecrypt