5th Bar-Ilan Winter School on Cryptography
Advances in Practical Multiparty Computation

"Tiny OT" – Part 3

A ~~New~~ (4 years old) Approach to Practical Active-Secure Two-Party Computation

Claudio Orlandi, Aarhus University

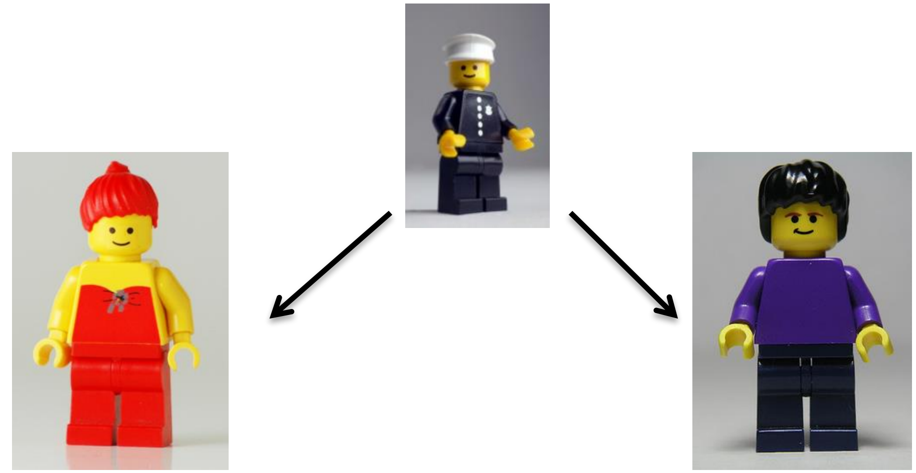
TinyOT authenticated bits

- $[x] = ((x_A, k_A, m_A) , (x_B, k_B, m_B))$ s.t.
  - $m_B = k_A + x_B \Delta_A$ (symmetric for m_A)
  - Δ_A, Δ_B is the same for all wires.
  - MACs, keys are k-bit strings.

(Maybe adversary knows a few bits of Δ)

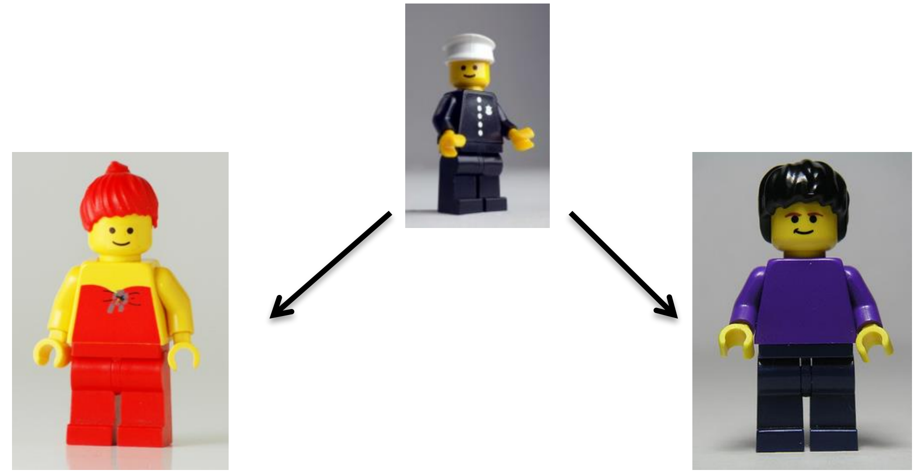
- Similarity with Oblivious Transfer
  - Sender has two messages $\boldsymbol{u_0, u_1}$
  - Receiver has a bit $\boldsymbol{b}$ and learns $\boldsymbol{u_b}$
  - Set $\boldsymbol{u_0 = k}$, $\boldsymbol{u_1 = k + \Delta}$, $\boldsymbol{b = x}$ then $\boldsymbol{u_b = k + x\Delta}$

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

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 TD.
   – e ← Open([a]+[x]), d ← Open([b]+[y])
   – Compute [z] = [c] + a[y] + b[x] - ed

Circuit Evaluation (Online phase)

3) [z]←Mul([x],[y]):

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

– *e=Open([a]+[x])*

– *d=Open([b]+[y])*

– Compute [z] = [c] + e[y] + d[x] - ed

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

Coming up…

- Given **authenticated bits,** produce ***authenticated multiplication triples***!

The problem

- **Input**: (random) [x], [y], [r], [s], …
- **Output**: [z] s.t. [z=xy]

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

How to authenticate local product?

How to authenticate cross product?

- **Remember**
  - [x] = ((x_A, k_A, m_A) , (x_B, k_B, m_B)) s.t.
  - $m_B = k_A + x_B \Delta_A$ (symmetric for m_A)
  - Δ_A, Δ_B is the same for all wires.
  - MACs, keys are k-bit strings.

Part 3:
From “Auth. Bits” to “Auth. Triples”

- **Authenticated local-products (*aAND*)**
- Authenticated cross-products (*aOT*)
- “LEGO” bucketing

Authenticate local products

- **Input:** [x], [y], [r]**; Alice private input:** x,y
- **Output:** [z] s.t. z=xy
- **First Attempt**: (like **Input**)
  - $r \leftarrow Open(A,[r])$
  - Alice sends Bob $d = r + xy + e$
  - $[z]=[xy]+r + e$

- **Corrupted Alice, what if e ≠ 0 ?**

Authenticate local products

- Δ is the same for all wires.
- $[x] = ((x,\ldots,m_x) , (\ldots,k_x,\ldots))$ s.t. $m_x = k_x + x\,\Delta$
- $[y] = ((y,\ldots,m_y) , (\ldots,k_y,\ldots))$ s.t. $m_y = k_y + y\,\Delta$
- $[z] = ((z,\ldots,m_z) , (\ldots,k_z,\ldots))$ s.t. $m_z = k_z + z\,\Delta$

- When $x = 0$

  $(m_x = k_x , m_z = k_z)$ iff $z = 0$
- When $x = 1$

  $(m_x = k_x + \Delta , m_z + m_y = k_z + k_y)$ iff $z = y$

Authenticate local products

- **Bob knows**

  $U_0 = (k_x , k_z)$ and

  $U_1 = (k_x + \Delta, k_z + k_y)$

- **Alice knows**

  U_x if $xy = z$

  neither if $xy \neq z$

- How can Alice prove she knows U_x without revealing x?

Proof of 1-out-of-2 strings

U_x

$B=H(U_0)+H(U_1)$ ←

U_0,U_1

if(x=0) $A = H(U_x)$
else $A = C+H(U_x)$

A →

$A = H(U_0)$

Proof of 1-out-of-2 strings

U_x

$B=H(U_0)+H(U_1)+e$

U_0,U_1

if(x=0) $A = H(U_x)$

else $A = C+H(U_x)$

A

$A = H(U_0) + xe$

Proof of 1-out-of-2 strings

U_x

$B = H(U_0) + H(U_1) + e$

U_0, U_1

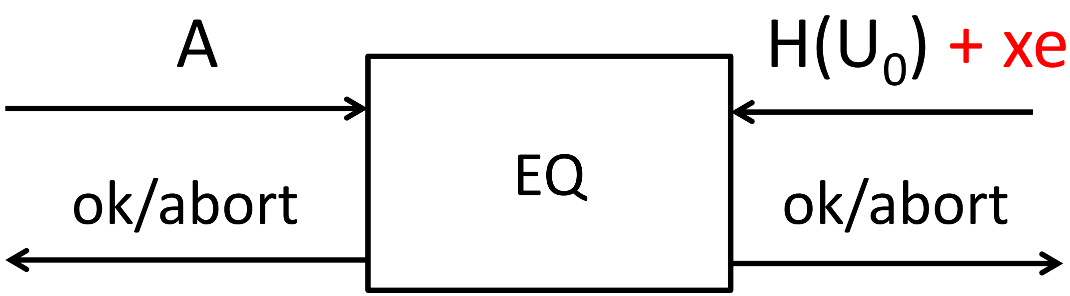
if(x=0) $A = H(U_x)$

else $A = C + H(U_x)$

If e ≠ 0

w.p. ½ abort with probability ½

w.p. ½ continue and Bob learns x



Combine local multiplications

- **Input**: (random) $[x_1]$, $[y_1]$, $[z_1]$, $[x_2]$, $[y_2]$, $[z_2]$
  // $z_i = x_i y_i$, Alice knows all
  // Bob knows: x_1 or x_2 (not both)
- **Output:** $[a]$, $[b]$, $[c]$ // Bob knows nothing

1. $[a] = [x_1] + [x_2]$ // Now a random
2. $[b] = [y_1]$
3. $d = \text{Open}([y_1]+[y_2])$
4. $[c] = [z_1] + [z_2] + d[x_2]$
   // $x_1y_1+x_2y_2+x_2y_1+x_2y_2 = (x_1+x_2)y_1 = ab$

Part 3:
From “Auth. Bits” to “Auth. Triples”

- Authenticated local-products (*aAND*)
- **Authenticated cross-products (*aOT*)**
- “LEGO” bucketing

The problem

- **Input**: (random) [x], [y], [r], [s], …
- **Output**: [z] s.t. [z=xy]

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

How to authenticate local product?

How to authenticate cross product?

- **Remember**
  - [x] = ((x_A, k_A, m_A) , (x_B, k_B, m_B)) s.t.
  - $m_B = k_A + x_B \Delta_A$ (symmetric for m_A)
  - Δ_A, Δ_B is the same for all wires.
  - MACs, keys are k-bit strings.

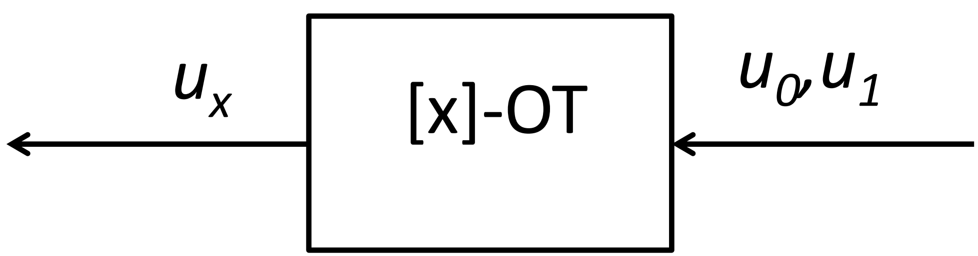
Use auth. bit to do OT

- Alice knows x
- $[x] = ((x, \ldots, m_x) , (\ldots, k_x, \ldots))$ s.t. $m_x = k_x + x\,\Delta$

$$c_0 = H(k_x) + u_0$$

$$c_1 = H(k_x + \Delta) + u_1$$

$$u_x = c_x + H(m_x) \longleftarrow$$



Authenticated cross-products

- **Input:** [x], [y], [z], [r]**;**
- **Alice has private input:** x, r
- **Bob has private input:** y, z
- **Output:** [s] s.t. $s = xy + z$

Authenticated cross-products

x, r

y, z

$z,$

$y+z,$

$s = xy + z$

[x]-OT

$d = r + s$

$[s] = [r] + d$

Authenticated cross-products

x, r

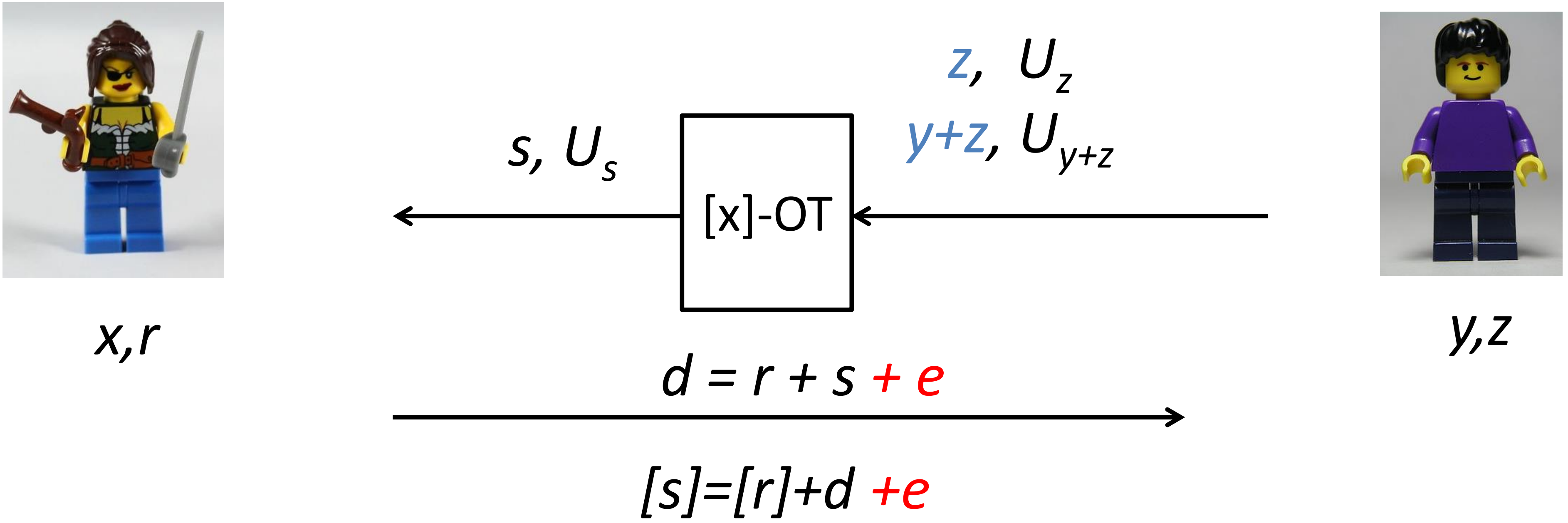
y, z

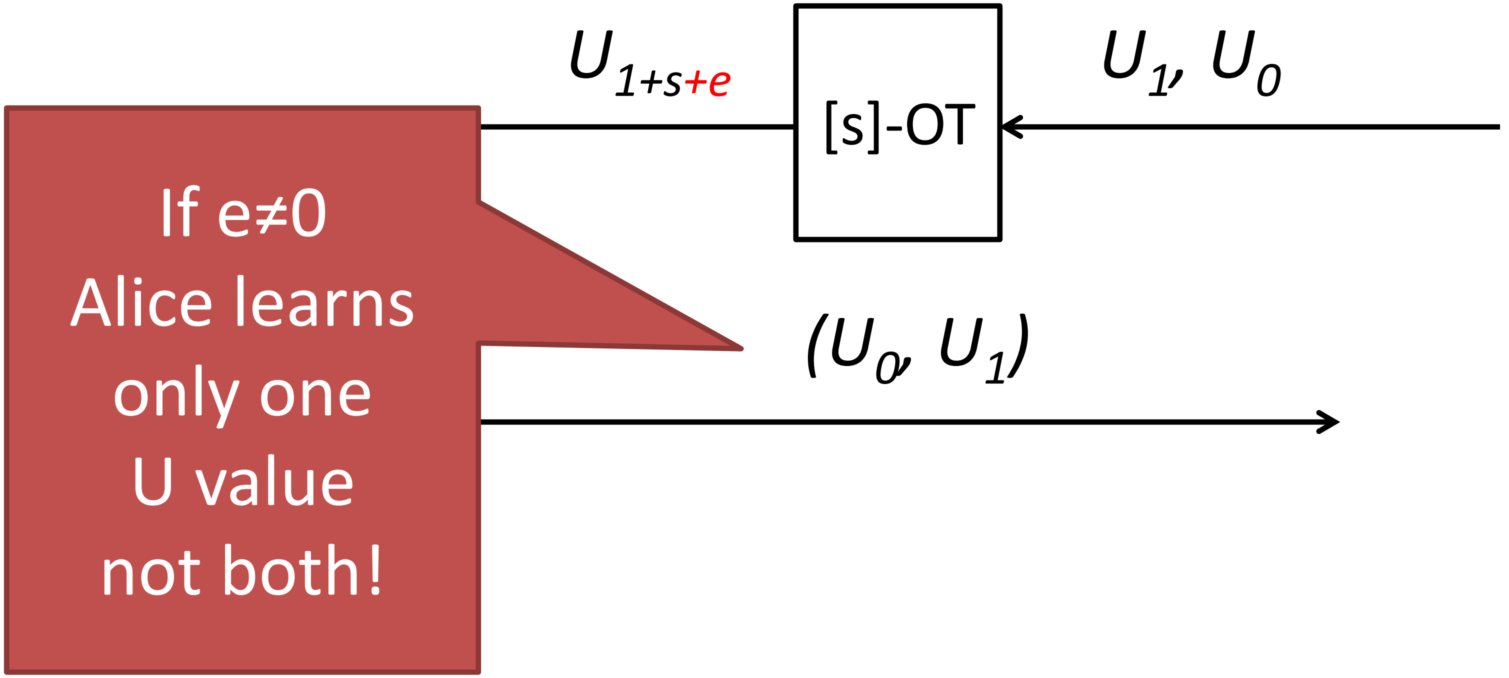
$s = xy + z$ ← [x]-OT ← z, $y + z$,

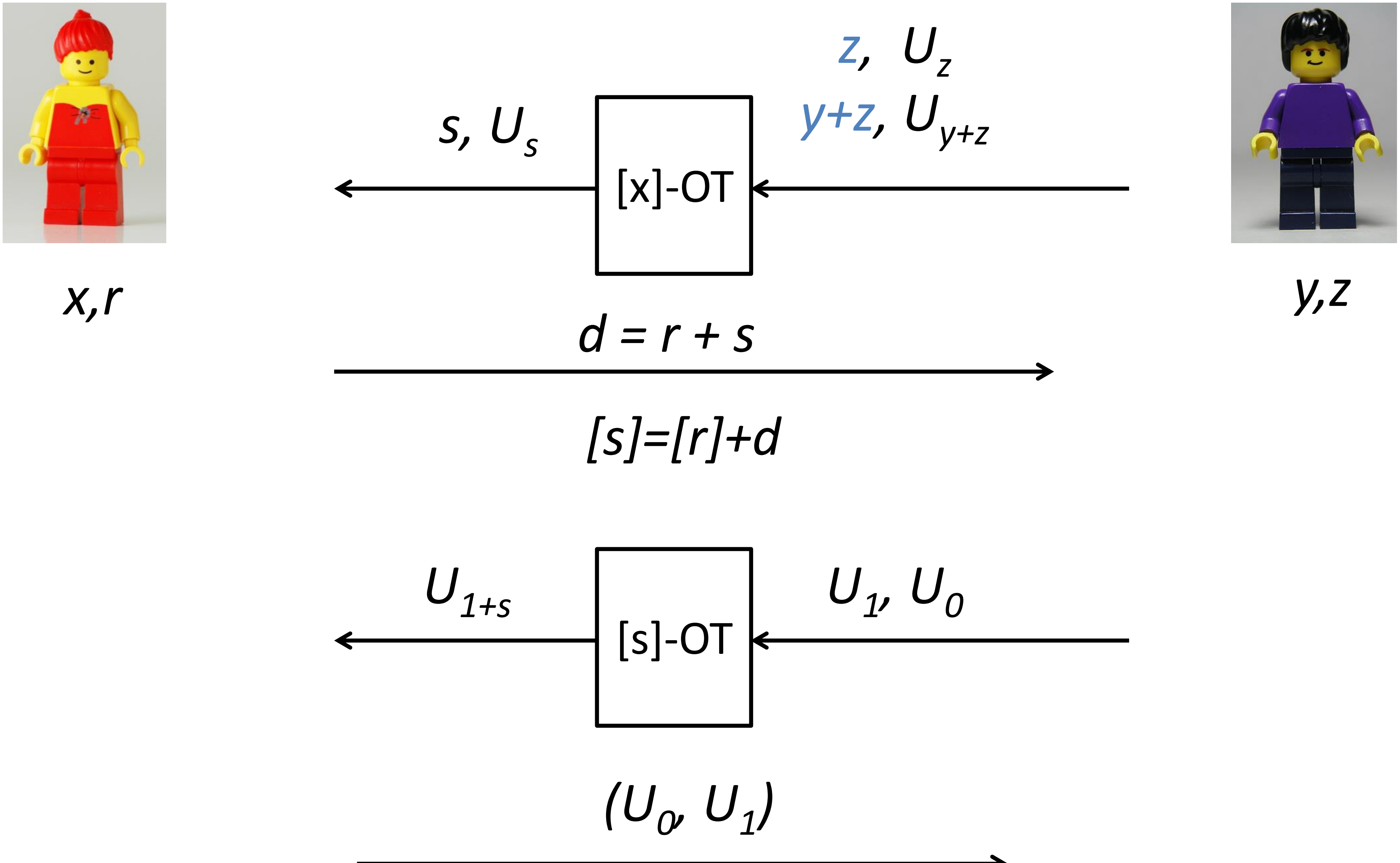
What if $e \neq 0$?

$d = r + s + e$ →

$[s] = [r] + d + e$


x,r
s, U_s
[x]-OT
z, U_z
y+z, U_y+z
y,z
d = r + s + e
[s]=[r]+d +e


U_1+s+e
[s]-OT
U_1, U_0
(U_0, U_1)
If e≠0
Alice learns
only one
U value
not both!


x,r
y,z
z, U_z
y+z, U_{y+z}
s, U_s
[x]-OT
d = r + s
[s]=[r]+d
U_{1+s}
U_1, U_0
[s]-OT
(U_0, U_1)

x,r

y,z

$z+e,\ U_z$

$y+z+e,\ U_{y+z}$

$s+f(x,e),\ U_s$ ← [x]-OT ←

$d = r + s + f(x,e)$ →

Leads to wrong result!

$[s+f(x,e)]=[r]+d$

U_{1+s} ← [s]-OT ← $U_1+e,\ U_0+e$

$+f(s,e),\ U_1+f(s,e))$ →

Bob learns s!

Solution: make sure that cheating leads to aborts w.p. ½

z, U_z

$y+z$, U_{y+z}

s, U_s ← [x]-OT ←

x,r

y,z

$d = r + s$ →

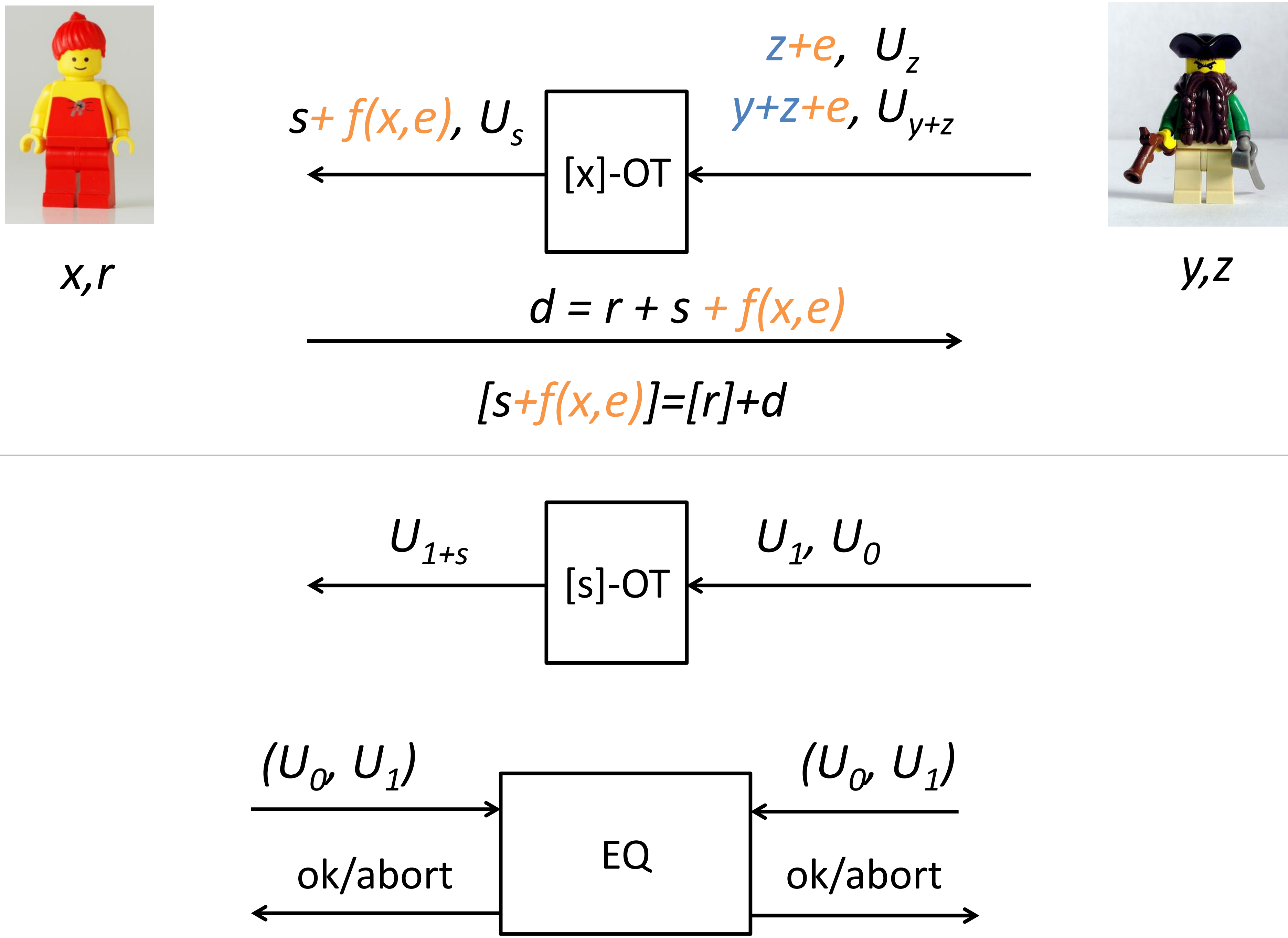
$[s]=[r]+d$

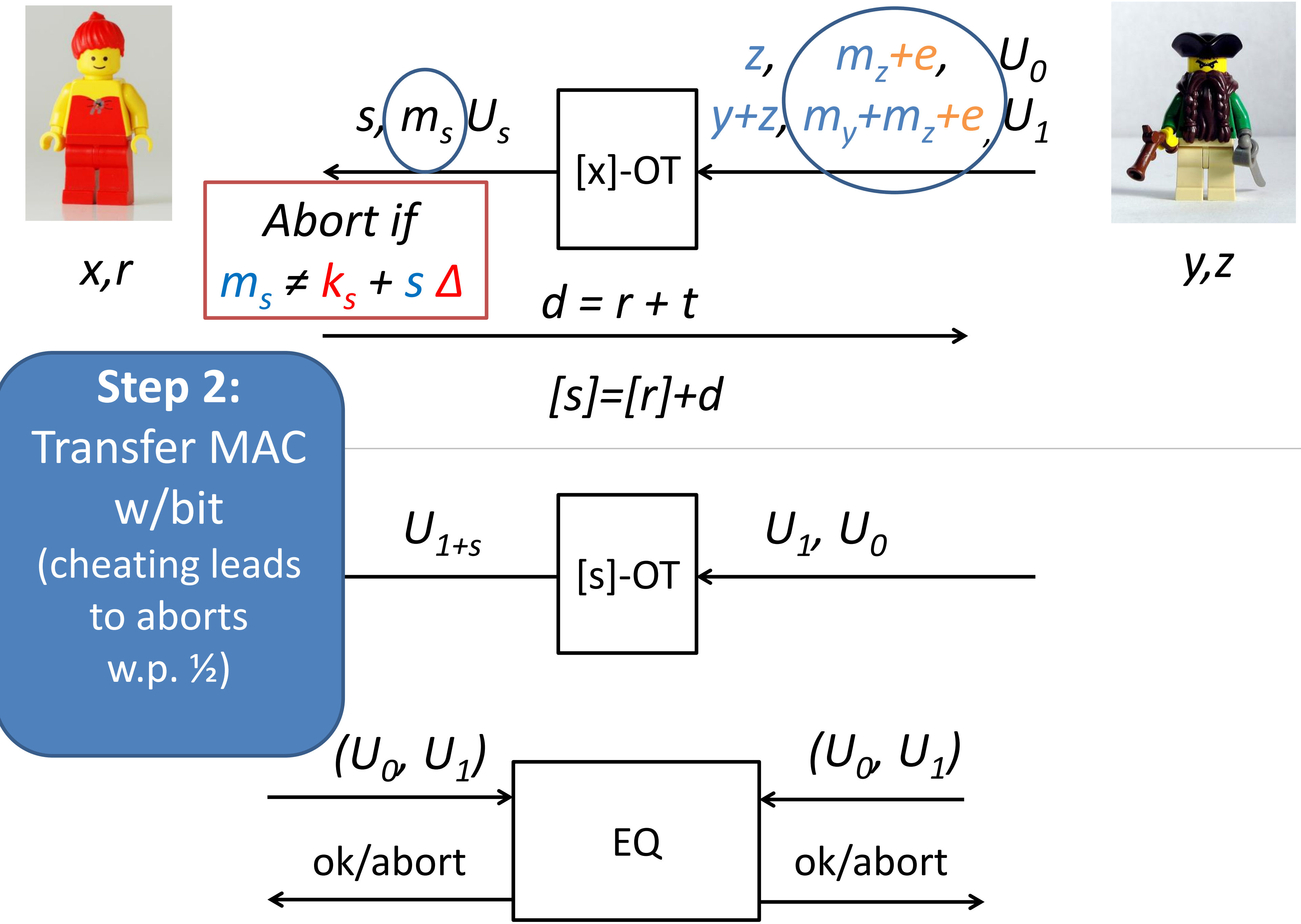
Step 1:
check U_0,U_1
w/EQ
(cheating leads to aborts w.p. ½)

U_{1+s} ← [s]-OT ← U_1+e, U_0+e

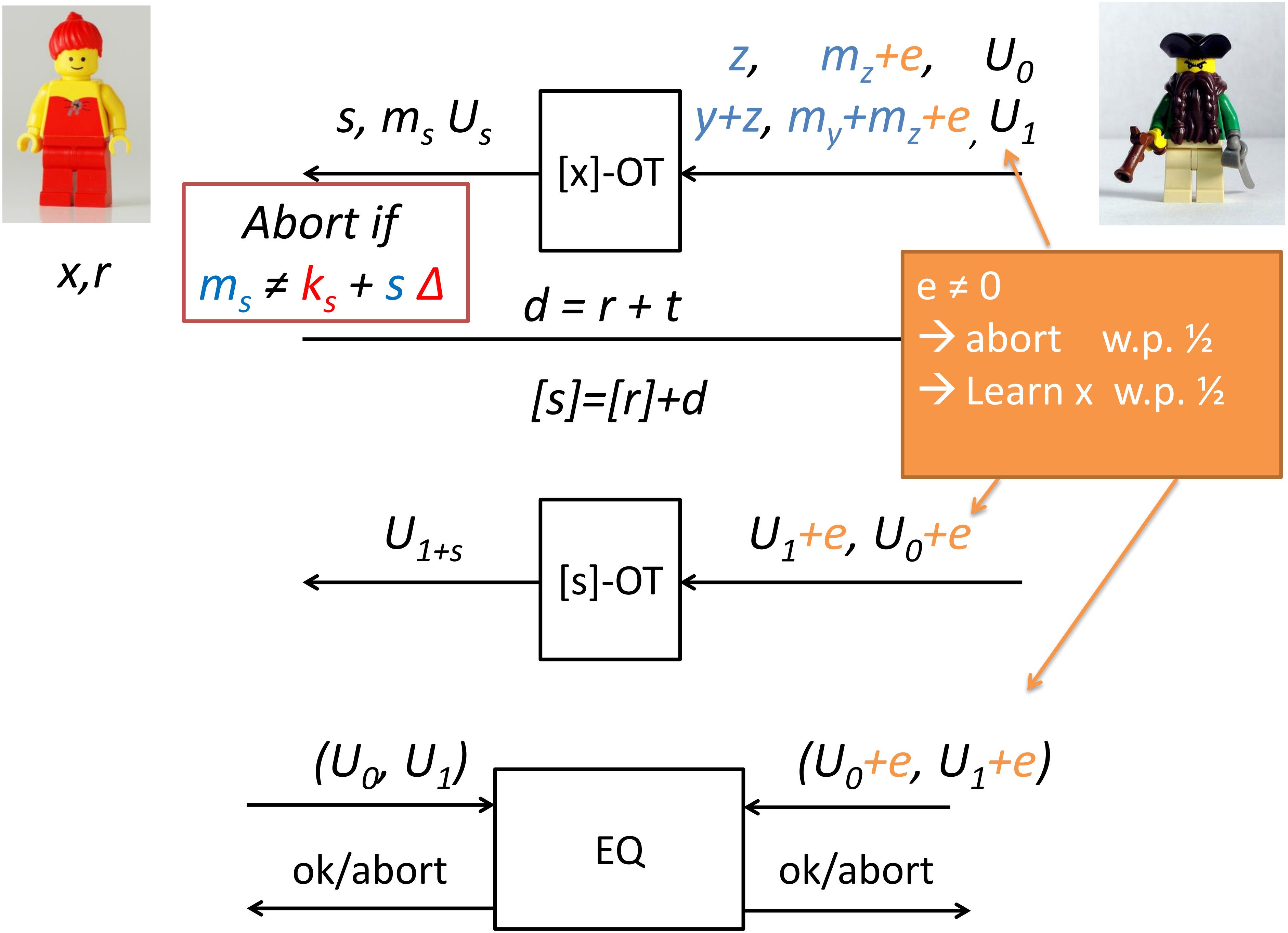
$(U_0+f(s,e), U_1+f(s,e))$ → EQ ← $(U_0+f(s,e), U_1+f(s,e))$

ok/abort ← EQ → ok/abort


x,r
y,z
s+ f(x,e), U_s
[x]-OT
z+e, U_z
y+z+e, U_{y+z}
d = r + s + f(x,e)
[s+f(x,e)]=[r]+d
U_{1+s}
[s]-OT
U_1, U_0
(U_0, U_1)
EQ
(U_0, U_1)
ok/abort
ok/abort



x,r
y,z
s, m_s, U_s
[x]-OT
z, m_z+e, U_0
y+z, m_y+m_z+e, U_1
Abort if
$m_s \neq k_s + s\,\Delta$
$d = r + t$
$[s]=[r]+d$
Step 2:
Transfer MAC w/bit (cheating leads to aborts w.p. ½)
U_{1+s}
[s]-OT
U_1, U_0
(U_0, U_1)
EQ
(U_0, U_1)
ok/abort
ok/abort


x,r
s, m_s U_s
[x]-OT
z, m_z+e, U_0
y+z, m_y+m_z+e, U_1
Abort if
m_s ≠ k_s + s Δ
d = r + t
[s]=[r]+d
e ≠ 0
→ abort w.p. ½
→ Learn x w.p. ½
U_1+s
[s]-OT
U_1+e, U_0+e
(U_0, U_1)
(U_0+e, U_1+e)
EQ
ok/abort
ok/abort

Combine local multiplications

- **Input**: $[x_1]$, $[y_1]$, $[z_1]$, $[s_1]$,$[x_2]$, $[y_2]$, $[z_2]$, $[s_2]$
  // $s_i = x_i y_i + z_i$, Alice knows x_i s_i, Bob knows y_i,z_i
  // Bob knows: x_1 or x_2 (not both)
- **Output:** [a], [b], [c], [t]   // Bob knows nothing

1. $[a] = [x_1] + [x_2]$ // Now a random
2. $[b] = [y_1]$, $[c] = [z_1] + [z_2]$
3. $d = \text{Open}([y_1]+[y_2])$
4. $[t] = [z_1] + [z_2] + d[x_2]$

//$x_1y_1+z_1+x_2y_2+z_2+x_2y_1+x_2y_2 =(x_1+x_2)y_1+z_1+z_2=ab+c$

Part 3:
From “Auth. Bits” to “Auth. Triples”

- Authenticated local-products (*aAND*)
- Authenticated cross-products (*aOT*)
- **“LEGO” bucketing**

Finishing Up

- We can compute **local-products** and **cross-products** where if **one party cheats**
  - **w.p. ½** protocol **aborts**
  - **w.p. ½** protocol **continues** and cheating party **learns 1 bit**

- If protocol continues

  ➔ There are at most σ leaked bits (w.p. $2^{-\sigma}$)

  ➔ Let *M* #multiplication gates

  ➔ Typically $M >> \sigma$

"LEGO" bucketing

- **Bucket size *B*, *M* buckets**
  - *overhead, # of multiplications*
- **Total work *BM*, randomly assign in buckets**
  - #of generated triples
- **Secure if ≥ 1 "good" in each bucket**
  - using combiners presented before
- **Stat. Sec. $2^{-\sigma}$ with bucket size $B = \frac{\sigma}{\log_2 N}$**
  - Larger circuits → more efficiency!

Tiny OT - Recap

- **Preprocessing**
  - Generate authenticated bits (OT extension)
  - Exploit **duality authenticated bit/OT** to perform **local multiplications** and **cross multiplications** efficiently *(but with some limited leakage)*
  - Randomly assign in small buckets (e.g., B=4)
  - Combine to get rid of leakage
- **Online phase**
  - Use precomputed triples to evaluate any circuit.