Efficient MPC

Oblivious Transfer and
Oblivious Linear Evaluation
aka "How to Multiply"

SODA
Scalable Oblivious Data Analytics

erc
European Research Council
Established by the European Commission

DANMARKS FRIE FORSKNINGSFOND
INDEPENDENT RESEARCH FUND DENMARK

Claudio Orlandi, Aarhus University

Circuit Evaluation

3) Multiplication?

How to compute [z]=[xy] ?

Alice, Bob should compute

$$z_1 + z_2 = (x_1+x_2)(y_1+y_2)$$

$$= x_1y_1 + x_2y_1 + x_1y_2 + x_2y_2$$

How do we compute this?

Alice can compute this

Bob can compute this

On the use of computational assumptions

- How much can we ask users to trust crypto?
  1. **Necessary** (one way functions are needed for symmetric crypto, public key crypto is probably needed for 2PC)
  2. **We must believe that some problems are hard** (e.g., breaking RSA or breaking AES). But we should not ask for more trust than needed!
  3. Construct complex systems based on well studied assumptions. Then prove (via reduction), that ***any adv that can break property X of system S can be used to solve computational problem P.***
  4. If **we believe problem P to be hard, then we conclude that system S has property X.**

The Crypto Toolbox

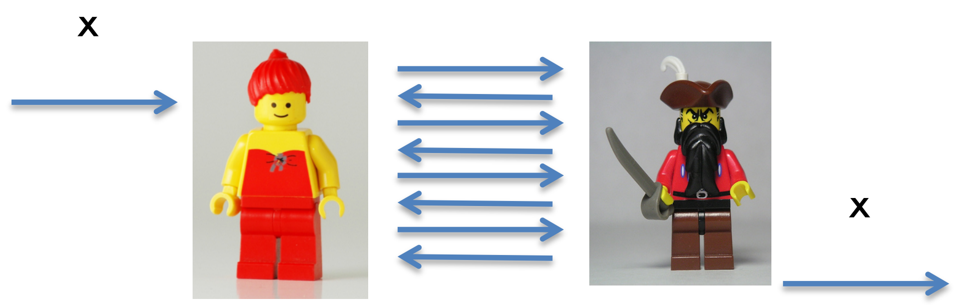
Weaker assumption ← Stronger assumption

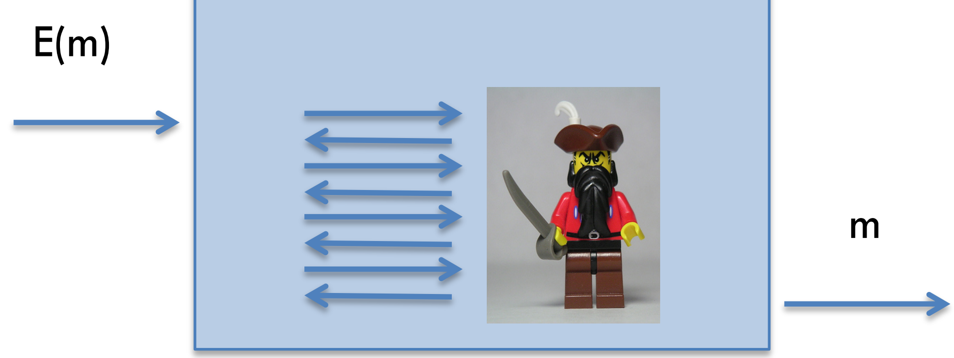
OTP >> SKE >> PKE >> FHE >> Obfuscation

More efficient ← Less efficient

Reduction Proof

- **If:** an adversary can break the security (e.g., learn the secret input x)
- **Then:** use this adversary as a subroutine to break the security of some hard problem (e.g., RSA)
- **But:** the problem is hard
- **So**: the protocol must be secure





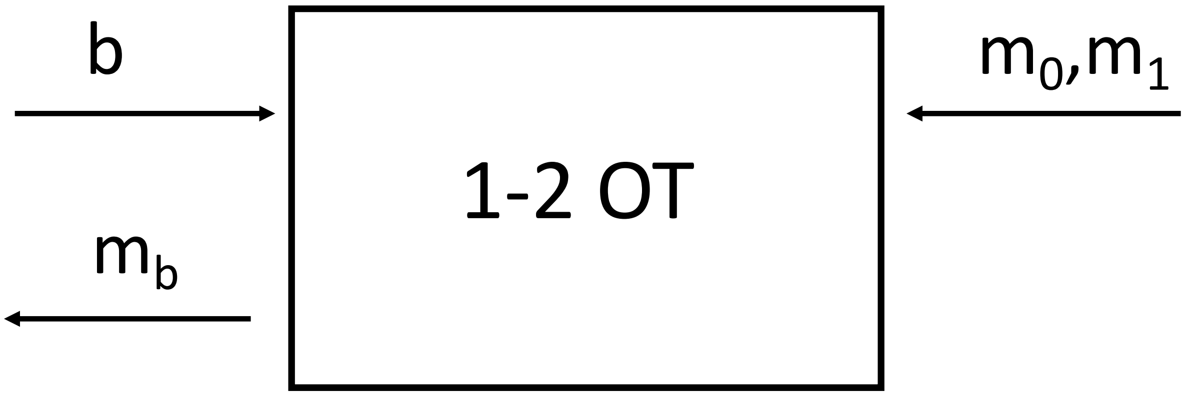
Part 2: How to multiply

- **Warmup: Useful OT Properties**
- OT Extension
- Multiplication Protocols
  - OT-based
  - Pailler Encryption
  - Noisy Encodings

1-2 OT

Receiver

Sender

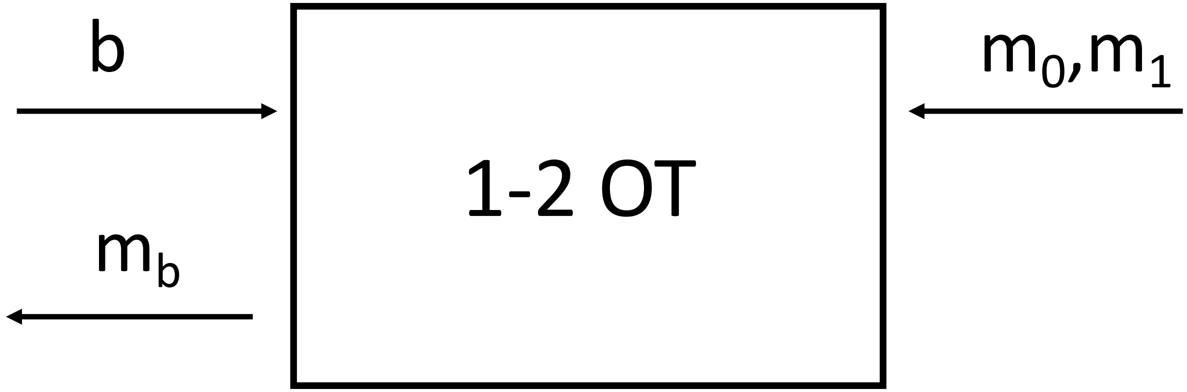


- Receiver does not learn m_{1-b}
- Sender does not learn b

1-2 OT

Receiver

Sender

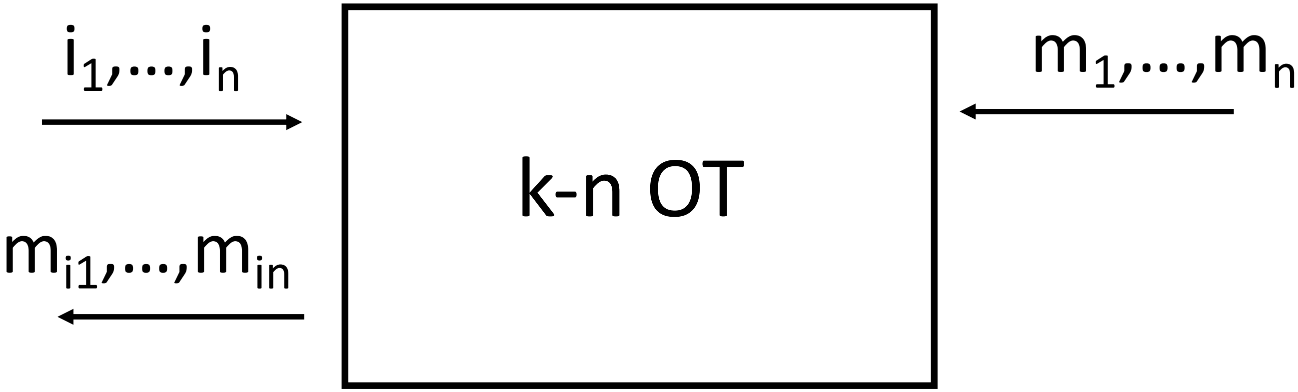


- $m_b = (1-b)\, m_0 + b\, m_1$
- $m_b = m_0 + b\, (m_1 - m_0)$

k-n OT

Receiver

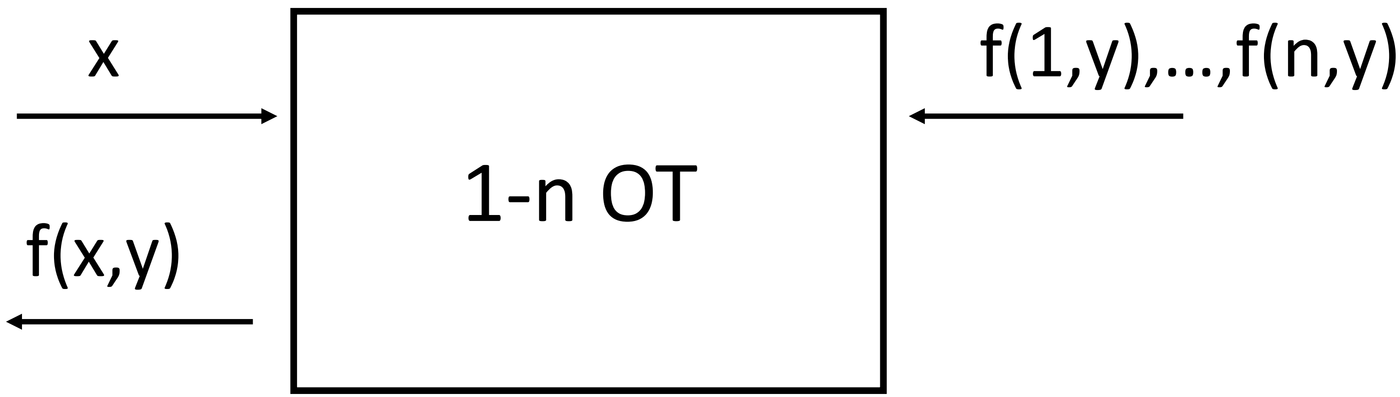
Sender



2PC via 1-n OT

Receiver

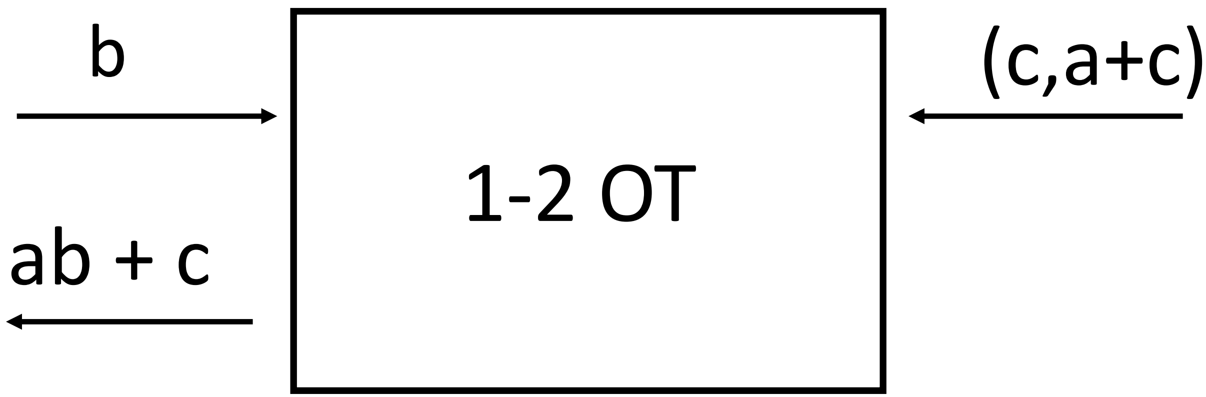
Sender



Oblivious Transfer
=
bit multiplication

Receiver

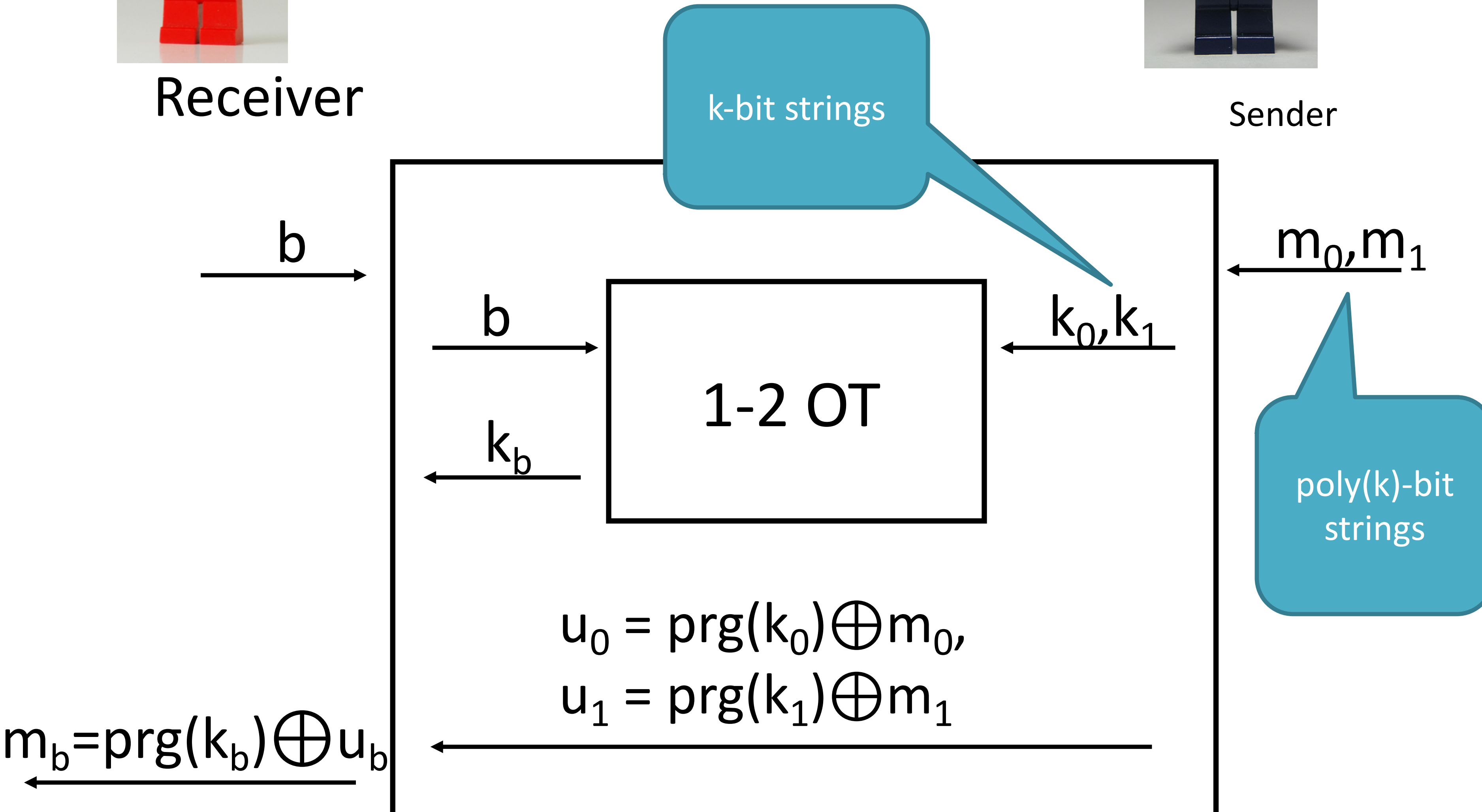
Sender



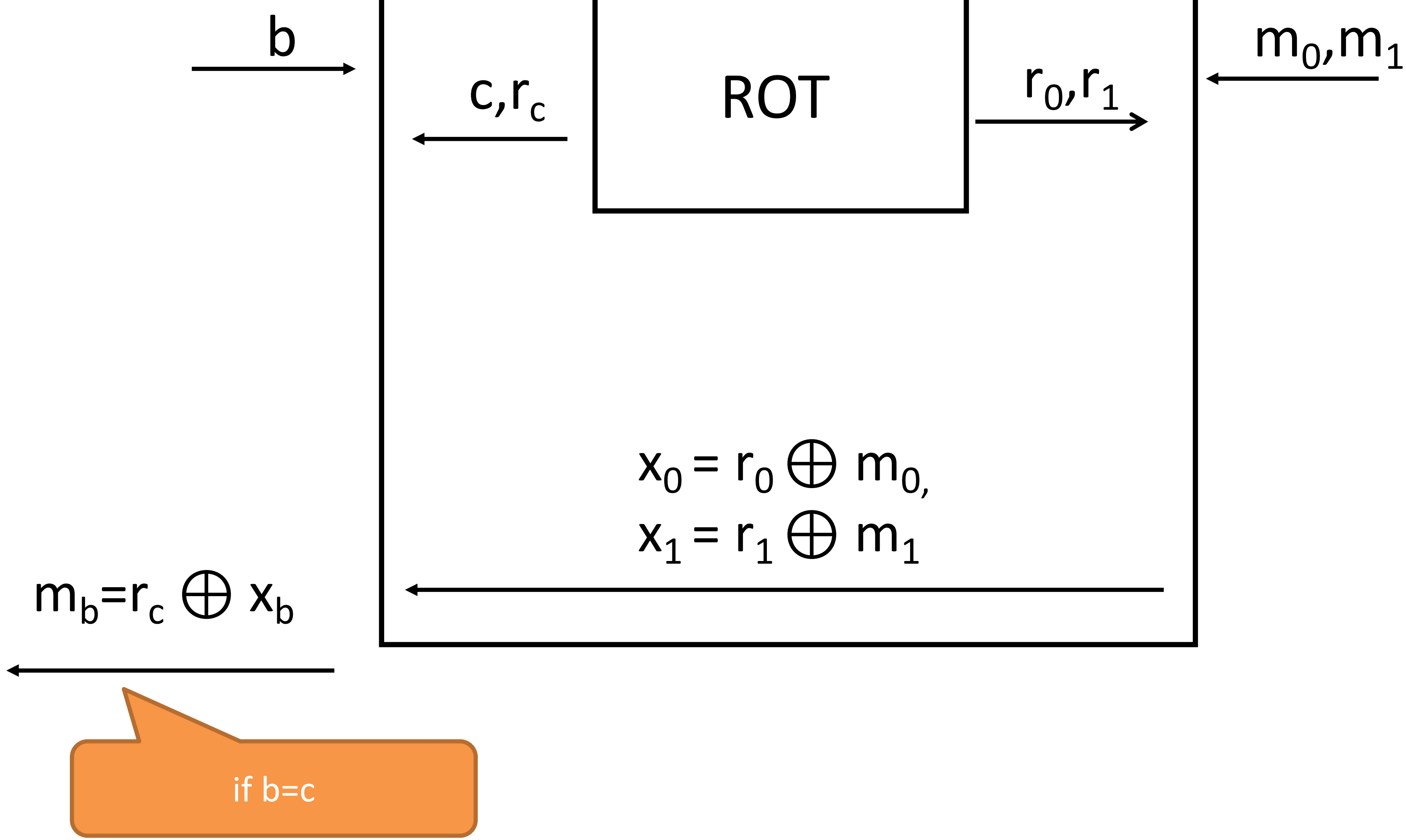
Short OT → Long OT

Receiver

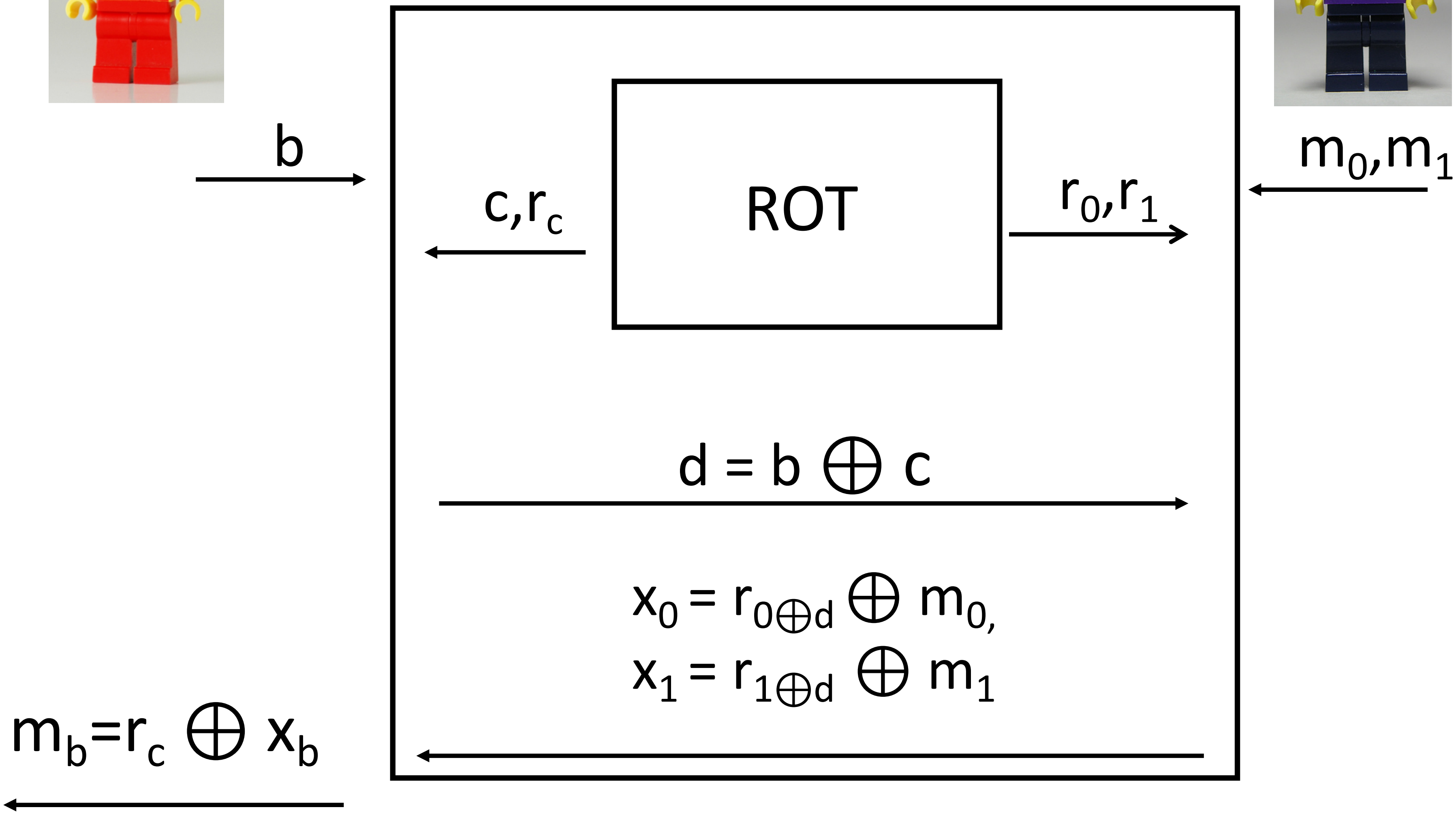
Sender



Random OT = OT

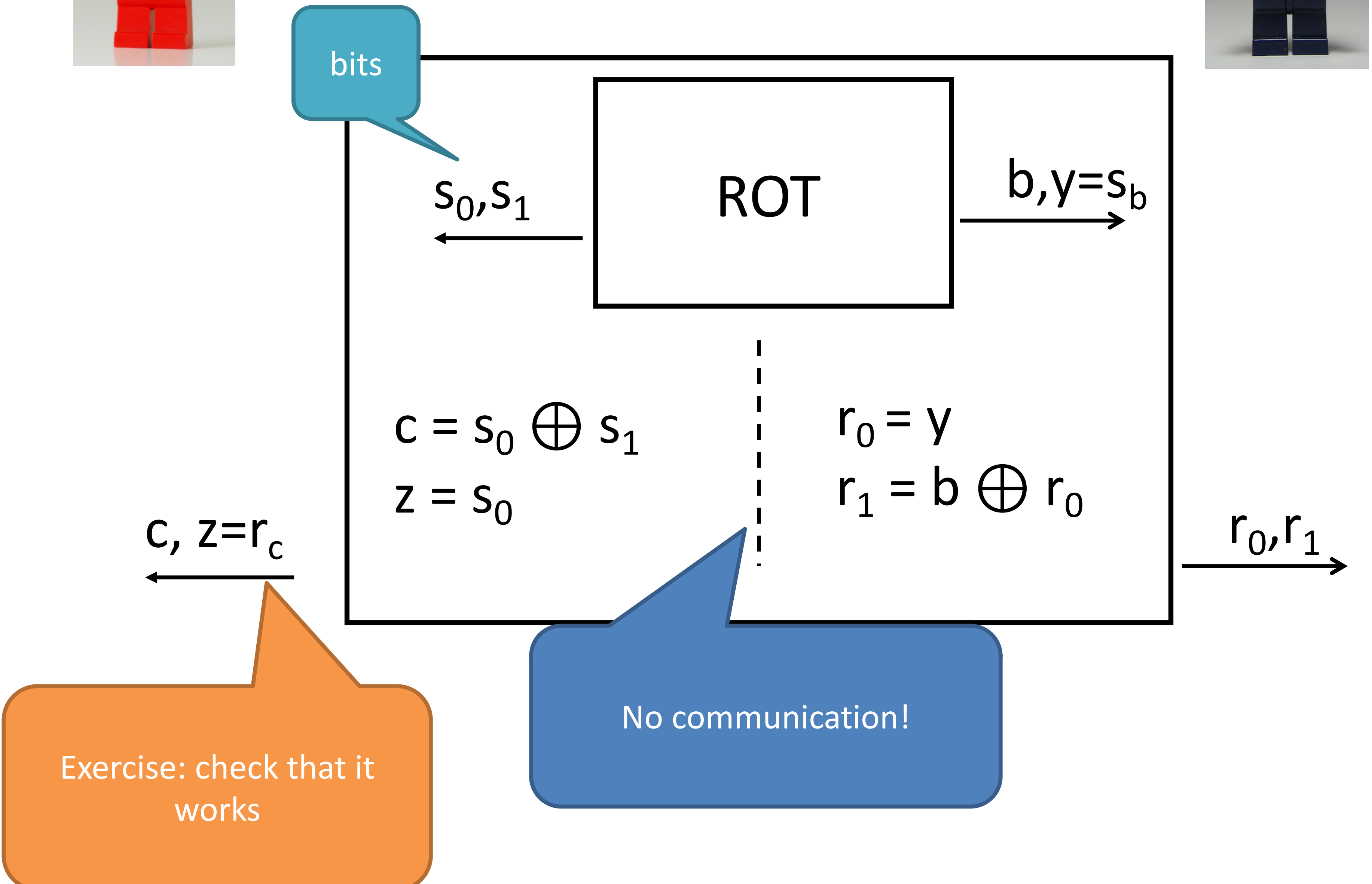


Random OT = OT



Exercise: check that it works!

(R)OT is symmetric



Part 2: How to multiply

- Warmup: Useful OT Properties
- **OT Extension**
- Multiplication Protocols
  - OT-based
  - Pailler Encryption
  - Noisy Encodings

Efficiency

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

The Crypto Toolbox

Weaker assumption ← Stronger assumption

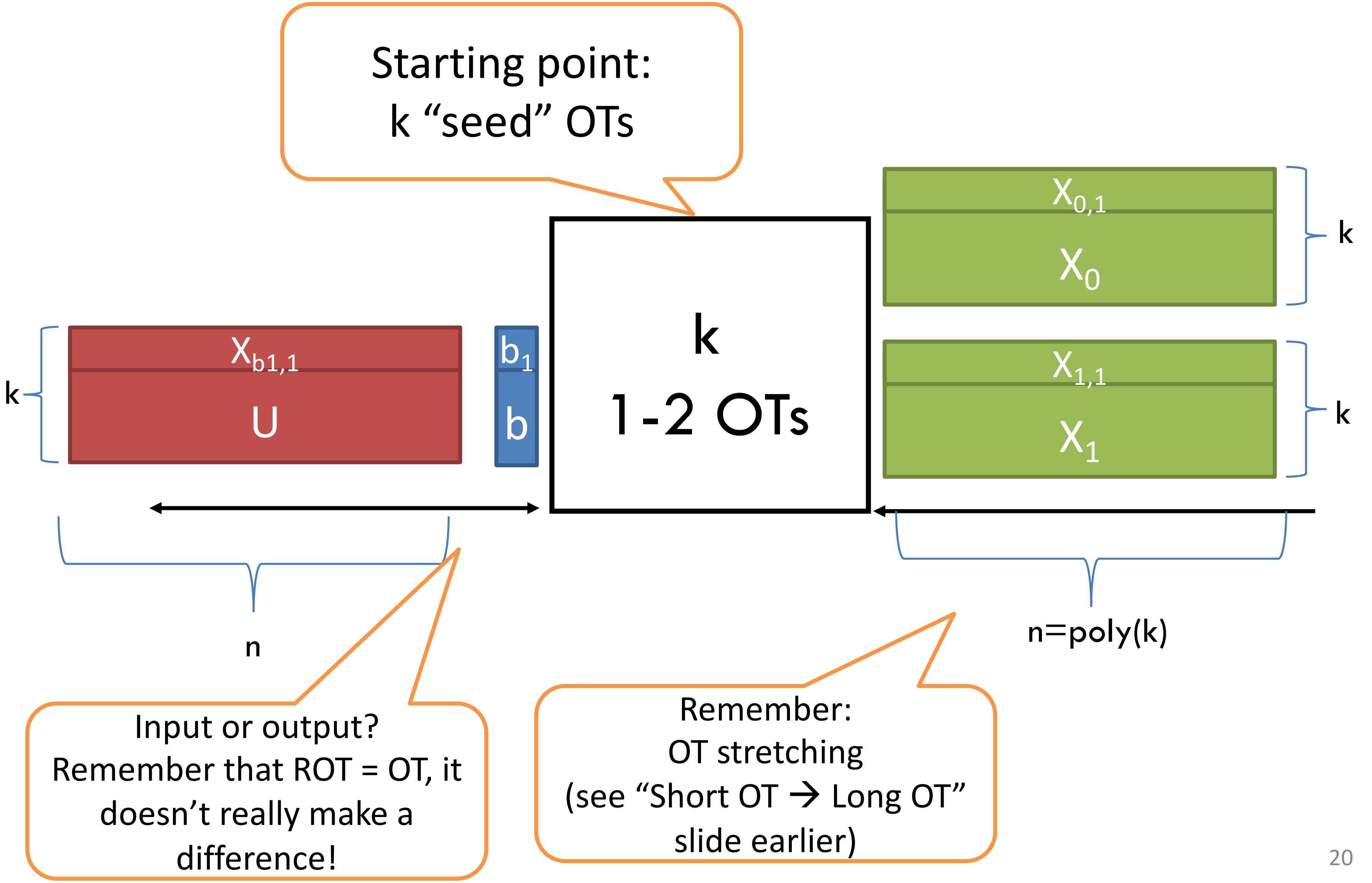
OTP >> SKE >> PKE >> FHE >> Obfuscation

More efficient ← Less efficient

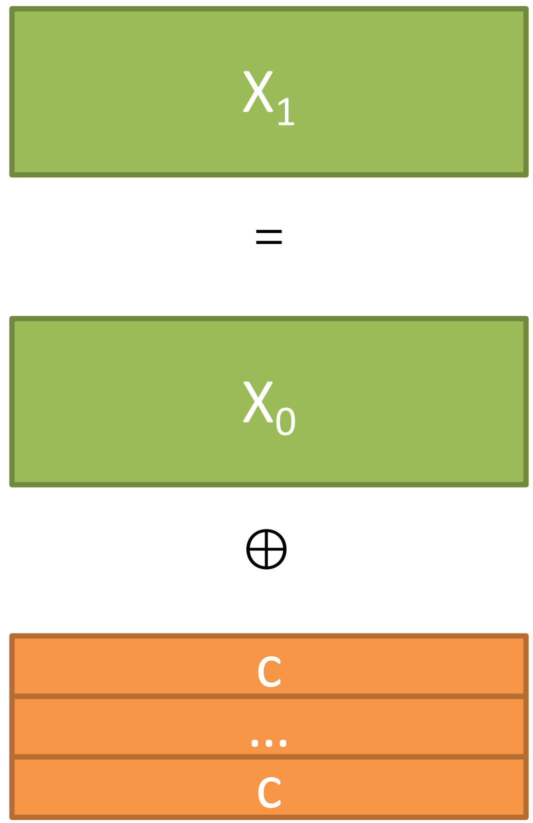
Efficiency

- ***Problem:*** OT requires public key primitives, inherently inefficient
- ***Solution:*** OT extension
  - Like hybrid encryption!
  - Start with few (expensive) OT based on PKE
  - Get many (inexpensive) OT using only SKE

OT Extension, Pictorially

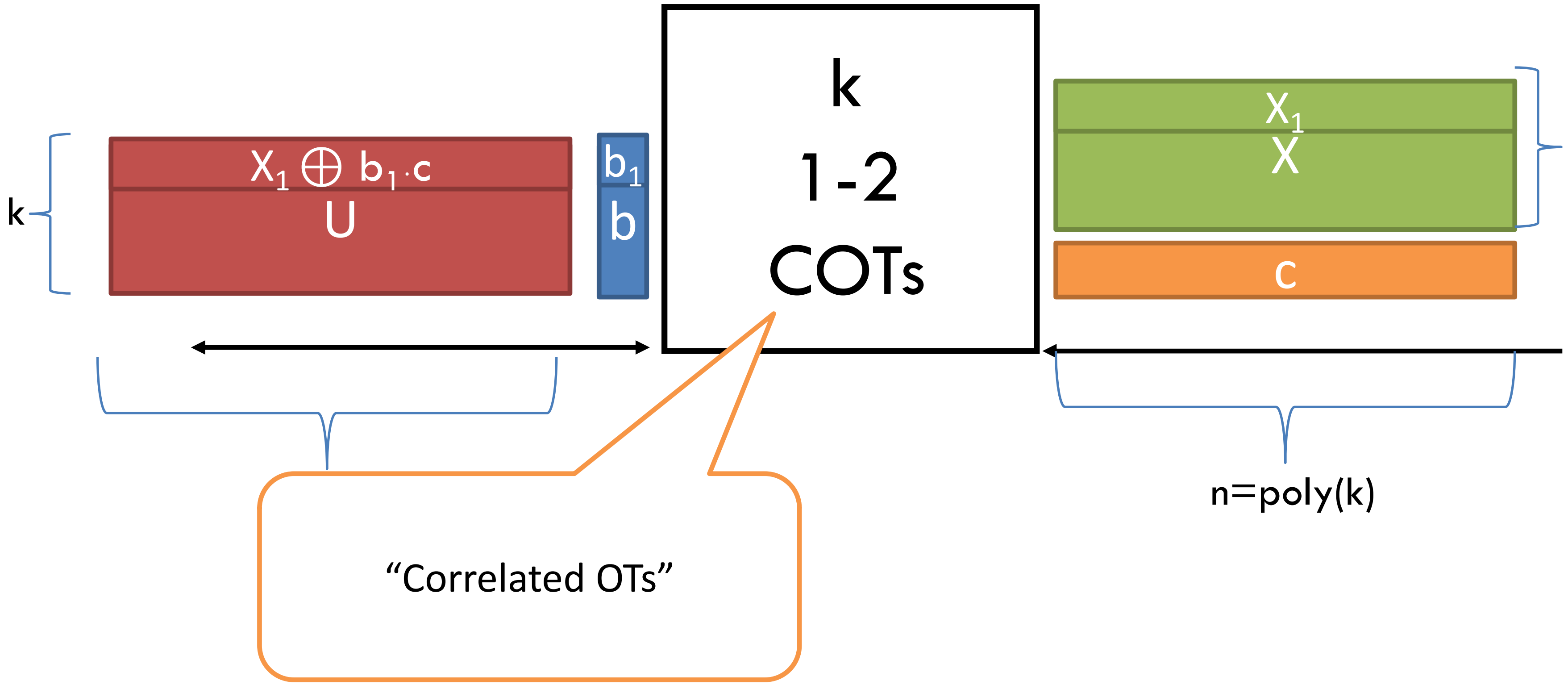


Condition for OT extension

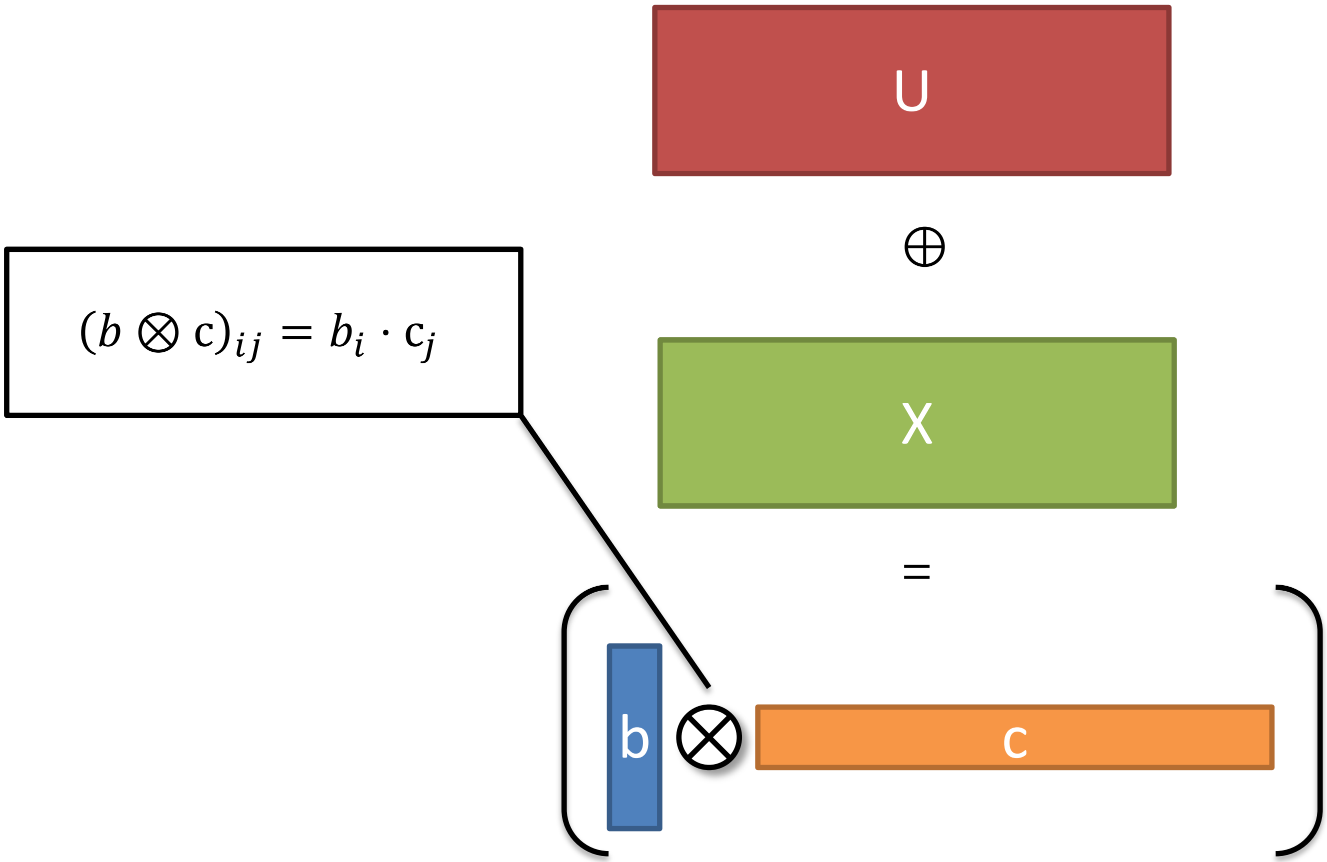


Problem for active security!

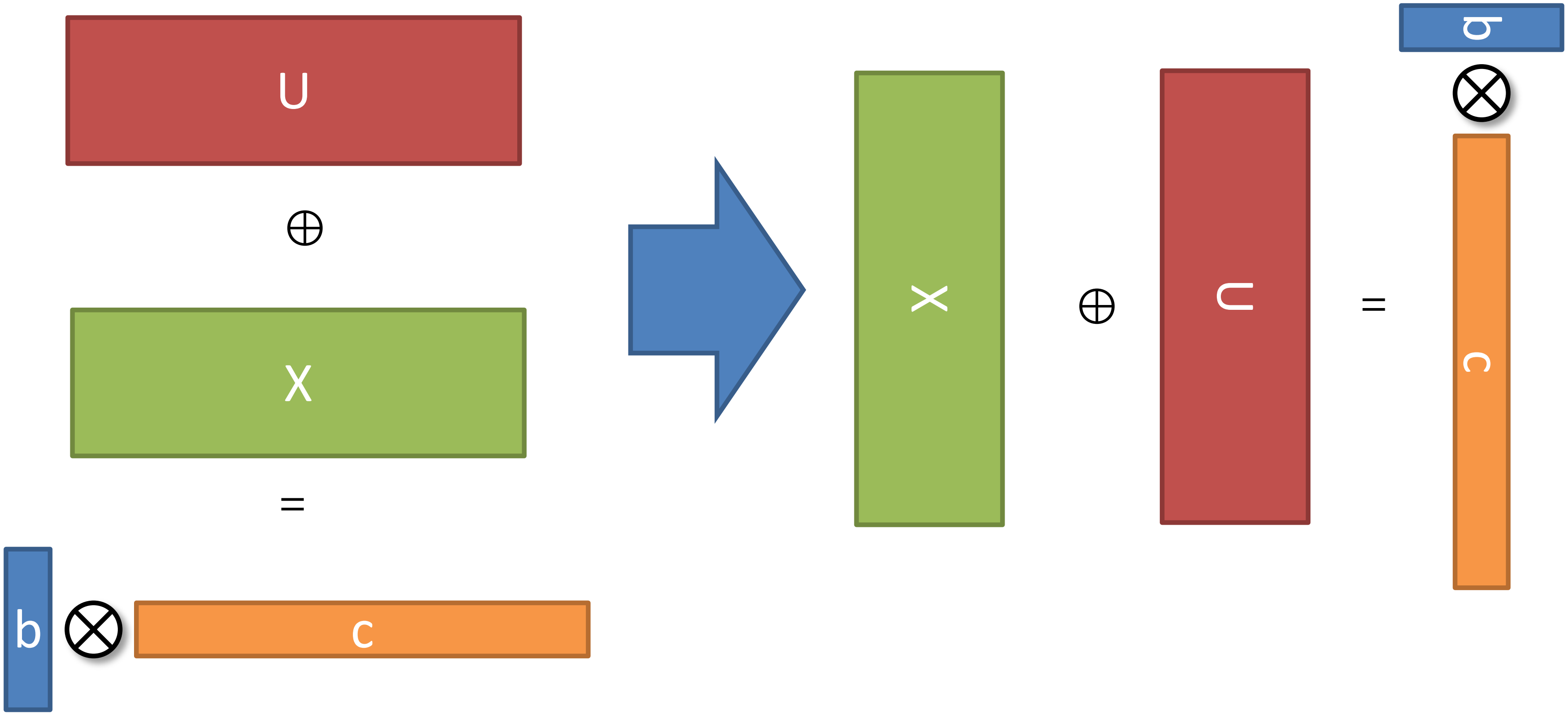
OT Extension, Pictorially



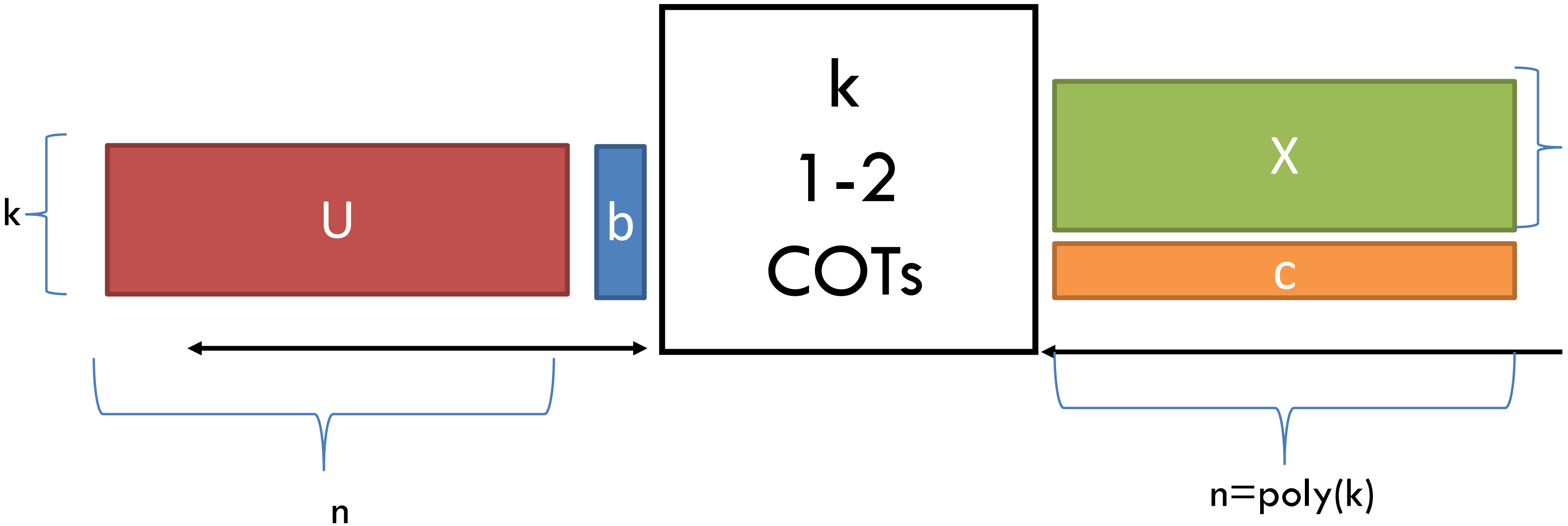
OT Extension, Pictorially



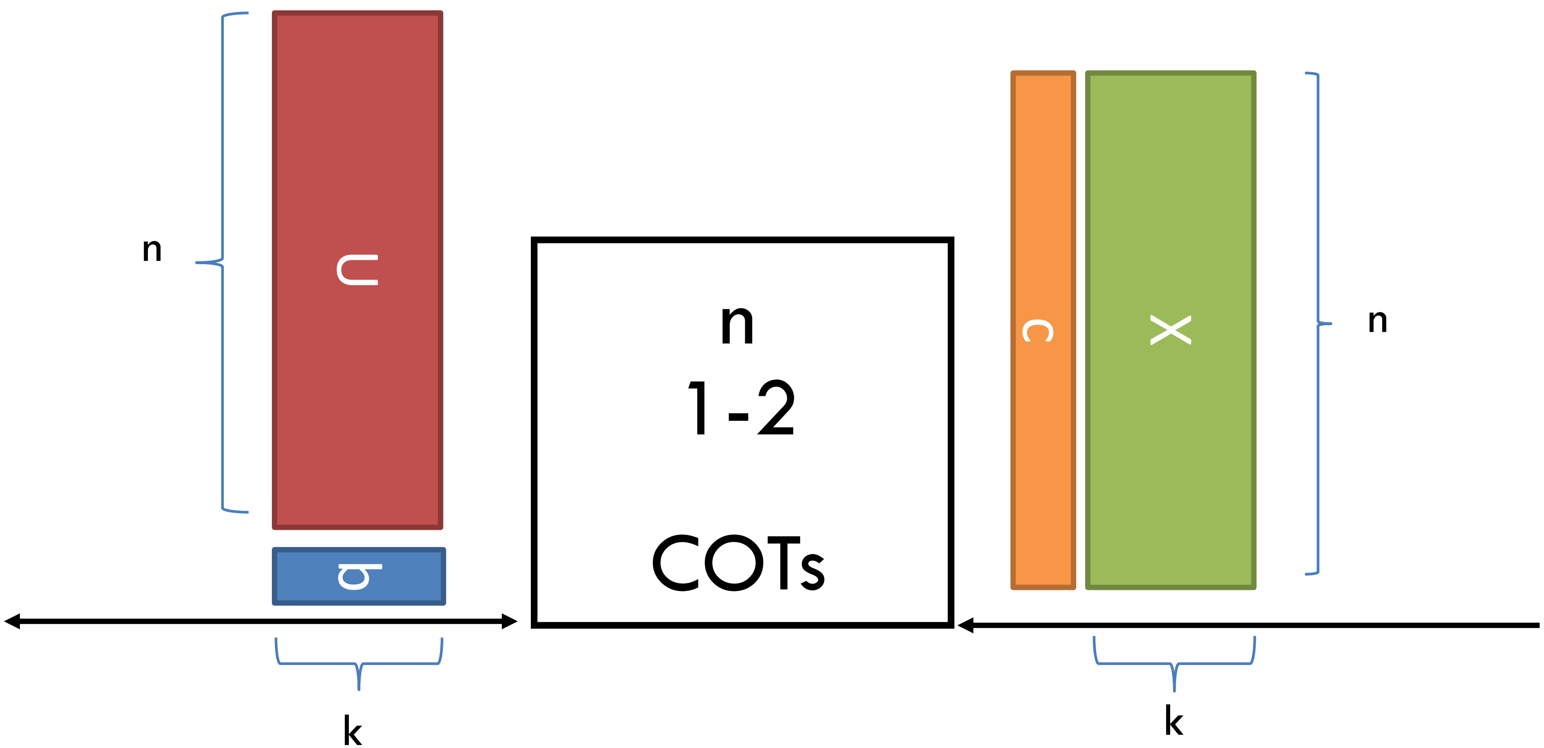
OT Extension, Turn your head!



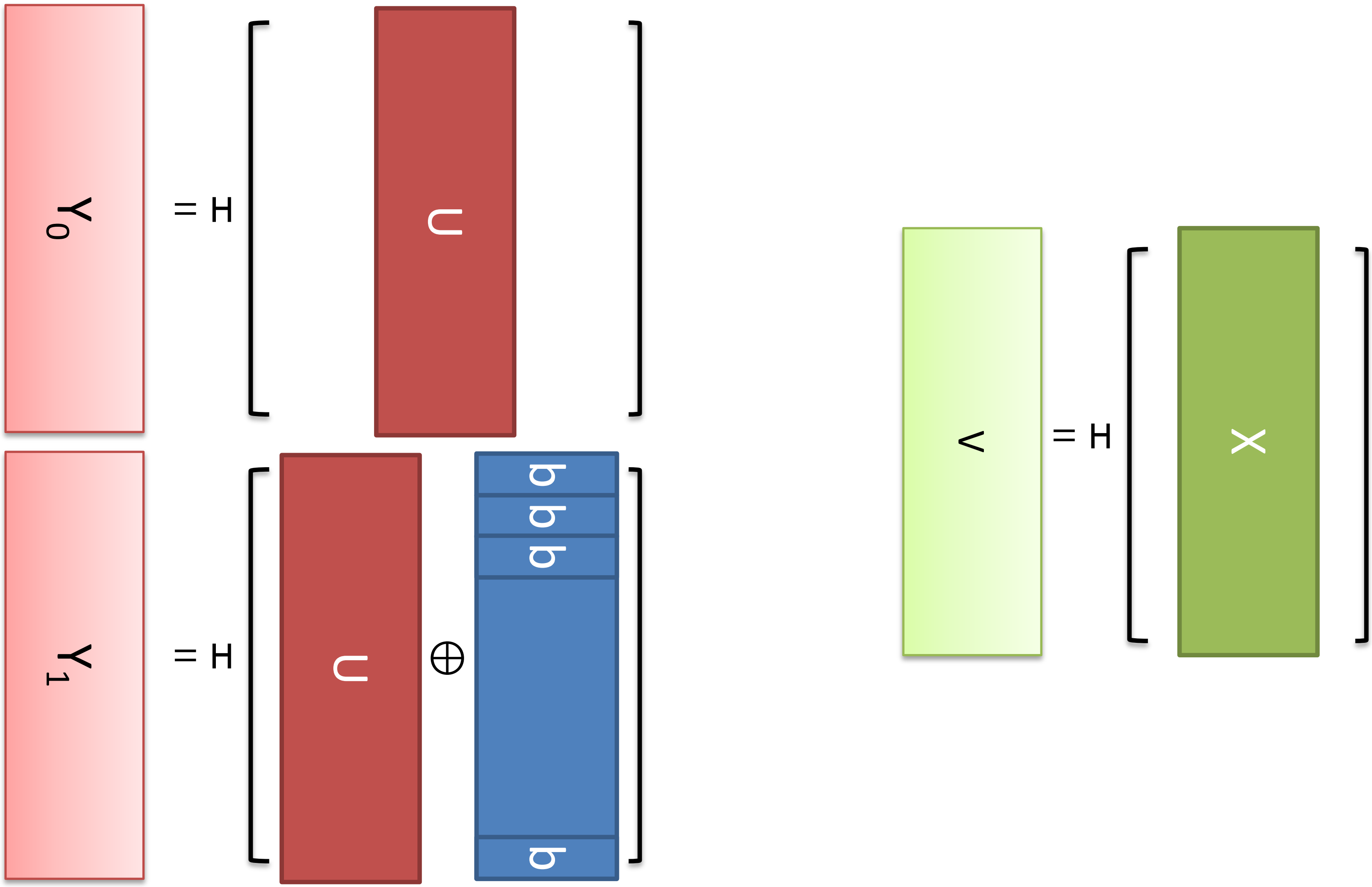
OT Extension, Pictorially



OT Extension, Pictorially



Break the correlation!

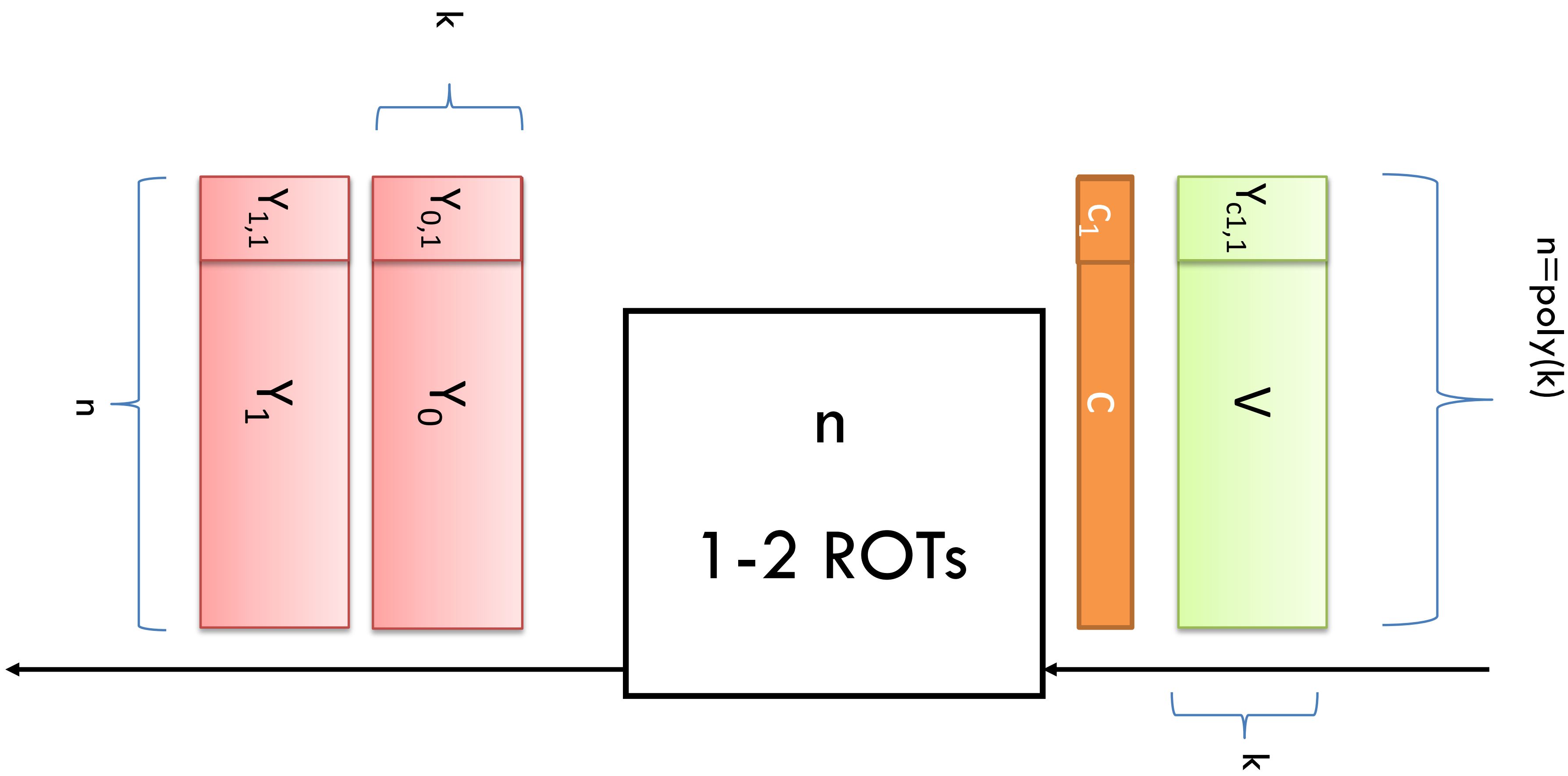


Breaking the correlation

- 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

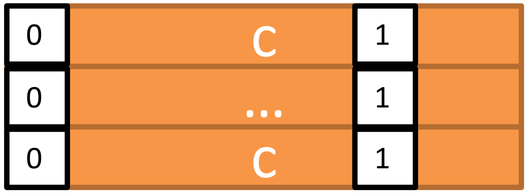
0. Strech **k OTs** from *k- to poly(k)=n-bitlong strings*
1. Send correction for each pair of messages x^i_0, x^i_1 s.t., $\mathbf{x^i_0 \oplus x^i_1 = c}$
2. **Turn your head** (S/R swap roles)
3. The bits of **c** are the new **choice bits**
4. Break the correlation: $\mathbf{y^j_0 = H(u^j),\ y^j_1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

- **Not secure against active adversaries**

Recent Results in OT Extension

(see references at the end)

- Active secure OT extension ”essentially” as efficient as passive OT.
  - Asharov et al.
  - Keller et al.
- The columns of the matrix



- Can be seen as a simple replica encoding of a bit. Better encodings can be used for better efficiency, see e.g.,
  - Kolesnikov et al.
  - Cascudo et al.

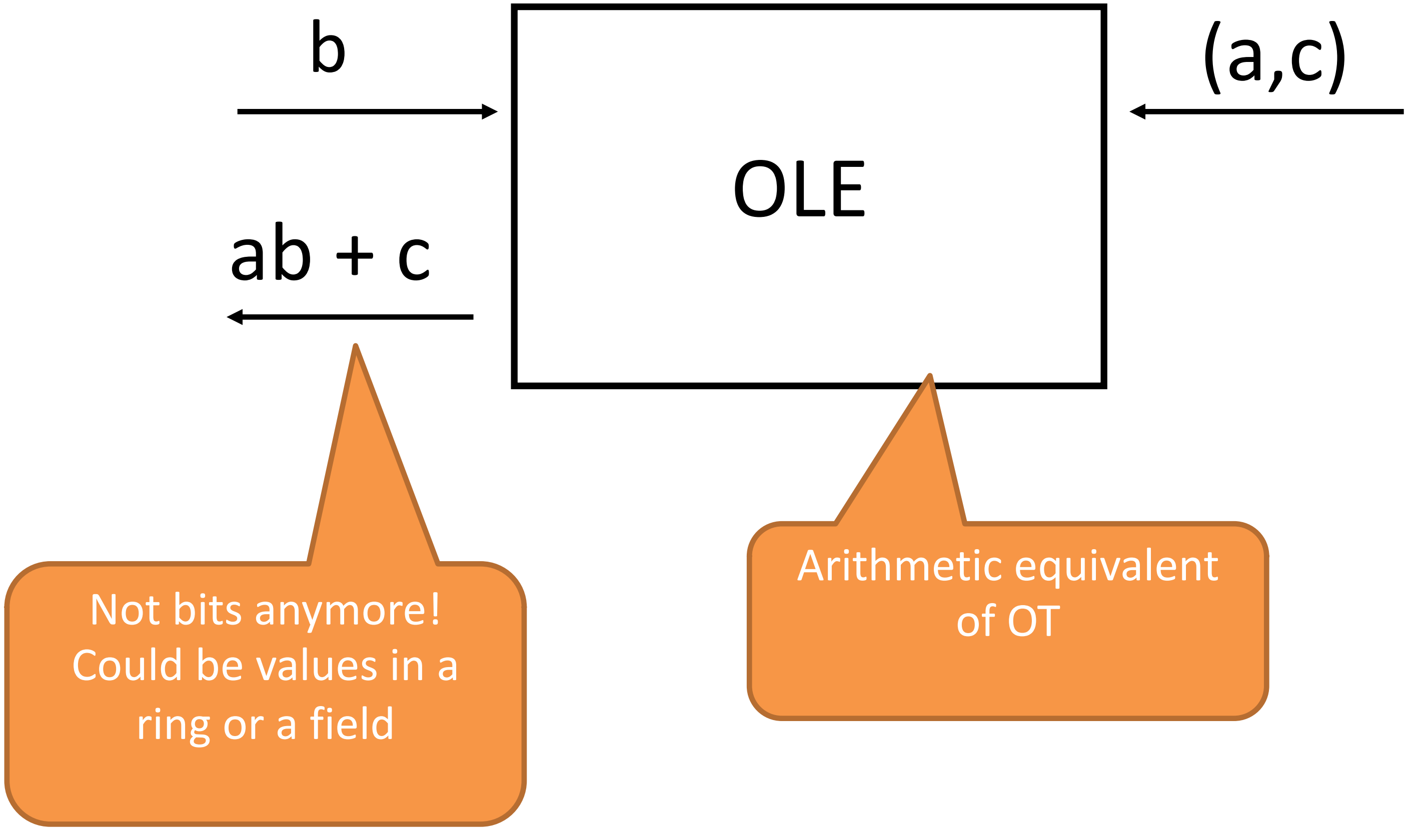
Part 2: How to multiply

- Warmup: Useful OT Properties
- OT Extension
- **Multiplication Protocols**
  - **OT-based**
  - Pailler Encryption
  - Noisy Encodings

Oblivious Linear Evaluation

Receiver

Sender



n OTs = OLE (Gilboa)

Receiver

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

Sender

a (n bit number)

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

b_i →

1-2 OT

← $(c_i, a2^i+c_i)$

$d_i=a(2^i b_i)+c_i$ ←

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

Part 2: How to multiply

- Warmup: Useful OT Properties
- OT Extension
- **Multiplication Protocols**
  - OT-based
  - **Pailler Encryption**
  - Noisy Encodings

Additive (or Linear) Homomorphic Encryption

- Pailler is a AHE whose security is related to the hardness of factoring

- Still an important tool in the protocol designer toolbox!

(Simplified) Pailler

- Public key:
  - N = pq, with |p|=|q|
- Secret key:
  - Φ(N)=(p-1)(q-1)
- Note that due to choice of parameters gcd(Φ(N),N)=1

- Pailler works mod N^2

$$\mathbb{Z}^*_{N^2} = \mathbb{Z}_N \times \mathbb{Z}^*_N$$

(Simplified) Pailler

- $(c \in \mathbb{Z}_{N^2}) \leftarrow \text{Encrypt}(m \in \mathbb{Z}_N; r \in \mathbb{Z}^*_N)$
  - Output $c = \alpha(m) \cdot \beta(r) \bmod N^2$

- Where:
  - $\alpha(m)$ takes care of the homomorphism
  - $\beta(r)$ takes care of security

$\alpha(m)$ – For homomorphism

- $\alpha(m \in \mathbb{Z}_N) = (1+mN) \bmod N^2$

- For decryption:
  - $\alpha(m)$ efficiently invertible
  - $\alpha^{-1}(y \in \mathbb{Z}_{N^2}) = y-1 / N$ // Integer division

- For homomorphism:
  - $\alpha(m_1) \cdot \alpha(m_2) = \alpha(m_1 + m_2 \bmod N)$
  - **Exercise**: check this!

$\beta(r)$ – For security

- $\beta(r \in \mathbb{Z}_N^*) = r^N \bmod N^2$
- For decryption:
  - $\beta(r)^{\Phi(N)} = 1 \bmod N^2$
- Assumption for security
  - $\{\beta(r) \mid r \leftarrow \mathbb{Z}_N^*\} \approx \{s \leftarrow \mathbb{Z}_{N^\wedge 2}^*\}$
- For homomorphism
  - $\beta(r_1) \cdot \beta(r_2) = \beta(r_1 \cdot r_2)$

$\Phi(N^2) = N \cdot \Phi(N)$
and

$x^{\Phi(N^\wedge 2)} = 1 \bmod N^2$

for all x in $\mathbb{Z}_{N2}^*$

Putting Things Together

- Security:
  - $\mathrm{Enc}_{pk}(m;r)$ $= \alpha(m) \cdot \beta(r)$ // r unif. in $\mathbb{Z}_N^*$

comp.ind. from $\approx \alpha(m) \cdot s$ // s unif. in $\mathbb{Z}_{N2}^*$

distributed identically to $\equiv t$ // t unif. in $\mathbb{Z}_{N2}^*$

Putting Things Together

- Homomorphism:
  - $\mathrm{Enc}_{pk}(m_1;r_1) \cdot \mathrm{Enc}_{pk}(m_2;r_2)$

$$= \alpha(m_1) \cdot \beta(r_1) \cdot \alpha(m_2) \cdot \beta(r_2)$$

$$= \alpha(m_1 + m_2 \bmod N) \cdot \beta(r_1 \cdot r_2)$$

$$= \mathrm{Enc}_{pk}(m_1 + m_2 \bmod N;\, r_1 \cdot r_2)$$

Putting Things Together - Decryption

- Dec(sk,c):

1. $t_1 = c^{\Phi(N)} \bmod N^2$
2. $t_2 = \alpha^{-1}(t_1) \bmod N$
3. $t_3 = t_2 \cdot \Phi(N)^{-1} \bmod N$
4. Output $m=t_3$

- Correctness

1. $t_1 = \alpha(m)^{\Phi(N)} \cdot \beta(r)^{\Phi(N)} =$
   $= \alpha(m \cdot \Phi(N)) \cdot 1$
2. $t_2 = \alpha^{-1}(\alpha(m \cdot \Phi(N))) =$
   $= m \cdot \Phi(N)$
3. $t_3 = m \cdot \Phi(N) \cdot \Phi(N)^{-1} =$
   $= m$

How to Multiply with Paillier

Receiver

Sender

$pk,\ B = Enc_{pk}(b;r)$ →

← $D = c^{a} \cdot Enc_{pk}(c;s)$

$d = Dec_{sk}(D) = ab + c \bmod N$

How to Multiply with Pailler

Receiver

Sender

pk, B = $Enc_{pk}(b;r)$

$D = c^a \cdot Enc_{pk}(c;s$

Privacy for Alice:
$B \approx Enc_{pk}(0;r)$
due to IND-CPA of Pailler

Privacy for Bob?
Alice knows the secret key! But due to homomorphism of Pailler
$\{sk,D\} \approx \{sk,Enc_{pk}(ab+c;t)\}$

Part 2: How to multiply

- Warmup: Useful OT Properties
- OT Extension
- **Multiplication Protocols**
  - OT-based
  - Pailler Encryption
  - **Noisy Encodings**

Slide by Satrajit Gosh

OLE from Noisy Encodings

(Ishai et al. [IPS09], generalizing [NP06])

Noisy Encodings

- Encode:

Takes $a \in \mathbb{F}^m$, outputs a set L and encoding $v \in \mathbb{F}^n$

- Eval:

Takes $b, c \in \mathbb{F}^m$ and the encoding v, outputs an encoding w

- Decode:

Takes an encoding w and the set L, outputs $y = ab + c$

Slide by Satrajit Gosh

OLE from Noisy Encodings

Encode(a)

m=1 for simplicity

1. Pick a polynomial A of degree $k-1$ with $A(0)=a$, evaluate at $n=4k$ positions 1…n

$\tilde{a}$	$A(1)$	$A(2)$	…	…	…	…	$A(n)$

2. Pick a random error vector e with $\rho = 2k+1$ non-zero elements, $\boldsymbol{L} = \{\boldsymbol{i} | \boldsymbol{e_i} = \boldsymbol{0}\}$

e	0	e_2	e_3	0	e_i	0	0

3. Add the two together

v	A(1)	e'_2	…	…	…	…	$A(n)$

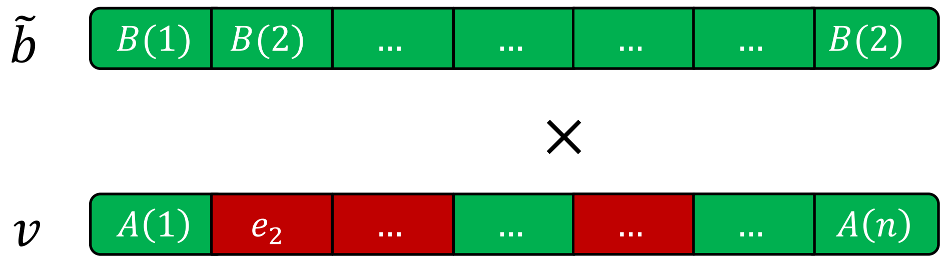
Assumption - Pseudorandomness

$$\boldsymbol{v} \leftarrow \boldsymbol{Encode}(\boldsymbol{a}) \equiv \mathcal{U}_{\boldsymbol{n}}$$

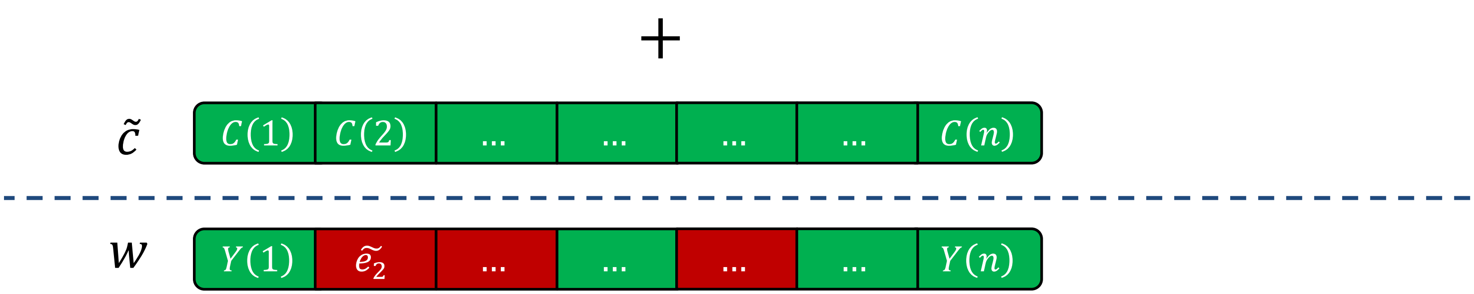
OLE from Noisy Encodings

Eval(v,b,r)

1. Pick a polynomial B of degree $k-1$
   with $B(0) = b$, evaluate at $n = 4k$ positions $1 \dots n$



2. Pick a polynomial R of degree $2k-2$ with $C(0) = c$,
   evaluate at $n = 4k$ positions 1…n



Slide by Satrajit Gosh

OLE from Noisy Encodings

Decode(w,L)

1. Ignore all $i \notin L$

w

2. Interpolate the polynomial $Y(x)$ and output $Y(0) = ab + c$

y

Slide by Satrajit Gosh

OLE from Noisy Encodings

$a \in \mathbb{F}$

$b, c \in \mathbb{F}$

$(v, L) \leftarrow Encode(a)$

$w \leftarrow Eval(v, b, c)$

L → $\binom{n}{k}$-OT ← w

$w_{|L}$

$y \leftarrow Decode(w_{|L}, L)\ (= ab + c)$

Constant overhead per multiplication!*

*using packed secret sharing

Summary

- OT properties
  - Symmetric
  - ROT and OT equivalence
  - OT can be stretched
- OT extension
  - Passive security
- Multiplication protocols
  - Gilboa (OT-based)
    - #OTs = #bits
    - (works on any ring)
  - AHE (Pailler)
  - Noisy Encoding
    - (works for fields)
    - #OTs independent on bitlength

Primary References

- Cryptographic Computing, lecture notes, http://orlandi.dk/crycom (with theory and programming exercises)
- Extending Oblivious Transfers Efficiently (Ishai et al.)
- A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System (Damgård et al.)
- Public-Key Cryptosystems Based on Composite Degree Residuosity Classes (Paillier)
- Secure Arithmetic Computation with No Honest Majority (Ishai et al.)
- Two Party RSA Key Generation (Gilboa)
- Extending Oblivious Transfers Efficiently - How to get Robustness Almost for Free (Nielsen)

Other References

- Oblivious Polynomial Evaluation (Naor et al.)
- More Efficient Oblivious Transfer Extensions with Security for Malicious Adversaries (Asharov et al.)
- Actively Secure OT Extension with Optimal Overhead (Keller et al.)
- Improved OT Extension for Transferring Short Secrets (Kolesnikov et al.)
- Efficient Batched Oblivious PRF with Applications to Private Set Intersection (Kolesnikov et al.)
- Actively Secure OT-Extension from q-ary Linear Codes (Cascudo et al.)
- Maliciously Secure Oblivious Linear Function Evaluation with Constant Overhead (Ghosh et al.)
- MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer (Keller et al.)
- A New Approach to Practical Active-Secure Two-Party Computation (Nielsen et al.)