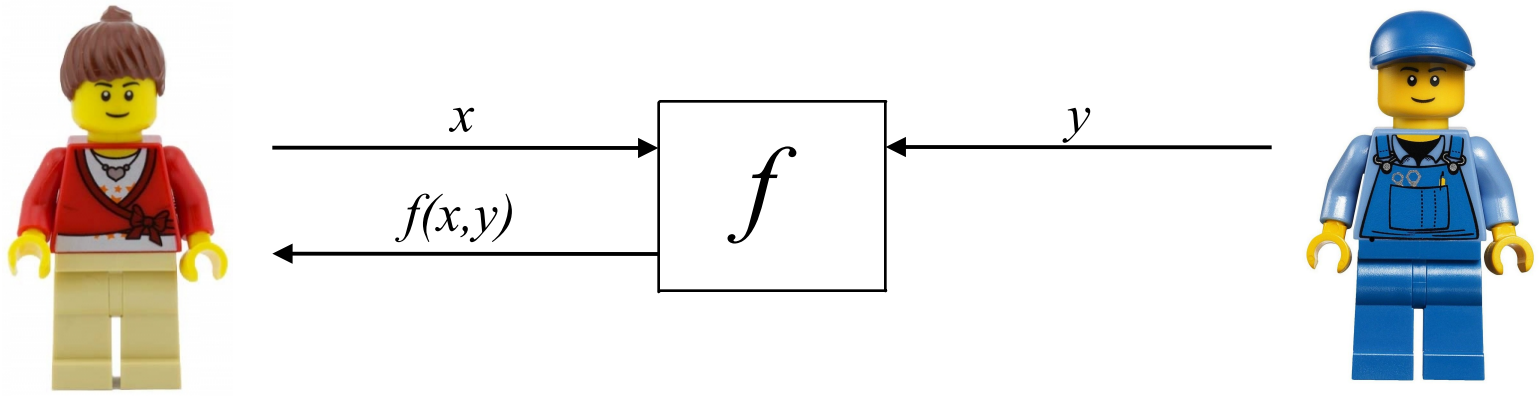
Optimizations of Generic Protocols for Semi-Honest Adversaries

TECHNISCHE UNIVERSITÄT DARMSTADT

Thomas Schneider (TU Darmstadt)

5th Bar-Ilan Winter School on Cryptography, Feb 2015

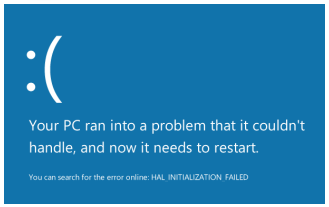
Secure Two-Party Computation



This Lecture: **Semi-Honest (Passive)** Adversaries

Secure Two-Party Computation

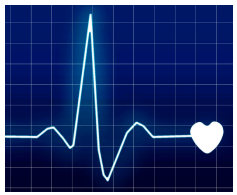
Auctions [NaorPS99], ...



Remote Diagnostics [BrickellPSW07], ...

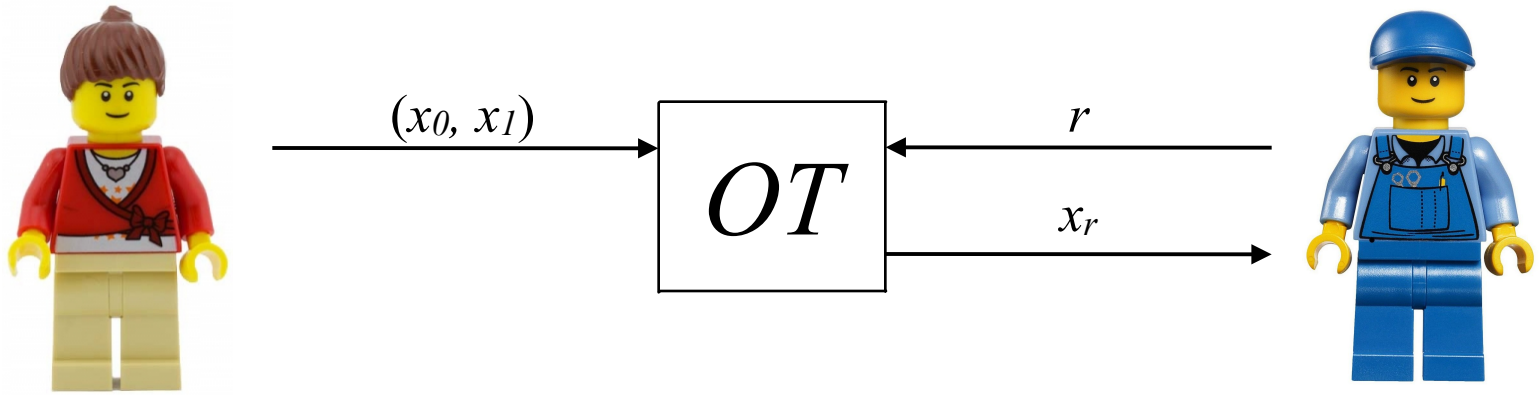
DNA Searching [Troncoso-PastorizaKC07], ...

Biometric Identification [ErkinFGKLT09], ...



Medical Diagnostics [BarniFKLSS09], ...

Oblivious Transfer (OT)



1-out-of-2 OT is an essential building block for secure computation.

How to Measure Efficiency of a Protocol?

- ✓ Runtime (depends on implementation & scenario)

How to Measure Efficiency of a Protocol?

- ✓ Runtime (depends on implementation & scenario)
- ✓ Communication
  - # bits sent (important for networks with low bandwidth)
  - # rounds (important for networks with high latency)

How to Measure Efficiency of a Protocol?

✓ Runtime (depends on implementation & scenario)

✓ Communication

- # bits sent (important for networks with low bandwidth)
- # rounds (important for networks with high latency)

? Computation

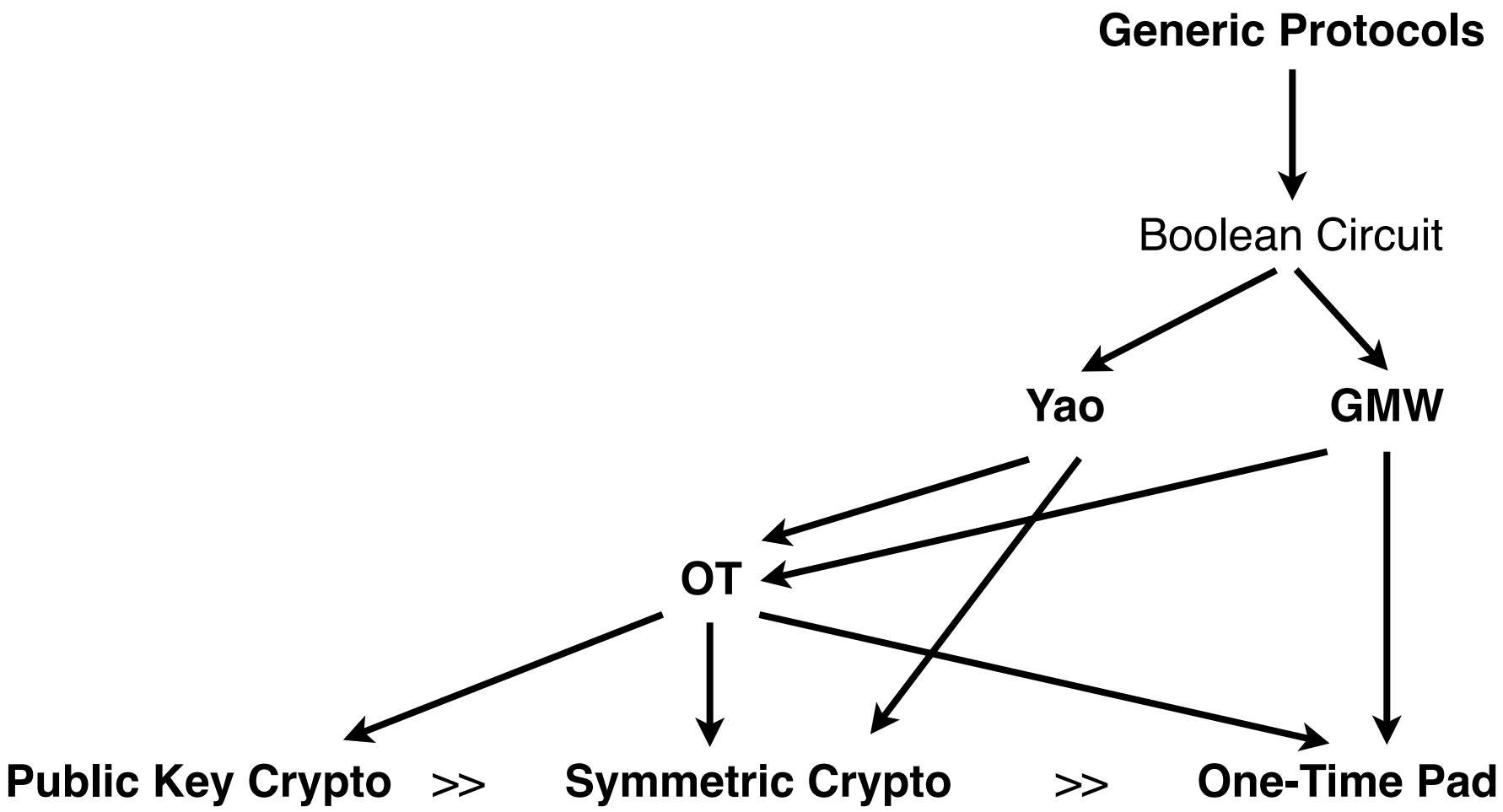
- Usually: count # crypto operations, e.g.,
  - # modular exponentiations
  - # point multiplications
  - # hash function evaluations (SHA)
  - # block cipher evaluations (AES)
  - # One-Time Pad evaluations

- **But also non-cryptographic operations do matter!**

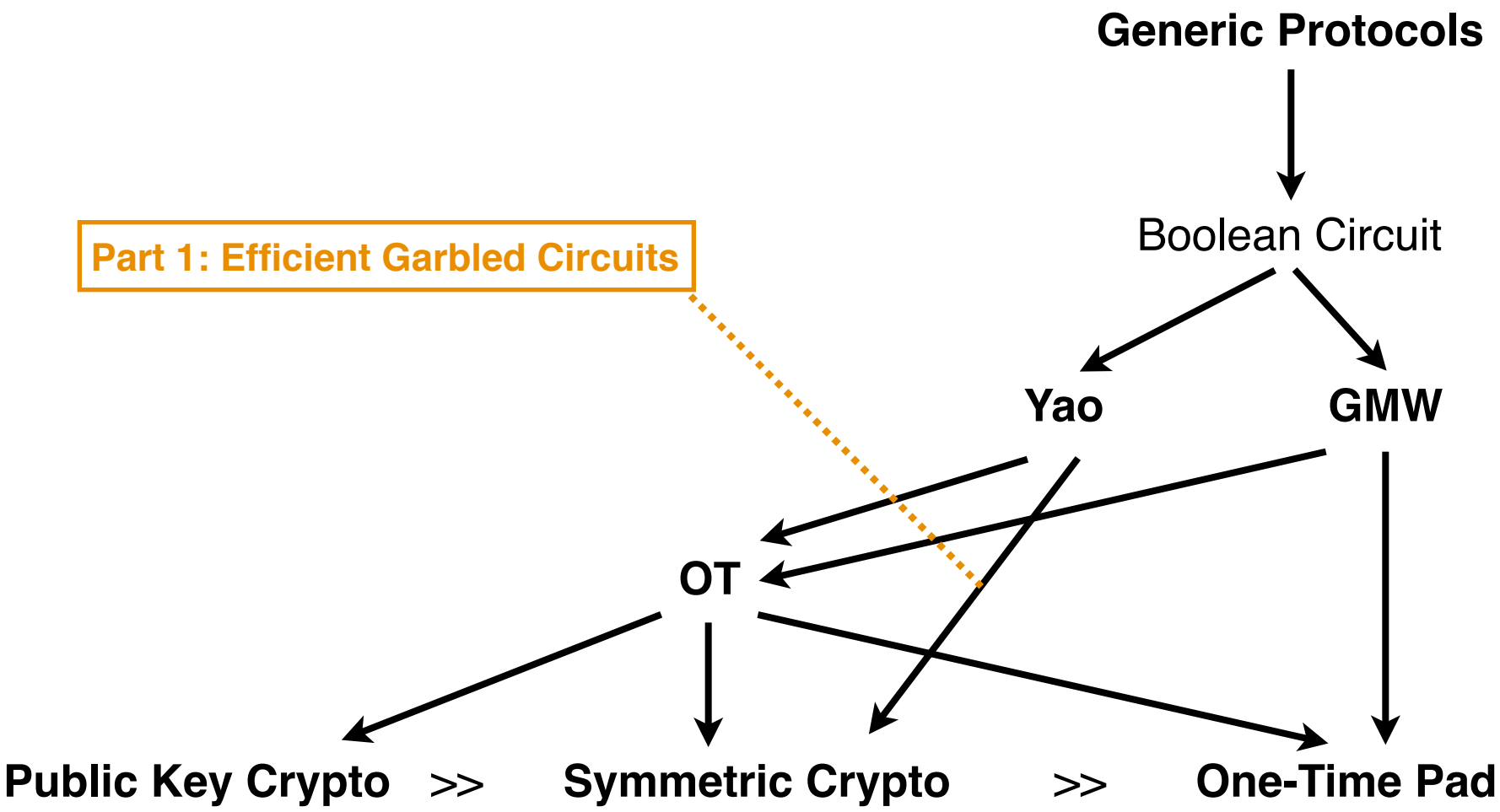
Overview of this Lecture

Public Key Crypto >> Symmetric Crypto >> One-Time Pad

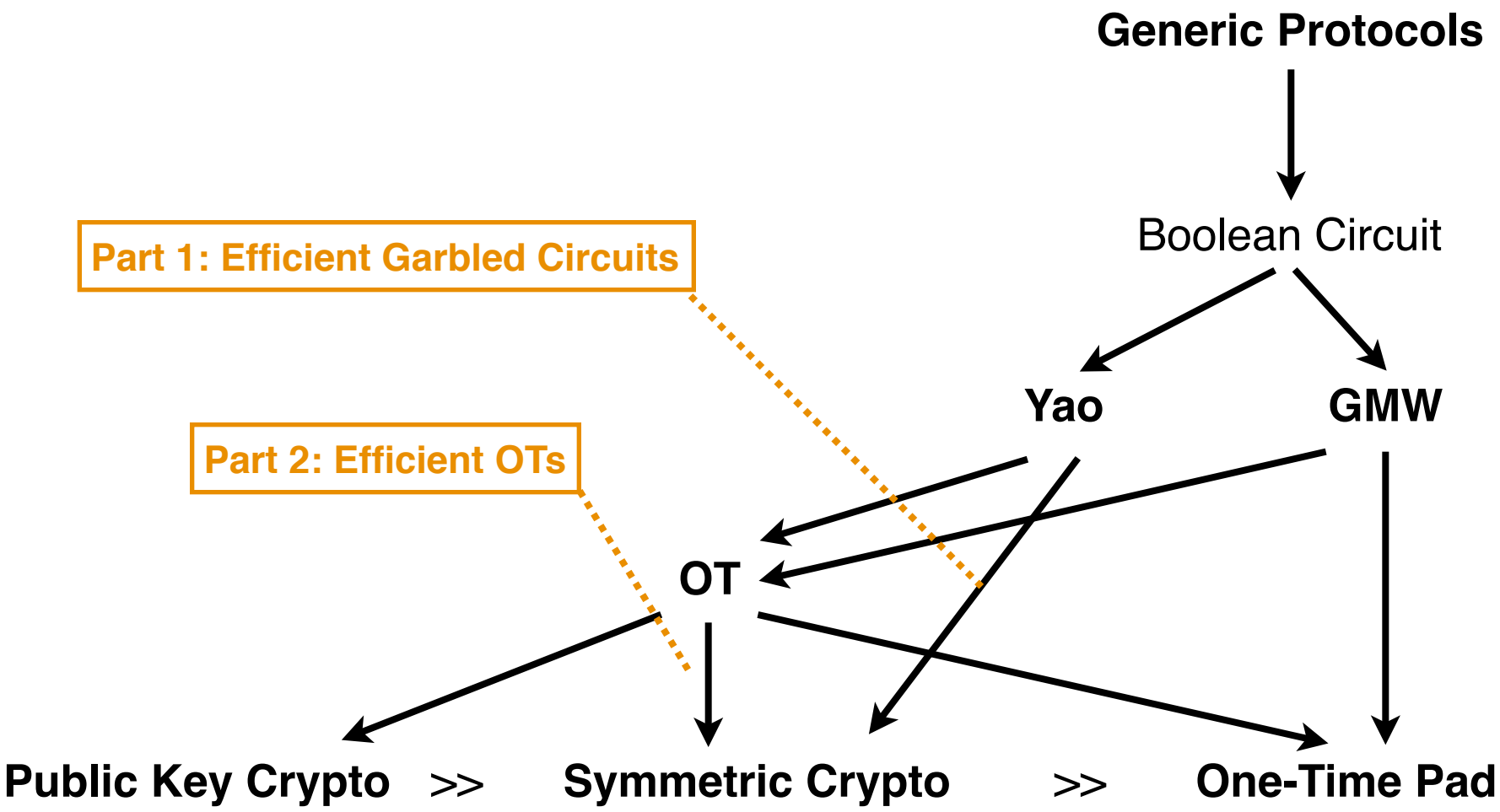
Overview of this Lecture



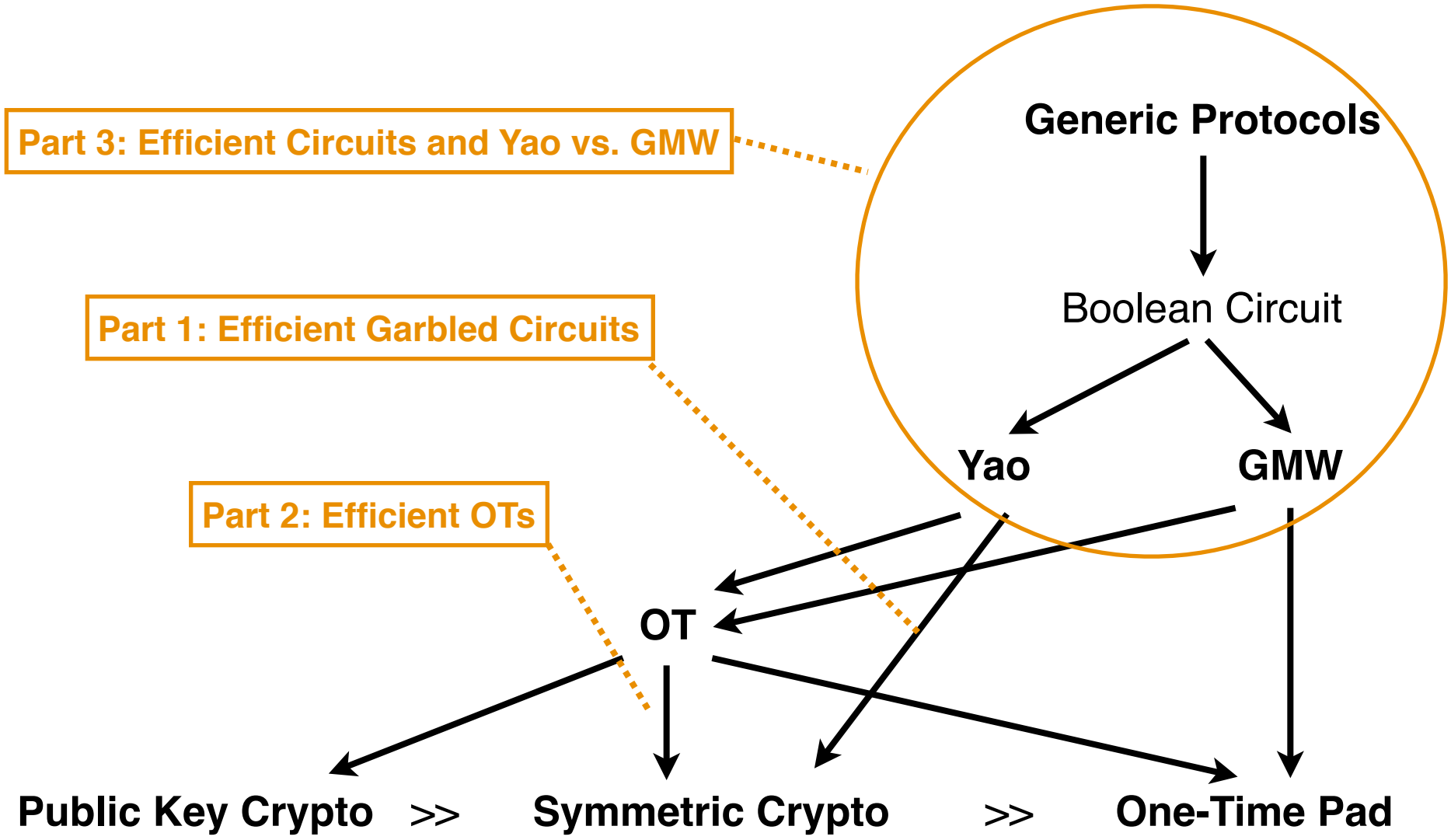
Overview of this Lecture



Overview of this Lecture



Overview of this Lecture



Yao's Garbled Circuits Protocol [Yao86]

$f(\cdot,\cdot)$ e.g., $\mathbf{x} < \mathbf{y}$

private data $\mathbf{x} = x_1, .., x_n$

private data $\mathbf{y} = y_1, .., y_n$

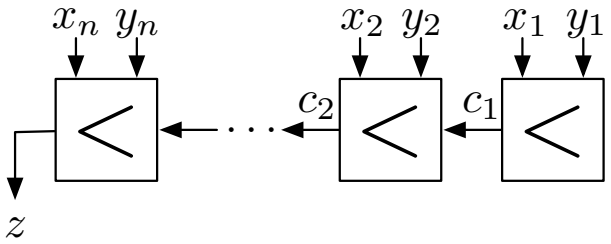
Yao's Garbled Circuits Protocol [Yao86]

$f(\cdot,\cdot)$ e.g., $\mathbf{x} < \mathbf{y}$

private data $\mathbf{x} = x_1, .., x_n$

private data $\mathbf{y} = y_1, .., y_n$

Circuit

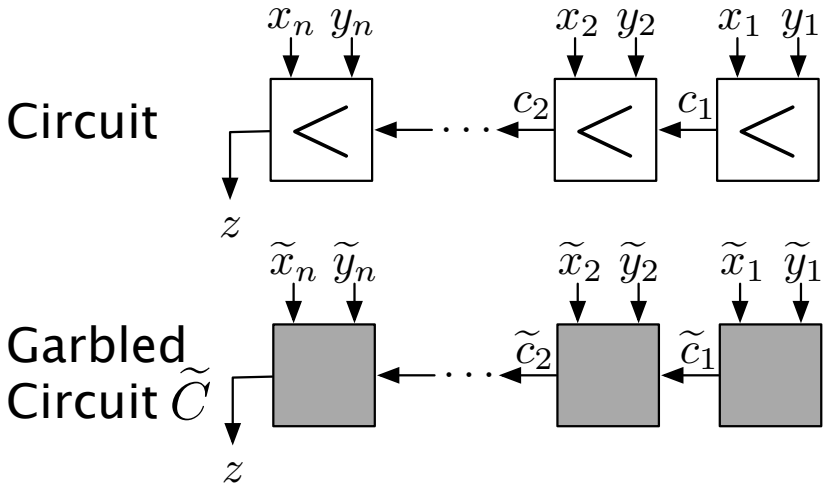


Yao's Garbled Circuits Protocol [Yao86]

private data $\mathbf{x} = x_1, .., x_n$

$f(\cdot, \cdot)$ e.g., $\mathbf{x} < \mathbf{y}$

private data $\mathbf{y} = y_1, .., y_n$

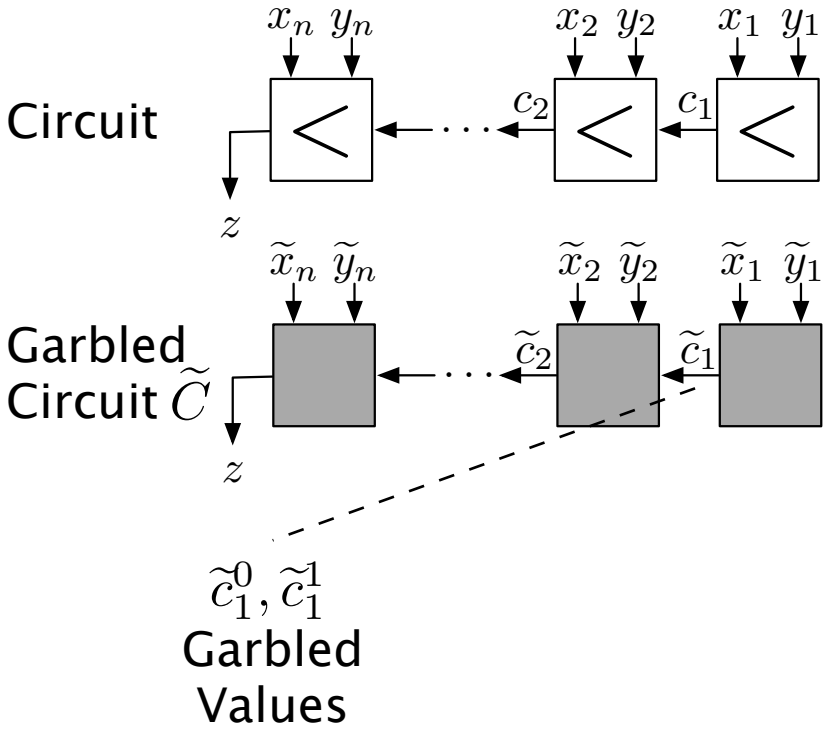


Yao's Garbled Circuits Protocol [Yao86]

$f(\cdot,\cdot)$ e.g., $\mathbf{x} < \mathbf{y}$

private data $\mathbf{x} = x_1, .., x_n$

private data $\mathbf{y} = y_1, .., y_n$

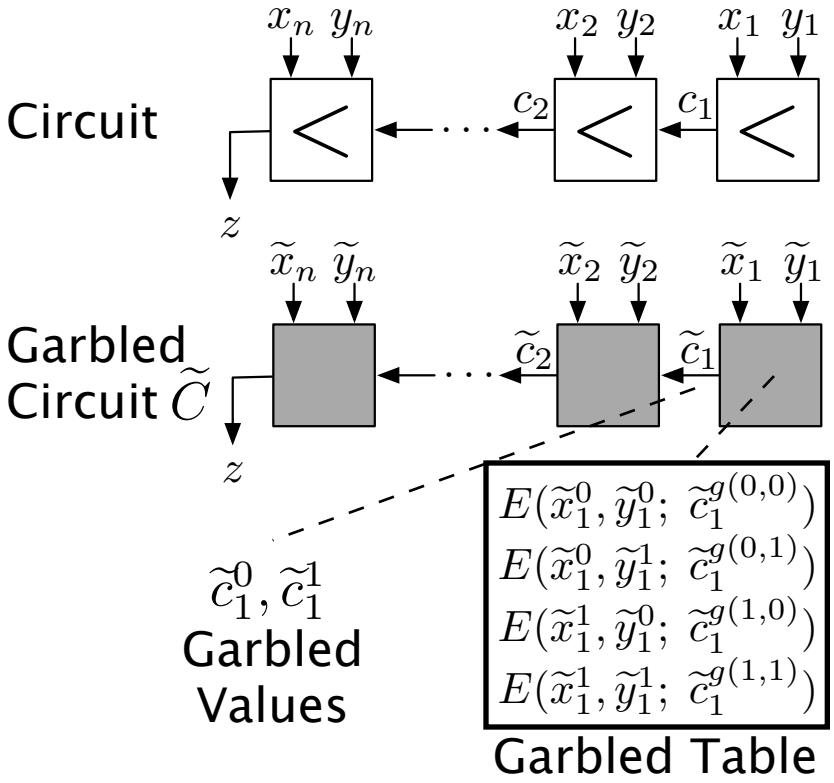


Yao's Garbled Circuits Protocol [Yao86]

$f(\cdot,\cdot)$ e.g., $\mathbf{x} < \mathbf{y}$

private data $\mathbf{x} = x_1, .., x_n$

private data $\mathbf{y} = y_1, .., y_n$

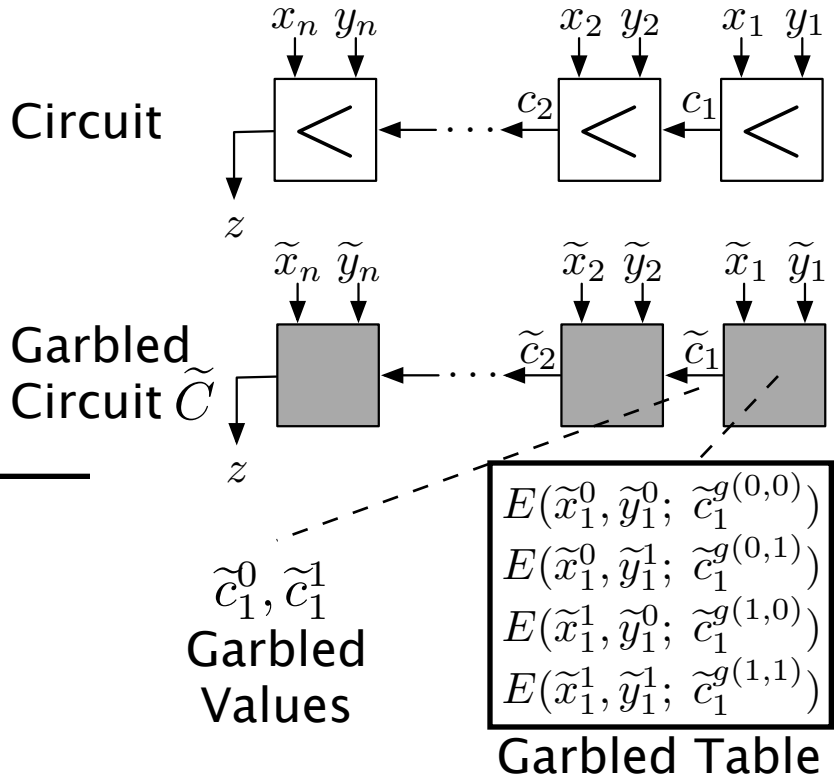


Yao's Garbled Circuits Protocol [Yao86]

$f(\cdot,\cdot)$ e.g., $\mathbf{x} < \mathbf{y}$

private data $\mathbf{x} = x_1, .., x_n$

private data $\mathbf{y} = y_1, .., y_n$

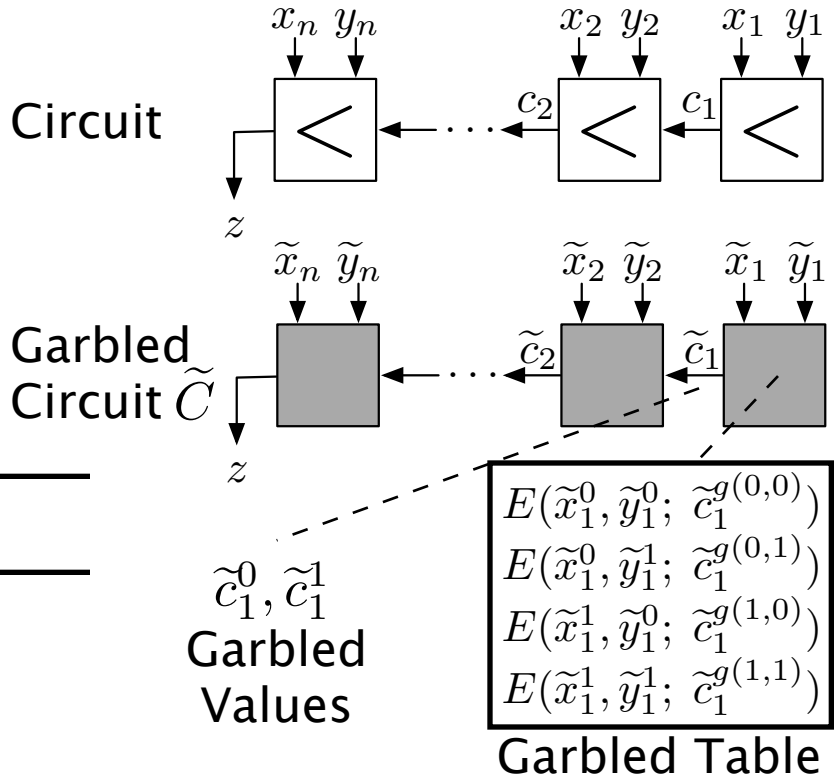


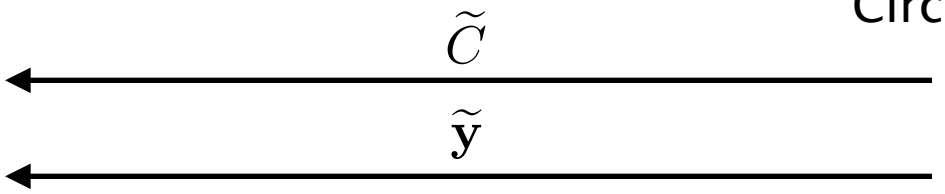
Yao's Garbled Circuits Protocol [Yao86]

$f(\cdot,\cdot)$ e.g., $\mathbf{x} < \mathbf{y}$

private data $\mathbf{x} = x_1, .., x_n$

private data $\mathbf{y} = y_1, .., y_n$



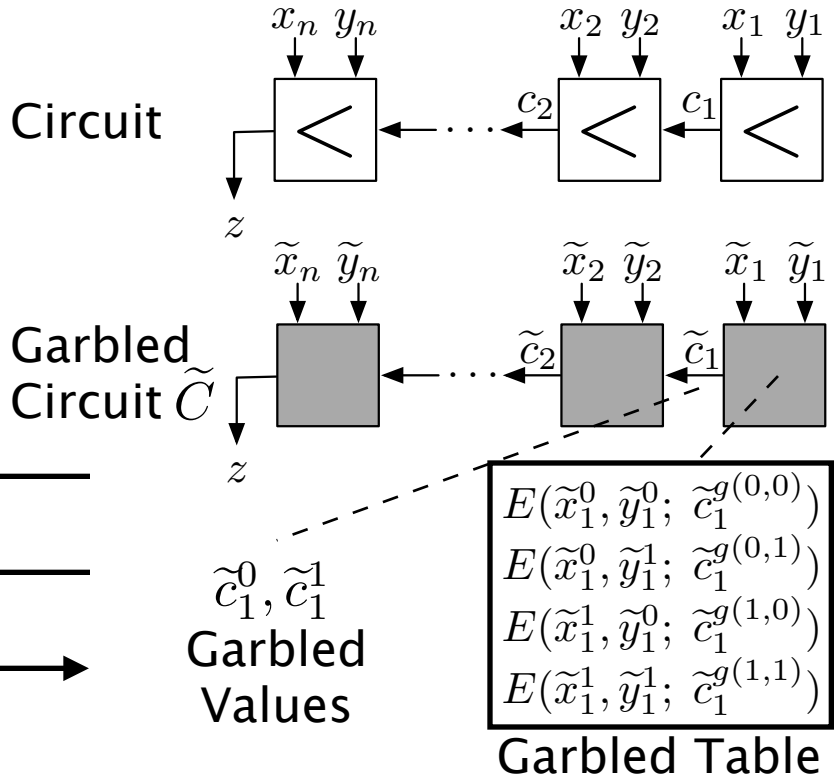


Yao's Garbled Circuits Protocol [Yao86]

$f(\cdot,\cdot)$ e.g., $\mathbf{x} < \mathbf{y}$

private data $\mathbf{x} = x_1, .., x_n$

private data $\mathbf{y} = y_1, .., y_n$



$\widetilde{C}$

$\widetilde{\mathbf{y}}$

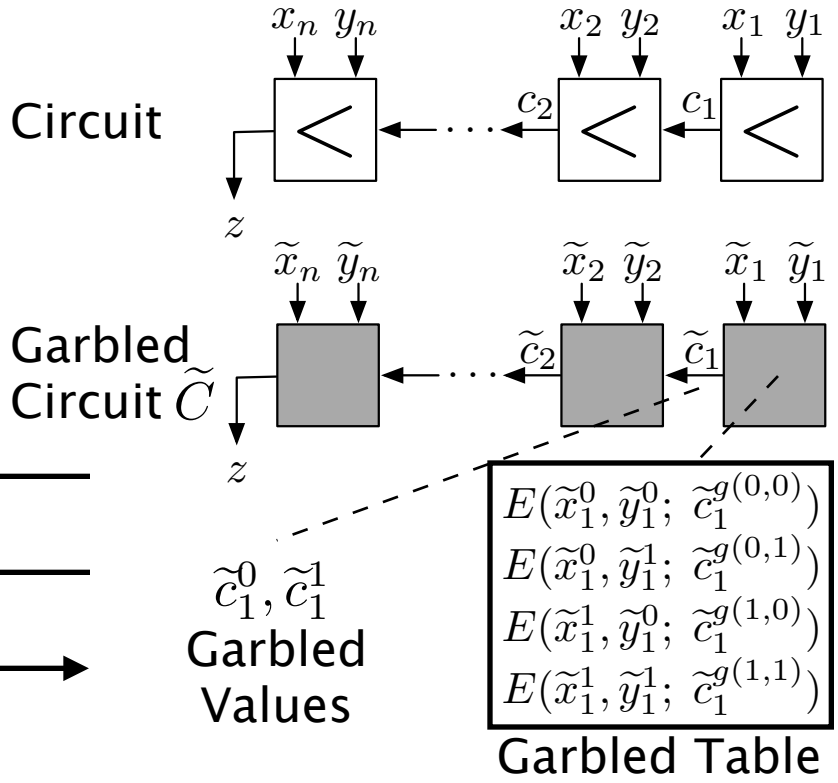
$(\widetilde{\mathbf{x}}; \perp) \leftarrow \mathsf{OT}(\mathbf{x}; (\widetilde{\mathbf{x}}^{\mathbf{0}}, \widetilde{\mathbf{x}}^{\mathbf{1}}))$

Yao's Garbled Circuits Protocol [Yao86]

$f(\cdot,\cdot)$ e.g., $\mathbf{x} < \mathbf{y}$

private data $\mathbf{x} = x_1, .., x_n$

private data $\mathbf{y} = y_1, .., y_n$



$\widetilde{C}$

$\widetilde{\mathbf{y}}$

$(\widetilde{\mathbf{x}}; \perp) \leftarrow \mathsf{OT}(\mathbf{x}; (\widetilde{\mathbf{x}}^{\mathbf{0}}, \widetilde{\mathbf{x}}^{\mathbf{1}}))$

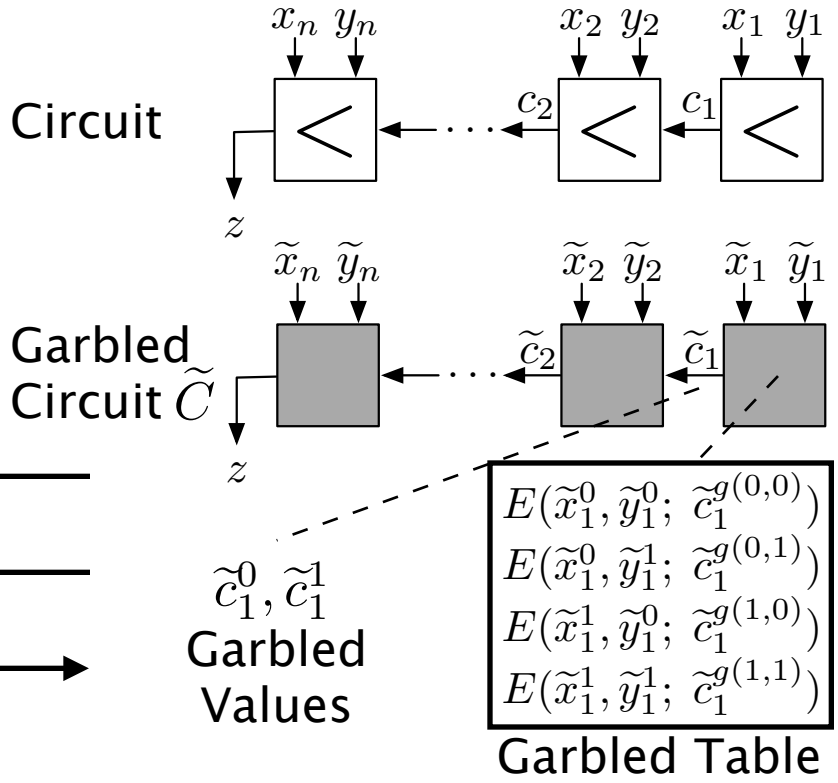
$f(\mathbf{x}, \mathbf{y}) = \widetilde{C}(\widetilde{\mathbf{x}}, \widetilde{\mathbf{y}})$

Yao's Garbled Circuits Protocol [Yao86]

$f(\cdot,\cdot)$ e.g., $\mathbf{x} < \mathbf{y}$

private data $\mathbf{x} = x_1, .., x_n$

private data $\mathbf{y} = y_1, .., y_n$



Part 1: Efficient Garbled Circuits

$\widetilde{C}$

$\widetilde{\mathbf{y}}$

$(\widetilde{\mathbf{x}}; \perp) \leftarrow \mathsf{OT}(\mathbf{x}; (\widetilde{\mathbf{x}}^{\mathbf{0}}, \widetilde{\mathbf{x}}^{\mathbf{1}}))$

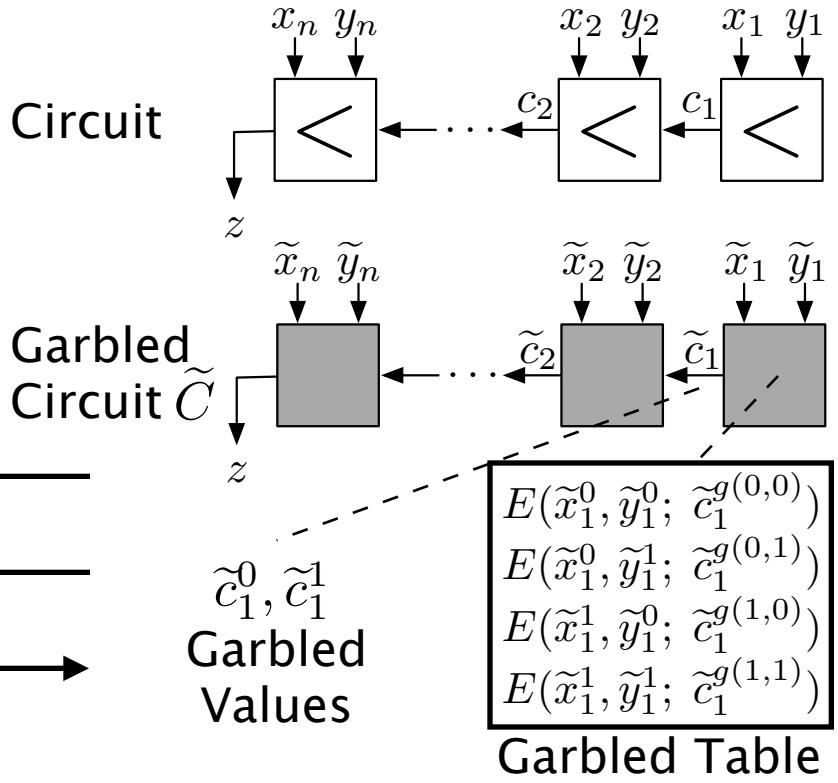
$f(\mathbf{x}, \mathbf{y}) = \widetilde{C}(\widetilde{\mathbf{x}}, \widetilde{\mathbf{y}})$

Yao's Garbled Circuits Protocol [Yao86]

$f(\cdot,\cdot)$ e.g., $\mathbf{x} < \mathbf{y}$

private data $\mathbf{x} = x_1, .., x_n$

private data $\mathbf{y} = y_1, .., y_n$



Part 1: Efficient Garbled Circuits

$\widetilde{C}$

$\widetilde{\mathbf{y}}$

$(\widetilde{\mathbf{x}}; \perp) \leftarrow \mathsf{OT}(\mathbf{x}; (\widetilde{\mathbf{x}}^{\mathbf{0}}, \widetilde{\mathbf{x}}^{\mathbf{1}}))$

$f(\mathbf{x}, \mathbf{y}) = \widetilde{C}(\widetilde{\mathbf{x}}, \widetilde{\mathbf{y}})$

Part 2: Efficient OT

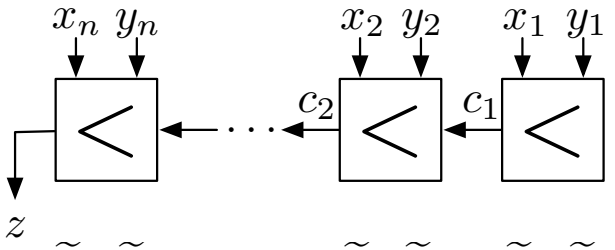
Yao's Garbled Circuits Protocol [Yao86]

$f(\cdot,\cdot)$ e.g., $\mathbf{x} < \mathbf{y}$

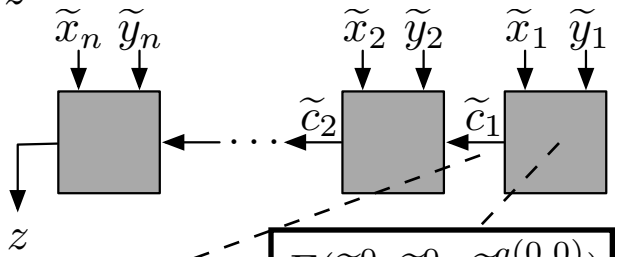
private data $\mathbf{x} = x_1, .., x_n$

private data $\mathbf{y} = y_1, .., y_n$

Part 3: Efficient Circuits ..... Circuit



Part 1: Efficient Garbled Circuits ..... Garbled Circuit $\widetilde{C}$



$\widetilde{C}$

$\widetilde{\mathbf{y}}$

$(\widetilde{\mathbf{x}}; \perp) \leftarrow \mathsf{OT}(\mathbf{x}; (\widetilde{\mathbf{x}}^{\mathbf{0}}, \widetilde{\mathbf{x}}^{\mathbf{1}}))$

$f(\mathbf{x}, \mathbf{y}) = \widetilde{C}(\widetilde{\mathbf{x}}, \widetilde{\mathbf{y}})$

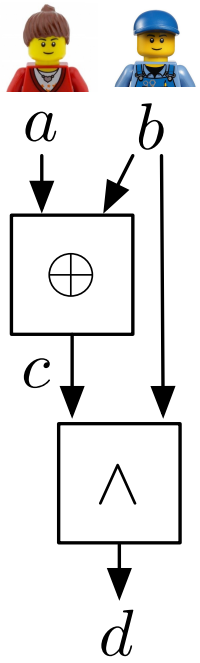
Part 2: Efficient OT

$\widetilde{c}_1^0, \widetilde{c}_1^1$ Garbled Values

$E(\widetilde{x}_1^0, \widetilde{y}_1^0; \widetilde{c}_1^{g(0,0)})$
$E(\widetilde{x}_1^0, \widetilde{y}_1^1; \widetilde{c}_1^{g(0,1)})$
$E(\widetilde{x}_1^1, \widetilde{y}_1^0; \widetilde{c}_1^{g(1,0)})$
$E(\widetilde{x}_1^1, \widetilde{y}_1^1; \widetilde{c}_1^{g(1,1)})$

Garbled Table

The GMW Protocol [GMW87]

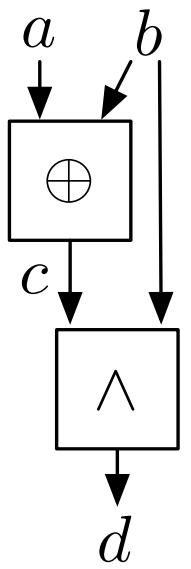


The GMW Protocol [GMW87]

Secret share inputs:

$a = a_1 \oplus a_2$

$b = b_1 \oplus b_2$

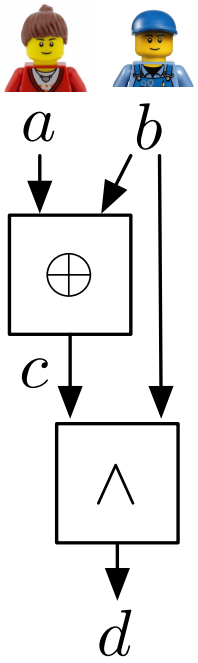


The GMW Protocol [GMW87]

Secret share inputs: $a = a_1 \oplus a_2$

$b = b_1 \oplus b_2$

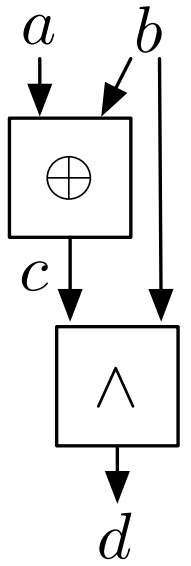
Non-Interactive XOR gates: $c_1 = a_1 \oplus b_1$; $c_2 = a_2 \oplus b_2$



The GMW Protocol [GMW87]

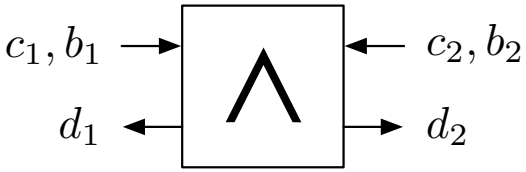
Secret share inputs: $a = a_1 \oplus a_2$

$b = b_1 \oplus b_2$



Non-Interactive XOR gates: $c_1 = a_1 \oplus b_1$; $c_2 = a_2 \oplus b_2$

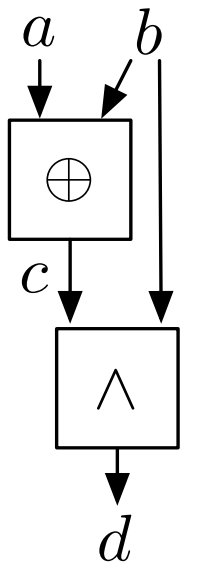
Interactive AND gates:



The GMW Protocol [GMW87]

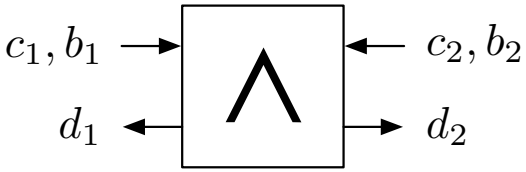
Secret share inputs: $a = a_1 \oplus a_2$

$b = b_1 \oplus b_2$



Non-Interactive XOR gates: $c_1 = a_1 \oplus b_1$; $c_2 = a_2 \oplus b_2$

Interactive AND gates:

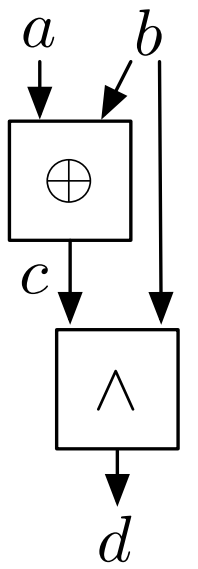


Recombine outputs: $d = d_1 \oplus d_2$

The GMW Protocol [GMW87]

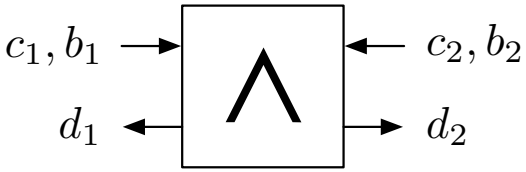
Secret share inputs: $a = a_1 \oplus a_2$

$b = b_1 \oplus b_2$



Non-Interactive XOR gates: $c_1 = a_1 \oplus b_1$; $c_2 = a_2 \oplus b_2$

Interactive AND gates:



Recombine outputs: $d = d_1 \oplus d_2$

Part 3: Efficient Circuits

Evaluating ANDs via Multiplication Triples [Beaver91]

Evaluating ANDs via Multiplication Triples [Beaver91]

The Aim: Generate a multiplication triple $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$

Evaluating ANDs via Multiplication Triples [Beaver91]

The Aim: Generate a multiplication triple $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$

- P_1's output: a_1, b_1, c_1

Evaluating ANDs via Multiplication Triples [Beaver91]

The Aim: Generate a multiplication triple $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$

- P_1's output: a_1, b_1, c_1
- P_2's output: a_2, b_2, c_2

Evaluating ANDs via Multiplication Triples [Beaver91]

The Aim: Generate a multiplication triple $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$

- P_1's output: a_1, b_1, c_1
- P_2's output: a_2, b_2, c_2
- Property: $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$

Evaluating ANDs via Multiplication Triples [Beaver91]

The Aim: Generate a multiplication triple $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$

- P_1's output: a_1, b_1, c_1
- P_2's output: a_2, b_2, c_2
- Property: $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$
  - Observe that $c_1 \oplus c_2 = a_1 b_1 \oplus a_2 b_1 \oplus a_1 b_2 \oplus a_2 b_2$

Evaluating ANDs via Multiplication Triples [Beaver91]

The Aim: Generate a multiplication triple $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$

- P_1's output: a_1, b_1, c_1
- P_2's output: a_2, b_2, c_2
- Property: $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$
  - Observe that $c_1 \oplus c_2 = a_1 b_1 \oplus a_2 b_1 \oplus a_1 b_2 \oplus a_2 b_2$

The Protocol:

Evaluating ANDs via Multiplication Triples [Beaver91]

The Aim: Generate a multiplication triple $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$

- P_1's output: a_1, b_1, c_1
- P_2's output: a_2, b_2, c_2
- Property: $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$
  - Observe that $c_1 \oplus c_2 = a_1 b_1 \oplus a_2 b_1 \oplus a_1 b_2 \oplus a_2 b_2$

The Protocol:

1. P_1: choose $m_0, m_1 \in_R \{0,1\}$; P_2: choose $a_2 \in_R \{0,1\}$

Evaluating ANDs via Multiplication Triples [Beaver91]

The Aim: Generate a multiplication triple $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$

- P_1's output: a_1, b_1, c_1
- P_2's output: a_2, b_2, c_2
- Property: $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$
  - Observe that $c_1 \oplus c_2 = a_1 b_1 \oplus a_2 b_1 \oplus a_1 b_2 \oplus a_2 b_2$

The Protocol:

1. P_1: choose $m_0, m_1 \in_R \{0,1\}$; P_2: choose $a_2 \in_R \{0,1\}$
2. P_1 and P_2 run OT: P_1 inputs (m_0, m_1), P_2 inputs a_2 and gets $u_2 = m_{a2}$

Evaluating ANDs via Multiplication Triples [Beaver91]

The Aim: Generate a multiplication triple $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$

- P_1's output: a_1, b_1, c_1
- P_2's output: a_2, b_2, c_2
- Property: $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$
  - Observe that $c_1 \oplus c_2 = a_1 b_1 \oplus a_2 b_1 \oplus a_1 b_2 \oplus a_2 b_2$

The Protocol:

1. P_1: choose $m_0, m_1 \in_R \{0,1\}$; P_2: choose $a_2 \in_R \{0,1\}$
2. P_1 and P_2 run OT: P_1 inputs (m_0, m_1), P_2 inputs a_2 and gets $u_2 = m_{a2}$
3. P_1 sets $b_1 = m_0 \oplus m_1$; $v_1 = m_0$

Evaluating ANDs via Multiplication Triples [Beaver91]

The Aim: Generate a multiplication triple $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$

- P_1's output: a_1, b_1, c_1
- P_2's output: a_2, b_2, c_2
- Property: $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$
  - Observe that $c_1 \oplus c_2 = a_1 b_1 \oplus a_2 b_1 \oplus a_1 b_2 \oplus a_2 b_2$

The Protocol:

1. P_1: choose $m_0, m_1 \in_R \{0,1\}$; P_2: choose $a_2 \in_R \{0,1\}$
2. P_1 and P_2 run OT: P_1 inputs (m_0, m_1), P_2 inputs a_2 and gets $u_2 = m_{a2}$
3. P_1 sets $b_1 = m_0 \oplus m_1$; $v_1 = m_0$
   - Observe: $v_1 \oplus u_2 = m_0 \oplus m_{a2}$

Evaluating ANDs via Multiplication Triples [Beaver91]

The Aim: Generate a multiplication triple $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$

- P_1's output: a_1, b_1, c_1
- P_2's output: a_2, b_2, c_2
- Property: $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$
  - Observe that $c_1 \oplus c_2 = a_1 b_1 \oplus a_2 b_1 \oplus a_1 b_2 \oplus a_2 b_2$

The Protocol:

1. P_1: choose $m_0, m_1 \in_R \{0,1\}$; P_2: choose $a_2 \in_R \{0,1\}$
2. P_1 and P_2 run OT: P_1 inputs (m_0, m_1), P_2 inputs a_2 and gets $u_2 = m_{a2}$
3. P_1 sets $b_1 = m_0 \oplus m_1$; $v_1 = m_0$
   - Observe: $v_1 \oplus u_2 = m_0 \oplus m_{a2}$
   - If $a_2 = 0$ then $v_1 \oplus u_2 = m_0 \oplus m_0 = 0 = a_2 b_1$

Evaluating ANDs via Multiplication Triples [Beaver91]

The Aim: Generate a multiplication triple $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$

- P_1's output: a_1, b_1, c_1
- P_2's output: a_2, b_2, c_2
- Property: $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$
  - Observe that $c_1 \oplus c_2 = a_1 b_1 \oplus a_2 b_1 \oplus a_1 b_2 \oplus a_2 b_2$

The Protocol:

1. P_1: choose $m_0, m_1 \in_R \{0,1\}$; P_2: choose $a_2 \in_R \{0,1\}$
2. P_1 and P_2 run OT: P_1 inputs (m_0, m_1), P_2 inputs a_2 and gets $u_2 = m_{a2}$
3. P_1 sets $b_1 = m_0 \oplus m_1$; $v_1 = m_0$
   - Observe: $v_1 \oplus u_2 = m_0 \oplus m_{a2}$
   - If $a_2 = 0$ then $v_1 \oplus u_2 = m_0 \oplus m_0 = 0 = a_2 b_1$
   - If $a_2 = 1$ then $v_1 \oplus u_2 = m_0 \oplus m_1 = b_1 = a_2 b_1$

Evaluating ANDs via Multiplication Triples [Beaver91]

The Aim: Generate a multiplication triple $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$

- P_1's output: a_1, b_1, c_1
- P_2's output: a_2, b_2, c_2
- Property: $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$
  - Observe that $c_1 \oplus c_2 = a_1 b_1 \oplus a_2 b_1 \oplus a_1 b_2 \oplus a_2 b_2$

The Protocol:

1. P_1: choose $m_0, m_1 \in_R \{0,1\}$; P_2: choose $a_2 \in_R \{0,1\}$
2. P_1 and P_2 run OT: P_1 inputs (m_0, m_1), P_2 inputs a_2 and gets $u_2 = m_{a2}$
3. P_1 sets $b_1 = m_0 \oplus m_1$; $v_1 = m_0$
   - Observe: $v_1 \oplus u_2 = m_0 \oplus m_{a2}$
   - If $a_2=0$ then $v_1 \oplus u_2 = m_0 \oplus m_0 = 0 = a_2 b_1$
   - If $a_2=1$ then $v_1 \oplus u_2 = m_0 \oplus m_1 = b_1 = a_2 b_1$
4. P_1 and P_2 repeat steps 1-3 with reversed roles to obtain (a_1, u_1); (b_2, v_2)

Evaluating ANDs via Multiplication Triples [Beaver91]

The Aim: Generate a multiplication triple $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$

- P_1's output: a_1, b_1, c_1
- P_2's output: a_2, b_2, c_2
- Property: $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$
  - Observe that $c_1 \oplus c_2 = a_1 b_1 \oplus a_2 b_1 \oplus a_1 b_2 \oplus a_2 b_2$

The Protocol:

1. P_1: choose $m_0, m_1 \in_R \{0,1\}$; P_2: choose $a_2 \in_R \{0,1\}$
2. P_1 and P_2 run OT: P_1 inputs (m_0, m_1), P_2 inputs a_2 and gets $u_2 = m_{a2}$
3. P_1 sets $b_1 = m_0 \oplus m_1$; $v_1 = m_0$
   - Observe: $v_1 \oplus u_2 = m_0 \oplus m_{a2}$
   - If $a_2=0$ then $v_1 \oplus u_2 = m_0 \oplus m_0 = 0 = a_2 b_1$
   - If $a_2=1$ then $v_1 \oplus u_2 = m_0 \oplus m_1 = b_1 = a_2 b_1$
4. P_1 and P_2 repeat steps 1-3 with reversed roles to obtain (a_1, u_1); (b_2, v_2)
5. P_i sets $c_i = (a_i b_i) \oplus u_i \oplus v_i$

Evaluating ANDs via Multiplication Triples [Beaver91]

The Aim: Generate a multiplication triple $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$

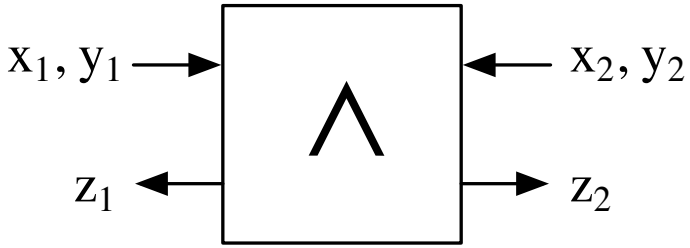
- P_1's output: a_1, b_1, c_1
- P_2's output: a_2, b_2, c_2
- Property: $(a_1 \oplus a_2)\ (b_1 \oplus b_2) = c_1 \oplus c_2$
  - Observe that $c_1 \oplus c_2 = a_1 b_1 \oplus a_2 b_1 \oplus a_1 b_2 \oplus a_2 b_2$

The Protocol:

1. P_1: choose $m_0, m_1 \in_R \{0,1\}$; P_2: choose $a_2 \in_R \{0,1\}$
2. P_1 and P_2 run OT: P_1 inputs (m_0, m_1), P_2 inputs a_2 and gets $u_2 = m_{a2}$
3. P_1 sets $b_1 = m_0 \oplus m_1$; $v_1 = m_0$
   - Observe: $v_1 \oplus u_2 = m_0 \oplus m_{a2}$
   - If $a_2=0$ then $v_1 \oplus u_2 = m_0 \oplus m_0 = 0 = a_2 b_1$
   - If $a_2=1$ then $v_1 \oplus u_2 = m_0 \oplus m_1 = b_1 = a_2 b_1$
4. P_1 and P_2 repeat steps 1-3 with reversed roles to obtain (a_1, u_1); (b_2, v_2)
5. P_i sets $c_i = (a_i b_i) \oplus u_i \oplus v_i$

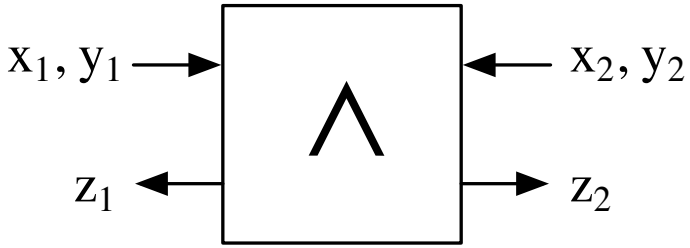
Observe: $c_1 \oplus c_2 = a_1 b_1 \oplus u_1 \oplus v_1 \oplus a_2 b_2 \oplus u_2 \oplus v_2 = a_1 b_1 \oplus a_2 b_1 \oplus a_1 b_2 \oplus a_2 b_2$

Evaluating ANDs via Multiplication Triples [Beaver91]



Evaluating ANDs via Multiplication Triples [Beaver91]

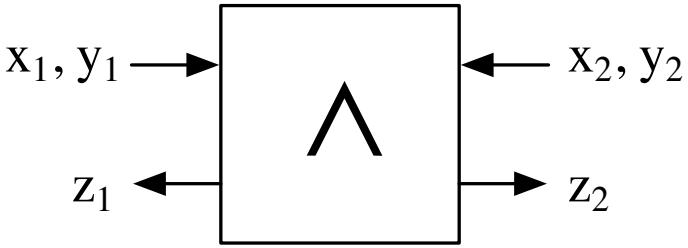
The aim: Compute AND using multiplication triple



Evaluating ANDs via Multiplication Triples [Beaver91]

The aim: Compute AND using multiplication triple

Given: a_1,b_1,c_1 and a_2,b_2,c_2 such that $c_1 \oplus c_2 = a_1b_1 \oplus a_2b_1 \oplus a_1b_2 \oplus a_2b_2$

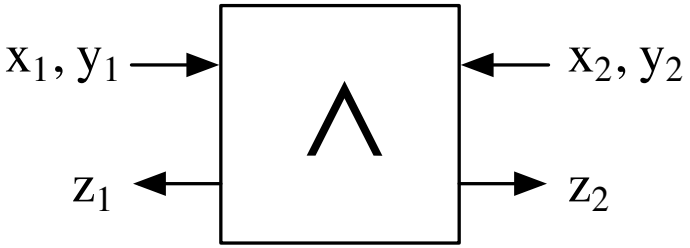


Evaluating ANDs via Multiplication Triples [Beaver91]

The aim: Compute AND using multiplication triple

Given: a_1,b_1,c_1 and a_2,b_2,c_2 such that $c_1 \oplus c_2 = a_1b_1 \oplus a_2b_1 \oplus a_1b_2 \oplus a_2b_2$

P_1 sends P_2: $d_1 = x_1 \oplus a_1$; $e_1 = y_1 \oplus b_1$



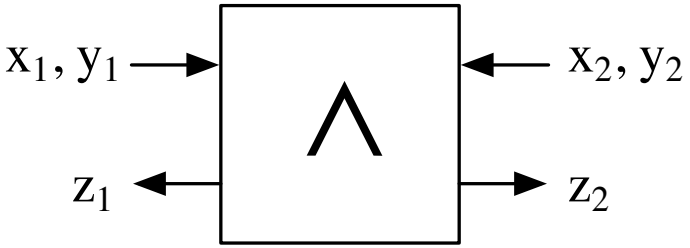
Evaluating ANDs via Multiplication Triples [Beaver91]

The aim: Compute AND using multiplication triple

Given: a_1,b_1,c_1 and a_2,b_2,c_2 such that $c_1 \oplus c_2 = a_1b_1 \oplus a_2b_1 \oplus a_1b_2 \oplus a_2b_2$

P_1 sends P_2: $d_1=x_1 \oplus a_1$; $e_1=y_1 \oplus b_1$

P_2 sends P_1: $d_2=x_2 \oplus a_2$; $e_2=y_2 \oplus b_2$

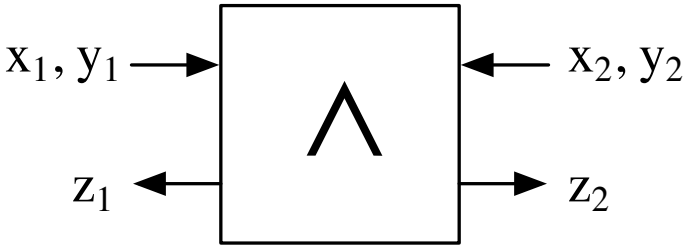


Evaluating ANDs via Multiplication Triples [Beaver91]

The aim: Compute AND using multiplication triple

Given: a_1,b_1,c_1 and a_2,b_2,c_2 such that $c_1 \oplus c_2 = a_1b_1 \oplus a_2b_1 \oplus a_1b_2 \oplus a_2b_2$

P_1 sends P_2: $d_1=x_1 \oplus a_1$; $e_1=y_1 \oplus b_1$
P_2 sends P_1: $d_2=x_2 \oplus a_2$; $e_2=y_2 \oplus b_2$

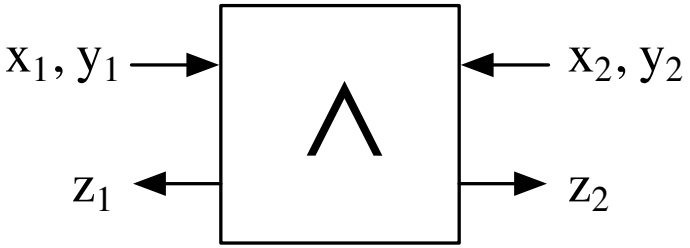


P_1 and P_2 locally compute: $d=d_1 \oplus d_2$; $e=e_1 \oplus e_2$

Evaluating ANDs via Multiplication Triples [Beaver91]

The aim: Compute AND using multiplication triple

Given: a_1,b_1,c_1 and a_2,b_2,c_2 such that $c_1 \oplus c_2 = a_1b_1 \oplus a_2b_1 \oplus a_1b_2 \oplus a_2b_2$



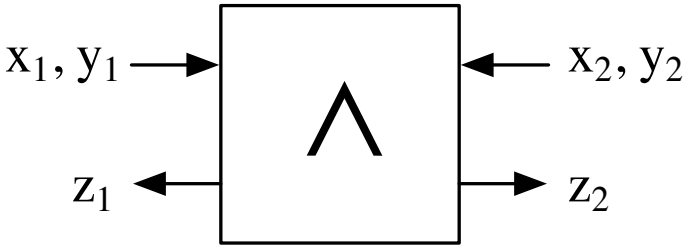
P_1 sends P_2: $d_1=x_1 \oplus a_1$; $e_1=y_1 \oplus b_1$
P_2 sends P_1: $d_2=x_2 \oplus a_2$; $e_2=y_2 \oplus b_2$

P_1 and P_2 locally compute: $d=d_1 \oplus d_2$; $e=e_1 \oplus e_2$
P_1 outputs: $z_1 = db_1 \oplus ea_1 \oplus c_1 \oplus de$

Evaluating ANDs via Multiplication Triples [Beaver91]

The aim: Compute AND using multiplication triple

Given: a_1,b_1,c_1 and a_2,b_2,c_2 such that $c_1 \oplus c_2 = a_1b_1 \oplus a_2b_1 \oplus a_1b_2 \oplus a_2b_2$



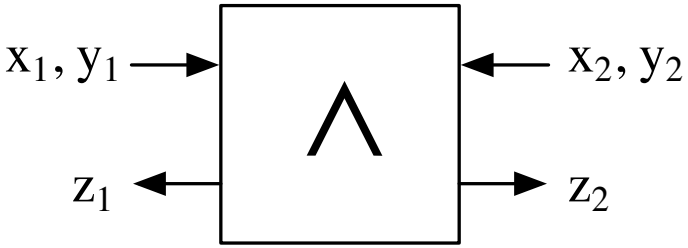
P_1 sends P_2: $d_1=x_1 \oplus a_1$; $e_1=y_1 \oplus b_1$
P_2 sends P_1: $d_2=x_2 \oplus a_2$; $e_2=y_2 \oplus b_2$

P_1 and P_2 locally compute: $d=d_1 \oplus d_2$; $e=e_1 \oplus e_2$
P_1 outputs: $z_1 = db_1 \oplus ea_1 \oplus c_1 \oplus de$
P_2 outputs: $z_2 = db_2 \oplus ea_2 \oplus c_2$

Evaluating ANDs via Multiplication Triples [Beaver91]

The aim: Compute AND using multiplication triple

Given: a_1,b_1,c_1 and a_2,b_2,c_2 such that $c_1 \oplus c_2 = a_1b_1 \oplus a_2b_1 \oplus a_1b_2 \oplus a_2b_2$

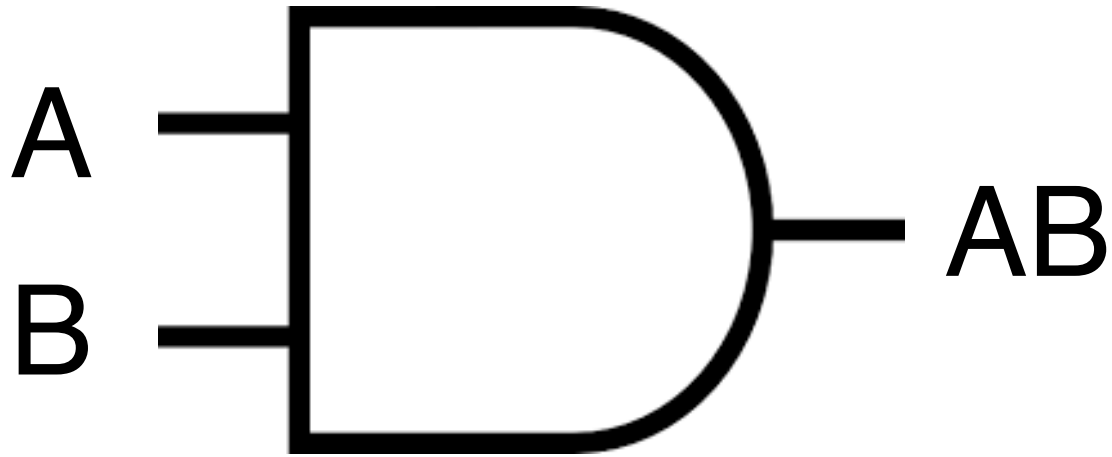


P_1 sends P_2: $d_1=x_1 \oplus a_1$; $e_1=y_1 \oplus b_1$
P_2 sends P_1: $d_2=x_2 \oplus a_2$; $e_2=y_2 \oplus b_2$

P_1 and P_2 locally compute: $d=d_1 \oplus d_2$; $e=e_1 \oplus e_2$
P_1 outputs: $z_1 = db_1 \oplus ea_1 \oplus c_1 \oplus de$
P_2 outputs: $z_2 = db_2 \oplus ea_2 \oplus c_2$

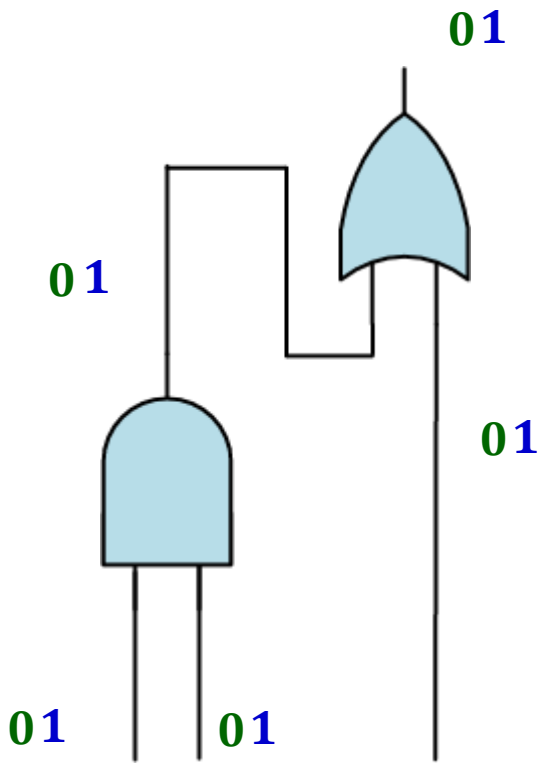
On the board: it holds $z_1 \oplus z_2 = (x_1 \oplus x_2)(y_1 \oplus y_2)$

Part 1: Efficient Garbled Circuits



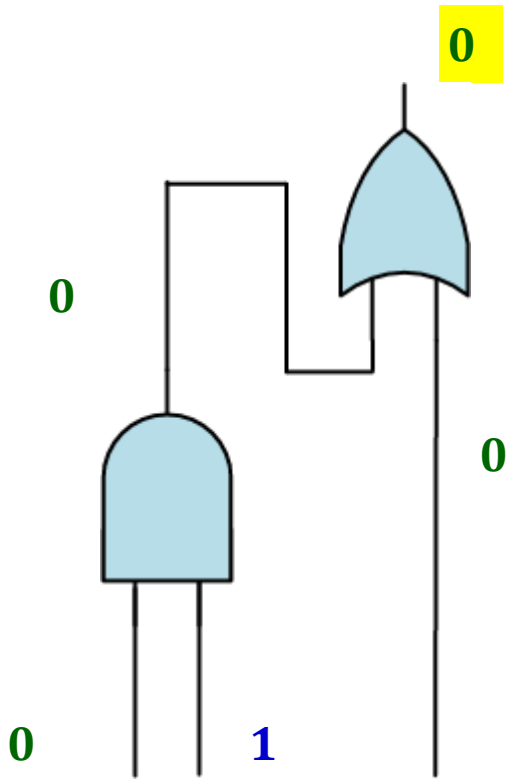
Garbled Circuits [Yao86]

Conventional circuit



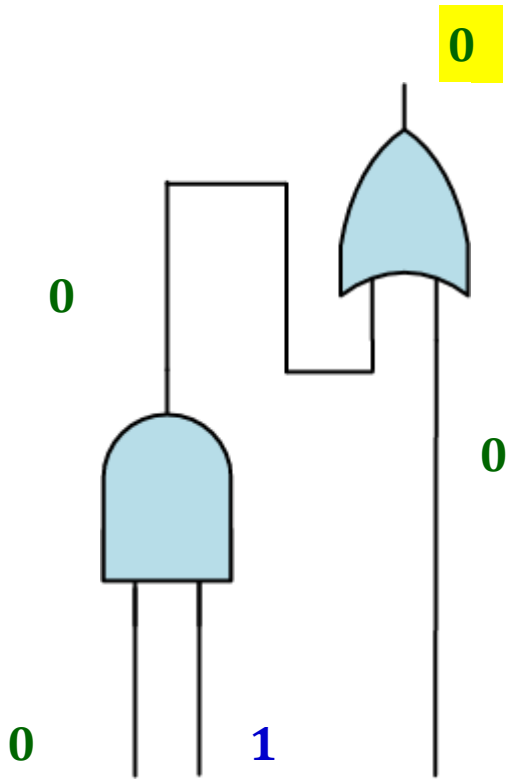
Garbled Circuits [Yao86]

Conventional circuit

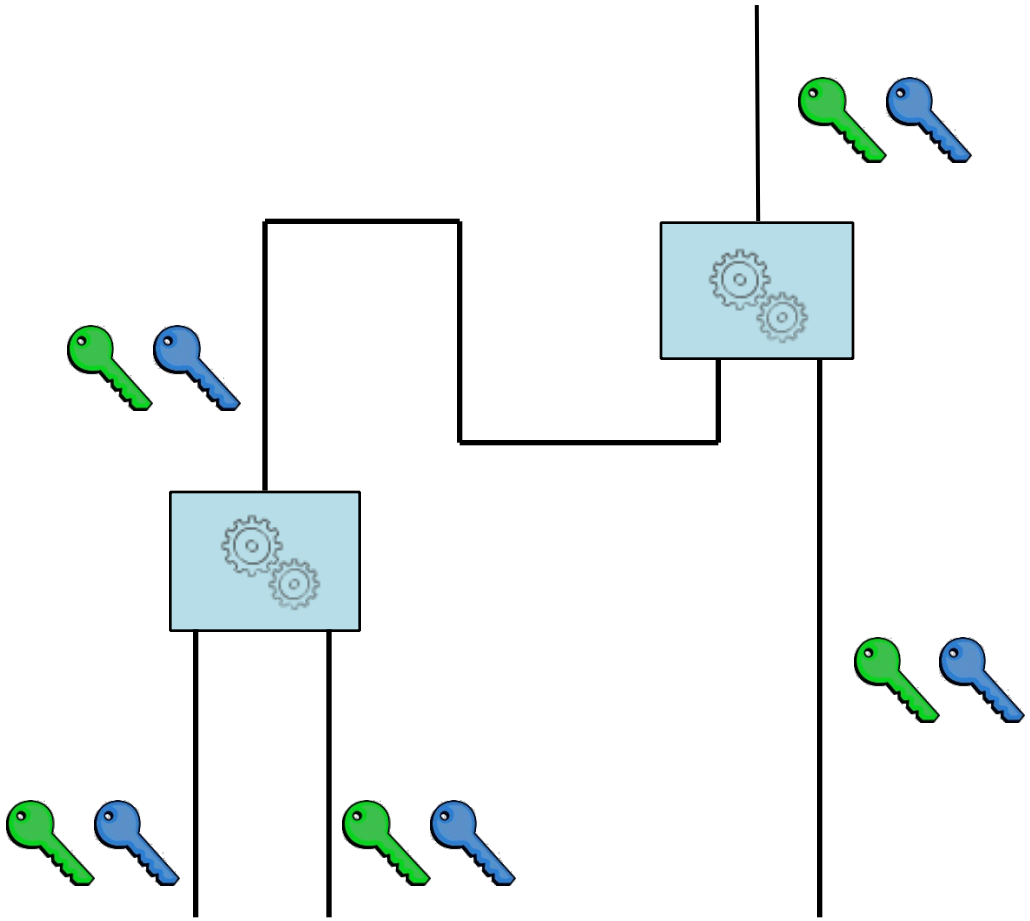


Garbled Circuits [Yao86]

Conventional circuit

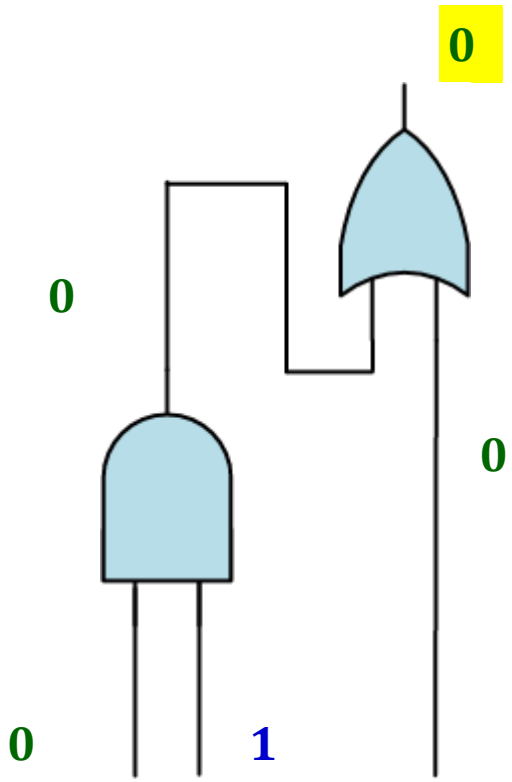


Garbled circuit

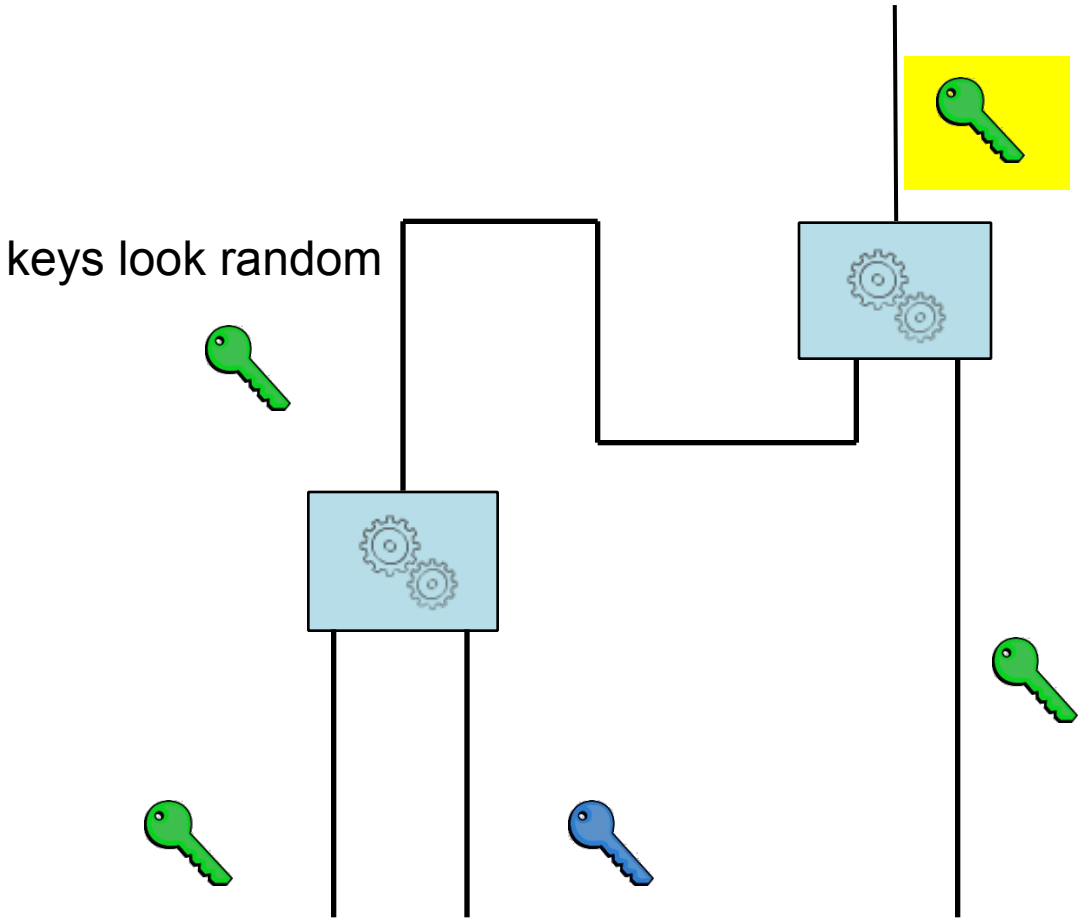


Garbled Circuits [Yao86]

Conventional circuit

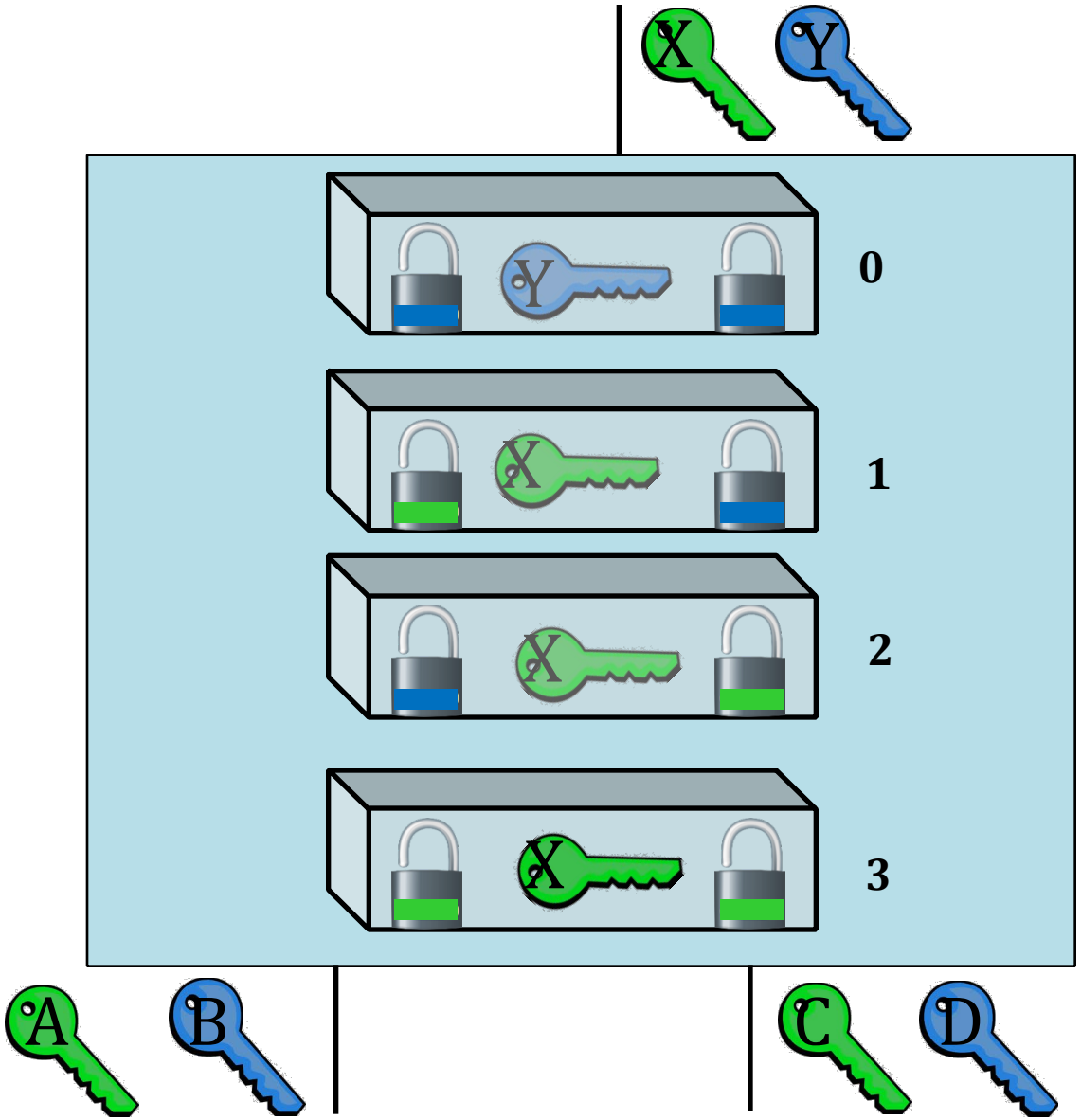


Garbled circuit



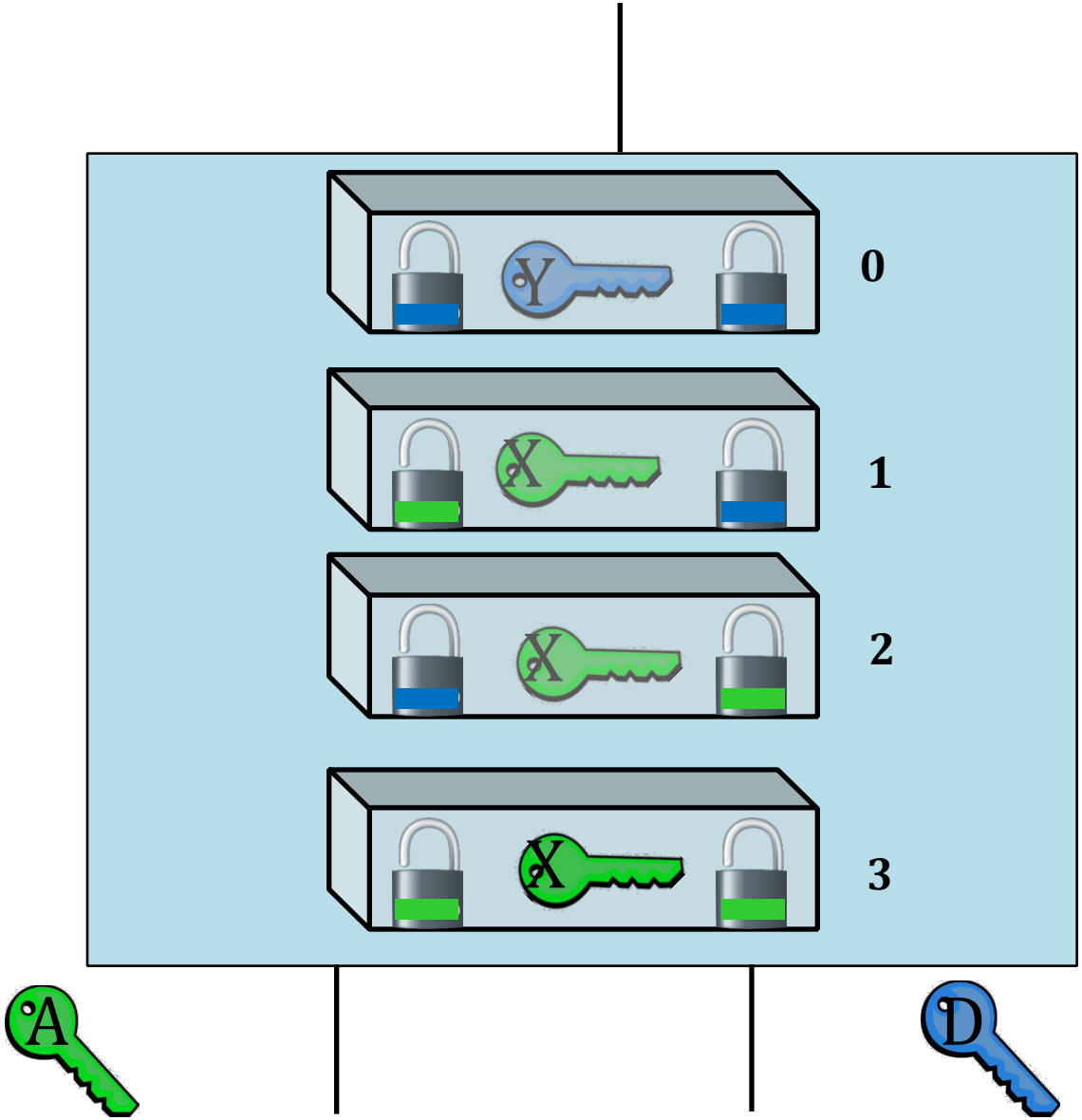
given input keys, can compute output key only

Garbled Gate [Yao86]



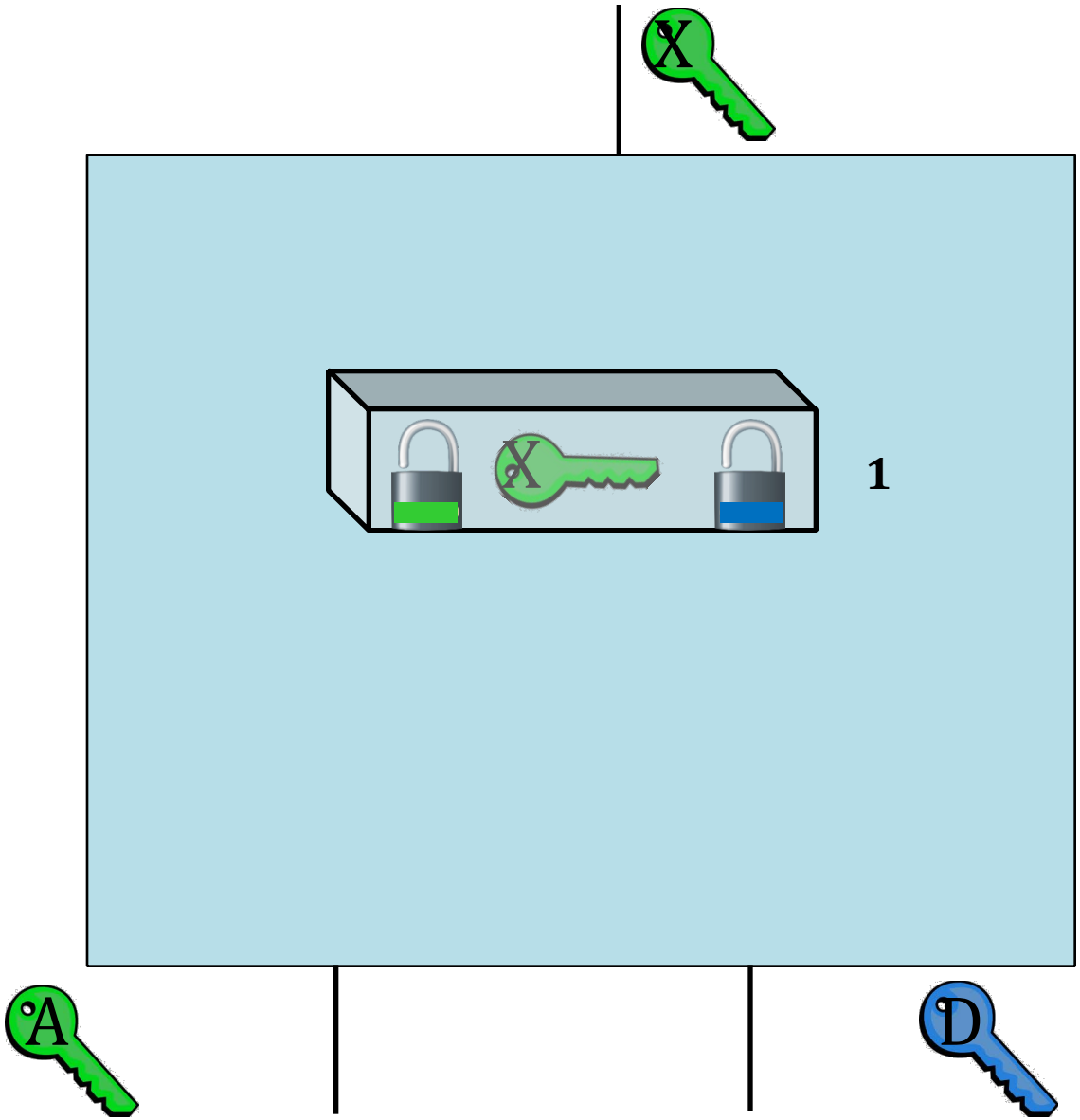
given two input keys,
can compute only output key

Garbled Gate [Yao86]



given two input keys,
can compute only output key

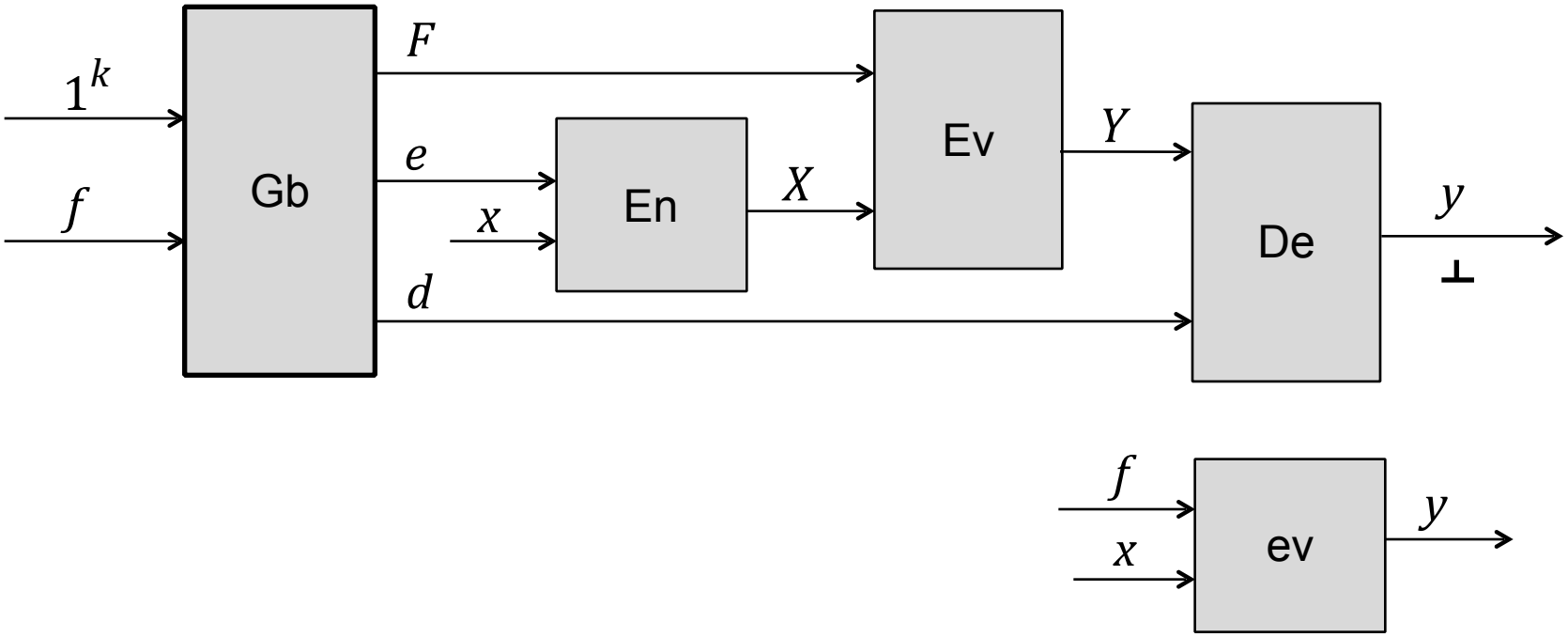
Garbled Gate [Yao86]



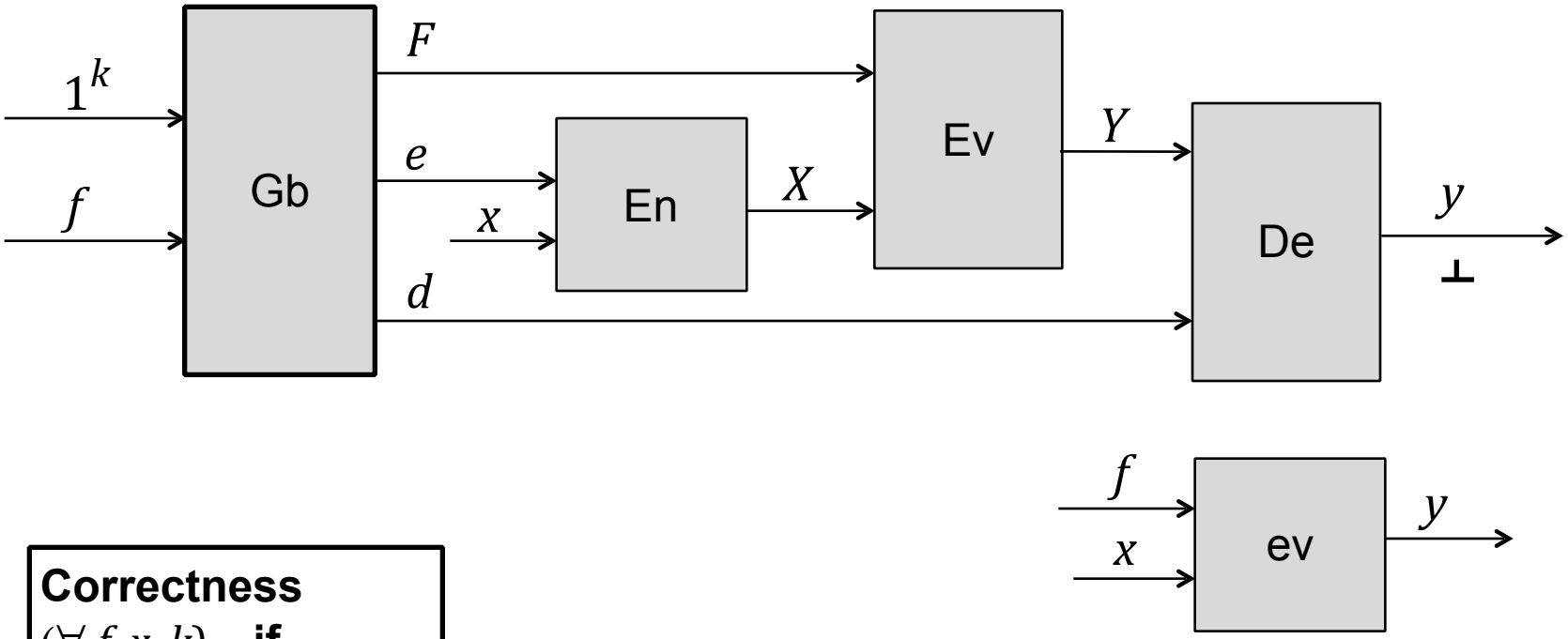
given two input keys,
can compute only output key

(Slide from Viet-Tung Hoang)

Formalization: Garbling Schemes [BellareHoangRogaway12]



Formalization: Garbling Schemes [BellareHoangRogaway12]



Correctness

$(\forall f, x, k)$, **if**

$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$,

$X \leftarrow \mathsf{En}(e, x)$,

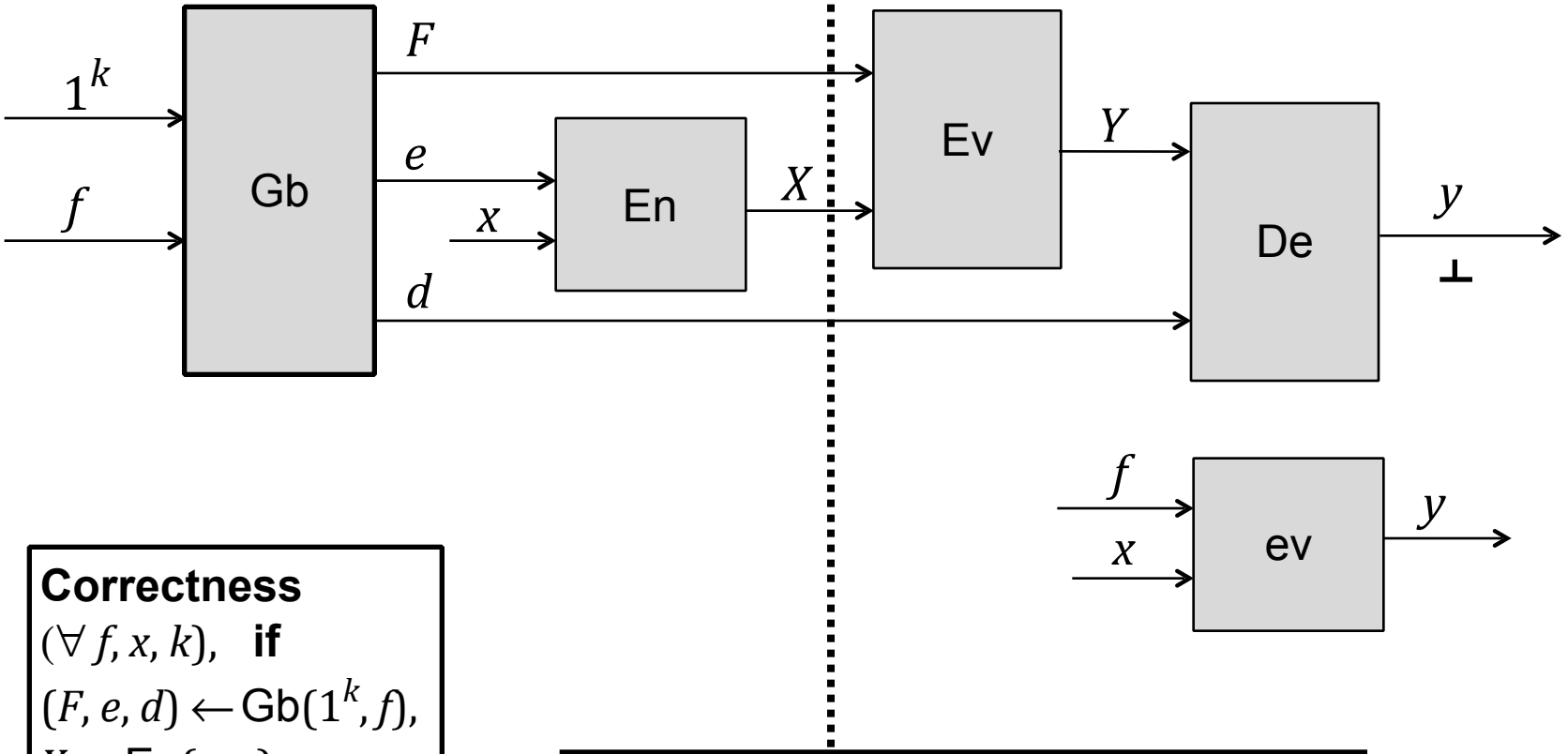
$Y \leftarrow \mathsf{Ev}(F, X)$,

$y \leftarrow \mathsf{De}(d, Y)$ **then**

$y = \mathsf{ev}(f, x)$

(Slide from Viet-Tung Hoang)

Formalization: Garbling Schemes [BellareHoangRogaway12]



Correctness
$(\forall f, x, k)$, **if**
$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$,
$X \leftarrow \mathsf{En}(e, x)$,
$Y \leftarrow \mathsf{Ev}(F, X)$,
$y \leftarrow \mathsf{De}(d, Y)$ **then**
$y = \mathsf{ev}(f, x)$

Privacy (informal):
Given (F, X, d) learn nothing but y=f(x).

(Slide from Viet-Tung Hoang)

Overview of Efficient Garbled Circuit Constructions

1990	**Point-and-Permute**	**[BeaverMicaliRogaway]**
1999	**3-row reduction**	**[NaorPinkasSumner]**
2008	**Free-XOR**	**[KolesnikovSchneider]**
2009	2-row reduction	[PinkasSchneiderSmartWilliams]
2012	**Garbling via AES**	**[KreuterShelatShen]**
2013	**Fixed-key AES**	**[BellareHoangKeelveedhiRogaway]**
2014	FleXor	[KolesnikovMohasselRosulek]
2015	**HalfGates**	**[ZahurRosulekEvans]**

(Slide from Payman Mohassel)

1) Encryption with Efficiently Verifiable Range [LindellPinkas04]

Wires:

Assign random keys k_i, K_i to all wires i

1) Encryption with Efficiently Verifiable Range [LindellPinkas04]

Wires:

Assign random keys k_i, K_i to all wires i

Garbling:

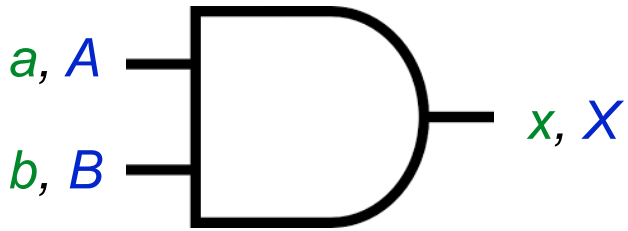
For each gate use double-encryption and randomly permute entries:

$E_a(E_b(x))$

$E_a(E_B(x))$

$E_A(E_b(x))$

$E_A(E_B(X))$



1) Encryption with Efficiently Verifiable Range [LindellPinkas04]

Wires:

Assign random keys k_i, K_i to all wires i

Garbling:

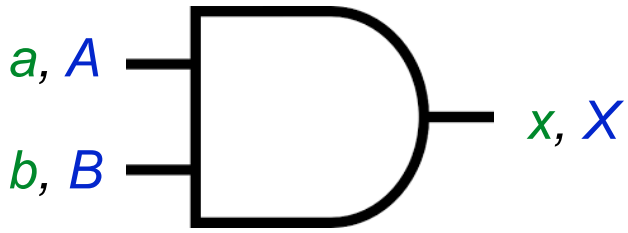
For each gate use double-encryption and randomly permute entries:

$E_a(E_b(x))$

$E_a(E_B(x))$

$E_A(E_b(x))$

$E_A(E_B(X))$



Outputs:

For each output wire *i*: provide mapping [(0, k_i), (1, K_i)]

1) Encryption with Efficiently Verifiable Range [LindellPinkas04]

Evaluator needs to know which entry was decrypted successfully

1) Encryption with Efficiently Verifiable Range [LindellPinkas04]

Evaluator needs to know which entry was decrypted successfully

⇒ Use encryption function with **efficiently verifiable range**:

$E_k(m) = [r, f_k(r) \oplus (m \,||\, 0^n)]$, where f is a pseudo-random function

(by pseudorandomness of f, prob. of obtaining 0^n with incorrect k is negl.)

1) Encryption with Efficiently Verifiable Range [LindellPinkas04]

Evaluator needs to know which entry was decrypted successfully

⇒ Use encryption function with **efficiently verifiable range**:

$E_k(m) = [r, f_k(r) \oplus (m \,||\, 0^n)]$, where f is a pseudo-random function

(by pseudorandomness of f, prob. of obtaining 0^n with incorrect k is negl.)

⇒ Need to **decrypt multiple entries** until decryption succeeds

1) Encryption with Efficiently Verifiable Range [LindellPinkas04]

Evaluator needs to know which entry was decrypted successfully

⇒ Use encryption function with **efficiently verifiable range**:

$E_k(m) = [r, f_k(r) \oplus (m \,||\, 0^n)]$, where f is a pseudo-random function

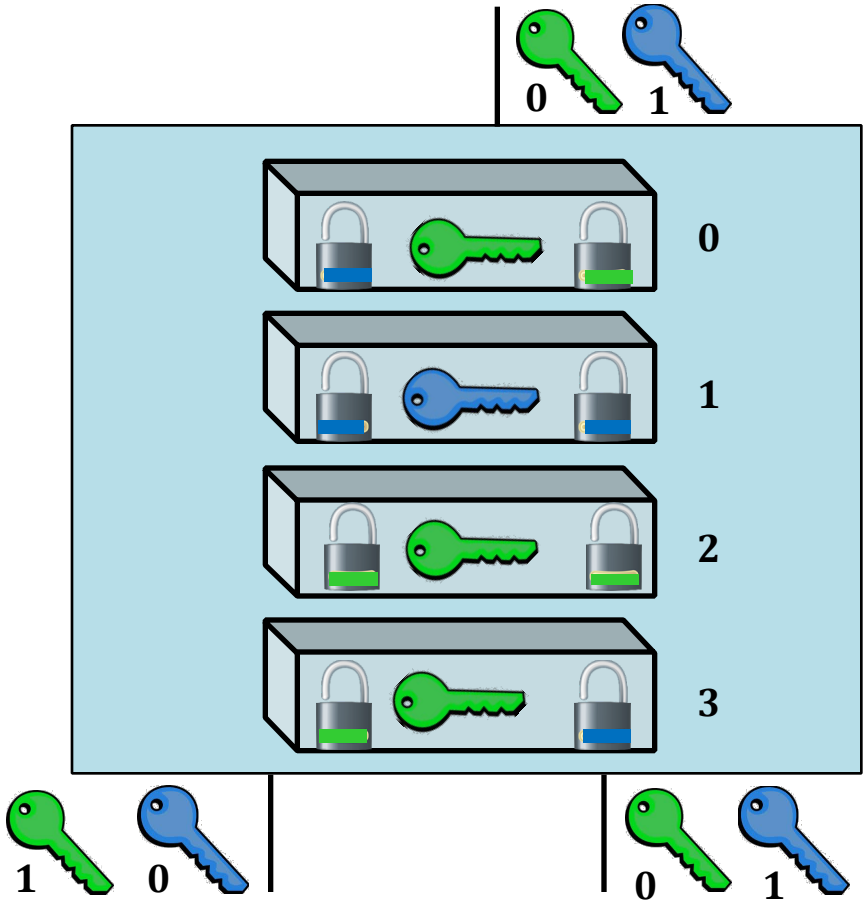
(by pseudorandomness of f, prob. of obtaining 0^n with incorrect k is negl.)

⇒ Need to **decrypt multiple entries** until decryption succeeds

- Expected number of decryptions requires is 2.5

2) Point and Permute [BeaverMicaliRogaway90]

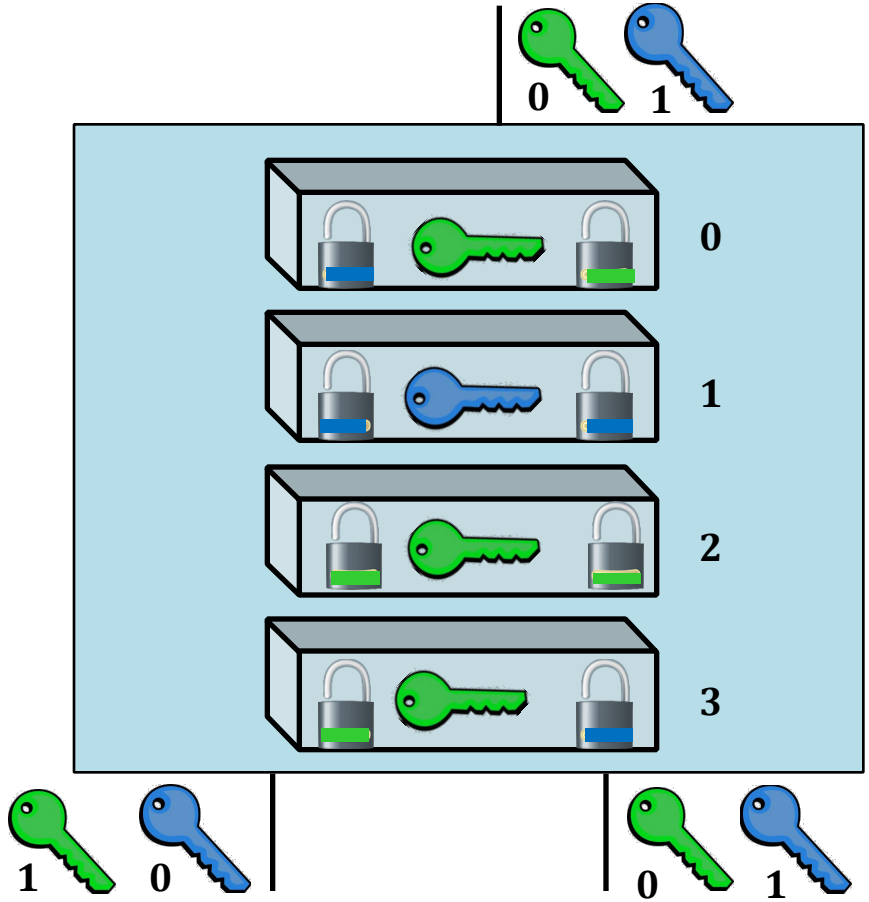
For every wire i, choose a random signal bit p_i together with the key.



2) Point and Permute [BeaverMicaliRogaway90]

For every wire *i*, choose a random signal bit p_i together with the key.

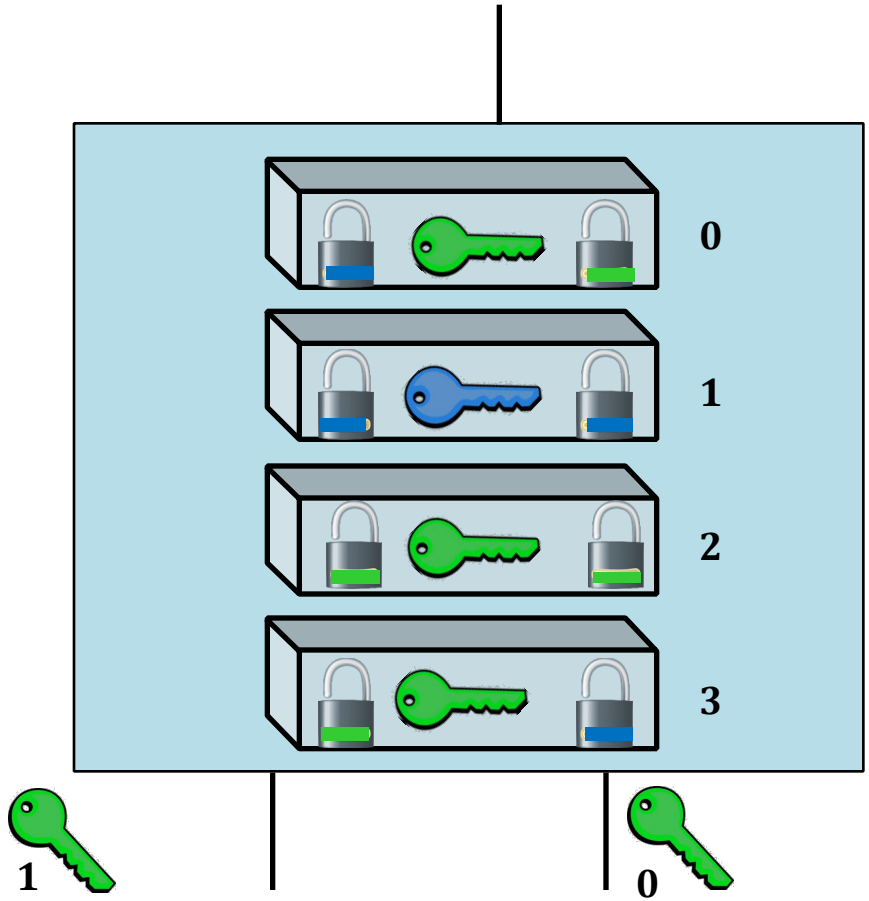
The signal bits of the input wires determine which entry to decrypt.



2) Point and Permute [BeaverMicaliRogaway90]

For every wire *i*, choose a random signal bit p_i together with the key.

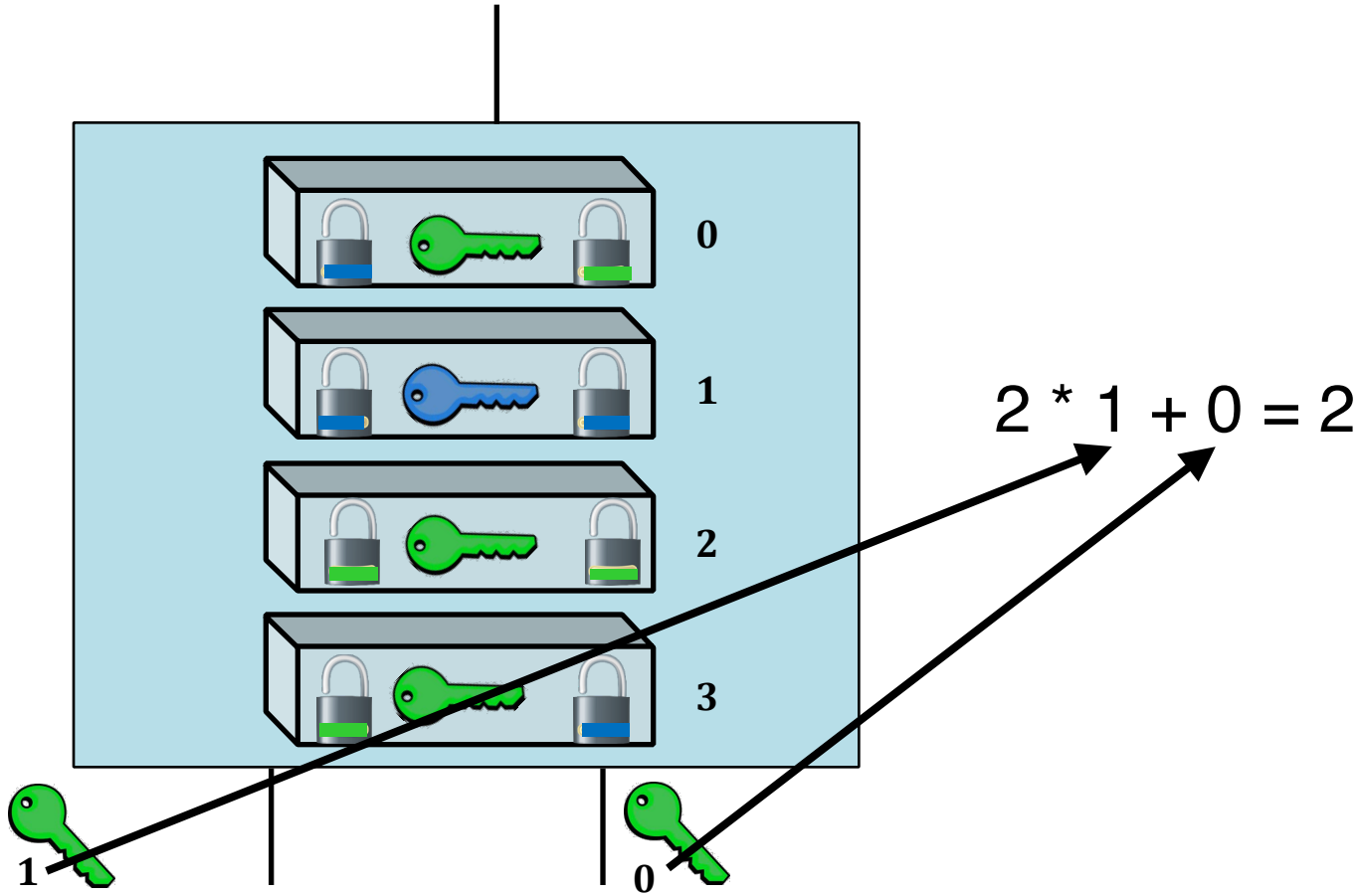
The signal bits of the input wires determine which entry to decrypt.



2) Point and Permute [BeaverMicaliRogaway90]

For every wire *i*, choose a random signal bit p_i together with the key.

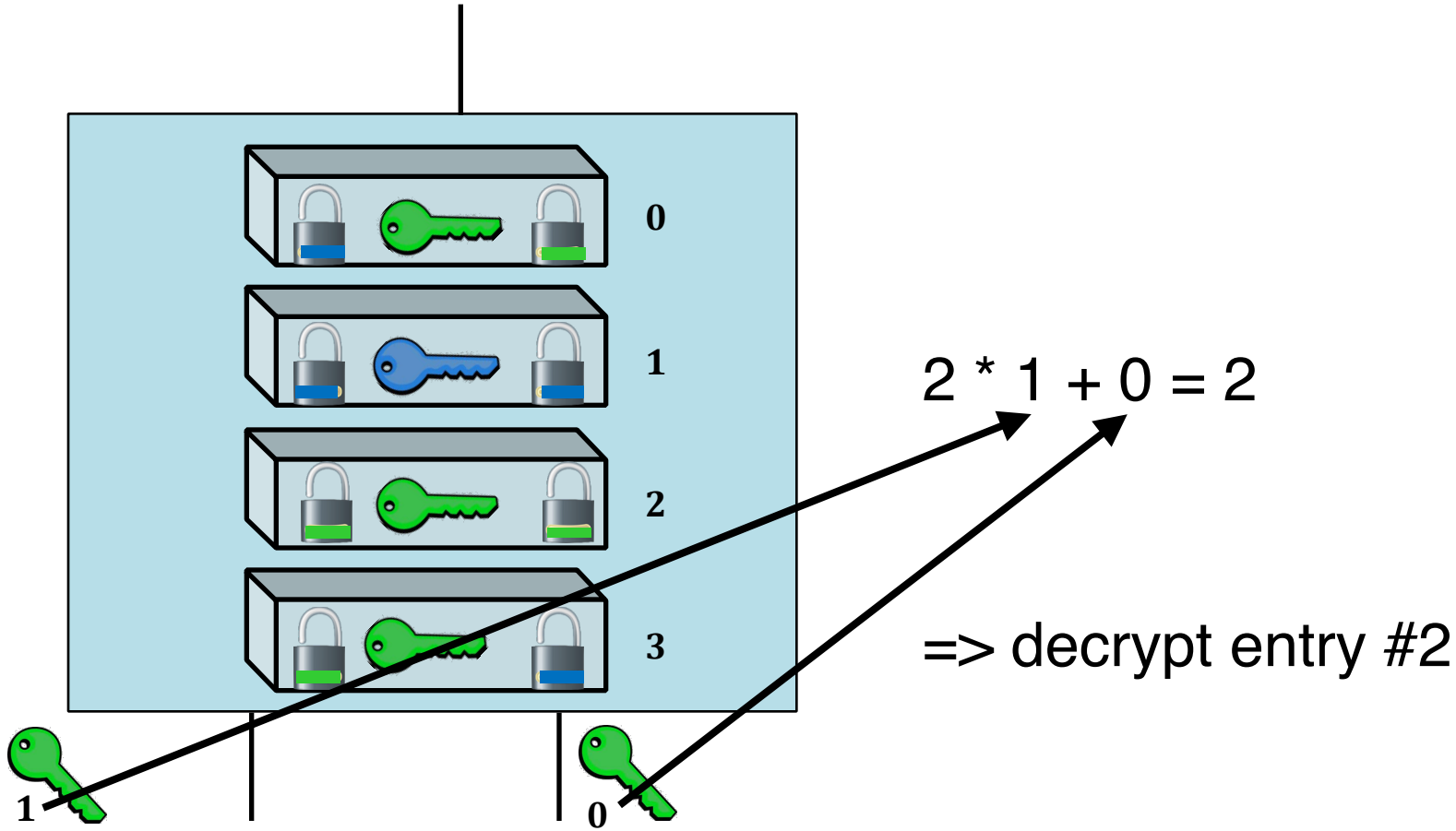
The signal bits of the input wires determine which entry to decrypt.



2) Point and Permute [BeaverMicaliRogaway90]

For every wire *i*, choose a random signal bit p_i together with the key.

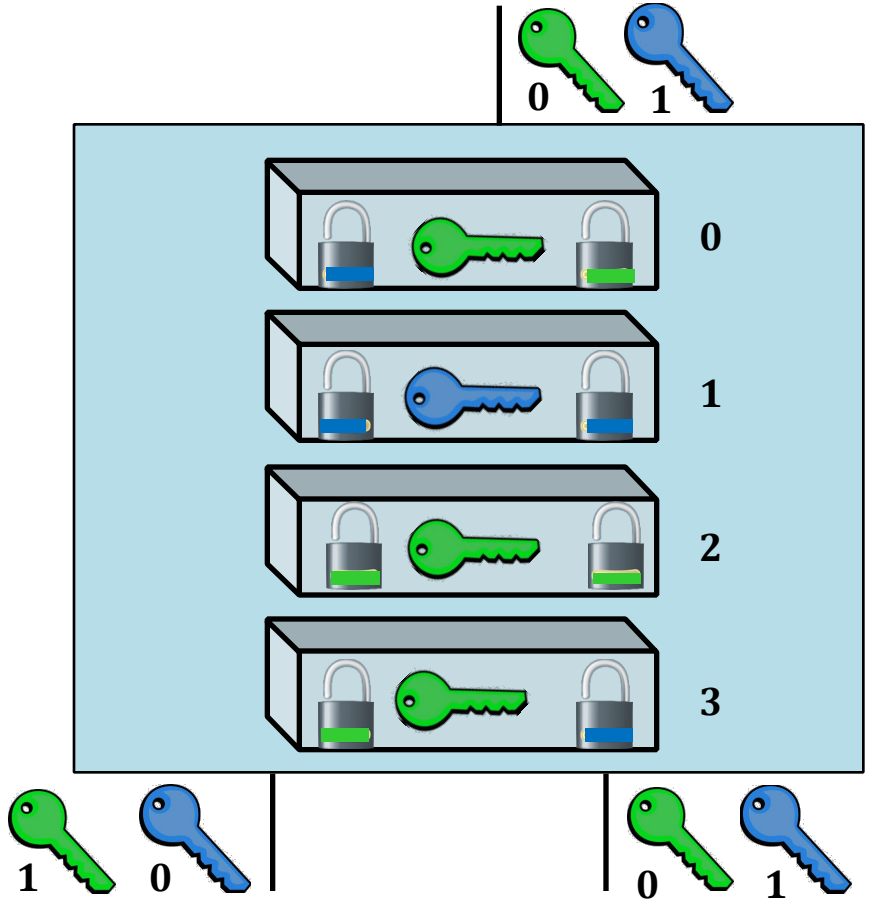
The signal bits of the input wires determine which entry to decrypt.



2) Point and Permute [BeaverMicaliRogaway90]

For every wire *i*, choose a random signal bit p_i together with the key.

The signal bits of the input wires determine which entry to decrypt.



2) Point and Permute [BeaverMicaliRogaway90]

Advantages of point-and-permute:

- Exactly one entry needs to be decrypted

2) Point and Permute [BeaverMicaliRogaway90]

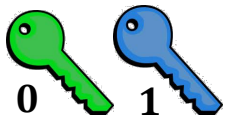
Advantages of point-and-permute:

- Exactly one entry needs to be decrypted
- Simplifies output decryption

2) Point and Permute [BeaverMicaliRogaway90]

Advantages of point-and-permute:

- Exactly one entry needs to be decrypted
- Simplifies output decryption



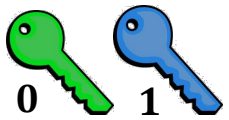
If output permutation $p_i = 0$ then output is permutation bit

If output permutation $p_i = 1$ then output is negated permutation bit

2) Point and Permute [BeaverMicaliRogaway90]

Advantages of point-and-permute:

- Exactly one entry needs to be decrypted
- Simplifies output decryption



If output permutation $p_i = 0$ then output is permutation bit

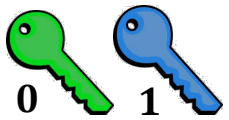
If output permutation $p_i = 1$ then output is negated permutation bit

$\Rightarrow$ Sender simply reveals for each output wire the bit p_i to receiver.

2) Point and Permute [BeaverMicaliRogaway90]

Advantages of point-and-permute:

- Exactly one entry needs to be decrypted
- Simplifies output decryption



If output permutation $p_i = 0$ then output is permutation bit

If output permutation $p_i = 1$ then output is negated permutation bit

⇒ Sender simply reveals for each output wire the bit p_i to receiver.

In the following we always assume usage of point and permute.

p(k) is the permutation bit of key k.

3) 3-Row Reduction [NaorPinkasSumner99]

Encryption function: $E^T(k_l, k_r; k_o) = k_o \oplus F(k_l, p(k_l) \,||\, T) \oplus F(k_r, p(k_r) \,||\, T)$,

where F is a pseudo-random function, e.g., instantiated with AES.

3) 3-Row Reduction [NaorPinkasSumner99]

Encryption function: $E^T(k_l, k_r; k_o) = k_o \oplus F(k_l, p(k_l) \,||\, T) \oplus F(k_r, p(k_r) \,||\, T)$,

where F is a pseudo-random function, e.g., instantiated with AES.

Idea: Eliminate first table entry by fixing it to be 0.

$E^T(k_l,k_r; c) = c \oplus F(k_l, p(k_l) \,||\, T) \oplus F(k_r, p(k_r) \,||\, i)$ **!= 0**

$\Rightarrow c = F(k_l, p(k_l) \,||\, T) \oplus F(k_r, p(k_r) \,||\, T)$.

$\Rightarrow$ One of the two output keys is derived from the input keys.

3) 3-Row Reduction [NaorPinkasSumner99]

Encryption function: $E^T(k_l, k_r; k_o) = k_o \oplus F(k_l, p(k_l) \,||\, T) \oplus F(k_r, p(k_r) \,||\, T)$,

where F is a pseudo-random function, e.g., instantiated with AES.

Idea: Eliminate first table entry by fixing it to be 0.

$E^T(k_l,k_r; c) = c \oplus F(k_l, p(k_l) \,||\, T) \oplus F(k_r, p(k_r) \,||\, i)$ **!= 0**

$\Rightarrow c = F(k_l, p(k_l) \,||\, T) \oplus F(k_r, p(k_r) \,||\, T)$.

$\Rightarrow$ One of the two output keys is derived from the input keys.

$E^T(a,B; c)$
$E^T(A,b; c)$
$E^T(A,B; C)$

} remaining 3 table entries as before

3) 3-Row Reduction [NaorPinkasSumner99]

Encryption function: $E^T(k_l, k_r; k_o) = k_o \oplus F(k_l, p(k_l) \,||\, T) \oplus F(k_r, p(k_r) \,||\, T)$,

where F is a pseudo-random function, e.g., instantiated with AES.

Idea: Eliminate first table entry by fixing it to be 0.

$E^T(k_l,k_r; c) = c \oplus F(k_l, p(k_l) \,||\, T) \oplus F(k_r, p(k_r) \,||\, i)$ **!= 0**

$\Rightarrow c = F(k_l, p(k_l) \,||\, T) \oplus F(k_r, p(k_r) \,||\, T)$.

$\Rightarrow$ One of the two output keys is derived from the input keys.

$E^T(a,B; c)$
$E^T(A,b; c)$
$E^T(A,B; C)$

} remaining 3 table entries as before

$\Rightarrow$ Communication is reduced from 4 to 3 table entries.

4) Free XOR [KolesnikovSchneider08]

Encryption function: $E^T(k_l, k_r; k_o) = k_o \oplus H(k_l \,||\, k_r \,||\, T)$,

where H is a random oracle, e.g., instantiated with SHA-2

4) Free XOR [KolesnikovSchneider08]

Encryption function: $E^T(k_l, k_r; k_o) = k_o \oplus H(k_l \,||\, k_r \,||\, T)$,

where H is a random oracle, e.g., instantiated with SHA-2

Idea: Choose keys s.t. each pair has distance R (unknown to evaluator).

$R = a \oplus A = b \oplus B = c \oplus C = \ldots$

4) Free XOR [KolesnikovSchneider08]

Encryption function: $E^T(k_l, k_r; k_o) = k_o \oplus H(k_l \,||\, k_r \,||\, T)$,

where H is a random oracle, e.g., instantiated with SHA-2

Idea: Choose keys s.t. each pair has distance R (unknown to evaluator).

$R = a \oplus A = b \oplus B = c \oplus C = \ldots$

Garble XOR: set output key $c = a \oplus b$

4) Free XOR [KolesnikovSchneider08]

Encryption function: $E^T(k_l, k_r; k_o) = k_o \oplus H(k_l \,||\, k_r \,||\, T)$,

where H is a random oracle, e.g., instantiated with SHA-2

Idea: Choose keys s.t. each pair has distance R (unknown to evaluator).

$R = a \oplus A = b \oplus B = c \oplus C = \ldots$

Garble XOR: set output key $c = a \oplus b$

Evaluate XOR: set output key $k_c = k_a \oplus k_b$

4) Free XOR [KolesnikovSchneider08]

Encryption function: $E^T(k_l, k_r; k_o) = k_o \oplus H(k_l \,||\, k_r \,||\, T)$,

where H is a random oracle, e.g., instantiated with SHA-2

Idea: Choose keys s.t. each pair has distance R (unknown to evaluator).

$R = a \oplus A = b \oplus B = c \oplus C = \ldots$

Garble XOR: set output key $c = a \oplus b$

Evaluate XOR: set output key $k_c = k_a \oplus k_b$

Correctness: $c = a \oplus b = (R \oplus a) \oplus (R \oplus b) = A \oplus B$

$C = c \oplus R = a \oplus b \oplus R = a \oplus B = A \oplus b$

4) Free XOR [KolesnikovSchneider08]

Encryption function: $E^T(k_l, k_r; k_o) = k_o \oplus H(k_l \,||\, k_r \,||\, T)$,

where H is a random oracle, e.g., instantiated with SHA-2

Idea: Choose keys s.t. each pair has distance R (unknown to evaluator).

$R = a \oplus A = b \oplus B = c \oplus C = \ldots$

Garble XOR: set output key $c = a \oplus b$

Evaluate XOR: set output key $k_c = k_a \oplus k_b$

Correctness: $c = a \oplus b = (R \oplus a) \oplus (R \oplus b) = A \oplus B$

$C = c \oplus R = a \oplus b \oplus R = a \oplus B = A \oplus b$

Security (intuitively): Evaluator knows one key per wire, so never learns R

- *Requires random oracle or non-standard circularity assumption*

4) Free XOR [KolesnikovSchneider08]

Encryption function: $E^T(k_l, k_r; k_o) = k_o \oplus H(k_l \,||\, k_r \,||\, T)$,

where H is a random oracle, e.g., instantiated with SHA-2

Idea: Choose keys s.t. each pair has distance R (unknown to evaluator).

$R = a \oplus A = b \oplus B = c \oplus C = \ldots$

Garble XOR: set output key $c = a \oplus b$

Evaluate XOR: set output key $k_c = k_a \oplus k_b$

Correctness: $c = a \oplus b = (R \oplus a) \oplus (R \oplus b) = A \oplus B$

$C = c \oplus R = a \oplus b \oplus R = a \oplus B = A \oplus b$

Security (intuitively): Evaluator knows one key per wire, so never learns R

- *Requires random oracle or non-standard circularity assumption*

Can be combined with 3-row reduction.

5) Garbling via AES

Since 2008 many Intel and AMD CPUs have hardware support for AES:

Advanced Encryption Standard New Instructions (AES-NI)

5) Garbling via AES

Since 2008 many Intel and AMD CPUs have hardware support for AES:

Advanced Encryption Standard New Instructions (AES-NI)

[KreuterShelatShen12]: $E^T(k_l, k_r; k_o) = k_o \oplus \text{AES-256}(k_l \,||\, k_r; T)$

5) Garbling via AES

Since 2008 many Intel and AMD CPUs have hardware support for AES:

Advanced Encryption Standard New Instructions (AES-NI)

[KreuterShelatShen12]: $E^T(k_l, k_r; k_o) = k_o \oplus \text{AES-256}(k_l \,||\, k_r; T)$

=> Needs to run expensive AES key schedule per gate

=> Also assumes a related-key assumptions (not great for AES)

5) Garbling via AES

Since 2008 many Intel and AMD CPUs have hardware support for AES:

Advanced Encryption Standard New Instructions (AES-NI)

[KreuterShelatShen12]: $E^T(k_l, k_r; k_o) = k_o \oplus \text{AES-256}(k_l \,||\, k_r; T)$

=> Needs to run expensive AES key schedule per gate

=> Also assumes a related-key assumptions (not great for AES)

[BellareHoangKeelveedhiRogaway13]:

Choose fixed key X and run AES key schedule once

$E^T(k_l, k_r; k_o) = k_o \oplus \text{AES-128}(X; K) \oplus K$ with $K = 2k_l \oplus 4k_r \oplus T$

5) Garbling via AES

Since 2008 many Intel and AMD CPUs have hardware support for AES:

Advanced Encryption Standard New Instructions (AES-NI)

[KreuterShelatShen12]: $E^T(k_l, k_r; k_o) = k_o \oplus \text{AES-256}(k_l \,||\, k_r; T)$

=> Needs to run expensive AES key schedule per gate

=> Also assumes a related-key assumptions (not great for AES)

[BellareHoangKeelveedhiRogaway13]:

Choose fixed key X and run AES key schedule once

$E^T(k_l, k_r; k_o) = k_o \oplus \text{AES-128}(X; K) \oplus K$ with $K = 2k_l \oplus 4k_r \oplus T$

Requires assuming an “ideal cipher” assumption on AES

5) Garbling via AES

Since 2008 many Intel and AMD CPUs have hardware support for AES:

Advanced Encryption Standard New Instructions (AES-NI)

[KreuterShelatShen12]: $E^T(k_l, k_r; k_o) = k_o \oplus \text{AES-256}(k_l \,||\, k_r; T)$

=> Needs to run expensive AES key schedule per gate

=> Also assumes a related-key assumptions (not great for AES)

[BellareHoangKeelveedhiRogaway13]:

Choose fixed key X and run AES key schedule once

$E^T(k_l, k_r; k_o) = k_o \oplus \text{AES-128}(X; K) \oplus K$ with $K = 2k_l \oplus 4k_r \oplus T$

Requires assuming an “ideal cipher” assumption on AES

Can be combined with free XOR and 3-row reduction

6) HalfGates - Today's Most Efficient Scheme [ZRE14]

procedure $\mathsf{Gb}(1^k, f)$:
  $R \twoheadleftarrow \{0,1\}^{k-1}1$
  for $i \in \mathsf{Inputs}(f)$ **do**
    $W_i^0 \twoheadleftarrow \{0,1\}^k$
    $W_i^1 \leftarrow W_i^0 \oplus R$
    $e_i \leftarrow W_i^0$
  for $i \notin \mathsf{Inputs}(f)$ {*in topo. order*} **do**
    $\{a, b\} \leftarrow \mathsf{GateInputs}(f, i)$
    if $i \in \mathsf{XorGates}(f)$ **then**
      $W_i^0 \leftarrow W_a^0 \oplus W_b^0$
    else
      $(W_i^0, T_{Gi}, T_{Ei}) \leftarrow \mathsf{GbAnd}(W_a^0, W_b^0)$
      $F_i \leftarrow (T_{Gi}, T_{Ei})$
    end if
    $W_i^1 \leftarrow W_i^0 \oplus R$
  for $i \in \mathsf{Outputs}(f)$ **do**
    $d_i \leftarrow \mathsf{lsb}(W_i^0)$
  return $(\hat{F}, \hat{e}, \hat{d})$

private procedure $\mathsf{GbAnd}(W_a^0, W_b^0)$:
  $p_a \leftarrow \mathsf{lsb}\, W_a^0$; $p_b \leftarrow \mathsf{lsb}\, W_b^0$
  $j \leftarrow \mathsf{NextIndex}()$; $j' \leftarrow \mathsf{NextIndex}()$
  {*First half gate*}
  $T_G \leftarrow H(W_a^0, j) \oplus H(W_a^1, j) \oplus p_b R$
  $W_G^0 \leftarrow H(W_a^0, j) \oplus p_a T_G$
  {*Second half gate*}
  $T_E \leftarrow H(W_b^0, j') \oplus H(W_b^1, j') \oplus W_a^0$
  $W_E^0 \leftarrow H(W_b^0, j') \oplus p_b(T_E \oplus W_a^0)$
  {*Combine halves*}
  $W^0 \leftarrow W_G^0 \oplus W_E^0$
  return (W^0, T_G, T_E)

procedure $\mathsf{En}(\hat{e}, \hat{x})$:
  for $e_i \in \hat{e}$ **do**
    $X_i \leftarrow e_i \oplus x_i R$
  return $\hat{X}$

procedure $\mathsf{De}(\hat{d}, \hat{Y})$:
  for $d_i \in \hat{d}$ **do**
    $y_i \leftarrow d_i \oplus \mathsf{lsb}\, Y_i$
  return $\hat{y}$

procedure $\mathsf{Ev}(\hat{F}, \hat{X})$:
  for $i \in \mathsf{Inputs}(\hat{F})$ **do**
    $W_i \leftarrow X_i$
  for $i \notin \mathsf{Inputs}(\hat{F})$ {*in topo. order*} **do**
    $\{a, b\} \leftarrow \mathsf{GateInputs}(\hat{F}, i)$
    if $i \in \mathsf{XorGates}(\hat{F})$ **then**
      $W_i \leftarrow W_a \oplus W_b$
    else
      $s_a \leftarrow \mathsf{lsb}\, W_a$; $s_b \leftarrow \mathsf{lsb}\, W_b$
      $j_1 \leftarrow \mathsf{NextIndex}()$; $j_2 \leftarrow \mathsf{NextIndex}()$
      $(T_{Gi}, T_{Ei}) \leftarrow F_i$
      $W_{Gi} \leftarrow H(W_a, j_1) \oplus s_a T_{Gi}$
      $W_{Ei} \leftarrow H(W_b, j_2) \oplus s_b(T_{Ei} \oplus W_a)$
      $W_i \leftarrow W_{Gi} \oplus W_{Ei}$
    end if
  for $i \in \mathsf{Outputs}(\hat{F})$ **do**
    $Y_i \leftarrow W_i$
  return $\hat{Y}$

6) HalfGates - Today's Most Efficient Scheme [ZRE14]

Free XOR

procedure $\mathsf{Gb}(1^k, f)$:
- $R \twoheadleftarrow \{0,1\}^{k-1}1$
- **for** $i \in \mathsf{Inputs}(f)$ **do**
  - $W_i^0 \twoheadleftarrow \{0,1\}^k$
  - $W_i^1 \leftarrow W_i^0 \oplus R$
  - $e_i \leftarrow W_i^0$
- **for** $i \notin \mathsf{Inputs}(f)$ {*in topo. order*} **do**
  - $\{a, b\} \leftarrow \mathsf{GateInputs}(f, i)$
  - **if** $i \in \mathsf{XorGates}(f)$ **then**
    - $W_i^0 \leftarrow W_a^0 \oplus W_b^0$
  - **else**
    - $(W_i^0, T_{Gi}, T_{Ei}) \leftarrow \mathsf{GbAnd}(W_a^0, W_b^0)$
    - $F_i \leftarrow (T_{Gi}, T_{Ei})$
  - **end if**
  - $W_i^1 \leftarrow W_i^0 \oplus R$
- **for** $i \in \mathsf{Outputs}(f)$ **do**
  - $d_i \leftarrow \mathsf{lsb}(W_i^0)$
- **return** $(\hat{F}, \hat{e}, \hat{d})$

private procedure $\mathsf{GbAnd}(W_a^0, W_b^0)$:
- $p_a \leftarrow \mathsf{lsb}\, W_a^0$; $p_b \leftarrow \mathsf{lsb}\, W_b^0$
- $j \leftarrow \mathsf{NextIndex}()$; $j' \leftarrow \mathsf{NextIndex}()$
- {*First half gate*}
- $T_G \leftarrow H(W_a^0, j) \oplus H(W_a^1, j) \oplus p_b R$
- $W_G^0 \leftarrow H(W_a^0, j) \oplus p_a T_G$
- {*Second half gate*}
- $T_E \leftarrow H(W_b^0, j') \oplus H(W_b^1, j') \oplus W_a^0$
- $W_E^0 \leftarrow H(W_b^0, j') \oplus p_b(T_E \oplus W_a^0)$
- {*Combine halves*}
- $W^0 \leftarrow W_G^0 \oplus W_E^0$
- **return** (W^0, T_G, T_E)

procedure $\mathsf{En}(\hat{e}, \hat{x})$:
- **for** $e_i \in \hat{e}$ **do**
  - $X_i \leftarrow e_i \oplus x_i R$
- **return** $\hat{X}$

procedure $\mathsf{De}(\hat{d}, \hat{Y})$:
- **for** $d_i \in \hat{d}$ **do**
  - $y_i \leftarrow d_i \oplus \mathsf{lsb}\, Y_i$
- **return** $\hat{y}$

procedure $\mathsf{Ev}(\hat{F}, \hat{X})$:
- **for** $i \in \mathsf{Inputs}(\hat{F})$ **do**
  - $W_i \leftarrow X_i$
- **for** $i \notin \mathsf{Inputs}(\hat{F})$ {*in topo. order*} **do**
  - $\{a, b\} \leftarrow \mathsf{GateInputs}(\hat{F}, i)$
  - **if** $i \in \mathsf{XorGates}(\hat{F})$ **then**
    - $W_i \leftarrow W_a \oplus W_b$
  - **else**
    - $s_a \leftarrow \mathsf{lsb}\, W_a$; $s_b \leftarrow \mathsf{lsb}\, W_b$
    - $j_1 \leftarrow \mathsf{NextIndex}()$; $j_2 \leftarrow \mathsf{NextIndex}()$
    - $(T_{Gi}, T_{Ei}) \leftarrow F_i$
    - $W_{Gi} \leftarrow H(W_a, j_1) \oplus s_a T_{Gi}$
    - $W_{Ei} \leftarrow H(W_b, j_2) \oplus s_b(T_{Ei} \oplus W_a)$
    - $W_i \leftarrow W_{Gi} \oplus W_{Ei}$
  - **end if**
- **for** $i \in \mathsf{Outputs}(\hat{F})$ **do**
  - $Y_i \leftarrow W_i$
- **return** $\hat{Y}$

6) HalfGates - Today's Most Efficient Scheme [ZRE14]

Free XOR

4 calls of H for garbling AND

procedure $\mathsf{Gb}(1^k, f)$:
- $R \twoheadleftarrow \{0,1\}^{k-1}1$
- **for** $i \in \mathsf{Inputs}(f)$ **do**
  - $W_i^0 \twoheadleftarrow \{0,1\}^k$
  - $W_i^1 \leftarrow W_i^0 \oplus R$
  - $e_i \leftarrow W_i^0$
- **for** $i \notin \mathsf{Inputs}(f)$ {*in topo. order*} **do**
  - $\{a, b\} \leftarrow \mathsf{GateInputs}(f, i)$
  - **if** $i \in \mathsf{XorGates}(f)$ **then**
    - $W_i^0 \leftarrow W_a^0 \oplus W_b^0$
  - **else**
    - $(W_i^0, T_{Gi}, T_{Ei}) \leftarrow \mathsf{GbAnd}(W_a^0, W_b^0)$
    - $F_i \leftarrow (T_{Gi}, T_{Ei})$
  - **end if**
  - $W_i^1 \leftarrow W_i^0 \oplus R$
- **for** $i \in \mathsf{Outputs}(f)$ **do**
  - $d_i \leftarrow \mathsf{lsb}(W_i^0)$
- **return** $(\hat{F}, \hat{e}, \hat{d})$

private procedure $\mathsf{GbAnd}(W_a^0, W_b^0)$:
- $p_a \leftarrow \mathsf{lsb}\, W_a^0$; $p_b \leftarrow \mathsf{lsb}\, W_b^0$
- $j \leftarrow \mathsf{NextIndex}()$; $j' \leftarrow \mathsf{NextIndex}()$
- {*First half gate*}
- $T_G \leftarrow H(W_a^0, j) \oplus H(W_a^1, j) \oplus p_b R$
- $W_G^0 \leftarrow H(W_a^0, j) \oplus p_a T_G$
- {*Second half gate*}
- $T_E \leftarrow H(W_b^0, j') \oplus H(W_b^1, j') \oplus W_a^0$
- $W_E^0 \leftarrow H(W_b^0, j') \oplus p_b(T_E \oplus W_a^0)$
- {*Combine halves*}
- $W^0 \leftarrow W_G^0 \oplus W_E^0$
- **return** (W^0, T_G, T_E)

procedure $\mathsf{En}(\hat{e}, \hat{x})$:
- **for** $e_i \in \hat{e}$ **do**
  - $X_i \leftarrow e_i \oplus x_i R$
- **return** $\hat{X}$

procedure $\mathsf{De}(\hat{d}, \hat{Y})$:
- **for** $d_i \in \hat{d}$ **do**
  - $y_i \leftarrow d_i \oplus \mathsf{lsb}\, Y_i$
- **return** $\hat{y}$

procedure $\mathsf{Ev}(\hat{F}, \hat{X})$:
- **for** $i \in \mathsf{Inputs}(\hat{F})$ **do**
  - $W_i \leftarrow X_i$
- **for** $i \notin \mathsf{Inputs}(\hat{F})$ {*in topo. order*} **do**
  - $\{a, b\} \leftarrow \mathsf{GateInputs}(\hat{F}, i)$
  - **if** $i \in \mathsf{XorGates}(\hat{F})$ **then**
    - $W_i \leftarrow W_a \oplus W_b$
  - **else**
    - $s_a \leftarrow \mathsf{lsb}\, W_a$; $s_b \leftarrow \mathsf{lsb}\, W_b$
    - $j_1 \leftarrow \mathsf{NextIndex}()$; $j_2 \leftarrow \mathsf{NextIndex}()$
    - $(T_{Gi}, T_{Ei}) \leftarrow F_i$
    - $W_{Gi} \leftarrow H(W_a, j_1) \oplus s_a T_{Gi}$
    - $W_{Ei} \leftarrow H(W_b, j_2) \oplus s_b(T_{Ei} \oplus W_a)$
    - $W_i \leftarrow W_{Gi} \oplus W_{Ei}$
  - **end if**
- **for** $i \in \mathsf{Outputs}(\hat{F})$ **do**
  - $Y_i \leftarrow W_i$
- **return** $\hat{Y}$

6) HalfGates - Today's Most Efficient Scheme [ZRE14]

Free XOR

4 calls of H for garbling AND

2 table entries per AND!

procedure $\mathsf{Gb}(1^k, f)$:
- $R \twoheadleftarrow \{0,1\}^{k-1}1$
- **for** $i \in \mathsf{Inputs}(f)$ **do**
  - $W_i^0 \twoheadleftarrow \{0,1\}^k$
  - $W_i^1 \leftarrow W_i^0 \oplus R$
  - $e_i \leftarrow W_i^0$
- **for** $i \notin \mathsf{Inputs}(f)$ {*in topo. order*} **do**
  - $\{a, b\} \leftarrow \mathsf{GateInputs}(f, i)$
  - **if** $i \in \mathsf{XorGates}(f)$ **then**
    - $W_i^0 \leftarrow W_a^0 \oplus W_b^0$
  - **else**
    - $(W_i^0, T_{Gi}, T_{Ei}) \leftarrow \mathsf{GbAnd}(W_a^0, W_b^0)$
    - $F_i \leftarrow (T_{Gi}, T_{Ei})$
  - **end if**
  - $W_i^1 \leftarrow W_i^0 \oplus R$
- **for** $i \in \mathsf{Outputs}(f)$ **do**
  - $d_i \leftarrow \mathsf{lsb}(W_i^0)$
- **return** $(\hat{F}, \hat{e}, \hat{d})$

private procedure $\mathsf{GbAnd}(W_a^0, W_b^0)$:
- $p_a \leftarrow \mathsf{lsb}\, W_a^0$; $p_b \leftarrow \mathsf{lsb}\, W_b^0$
- $j \leftarrow \mathsf{NextIndex}()$; $j' \leftarrow \mathsf{NextIndex}()$
- {*First half gate*}
- $T_G \leftarrow H(W_a^0, j) \oplus H(W_a^1, j) \oplus p_b R$
- $W_G^0 \leftarrow H(W_a^0, j) \oplus p_a T_G$
- {*Second half gate*}
- $T_E \leftarrow H(W_b^0, j') \oplus H(W_b^1, j') \oplus W_a^0$
- $W_E^0 \leftarrow H(W_b^0, j') \oplus p_b(T_E \oplus W_a^0)$
- {*Combine halves*}
- $W^0 \leftarrow W_G^0 \oplus W_E^0$
- **return** (W^0, T_G, T_E)

procedure $\mathsf{En}(\hat{e}, \hat{x})$:
- **for** $e_i \in \hat{e}$ **do**
  - $X_i \leftarrow e_i \oplus x_i R$
- **return** $\hat{X}$

procedure $\mathsf{De}(\hat{d}, \hat{Y})$:
- **for** $d_i \in \hat{d}$ **do**
  - $y_i \leftarrow d_i \oplus \mathsf{lsb}\, Y_i$
- **return** $\hat{y}$

procedure $\mathsf{Ev}(\hat{F}, \hat{X})$:
- **for** $i \in \mathsf{Inputs}(\hat{F})$ **do**
  - $W_i \leftarrow X_i$
- **for** $i \notin \mathsf{Inputs}(\hat{F})$ {*in topo. order*} **do**
  - $\{a, b\} \leftarrow \mathsf{GateInputs}(\hat{F}, i)$
  - **if** $i \in \mathsf{XorGates}(\hat{F})$ **then**
    - $W_i \leftarrow W_a \oplus W_b$
  - **else**
    - $s_a \leftarrow \mathsf{lsb}\, W_a$; $s_b \leftarrow \mathsf{lsb}\, W_b$
    - $j_1 \leftarrow \mathsf{NextIndex}()$; $j_2 \leftarrow \mathsf{NextIndex}()$
    - $(T_{Gi}, T_{Ei}) \leftarrow F_i$
    - $W_{Gi} \leftarrow H(W_a, j_1) \oplus s_a T_{Gi}$
    - $W_{Ei} \leftarrow H(W_b, j_2) \oplus s_b(T_{Ei} \oplus W_a)$
    - $W_i \leftarrow W_{Gi} \oplus W_{Ei}$
  - **end if**
- **for** $i \in \mathsf{Outputs}(\hat{F})$ **do**
  - $Y_i \leftarrow W_i$
- **return** $\hat{Y}$

6) HalfGates - Today's Most Efficient Scheme [ZRE14]

procedure $\mathsf{Gb}(1^k, f)$:
- $R \twoheadleftarrow \{0,1\}^{k-1}1$
- **for** $i \in \mathsf{Inputs}(f)$ **do**
  - $W_i^0 \twoheadleftarrow \{0,1\}^k$
  - $W_i^1 \leftarrow W_i^0 \oplus R$
  - $e_i \leftarrow W_i^0$
- **for** $i \notin \mathsf{Inputs}(f)$ {*in topo. order*} **do**
  - $\{a, b\} \leftarrow \mathsf{GateInputs}(f, i)$
  - **if** $i \in \mathsf{XorGates}(f)$ **then**
    - $W_i^0 \leftarrow W_a^0 \oplus W_b^0$
  - **else**
    - $(W_i^0, T_{Gi}, T_{Ei}) \leftarrow \mathsf{GbAnd}(W_a^0, W_b^0)$
    - $F_i \leftarrow (T_{Gi}, T_{Ei})$
  - **end if**
  - $W_i^1 \leftarrow W_i^0 \oplus R$
- **for** $i \in \mathsf{Outputs}(f)$ **do**
  - $d_i \leftarrow \mathsf{lsb}(W_i^0)$
- **return** $(\hat{F}, \hat{e}, \hat{d})$

private procedure $\mathsf{GbAnd}(W_a^0, W_b^0)$:
- $p_a \leftarrow \mathsf{lsb}\, W_a^0$; $p_b \leftarrow \mathsf{lsb}\, W_b^0$
- $j \leftarrow \mathsf{NextIndex}()$; $j' \leftarrow \mathsf{NextIndex}()$
- {*First half gate*}
- $T_G \leftarrow H(W_a^0, j) \oplus H(W_a^1, j) \oplus p_b R$
- $W_G^0 \leftarrow H(W_a^0, j) \oplus p_a T_G$
- {*Second half gate*}
- $T_E \leftarrow H(W_b^0, j') \oplus H(W_b^1, j') \oplus W_a^0$
- $W_E^0 \leftarrow H(W_b^0, j') \oplus p_b(T_E \oplus W_a^0)$
- {*Combine halves*}
- $W^0 \leftarrow W_G^0 \oplus W_E^0$
- **return** (W^0, T_G, T_E)

procedure $\mathsf{En}(\hat{e}, \hat{x})$:
- **for** $e_i \in \hat{e}$ **do**
  - $X_i \leftarrow e_i \oplus x_i R$
- **return** $\hat{X}$

procedure $\mathsf{De}(\hat{d}, \hat{Y})$:
- **for** $d_i \in \hat{d}$ **do**
  - $y_i \leftarrow d_i \oplus \mathsf{lsb}\, Y_i$
- **return** $\hat{y}$

procedure $\mathsf{Ev}(\hat{F}, \hat{X})$:
- **for** $i \in \mathsf{Inputs}(\hat{F})$ **do**
  - $W_i \leftarrow X_i$
- **for** $i \notin \mathsf{Inputs}(\hat{F})$ {*in topo. order*} **do**
  - $\{a, b\} \leftarrow \mathsf{GateInputs}(\hat{F}, i)$
  - **if** $i \in \mathsf{XorGates}(\hat{F})$ **then**
    - $W_i \leftarrow W_a \oplus W_b$
  - **else**
    - $s_a \leftarrow \mathsf{lsb}\, W_a$; $s_b \leftarrow \mathsf{lsb}\, W_b$
    - $j_1 \leftarrow \mathsf{NextIndex}()$; $j_2 \leftarrow \mathsf{NextIndex}()$
    - $(T_{Gi}, T_{Ei}) \leftarrow F_i$
    - $W_{Gi} \leftarrow H(W_a, j_1) \oplus s_a T_{Gi}$
    - $W_{Ei} \leftarrow H(W_b, j_2) \oplus s_b(T_{Ei} \oplus W_a)$
    - $W_i \leftarrow W_{Gi} \oplus W_{Ei}$
  - **end if**
- **for** $i \in \mathsf{Outputs}(\hat{F})$ **do**
  - $Y_i \leftarrow W_i$
- **return** $\hat{Y}$

Free XOR

4 calls of H for garbling AND

2 table entries per AND!

2 calls of H for evaluating AND

Part 2: Efficient OTs

http://encrypto.de/code/OTExtension

G. Asharov, Y. Lindell, T. Schneider, M. Zohner:
More efficient oblivious transfer and extensions for faster secure computation.
In ACM CCS'13.

OT - Bad News

- [ImpagliazzoRudich89]: there’s no black-box reduction from OT to OWFs

OT - Bad News

- [ImpagliazzoRudich89]: there's no black-box reduction from OT to OWFs

- Several OT protocols based on public-key cryptography
    - e.g., [NaorPinkas01] yields ~1,000 OTs per second

OT - Bad News

- [ImpagliazzoRudich89]: there's no black-box reduction from OT to OWFs

- Several OT protocols based on public-key cryptography
    - e.g., [NaorPinkas01] yields ~1,000 OTs per second

- Since public-key crypto is expensive, OT was believed to be inefficient

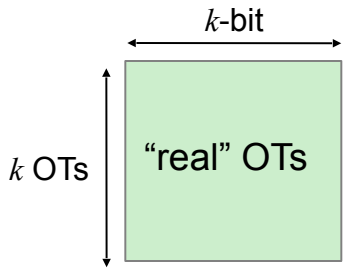
OT - Good News

- [Beaver95]: OTs can be precomputed (only OTP in online phase)

OT - Good News

- [Beaver95]: OTs can be precomputed (only OTP in online phase)

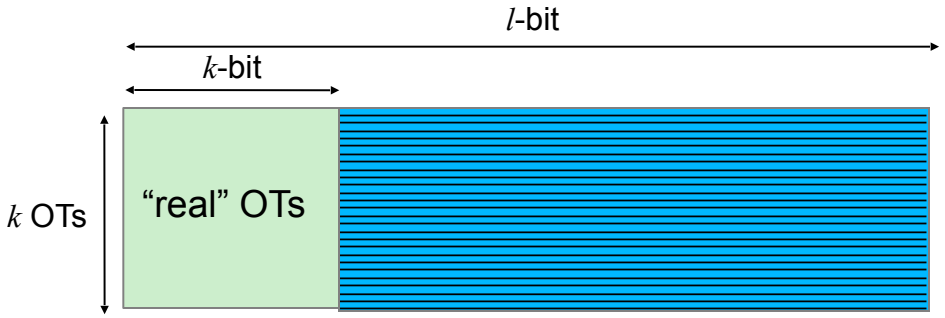
- OT Extensions (similar to hybrid encryption):
  use symmetric crypto to stretch few “real” OTs into longer/many OTs



OT - Good News

- [Beaver95]: OTs can be precomputed (only OTP in online phase)

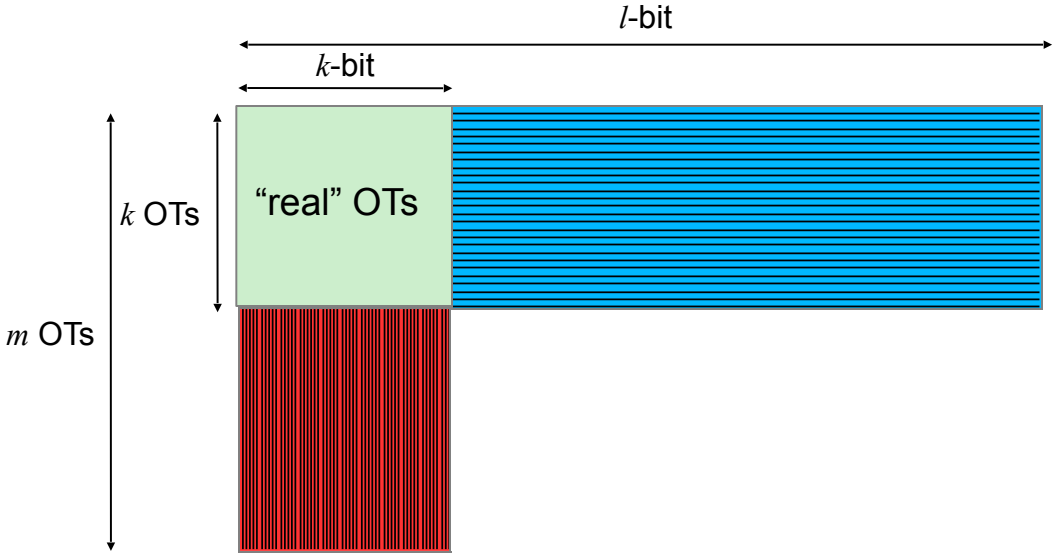
- OT Extensions (similar to hybrid encryption):
  use symmetric crypto to stretch few “real” OTs into longer/many OTs
  - [Beaver96]: OT on long strings from short seeds



OT - Good News

- [Beaver95]: OTs can be precomputed (only OTP in online phase)

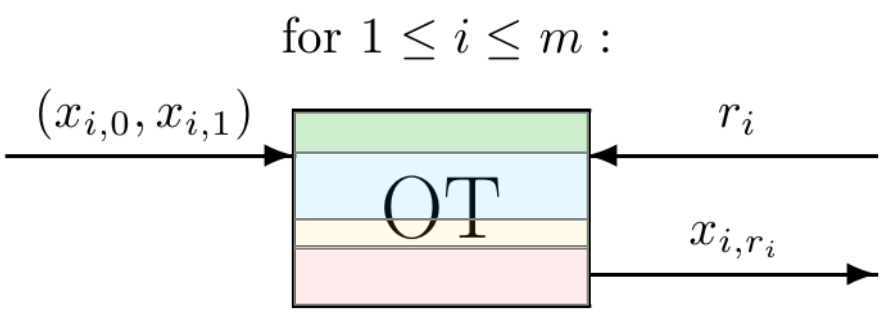
- OT Extensions (similar to hybrid encryption):
  use symmetric crypto to stretch few “real” OTs into longer/many OTs
  - [Beaver96]: OT on long strings from short seeds
  - [IshaiKilianNissimPetrank03]: many OTs from few OTs



OT Extension of [IKNP03] (1)

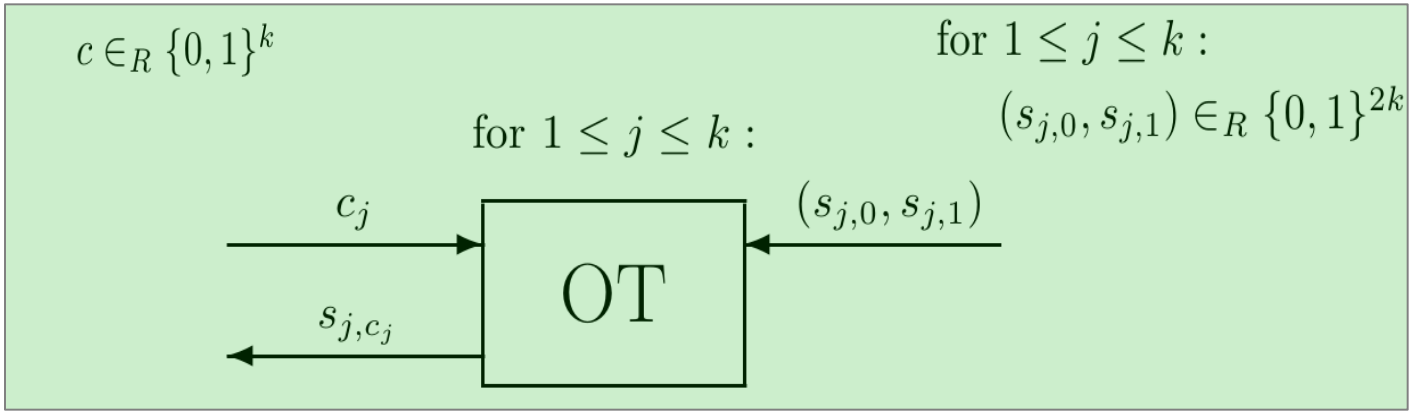
- Alice inputs m pairs of ℓ-bit strings $(x_{i,0}, x_{i,1})$

- Bob inputs m-bit string r and obtains x_{i,r_i} in i-th OT



OT Extension of [IKNP03] (2)

- Alice and Bob perform *k* “real” OTs on random seeds with reverse roles (*k*: security parameter)



OT Extension of [IKNP03] (3)

- Bob generates a random *m* x *k* bit matrix ***T*** and masks his choices *r*

- The matrix is masked with the stretched seeds of the “real” OTs

$$\mathbf{T} \in_R \{0,1\}^{m \times k}$$

for $1 \leq j \leq k$:

$$u_{j,0} = PRG(s_{j,0}) \oplus \mathbf{T}[\mathrm{j}]$$

$$u_{j,1} = PRG(s_{j,1}) \oplus \mathbf{T}[\mathrm{j}] \oplus \mathbf{r}$$

$(u_{j,0}, u_{j,1}),\ 1 \leq i \leq k$ ⟵

for $1 \leq j \leq k$:

$$\mathbf{V}[\mathrm{j}] = u_{j,c_j} \oplus PRG(s_{j,c_j})$$

PRG: pseudo-random generator (instantiated with AES)

OT Extension of [IKNP03] (4)

- Transpose matrices ***V*** and ***T***

- Alice masks her inputs and obliviously sends them to Bob

$\mathbf{V'} = \mathbf{V}^T$	$\mathbf{T'} = \mathbf{T}^T$
for $1 \leq i \leq m$:	
$y_{i,0} = x_{i,0} \oplus H(i, \mathbf{V'}[\mathrm{i}])$	
$y_{i,1} = x_{i,1} \oplus H(i, \mathbf{V'}[\mathrm{i}] \oplus c)$	
$\xrightarrow{(y_{i,0}, y_{i,1}), 1 \leq i \leq m}$	for $1 \leq i \leq m$:
	$x_{i,r_i} = y_{i,r_i} \oplus H(i, \mathbf{T'}[\mathrm{i}])$

H: correlation robust function (instantiated with hash function)

Computation Complexity of OT Extension

m pairs $(x_{i,0}, x_{i,1}) \in \{0,1\}^{2\ell}$

$\mathbf{r} = (r_1, ..., r_m) \in \{0,1\}^m$

for $1 \le j \le k$:

$c_j \in_R \{0,1\}$ → OT ← $(s_{j,0}, s_{j,1}) \in_R \{0,1\}^{2k}$

s_{j,c_j} ←

$\mathbf{T} \in_R \{0,1\}^{m \times k}$

for $1 \le j \le k$:

$u_{j,0} = PRG(s_{j,0}) \oplus \mathbf{T}[j]$

$u_{j,1} = PRG(s_{j,1}) \oplus \mathbf{T}[j] \oplus \mathbf{r}$

$(u_{j,0}, u_{j,1}),\ 1 \le i \le k$ ←

for $1 \le j \le k$:

$\mathbf{V}[j] = u_{j,c_j} \oplus PRG(s_{j,c_j})$

$\mathbf{V'} = \mathbf{V}^T$ $\quad$ $\mathbf{T'} = \mathbf{T}^T$

for $1 \le i \le m$:

$y_{i,0} = x_{i,0} \oplus H(i, \mathbf{V'}[i])$

$y_{i,1} = x_{i,1} \oplus H(i, \mathbf{V'}[i] \oplus c)$

$(y_{i,0}, y_{i,1}), 1 \le i \le m$ →

for $1 \le i \le m$:

$x_{i,r_i} = y_{i,r_i} \oplus H(i, \mathbf{T'}[i])$

Computation Complexity of OT Extension

m pairs $(x_{i,0}, x_{i,1}) \in \{0,1\}^{2\ell}$

$\mathbf{r} = (r_1, ..., r_m) \in \{0,1\}^m$

for $1 \le j \le k$:

$c_j \in_R \{0,1\}$ → OT ← $(s_{j,0}, s_{j,1}) \in_R \{0,1\}^{2k}$

s_{j,c_j} ←

$\mathbf{T} \in_R \{0,1\}^{m \times k}$

for $1 \le j \le k$:

$u_{j,0} = PRG(s_{j,0}) \oplus \mathbf{T}[j]$

$u_{j,1} = PRG(s_{j,1}) \oplus \mathbf{T}[j] \oplus \mathbf{r}$

$(u_{j,0}, u_{j,1}),\ 1 \le i \le k$ ←

for $1 \le j \le k$:

$\mathbf{V}[j] = u_{j,c_j} \oplus PRG(s_{j,c_j})$

$\mathbf{V'} = \mathbf{V}^T$ $\quad$ $\mathbf{T'} = \mathbf{T}^T$

for $1 \le i \le m$:

$y_{i,0} = x_{i,0} \oplus H(i, \mathbf{V'}[i])$

$y_{i,1} = x_{i,1} \oplus H(i, \mathbf{V'}[i] \oplus c)$

$(y_{i,0}, y_{i,1}), 1 \le i \le m$ →

for $1 \le i \le m$:

$x_{i,r_i} = y_{i,r_i} \oplus H(i, \mathbf{T'}[i])$

Per OT:

1	# PRG evaluations	2
2	# H evaluations	1

Computation Complexity of OT Extension

m pairs $(x_{i,0}, x_{i,1}) \in \{0,1\}^{2\ell}$

$\mathbf{r} = (r_1, ..., r_m) \in \{0,1\}^m$

for $1 \le j \le k$:

$c_j \in_R \{0,1\}$ → OT ← $(s_{j,0}, s_{j,1}) \in_R \{0,1\}^{2k}$

OT → s_{j,c_j}

$\mathbf{T} \in_R \{0,1\}^{m \times k}$

for $1 \le j \le k$:

$u_{j,0} = PRG(s_{j,0}) \oplus \mathbf{T}[j]$

$u_{j,1} = PRG(s_{j,1}) \oplus \mathbf{T}[j] \oplus \mathbf{r}$

$(u_{j,0}, u_{j,1}),\ 1 \le i \le k$ ←

for $1 \le j \le k$:

$\mathbf{V}[j] = u_{j,c_j} \oplus PRG(s_{j,c_j})$

$\mathbf{V'} = \mathbf{V}^T$ $\quad$ $\mathbf{T'} = \mathbf{T}^T$

for $1 \le i \le m$:

$y_{i,0} = x_{i,0} \oplus H(i, \mathbf{V'}[i])$

$y_{i,1} = x_{i,1} \oplus H(i, \mathbf{V'}[i] \oplus c)$

$(y_{i,0}, y_{i,1}), 1 \le i \le m$ →

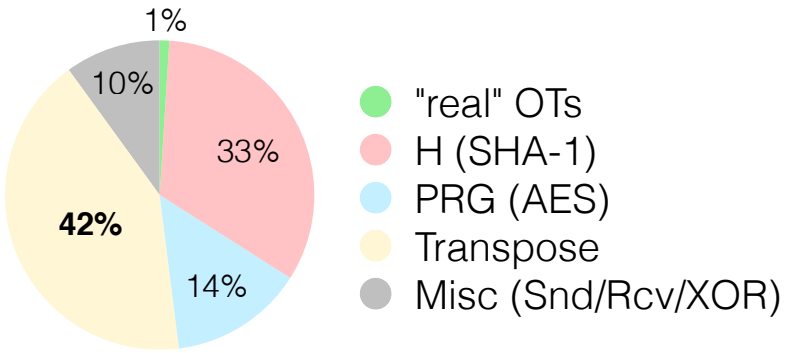
for $1 \le i \le m$:

$x_{i,r_i} = y_{i,r_i} \oplus H(i, \mathbf{T'}[i])$

Per OT:

1	# PRG evaluations	2
2	# H evaluations	1

Time distribution for 10 Million OTs (in 21s):
2.1 microseconds per OT



Computation Complexity of OT Extension

m pairs $(x_{i,0}, x_{i,1}) \in \{0,1\}^{2\ell}$

$\mathbf{r} = (r_1, ..., r_m) \in \{0,1\}^m$

for $1 \le j \le k$:

$c_j \in_R \{0,1\}$ → OT ← $(s_{j,0}, s_{j,1}) \in_R \{0,1\}^{2k}$

s_{j,c_j} ←

$\mathbf{T} \in_R \{0,1\}^{m \times k}$

for $1 \le j \le k$:

$u_{j,0} = PRG(s_{j,0}) \oplus \mathbf{T}[j]$

$u_{j,1} = PRG(s_{j,1}) \oplus \mathbf{T}[j] \oplus \mathbf{r}$

$(u_{j,0}, u_{j,1}),\ 1 \le i \le k$ ←

for $1 \le j \le k$:

$\mathbf{V}[j] = u_{j,c_j} \oplus PRG(s_{j,c_j})$

$\mathbf{V'} = \mathbf{V}^T$ $\quad$ $\mathbf{T'} = \mathbf{T}^T$

for $1 \le i \le m$:

$y_{i,0} = x_{i,0} \oplus H(i, \mathbf{V'}[i])$

$y_{i,1} = x_{i,1} \oplus H(i, \mathbf{V'}[i] \oplus c)$

$(y_{i,0}, y_{i,1}), 1 \le i \le m$ →

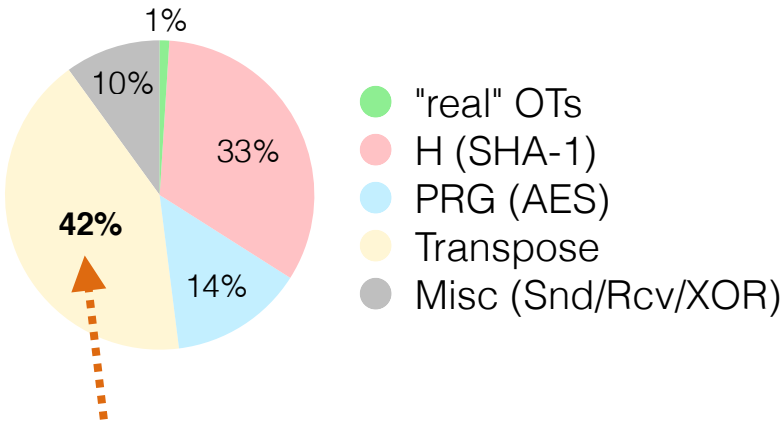
for $1 \le i \le m$:

$x_{i,r_i} = y_{i,r_i} \oplus H(i, \mathbf{T'}[i])$

Per OT:

1	# PRG evaluations	2
2	# H evaluations	1

Time distribution for 10 Million OTs (in 21s):
2.1 microseconds per OT



Non-crypto part was bottleneck!!!

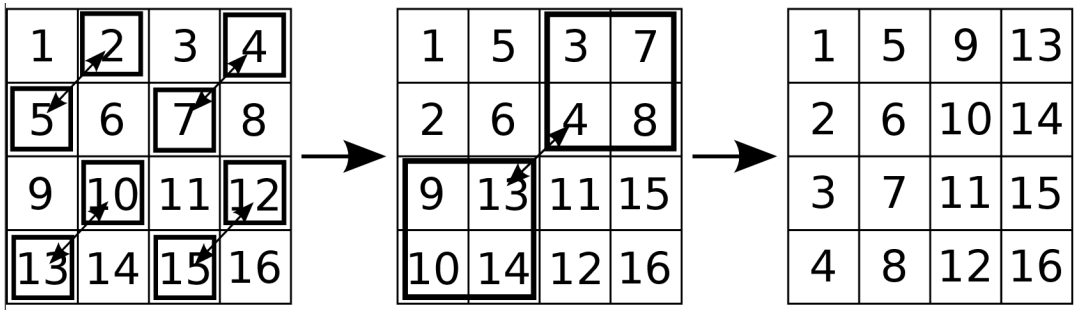
Algorithmic Optimization: Efficient Matrix Transposition

- Naive matrix transposition performs *mk* load/process/store operations

Algorithmic Optimization: Efficient Matrix Transposition

- Naive matrix transposition performs *mk* load/process/store operations

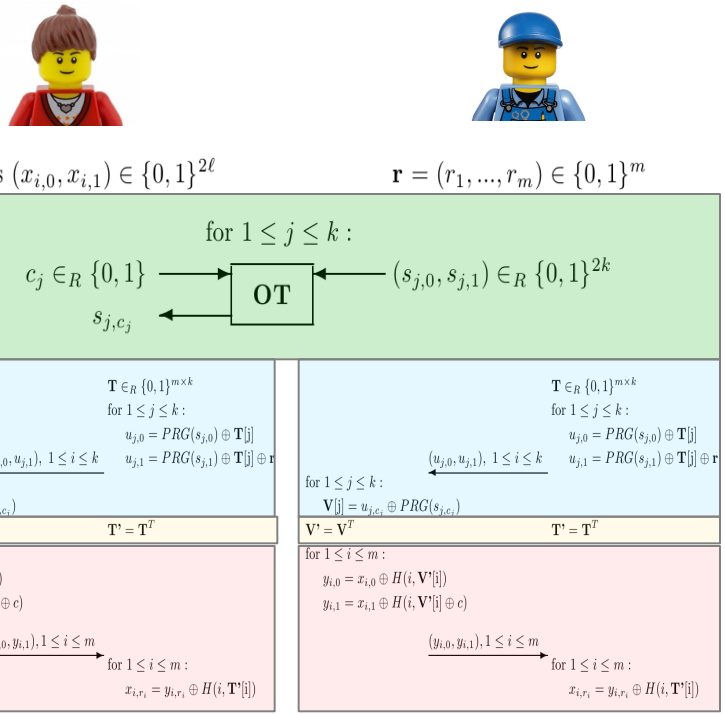
- Eklundh's algorithm reduces number of operations to O(*m* $\log_2$ *k*) swaps
  - Swap whole registers instead of bits
  - Transposing 10 times faster



Algorithmic Optimization: Parallelization

- OT extension can easily be parallelized by splitting the ***T*** matrix into sub-matrices

- Since columns are independent, OT is highly parallelizable



Communication Complexity of OT Extension

m pairs $(x_{i,0}, x_{i,1}) \in \{0,1\}^{2\ell}$

$\mathbf{r} = (r_1, ..., r_m) \in \{0,1\}^m$

for $1 \le j \le k$:

$c_j \in_R \{0,1\}$ → OT ← $(s_{j,0}, s_{j,1}) \in_R \{0,1\}^{2k}$

OT → s_{j,c_j}

$\mathbf{T} \in_R \{0,1\}^{m \times k}$

for $1 \le j \le k$:

$u_{j,0} = PRG(s_{j,0}) \oplus \mathbf{T}[j]$

$u_{j,1} = PRG(s_{j,1}) \oplus \mathbf{T}[j] \oplus \mathbf{r}$

$(u_{j,0}, u_{j,1}),\ 1 \le i \le k$ ←

for $1 \le j \le k$:

$\mathbf{V}[j] = u_{j,c_j} \oplus PRG(s_{j,c_j})$

$\mathbf{V'} = \mathbf{V}^T$ $\quad$ $\mathbf{T'} = \mathbf{T}^T$

for $1 \le i \le m$:

$y_{i,0} = x_{i,0} \oplus H(i, \mathbf{V'}[i])$

$y_{i,1} = x_{i,1} \oplus H(i, \mathbf{V'}[i] \oplus c)$

$(y_{i,0}, y_{i,1}), 1 \le i \le m$ →

for $1 \le i \le m$:

$x_{i,r_i} = y_{i,r_i} \oplus H(i, \mathbf{T'}[i])$

Per OT:

2ℓ Bits sent $2k$

Communication Complexity of OT Extension

m pairs $(x_{i,0}, x_{i,1}) \in \{0,1\}^{2\ell}$

$\mathbf{r} = (r_1, ..., r_m) \in \{0,1\}^m$

for $1 \le j \le k$:

$c_j \in_R \{0,1\}$ → OT ← $(s_{j,0}, s_{j,1}) \in_R \{0,1\}^{2k}$

OT → s_{j,c_j}

$\mathbf{T} \in_R \{0,1\}^{m \times k}$

for $1 \le j \le k$:

$u_{j,0} = PRG(s_{j,0}) \oplus \mathbf{T}[j]$

$u_{j,1} = PRG(s_{j,1}) \oplus \mathbf{T}[j] \oplus \mathbf{r}$

$(u_{j,0}, u_{j,1}),\ 1 \le i \le k$ ←

for $1 \le j \le k$:

$\mathbf{V}[j] = u_{j,c_j} \oplus PRG(s_{j,c_j})$

$\mathbf{V'} = \mathbf{V}^T$ $\quad$ $\mathbf{T'} = \mathbf{T}^T$

for $1 \le i \le m$:

$y_{i,0} = x_{i,0} \oplus H(i, \mathbf{V'}[i])$

$y_{i,1} = x_{i,1} \oplus H(i, \mathbf{V'}[i] \oplus c)$

$(y_{i,0}, y_{i,1}), 1 \le i \le m$ →

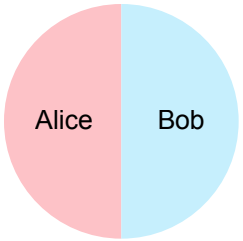
for $1 \le i \le m$:

$x_{i,r_i} = y_{i,r_i} \oplus H(i, \mathbf{T'}[i])$

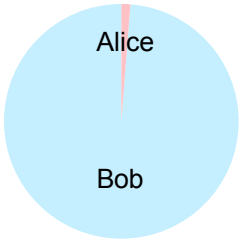
Per OT:

2ℓ Bits sent $2k$

Yao: $\ell = k = 80$



GMW: $\ell = 1,\ k = 80$



Protocol Optimization: General OT Extension

- Instead of generating a random ***T*** matrix, we derive it from $s_{j,0}$

- Reduces data sent by Bob by factor 2

$\mathbf{T} \in_R \{0,1\}^{m \times k}$

for $1 \le j \le k$:

$u_{j,0} = PRG(s_{j,0}) \oplus \mathbf{T}[\mathrm{j}]$

$u_{j,1} = PRG(s_{j,1}) \oplus \mathbf{T}[\mathrm{j}] \oplus \mathbf{r}$

$(u_{j,0}, u_{j,1}),\ 1 \le i \le k$ ←

for $1 \le j \le k$:

$\mathbf{V}[\mathrm{j}] = u_{j,c_j} \oplus PRG(s_{j,c_j})$

for $1 \le j \le k$:

$\mathbf{T}[\mathrm{j}] = PRG(s_{j,0})$

$u_j = PRG(s_{j,1}) \oplus \mathbf{T}[\mathrm{j}] \oplus \mathbf{r}$

$u_j,\ 1 \le i \le k$ ←

for $1 \le j \le k$:

$\mathbf{V}[\mathrm{j}] = c_j u_j \oplus PRG(s_{j,c_j})$

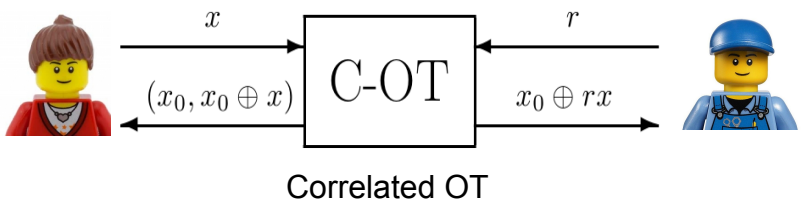
Specific OT Functionalities

- Secure computation protocols often require a specific OT functionality

Specific OT Functionalities

- Secure computation protocols often require a specific OT functionality
  - Yao with free XORs requires strings x_0, x_1 to be XOR-correlated

- Correlated OT: random x_0 and $x_1 = x_0 \oplus x$



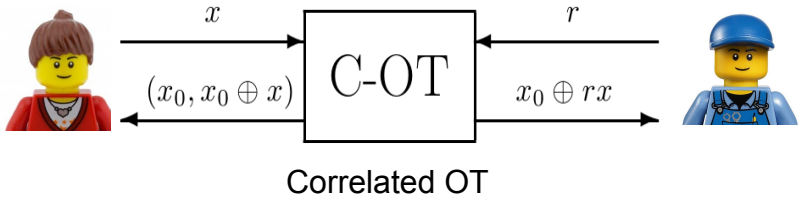
Correlated OT

e.g., for Yao

Specific OT Functionalities

- Secure computation protocols often require a specific OT functionality
  - Yao with free XORs requires strings x_0, x_1 to be XOR-correlated
  - GMW with multiplication triples can use random strings

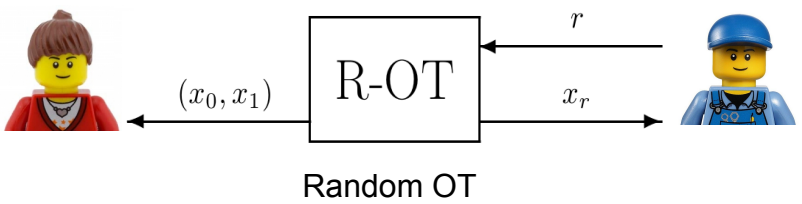
- Correlated OT: random x_0 and $x_1 = x_0 \oplus x$



Correlated OT

e.g., for Yao

- Random OT: random x_0 and x_1



Random OT

e.g., for GMW

Specific OT Functionalities: Correlated OT (C-OT)

- Choose $x_{i,0}$ as random output of H (modeled as RO here)

- Compute $x_{i,1}$ as $x_{i,0} \oplus x_i$ to obliviously transfer XOR-correlated values

- Reduces data sent by Alice by factor 2

$\mathbf{V'} = \mathbf{V}^T$ $\qquad\qquad$ $\mathbf{T'} = \mathbf{T}^T$

for $1 \le i \le m$:

$y_{i,0} = x_{i,0} \oplus H(i, \mathbf{V'}[\mathrm{i}])$

$y_{i,1} = x_{i,1} \oplus H(i, \mathbf{V'}[\mathrm{i}] \oplus c)$

$(y_{i,0}, y_{i,1}), 1 \le i \le m \longrightarrow$

for $1 \le i \le m$:

$x_{i,r_i} = y_{i,r_i} \oplus H(i, \mathbf{T'}[\mathrm{i}])$

$\mathbf{V'} = \mathbf{V}^T$ $\qquad\qquad$ $\mathbf{T'} = \mathbf{T}^T$

for $1 \le i \le m$:

$x_{i,0} = H(i, \mathbf{V'}[\mathrm{i}])$

$x_{i,1} = x_{i,0} \oplus x_i$

$y_i = x_{i,1} \oplus H(i, \mathbf{V'}[\mathrm{i}] \oplus c)$

$y_i, 1 \le i \le m \longrightarrow$

for $1 \le i \le m$:

$x_{i,r_i} = r_i y_i \oplus H(i, \mathbf{T'}[\mathrm{i}])$

Specific OT Functionalities: Random OT (R-OT)

- Choose $x_{i,0}$ and $x_{i,1}$ as random outputs of H (modeled as RO here)

- No data sent by Alice

$\mathbf{V'} = \mathbf{V}^T$

for $1 \le i \le m$:

$y_{i,0} = x_{i,0} \oplus H(i, \mathbf{V'}[\mathrm{i}])$

$y_{i,1} = x_{i,1} \oplus H(i, \mathbf{V'}[\mathrm{i}] \oplus c)$

$(y_{i,0}, y_{i,1}), 1 \le i \le m \longrightarrow$

$\mathbf{T'} = \mathbf{T}^T$

for $1 \le i \le m$:

$x_{i,r_i} = y_{i,r_i} \oplus H(i, \mathbf{T'}[\mathrm{i}])$

$\mathbf{V'} = \mathbf{V}^T$

for $1 \le i \le m$:

$x_{i,0} = H(i, \mathbf{V'}[\mathrm{i}])$

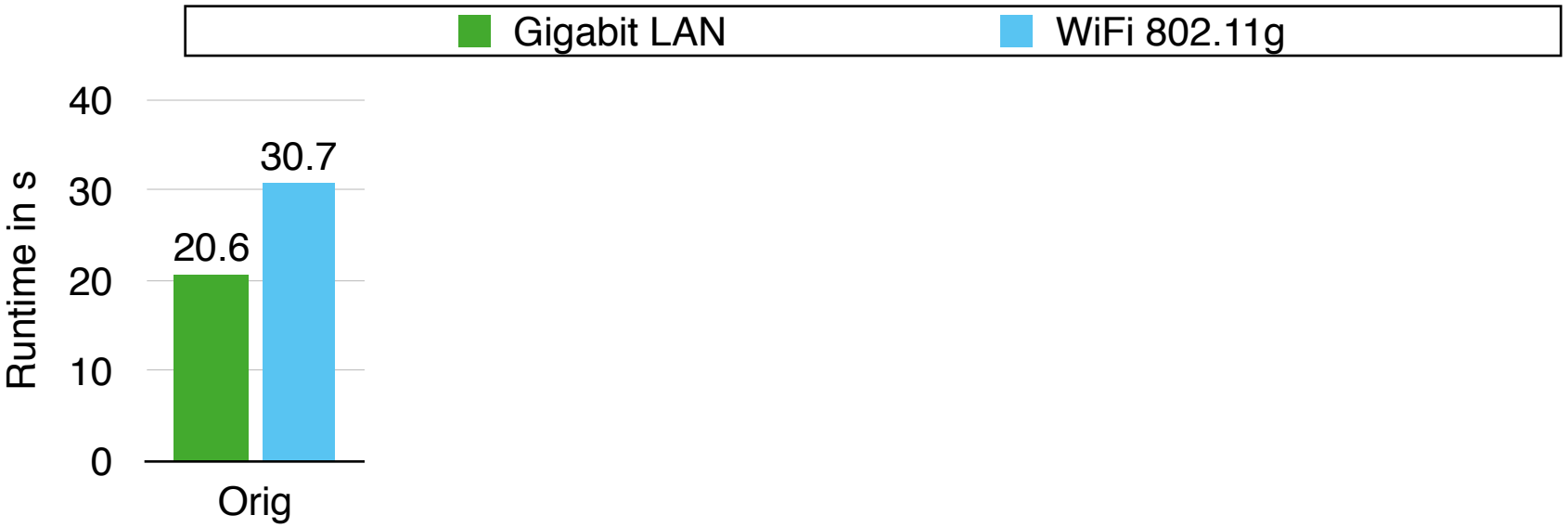
$x_{i,1} = H(i, \mathbf{V'}[\mathrm{i}] \oplus c)$

$\mathbf{T'} = \mathbf{T}^T$

for $1 \le i \le m$:

$x_{i,r_i} = H(i, \mathbf{T'}[\mathrm{i}])$

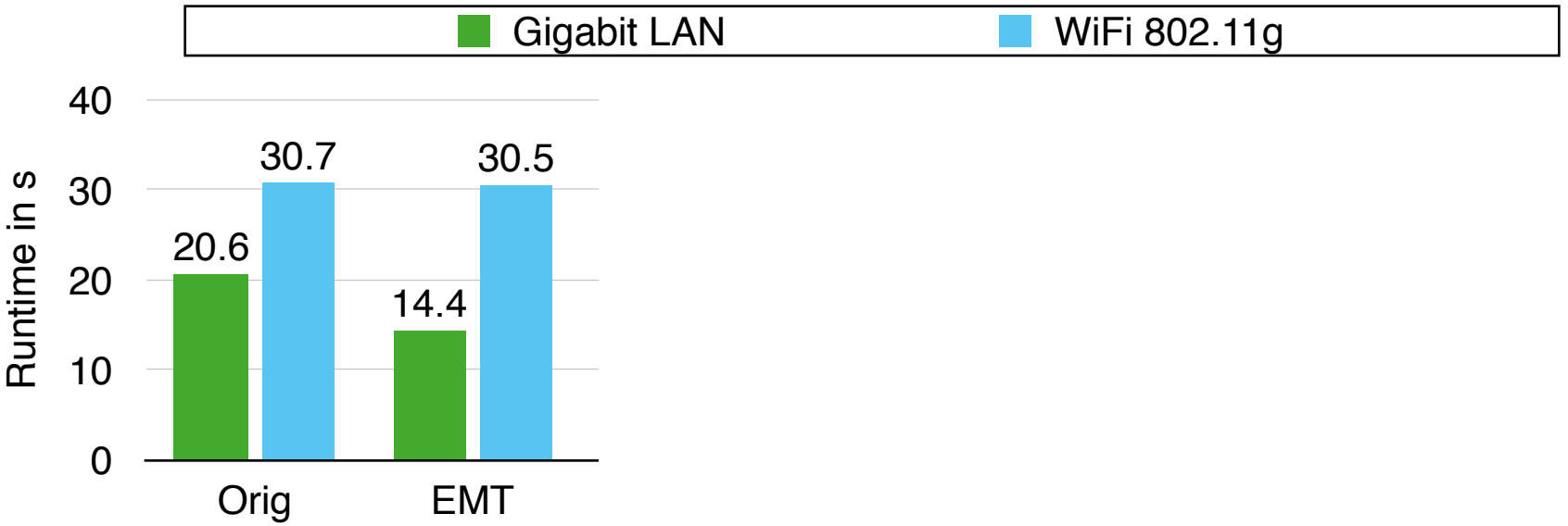
Performance Evaluation: Original Implementation



- C++ implementation of [SZ13] implementing OT extension of [IKNP03]

- Performance for 10 Million OTs on 80-bit strings

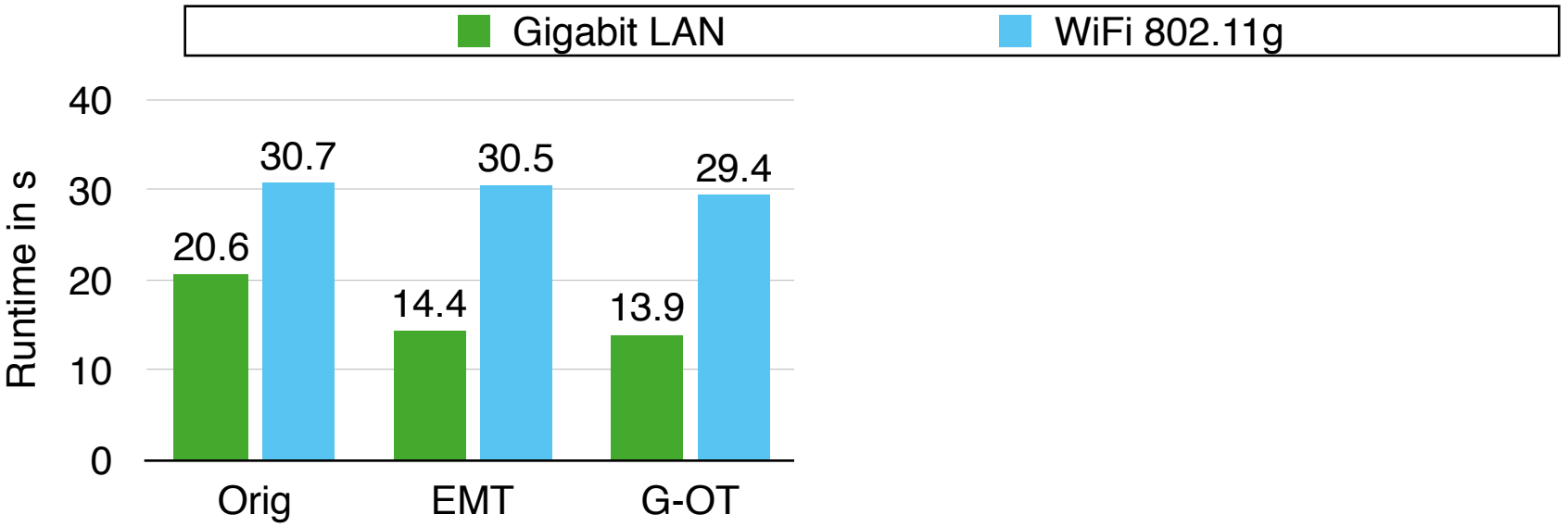
Performance Evaluation: Efficient Matrix Transposition



- Efficient matrix transposition – improves computation

- Only decreases runtime in LAN where computation is the bottleneck

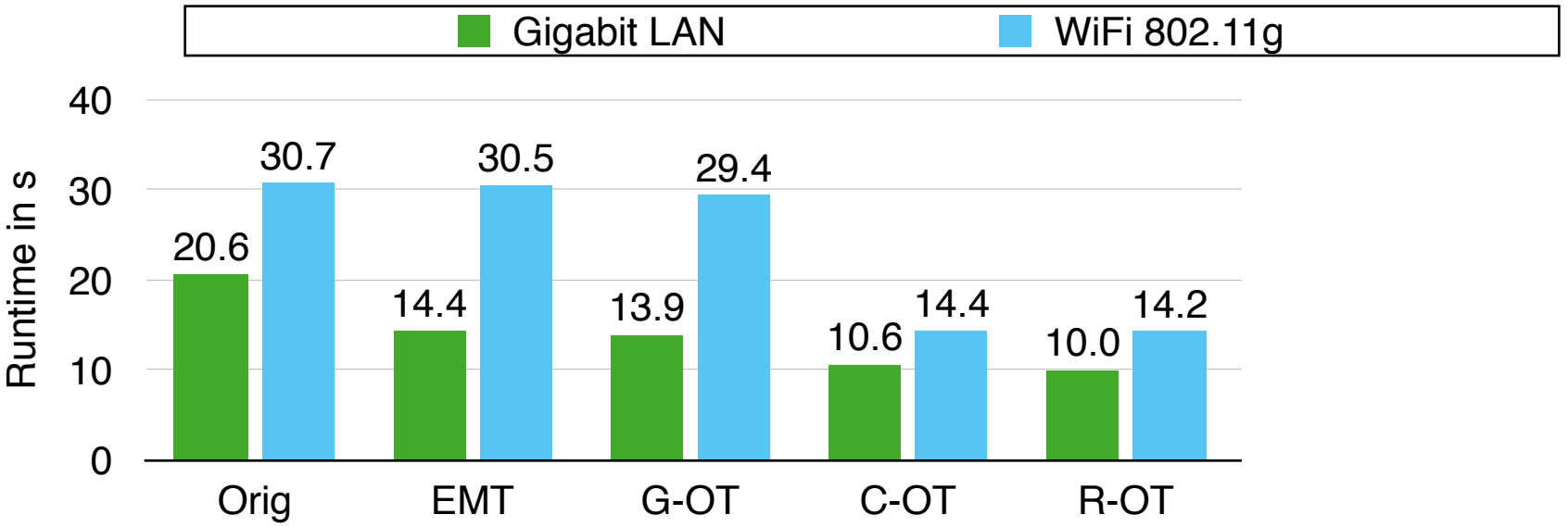
Performance Evaluation: General OT



- Generate T matrix from seeds – improves communication Bob → Alice

- Runtimes only slightly faster (bottleneck: communication Alice → Bob)

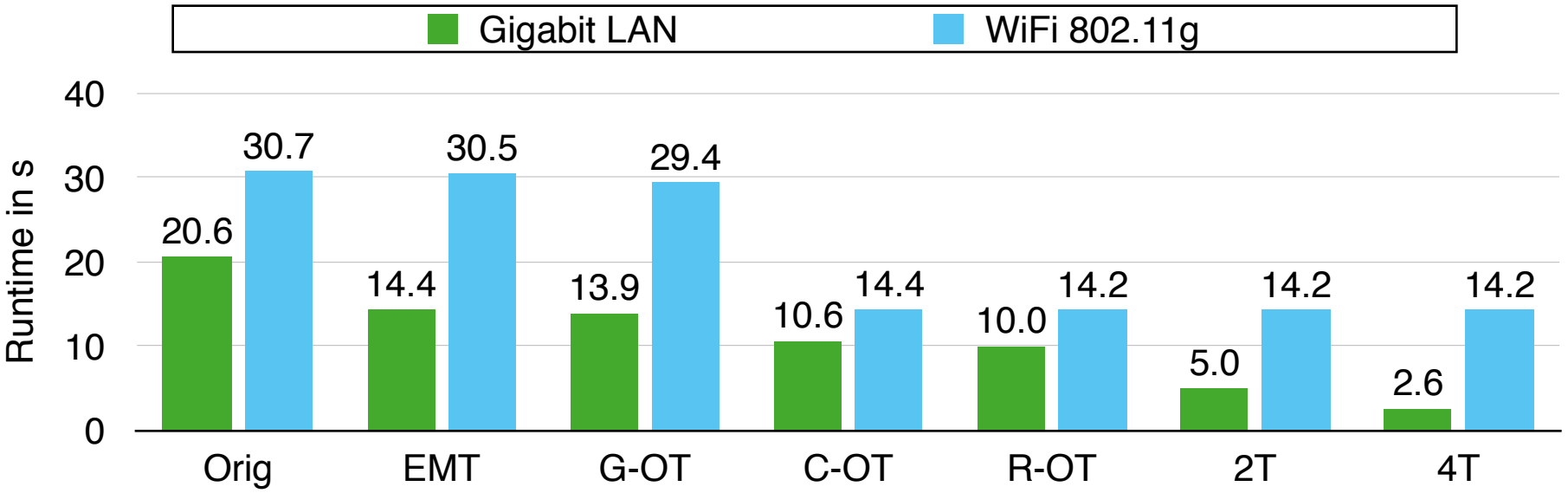
Performance Evaluation: Correlated/Random OT



- Correlated/Random OT – improved communication Alice → Bob

- WiFi runtime faster by factor 2 (bottleneck: communication Bob → Alice)

Performance Evaluation: Parallelization

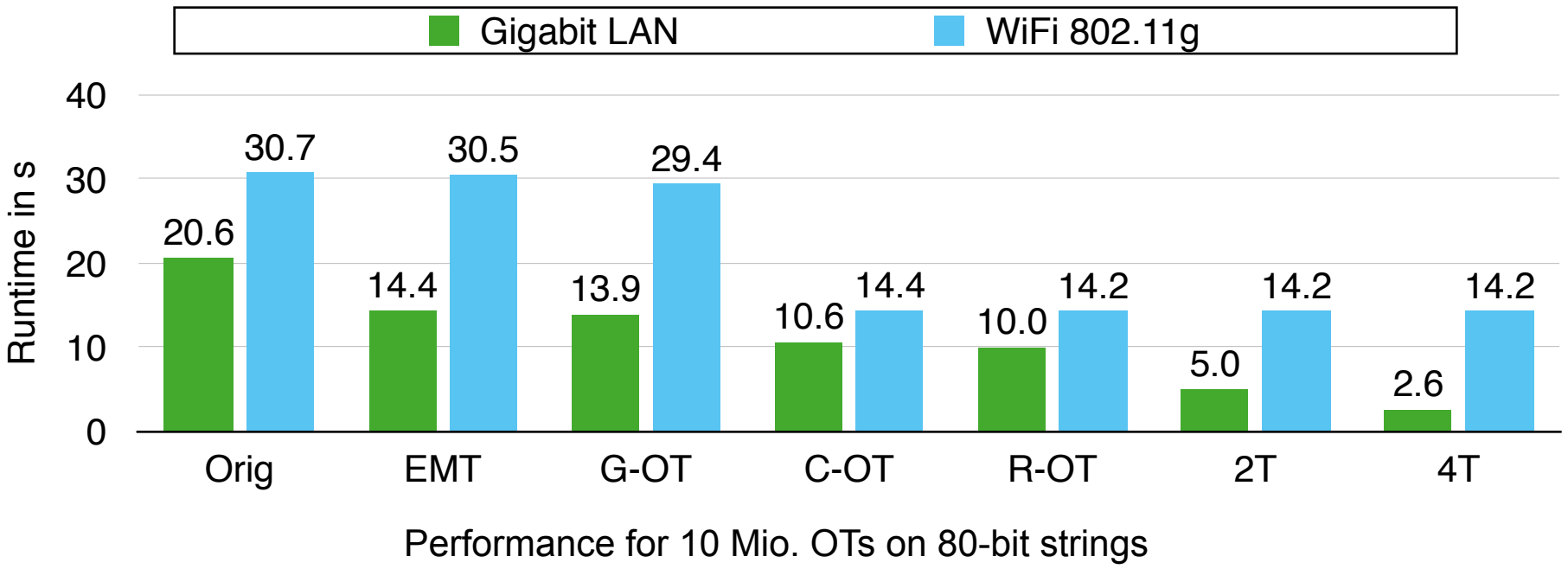


- Parallel OT extension with 2 and 4 threads – improved computation

- LAN runtime decreases linear in # of threads

- WiFi runtime remains the same (bottleneck: communication)

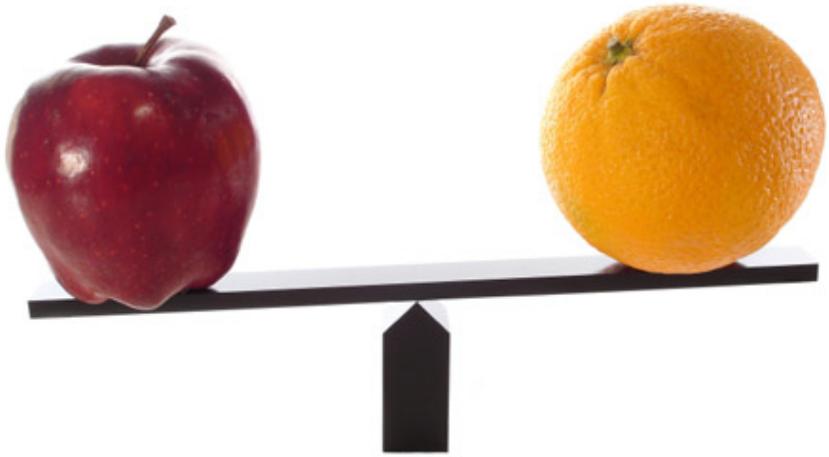
Performance Evaluation: Summary



- OT is very efficient

- **Communication** is the **bottleneck** for OT (even without using AES-NI)

Part 3: Efficient Circuits and Yao vs. GMW



T. Schneider, M. Zohner:
GMW vs. Yao? Efficient secure two-party computation with low depth circuits.
In FC'13.

Yao - the Apple

How to eat an apple?

Yao - the Apple

How to eat an apple?

bite-by-bite

Yao - the Apple

How to eat an apple?

bite-by-bite

+ Yao has constant #rounds

Yao - the Apple

How to eat an apple?

bite-by-bite

\+ Yao has constant #rounds

\- Evaluating a garbled gate requires symmetric crypto in the online phase

GMW - the Orange

How to eat an orange?

GMW - the Orange

How to eat an orange?

1) peel (almost all the effort)

GMW - the Orange

How to eat an orange?

1) peel (almost all the effort)

2) eat (easy)

GMW - the Orange

How to eat an orange?

1) peel (almost all the effort)

Setup phase:

- precompute multiplication triples for each AND gate using 2 R-OTs and constant #rounds

+ no need to know function, only max. #ANDs

2) eat (easy)

GMW - the Orange

How to eat an orange?

1) peel (almost all the effort)

Setup phase:

- precompute multiplication triples for each AND gate using 2 R-OTs and constant #rounds

+ no need to know function, only max. #ANDs

2) eat (easy)

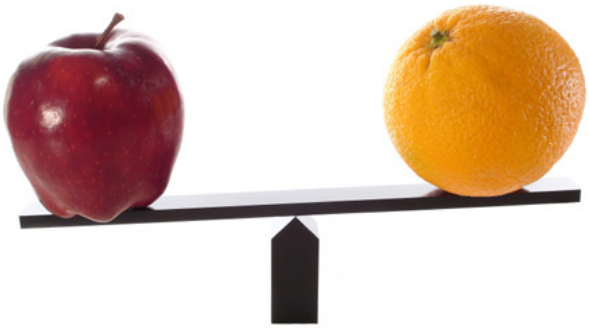
Online phase:

+ evaluating circuit needs OTP operations only

- communication per layer of AND gates

Yao vs. GMW

Yao

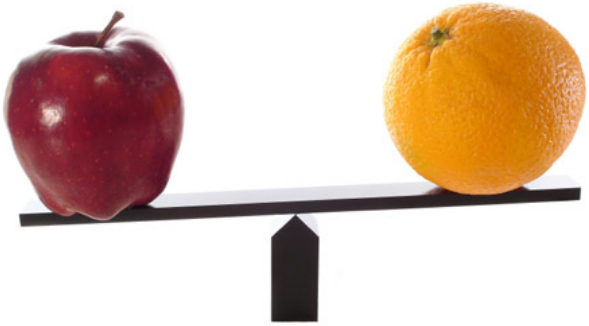


GMW

t: symmetric security parameter

Yao vs. GMW

Yao



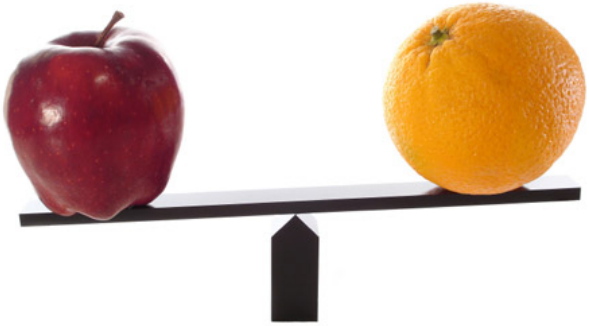
Free XOR

GMW

t: symmetric security parameter

Yao vs. GMW

Yao



GMW

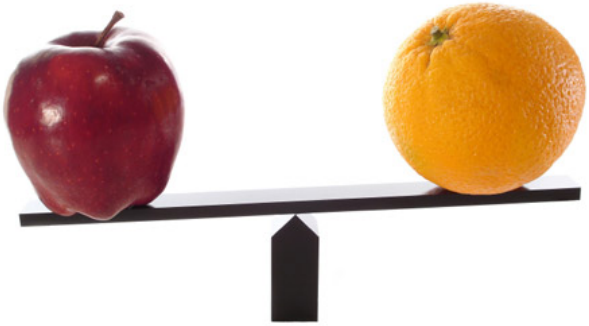
Free XOR

S: 4, R: 2 (online) symmetric crypto per AND setup: S: 6, R: 6

t: symmetric security parameter

Yao vs. GMW

Yao



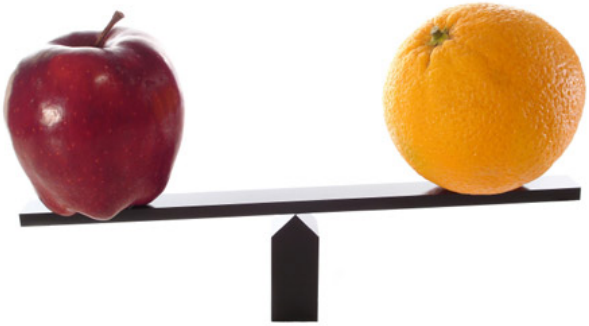
GMW

Free XOR

S: 4, R: 2 (online)	symmetric crypto per AND	setup: S: 6, R: 6
S→R: 2*t*	communication [bit] per AND	setup: S→R:t || R→S:*t* online: S→R:2 || R→S:*2*

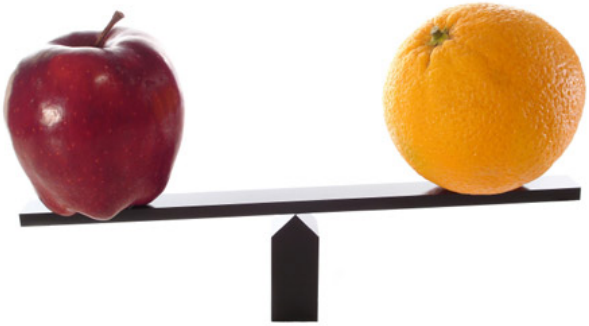
t: symmetric security parameter

Yao vs. GMW

Yao		GMW
	 Free XOR	
S: 4, R: 2 (online)	symmetric crypto per AND	setup: S: 6, R: 6
S→R: 2*t*	communication [bit] per AND	setup: S→R:t || R→S:*t* online: S→R:2 || R→S:*2*
O(1)	rounds	setup: O(1) online: O(ANDdepth(f))

t: symmetric security parameter

Yao vs. GMW

Yao		GMW
	 Free XOR	
S: 4, R: 2 (online)	symmetric crypto per AND	setup: S: 6, R: 6
S→R: 2*t*	communication [bit] per AND	setup: S→R:t || R→S:*t* online: S→R:2 || R→S:*2*
O(1)	rounds	setup: O(1) online: O(ANDdepth(f))
t	memory per wire [bit]	1

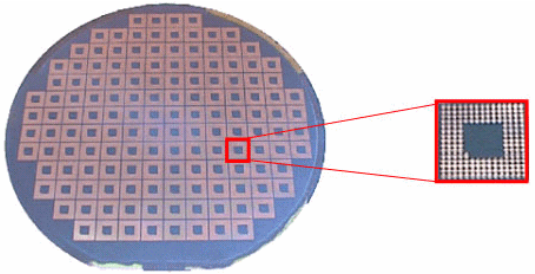
t: symmetric security parameter

Efficient Circuit Constructions for Secure Computation

Classical circuit design:

- few gates (⇒ small chip area)

- low depth (⇒ high clock frequency)

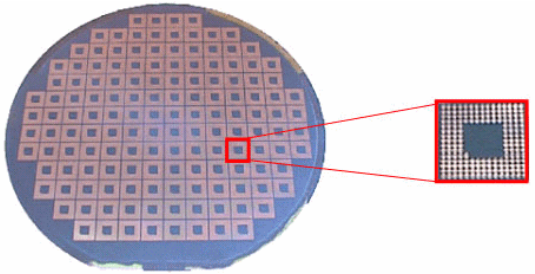


Efficient Circuit Constructions for Secure Computation

Classical circuit design:

- few gates (⇒ small chip area)

- low depth (⇒ high clock frequency)



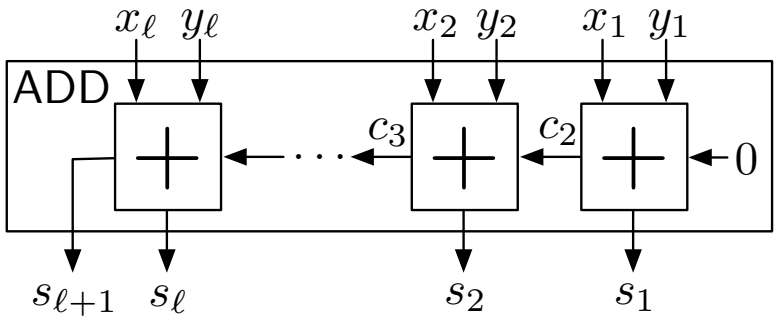
Circuits for secure computation:

- low ANDsize (#non-XORs ⇒ communication and symmetric crypto)

- low ANDdepth (#rounds in GMW's online phase)

Example Circuit: Addition

Ripple-Carry-Adder



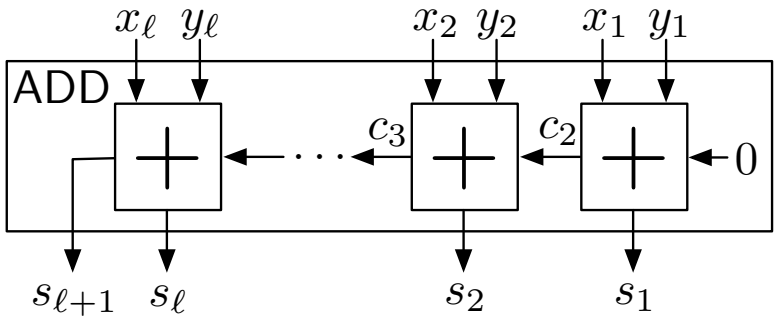
$s_i = x_i \oplus y_i \oplus c_i$

$c_{i+1} = ((x_i \oplus y_i) \wedge (x_i \oplus c_i)) \oplus x_i$ [BoyarPeraltaPochuev00]

ANDsize = ℓ, ANDdepth = ℓ

Example Circuit: Addition

Ripple-Carry-Adder

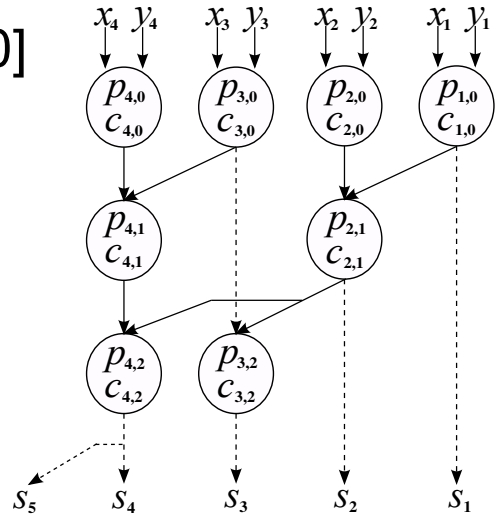


$s_i = x_i \oplus y_i \oplus c_i$

$c_{i+1} = ((x_i \oplus y_i) \wedge (x_i \oplus c_i)) \oplus x_i$ [BoyarPeraltaPochuev00]

ANDsize = ℓ, ANDdepth = ℓ

Ladner-Fischer-Adder [LF80]



$p_{i,0}=x_i \oplus y_i$, $c_{i,0}=x_i \wedge y_i$

$p_{i,j}=p_{i,j-1} \wedge p_{k,j-1}$

$c_{i,j}=(p_{i,j-1} \wedge c_{k,j-1}) \vee c_{i,j-1}$

ANDsize = ℓ+1.25 ℓ $\log_2(\ell)$, **ANDdepth = 1+2 $\log_2(\ell)$**

Example Circuits Summarized in [SchneiderZohner13]

Circuit	Size **S**	Depth **D**
Addition		
Ripple-carry ADD/SUB$^{\ell}_{RC}$	ℓ	ℓ
Ladner-Fischer ADD$^{\ell}_{LF}$	$1.25\ell\lceil\log_2 \ell\rceil + \ell$	$2\lceil\log_2 \ell\rceil + 1$
LF subtraction SUB$^{\ell}_{LF}$	$1.25\ell\lceil\log_2 \ell\rceil + 2\ell$	$2\lceil\log_2 \ell\rceil + 2$
Carry-save ADD$^{(\ell,3)}_{CSA}$	$\ell + \mathbf{S}(\text{ADD}^{\ell})$	$\mathbf{D}(\text{ADD}^{\ell})+1$
RC network ADD$^{(\ell,n)}_{RC}$	$\ell n - \ell + n - \lceil\log_2 n\rceil - 1$	$\lceil\log_2 n - 1\rceil + \ell$
CSA network ADD$^{(\ell,n)}_{CSA}$	$\ell n - 2\ell + n - \lceil\log_2 n\rceil$ $+\mathbf{S}(\text{ADD}^{\ell+\lceil\log_2 n\rceil}_{LF})$	$\lceil\log_2 n - 1\rceil$ $+\mathbf{D}(\text{ADD}^{\ell+\lceil\log_2 n\rceil}_{LF})$
Multiplication		
RCN school method MUL$^{\ell}_{RC}$	$2\ell^2 - \ell$	$2\ell - 1$
CSN school method MUL$^{\ell}_{CSN}$	$2\ell^2 + 1.25\ell\lceil\log_2 \ell\rceil - \ell + 2$	$3\lceil\log_2 \ell\rceil + 4$
RC squaring SQR$^{\ell}_{RC}$	$\ell^2 - \ell$	$2\ell - 3$
LF squaring SQR$^{\ell}_{LF}$	$\ell^2 + 1.25\ell\lceil\log_2 \ell\rceil - 1.5\ell - 2$	$3\lceil\log_2 \ell\rceil + 3$
Comparison		
Equality EQ$^{\ell}$	$\ell - 1$	$\lceil\log_2 \ell\rceil$
Sequential greater than GT$^{\ell}_{S}$	ℓ	ℓ
D&C greater than GT$^{\ell}_{DC}$	$3\ell - \lceil\log_2 \ell\rceil - 2$	$\lceil\log_2 \ell\rceil + 1$
Selection		
Multiplexer MUX$^{\ell}$	ℓ	1

Can trade-off larger size for better depth.

Example Circuits Summarized in [SchneiderZohner13]

Circuit	Size **S**	Depth **D**
Addition		
Ripple-carry ADD/SUB$^{\ell}_{RC}$	ℓ	ℓ
Ladner-Fischer ADD$^{\ell}_{LF}$	$1.25\ell\lceil\log_2 \ell\rceil + \ell$	$2\lceil\log_2 \ell\rceil + 1$
LF subtraction SUB$^{\ell}_{LF}$	$1.25\ell\lceil\log_2 \ell\rceil + 2\ell$	$2\lceil\log_2 \ell\rceil + 2$
Carry-save ADD$^{(\ell,3)}_{CSA}$	$\ell + \mathbf{S}(\text{ADD}^{\ell})$	$\mathbf{D}(\text{ADD}^{\ell})+1$
RC network ADD$^{(\ell,n)}_{RC}$	$\ell n - \ell + n - \lceil\log_2 n\rceil - 1$	$\lceil\log_2 n - 1\rceil + \ell$
CSA network ADD$^{(\ell,n)}_{CSA}$	$\ell n - 2\ell + n - \lceil\log_2 n\rceil$ $+\mathbf{S}(\text{ADD}^{\ell+\lceil\log_2 n\rceil}_{LF})$	$\lceil\log_2 n - 1\rceil$ $+\mathbf{D}(\text{ADD}^{\ell+\lceil\log_2 n\rceil}_{LF})$
Multiplication		
RCN school method MUL$^{\ell}_{RC}$	$2\ell^2 - \ell$	$2\ell - 1$
CSN school method MUL$^{\ell}_{CSN}$	$2\ell^2 + 1.25\ell\lceil\log_2 \ell\rceil - \ell + 2$	$3\lceil\log_2 \ell\rceil + 4$
RC squaring SQR$^{\ell}_{RC}$	$\ell^2 - \ell$	$2\ell - 3$
LF squaring SQR$^{\ell}_{LF}$	$\ell^2 + 1.25\ell\lceil\log_2 \ell\rceil - 1.5\ell - 2$	$3\lceil\log_2 \ell\rceil + 3$
Comparison		
Equality EQ$^{\ell}$	$\ell - 1$	$\lceil\log_2 \ell\rceil$
Sequential greater than GT$^{\ell}_{S}$	ℓ	ℓ
D&C greater than GT$^{\ell}_{DC}$	$3\ell - \lceil\log_2 \ell\rceil - 2$	$\lceil\log_2 \ell\rceil + 1$
Selection		
Multiplexer MUX$^{\ell}$	ℓ	1

Can trade-off larger size for better depth.

Example Circuits Summarized in [SchneiderZohner13]

Circuit	Size **S**	Depth **D**
Addition		
Ripple-carry ADD/SUB$^{\ell}_{RC}$	ℓ	ℓ
Ladner-Fischer ADD$^{\ell}_{LF}$	$1.25\ell\lceil\log_2 \ell\rceil + \ell$	$2\lceil\log_2 \ell\rceil + 1$
LF subtraction SUB$^{\ell}_{LF}$	$1.25\ell\lceil\log_2 \ell\rceil + 2\ell$	$2\lceil\log_2 \ell\rceil + 2$
Carry-save ADD$^{(\ell,3)}_{CSA}$	$\ell + \mathbf{S}(\text{ADD}^{\ell})$	$\mathbf{D}(\text{ADD}^{\ell})+1$
RC network ADD$^{(\ell,n)}_{RC}$	$\ell n - \ell + n - \lceil\log_2 n\rceil - 1$	$\lceil\log_2 n - 1\rceil + \ell$
CSA network ADD$^{(\ell,n)}_{CSA}$	$\ell n - 2\ell + n - \lceil\log_2 n\rceil$ $+\mathbf{S}(\text{ADD}^{\ell+\lceil\log_2 n\rceil}_{LF})$	$\lceil\log_2 n - 1\rceil$ $+\mathbf{D}(\text{ADD}^{\ell+\lceil\log_2 n\rceil}_{LF})$
Multiplication		
RCN school method MUL$^{\ell}_{RC}$	$2\ell^2 - \ell$	$2\ell - 1$
CSN school method MUL$^{\ell}_{CSN}$	$2\ell^2 + 1.25\ell\lceil\log_2 \ell\rceil - \ell + 2$	$3\lceil\log_2 \ell\rceil + 4$
RC squaring SQR$^{\ell}_{RC}$	$\ell^2 - \ell$	$2\ell - 3$
LF squaring SQR$^{\ell}_{LF}$	$\ell^2 + 1.25\ell\lceil\log_2 \ell\rceil - 1.5\ell - 2$	$3\lceil\log_2 \ell\rceil + 3$
Comparison		
Equality EQ$^{\ell}$	$\ell - 1$	$\lceil\log_2 \ell\rceil$
Sequential greater than GT$^{\ell}_{S}$	ℓ	ℓ
D&C greater than GT$^{\ell}_{DC}$	$3\ell - \lceil\log_2 \ell\rceil - 2$	$\lceil\log_2 \ell\rceil + 1$
Selection		
Multiplexer MUX$^{\ell}$	ℓ	1

Can trade-off larger size for better depth.

Example Circuits Summarized in [SchneiderZohner13]

Circuit	Size **S**	Depth **D**
Addition		
Ripple-carry ADD/SUB$^{\ell}_{RC}$	ℓ	ℓ
Ladner-Fischer ADD$^{\ell}_{LF}$	$1.25\ell\lceil\log_2 \ell\rceil + \ell$	$2\lceil\log_2 \ell\rceil + 1$
LF subtraction SUB$^{\ell}_{LF}$	$1.25\ell\lceil\log_2 \ell\rceil + 2\ell$	$2\lceil\log_2 \ell\rceil + 2$
Carry-save ADD$^{(\ell,3)}_{CSA}$	$\ell + \mathbf{S}(\text{ADD}^{\ell})$	$\mathbf{D}(\text{ADD}^{\ell})+1$
RC network ADD$^{(\ell,n)}_{RC}$	$\ell n - \ell + n - \lceil\log_2 n\rceil - 1$	$\lceil\log_2 n - 1\rceil + \ell$
CSA network ADD$^{(\ell,n)}_{CSA}$	$\ell n - 2\ell + n - \lceil\log_2 n\rceil$ $+\mathbf{S}(\text{ADD}^{\ell+\lceil\log_2 n\rceil}_{LF})$	$\lceil\log_2 n - 1\rceil$ $+\mathbf{D}(\text{ADD}^{\ell+\lceil\log_2 n\rceil}_{LF})$
Multiplication		
RCN school method MUL$^{\ell}_{RC}$	$2\ell^2 - \ell$	$2\ell - 1$
CSN school method MUL$^{\ell}_{CSN}$	$2\ell^2 + 1.25\ell\lceil\log_2 \ell\rceil - \ell + 2$	$3\lceil\log_2 \ell\rceil + 4$
RC squaring SQR$^{\ell}_{RC}$	$\ell^2 - \ell$	$2\ell - 3$
LF squaring SQR$^{\ell}_{LF}$	$\ell^2 + 1.25\ell\lceil\log_2 \ell\rceil - 1.5\ell - 2$	$3\lceil\log_2 \ell\rceil + 3$
Comparison		
Equality EQ$^{\ell}$	$\ell - 1$	$\lceil\log_2 \ell\rceil$
Sequential greater than GT$^{\ell}_{S}$	ℓ	ℓ
D&C greater than GT$^{\ell}_{DC}$	$3\ell - \lceil\log_2 \ell\rceil - 2$	$\lceil\log_2 \ell\rceil + 1$
Selection		
Multiplexer MUX$^{\ell}$	ℓ	1

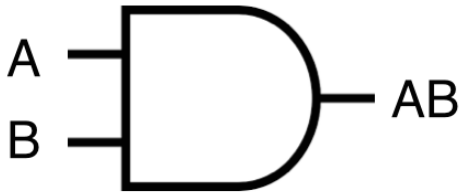
Can trade-off larger size for better depth.

Summary

Summary

Part 1: In Garbled Circuits, each non-XOR gate:

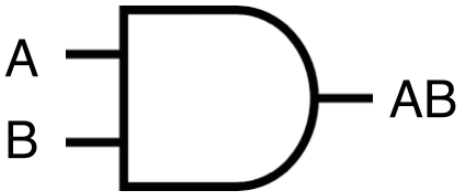
- small # of fixed-key AES evaluations
- send 2 ciphertexts



Summary

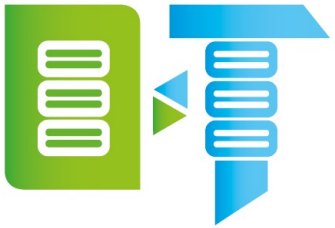
Part 1: In Garbled Circuits, each non-XOR gate:

- small # of fixed-key AES evaluations
- send 2 ciphertexts



Part 2: OT extension

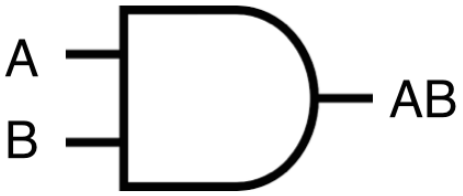
- send 1 ciphertext + |payload|
- communication is essentially the bottleneck



Summary

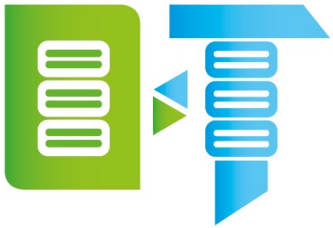
Part 1: In Garbled Circuits, each non-XOR gate:

- small # of fixed-key AES evaluations
- send 2 ciphertexts



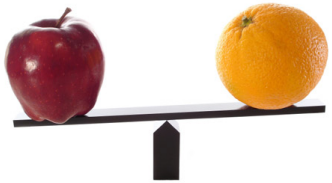
Part 2: OT extension

- send 1 ciphertext + |payload|
- communication is essentially the bottleneck



Part 3: Circuits and Yao vs. GMW

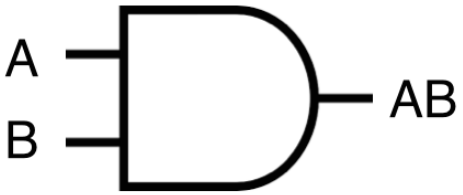
- can trade-off size for depth
- Yao has constant #rounds ⇒ good for networks with high latency (Internet)
- GMW can precompute all crypto, good for low-latency networks (LAN)



Summary

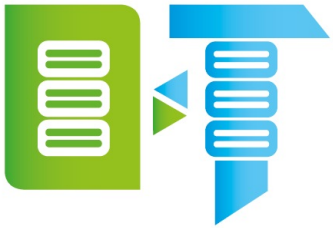
Part 1: In Garbled Circuits, each non-XOR gate:

- small # of fixed-key AES evaluations
- send 2 ciphertexts



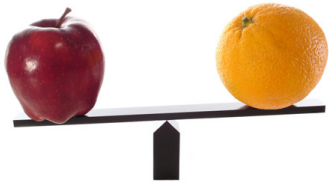
Part 2: OT extension

- send 1 ciphertext + |payload|
- communication is essentially the bottleneck



Part 3: Circuits and Yao vs. GMW

- can trade-off size for depth
- Yao has constant #rounds ⇒ good for networks with high latency (Internet)
- GMW can precompute all crypto, good for low-latency networks (LAN)



Symmetric crypto is so efficient that communication is the bottleneck.

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Thanks for your attention!

Questions?

Contact: http://encrypto.de