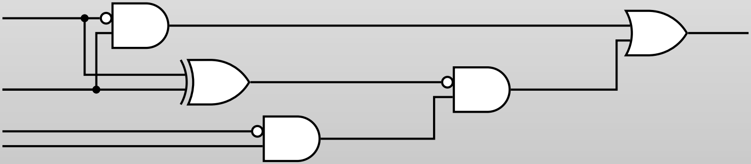
Garbled Circuits
For Secure Computation

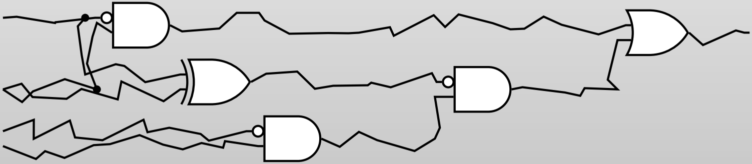
Mike Rosulek
Oregon State University OSU



Garbled Circuits
For Secure Computation

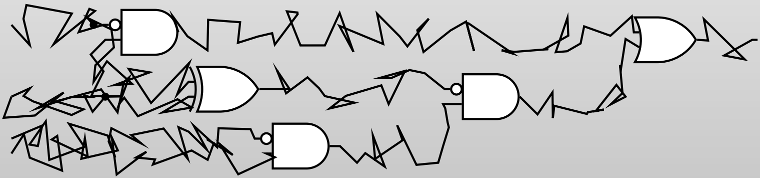
Mike Rosulek

Oregon State University



Garbled Circuits
For Secure Computation

Mike Rosulek
Oregon State University



Roadmap

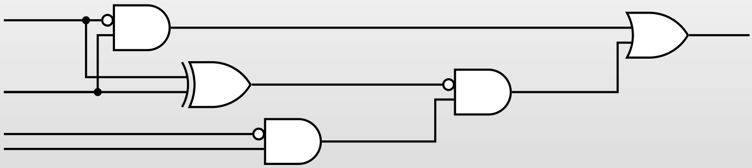
1 **Standard garbled circuits:** core concepts & constructions

- Yao's construction, security definitions, optimized constructions (row reduction, free XOR, half-gates)

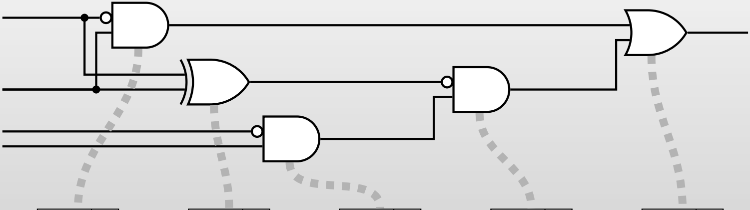
2 **New directions** beyond boolean circuits

- Garbled arithmetic circuits & RAM programs

Garbled circuit framework [Yao86]



Garbled circuit framework [Yao86]



0	0	0
0	1	1
1	0	0
1	1	0

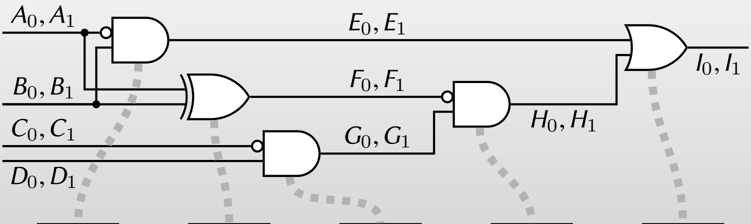
0	0	0
0	1	1
1	0	1
1	1	0

0	0	0
0	1	1
1	0	0
1	1	0

0	0	0
0	1	1
1	0	0
1	1	0

0	0	0
0	1	1
1	0	1
1	1	1

Garbled circuit framework [Yao86]



0	0	0
0	1	1
1	0	0
1	1	0

0	0	0
0	1	1
1	0	1
1	1	0

0	0	0
0	1	1
1	0	0
1	1	0

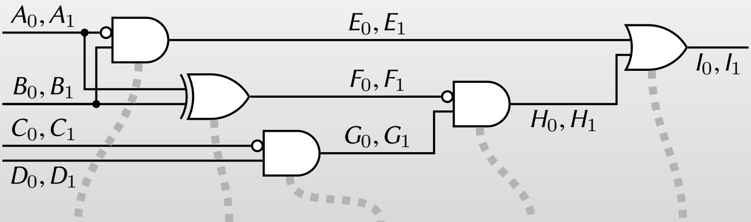
0	0	0
0	1	1
1	0	0
1	1	0

0	0	0
0	1	1
1	0	1
1	1	1

Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire

Garbled circuit framework [Yao86]



A_0	B_0	E_0
A_0	B_1	E_1
A_1	B_0	E_0
A_1	B_1	E_0

A_0	B_0	F_0
A_0	B_1	F_1
A_1	B_0	F_1
A_1	B_1	F_0

C_0	D_0	G_0
C_0	D_1	G_1
C_1	D_0	G_0
C_1	D_1	G_0

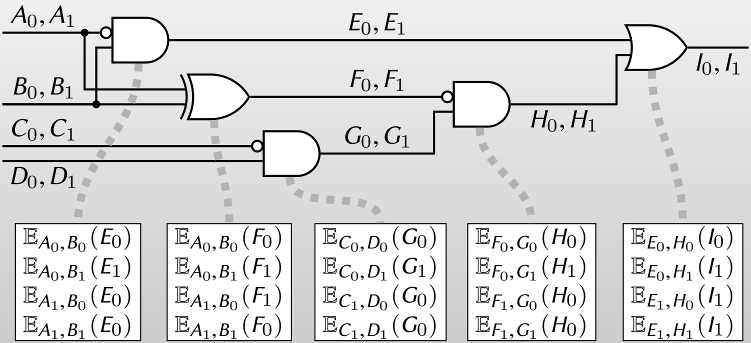
F_0	G_0	H_0
F_0	G_1	H_1
F_1	G_0	H_0
F_1	G_1	H_0

E_0	H_0	I_0
E_0	H_1	I_1
E_1	H_0	I_1
E_1	H_1	I_1

Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire

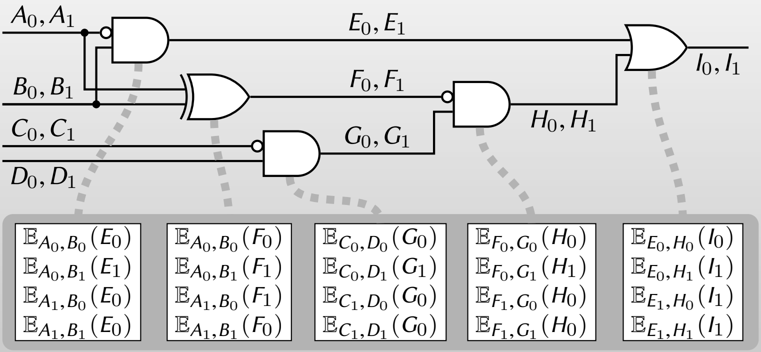
Garbled circuit framework [Yao86]



Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate

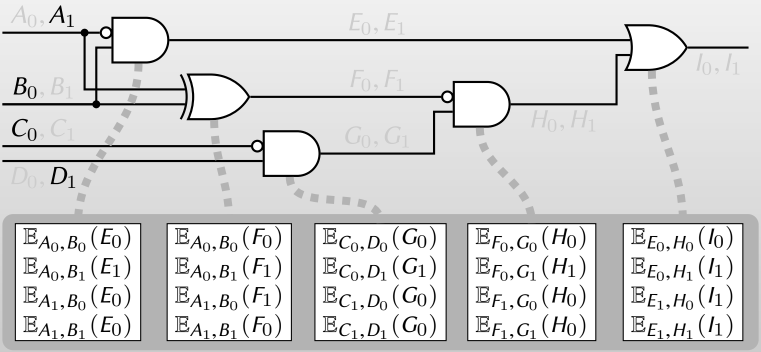
Garbled circuit framework [Yao86]



Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** ≡ all encrypted gates

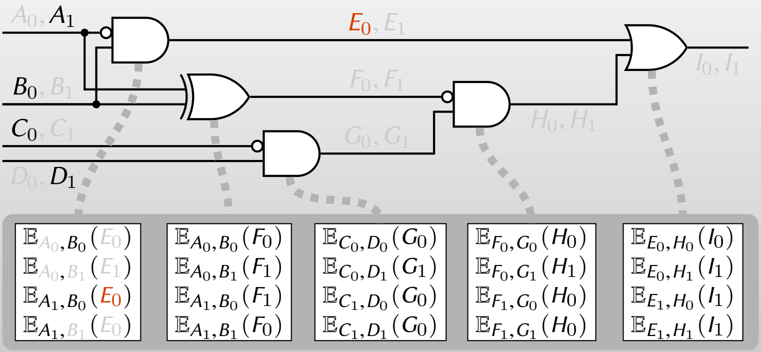
Garbled circuit framework [Yao86]



Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- “Encrypt” truth table of each gate
- **Garbled circuit** ≡ all encrypted gates
- **Garbled encoding** ≡ one label per wire

Garbled circuit framework [Yao86]



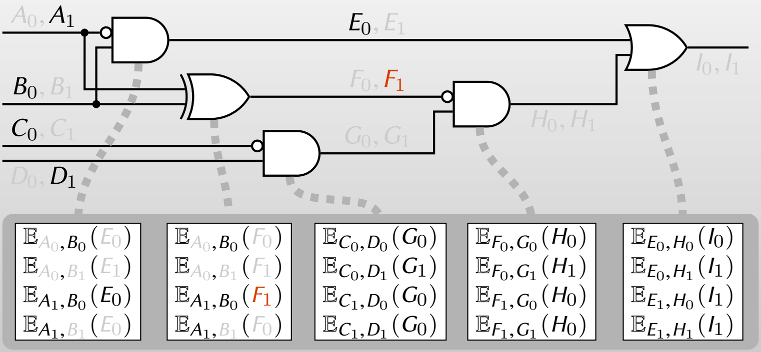
Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** ≡ all encrypted gates
- **Garbled encoding** ≡ one label per wire

Garbled evaluation:

- Only one ciphertext per gate is decryptable

Garbled circuit framework [Yao86]



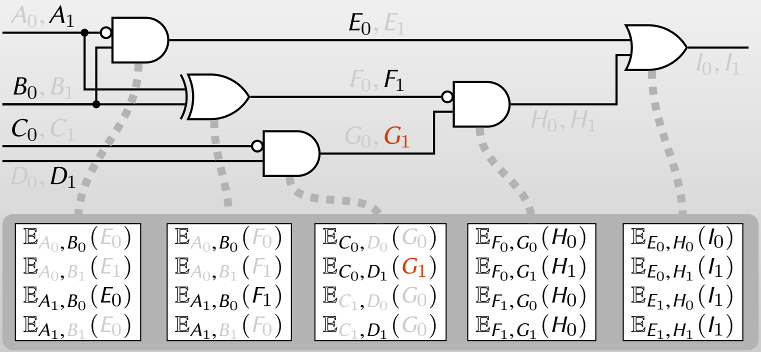
Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** ≡ all encrypted gates
- **Garbled encoding** ≡ one label per wire

Garbled evaluation:

- Only one ciphertext per gate is decryptable
- Result of decryption = value on outgoing wire

Garbled circuit framework [Yao86]



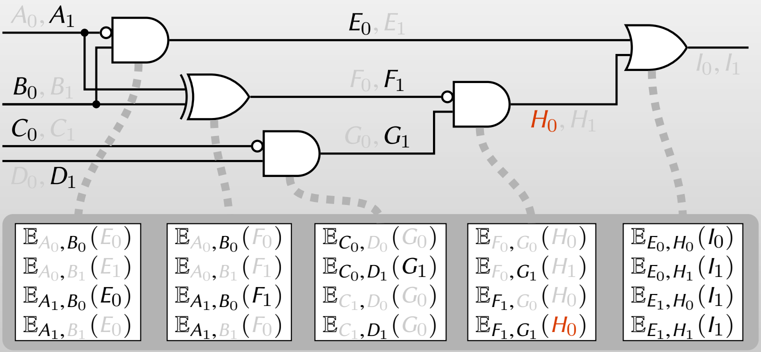
Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** ≡ all encrypted gates
- **Garbled encoding** ≡ one label per wire

Garbled evaluation:

- Only one ciphertext per gate is decryptable
- Result of decryption = value on outgoing wire

Garbled circuit framework [Yao86]



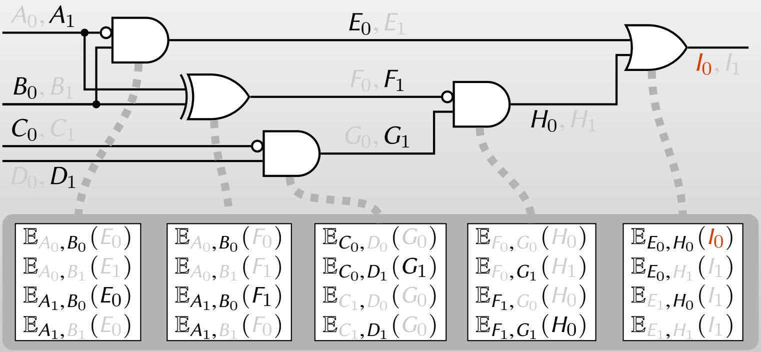
Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** ≡ all encrypted gates
- **Garbled encoding** ≡ one label per wire

Garbled evaluation:

- Only one ciphertext per gate is decryptable
- Result of decryption = value on outgoing wire

Garbled circuit framework [Yao86]



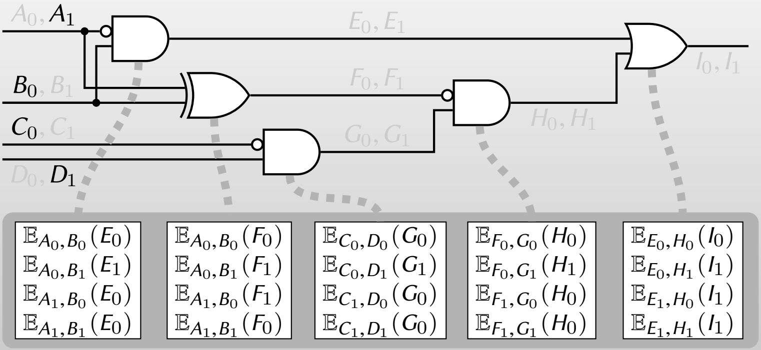
Garbling a circuit:

- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** ≡ all encrypted gates
- **Garbled encoding** ≡ one label per wire

Garbled evaluation:

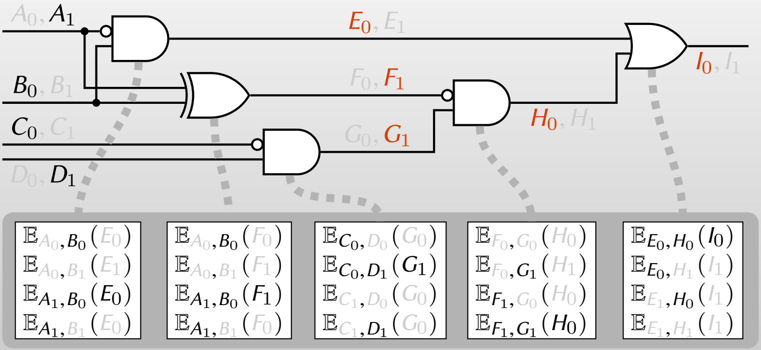
- Only one ciphertext per gate is decryptable
- Result of decryption = value on outgoing wire

Syntax & Security (informal)



Key idea: Given garbled circuit + garbled input . . .

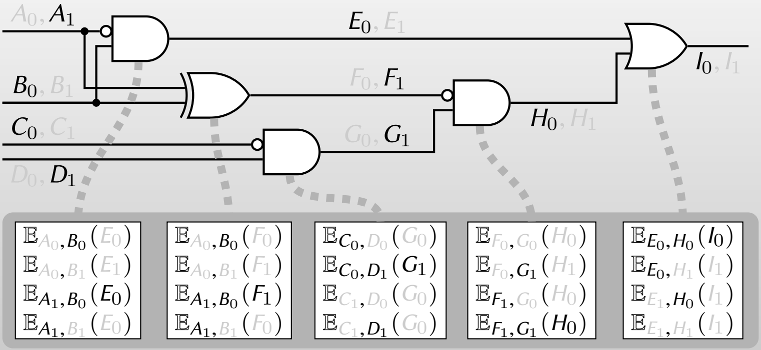
Syntax & Security (informal)



Key idea: Given garbled circuit + garbled input . . .

- . . . Only thing you can do is **(blindly) evaluate circuit** on that input

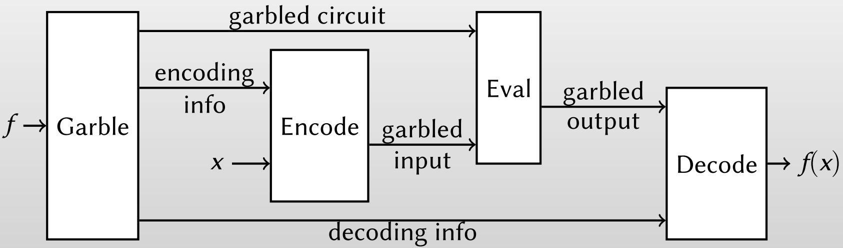
Syntax & Security (informal)



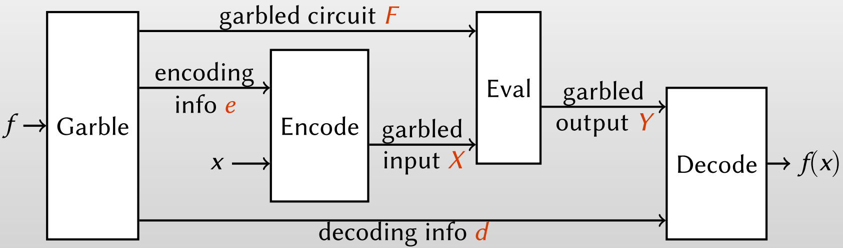
Key idea: Given garbled circuit + garbled input ...

- ... Only thing you can do is **(blindly) evaluate circuit** on that input
- Learn only 1 label per wire: hard to guess "complementary" label
- Seeing a single label hides logical value on wire, although ...
- Revealing both labels on *output wires* leaks *only* circuit output

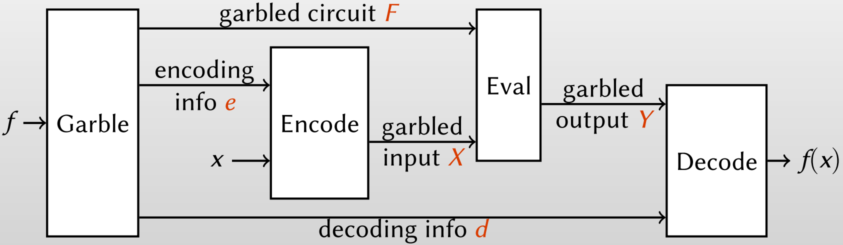
Syntax & Security [BellareHoangRogaway12]



Syntax & Security [BellareHoangRogaway12]



Syntax & Security [BellareHoangRogaway12]



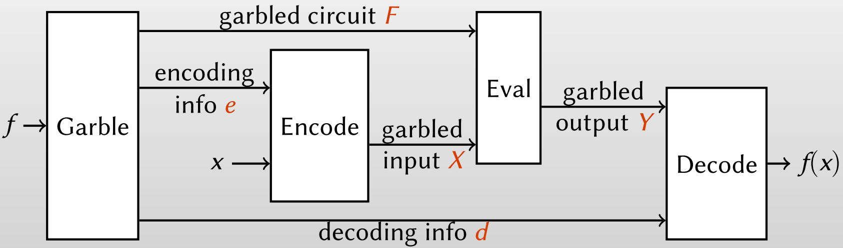
Formal security properties:

Privacy: (F, X, d) reveals nothing beyond f and $f(x)$

Obliviousness: (F, X) reveals nothing beyond f

Authenticity: given (F, X), hard to find $\widetilde{Y}$ that decodes $\notin \{f(x), \perp\}$

Syntax & Security [BellareHoangRogaway12]



Formal security properties:

Privacy: (F, X, d) reveals nothing beyond f and $f(x)$

Obliviousness: (F, X) reveals nothing beyond f

Authenticity: given (F, X), hard to find $\widetilde{Y}$ that decodes $\notin \{f(x), \perp\}$

Other interesting notions we won't discuss:

Adaptive security: choice of input can depend on *garbled* circuit

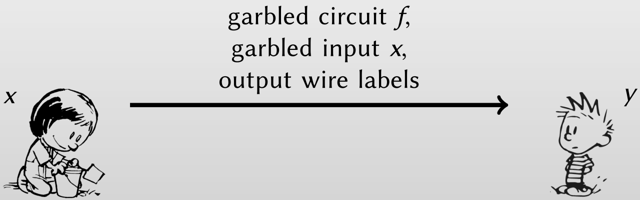
Gate-hiding: (F, X, d) reveals nothing beyond *topology of f* and $f(x)$

Yao's Protocol

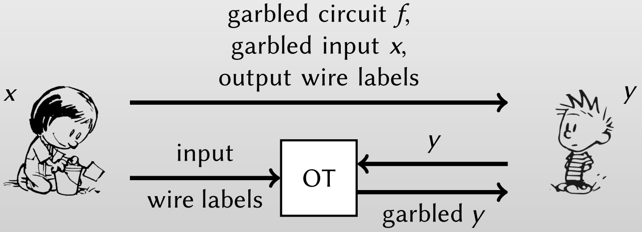
x

y

Yao's Protocol

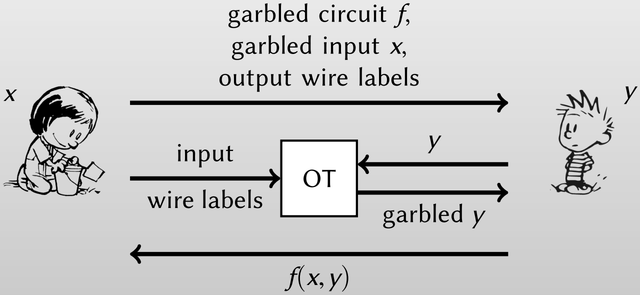


Yao's Protocol



- **Oblivious transfer:** Alice has m_0, m_1; Bob has b and learns m_b

Yao's Protocol



- **Oblivious transfer:** Alice has m_0, m_1; Bob has b and learns m_b
- Given garbled f + garbled inputs + all output labels ⇒ Bob learns **only** $f(x, y)$

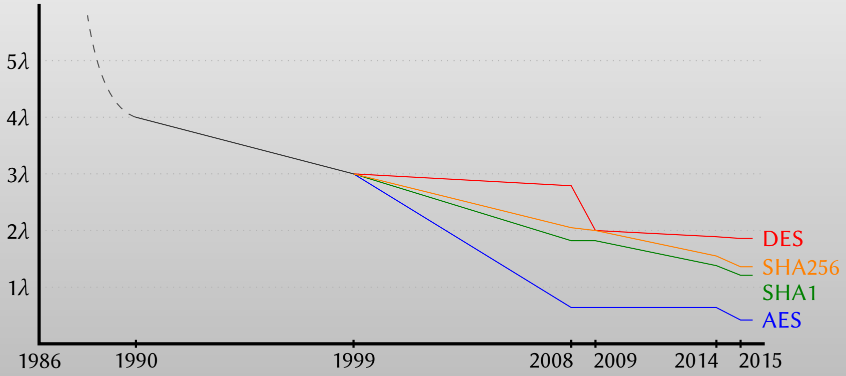
Optimizing garbled circuits

Size of garbled circuits . . .

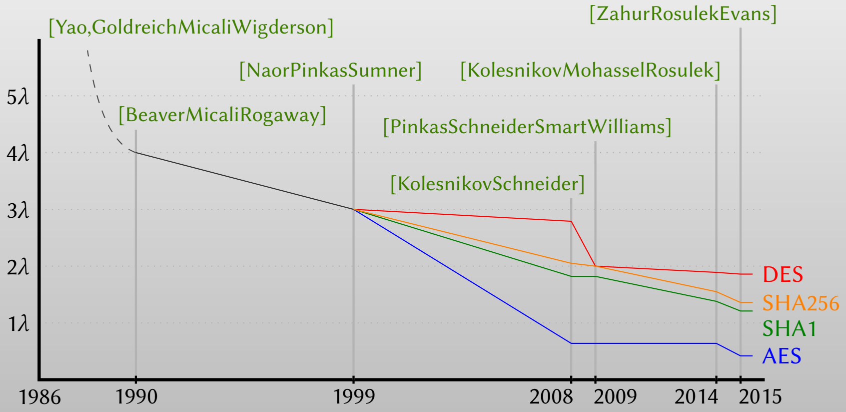
. . . is the most important parameter

- Applications of garbled circuits are **network-bound**
- Garbled circuit computations are very fast (typically hardware AES)

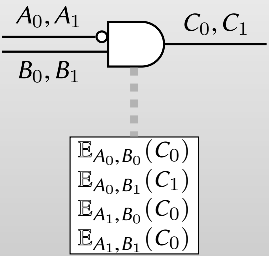
Average bits per garbled gate



Average bits per garbled gate

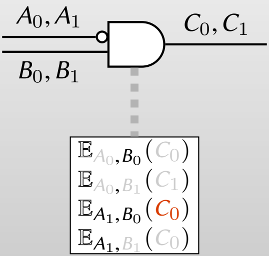


Ciphertext expansion [Yao86]



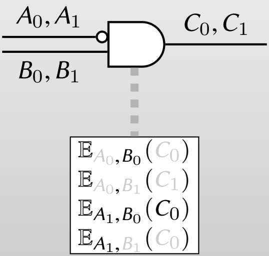
Position in this list leaks semantic value!

Ciphertext expansion [Yao86]



Position in this list leaks semantic value!

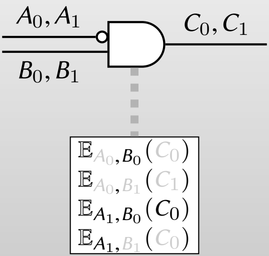
Ciphertext expansion [Yao86]



Position in this list leaks semantic value!

$\Rightarrow$ Need to randomly permute ciphertexts

Ciphertext expansion [Yao86]

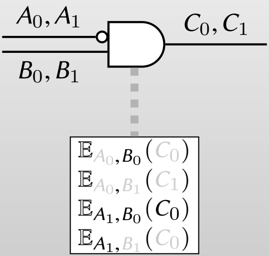


Position in this list leaks semantic value!

$\Rightarrow$ Need to randomly permute ciphertexts

$\Rightarrow$ Need to **detect** [in]correct decryption

Ciphertext expansion [Yao86]



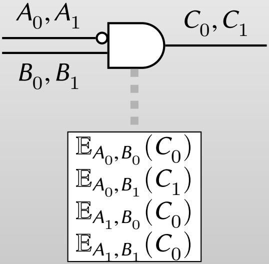
Position in this list leaks semantic value!

$\Rightarrow$ Need to randomly permute ciphertexts

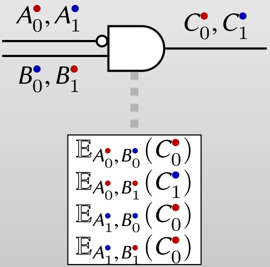
$\Rightarrow$ Need to **detect** [in]correct decryption

$\Rightarrow$ Need encryption scheme with *ciphertext expansion* (size doubles)

Point-and-permute [BeaverMicaliRogaway90]

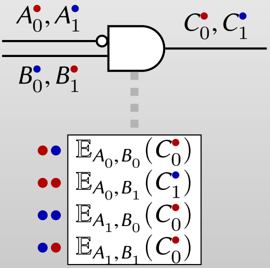


Point-and-permute [BeaverMicaliRogaway90]



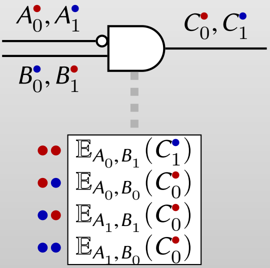
- Assign color bits • & • to wire labels
- Association between (•, •) ↔ (T, F) is random for each wire
- A wire label reveals its own color (e.g., as last bit)

Point-and-permute [BeaverMicaliRogaway90]



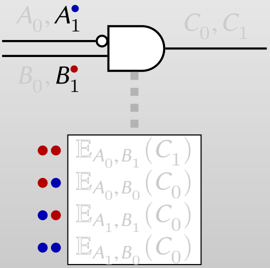
- Assign color bits • & • to wire labels
- Association between (•, •) ↔ (T, F) is random for each wire
- A wire label reveals its own color (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys

Point-and-permute [BeaverMicaliRogaway90]



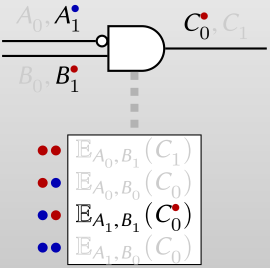
- Assign color bits • & • to wire labels
- Association between (•, •) $\leftrightarrow$ (T, F) is random for each wire
- A wire label reveals its own color (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys

Point-and-permute [BeaverMicaliRogaway90]



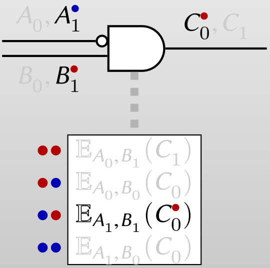
- Assign color bits • & • to wire labels
- Association between (•, •) ↔ (T, F) is random for each wire
- A wire label reveals its own color (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys
- Evaluate by decrypting ciphertext indexed by your colors

Point-and-permute [BeaverMicaliRogaway90]



- Assign color bits • & • to wire labels
- Association between (•, •) ↔ (T, F) is random for each wire
- A wire label reveals its own color (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys
- Evaluate by decrypting ciphertext indexed by your colors

Point-and-permute [BeaverMicaliRogaway90]



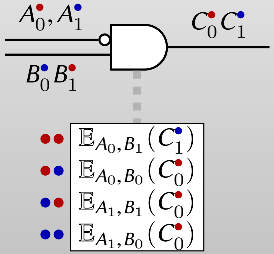
- Assign color bits ● & ● to wire labels
- Association between (●, ●) ↔ (T, F) is random for each wire
- A wire label reveals its own color (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys
- Evaluate by decrypting ciphertext indexed by your colors

No need for trial decryption ⇒ no need for ciphertext expansion!

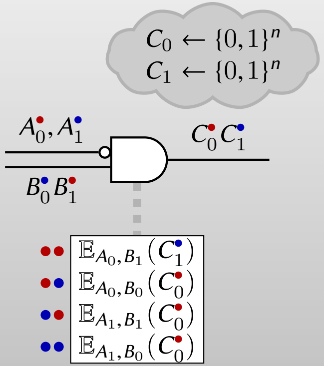
Scoreboard

	size ($\times\lambda$)	garble cost	eval cost
Classical [Yao86,GMW87]	8	4	2.5
P&P [BeaverMicaliRogaway90]	4	4	1

Garbled Row Reduction [NaorPinkasSumner99]

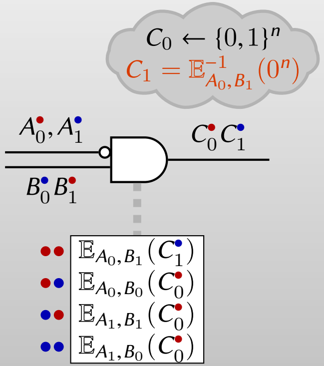


Garbled Row Reduction [NaorPinkasSumner99]



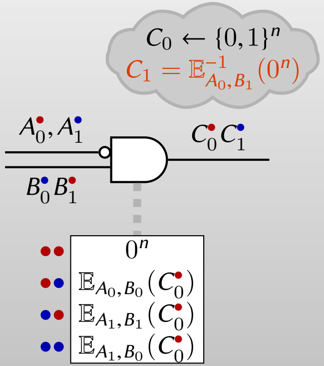
- Instead of choosing all wire labels uniformly . . .

Garbled Row Reduction [NaorPinkasSumner99]



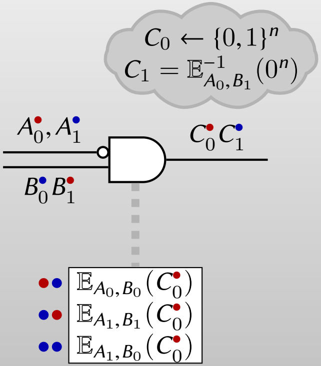
- Instead of choosing all wire labels uniformly ...
- ... choose so that first ciphertext is 0^n

Garbled Row Reduction [NaorPinkasSumner99]



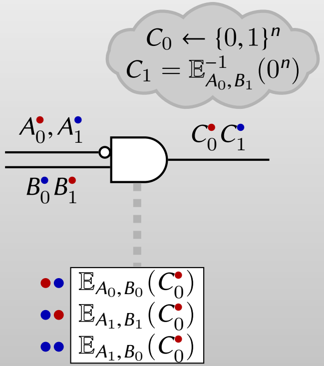
- Instead of choosing all wire labels uniformly ...
- ... choose so that first ciphertext is 0^n

Garbled Row Reduction [NaorPinkasSumner99]



- Instead of choosing all wire labels uniformly . . .
- . . . choose so that first ciphertext is 0^n
- No need to include 1st ciphertext in garbled gate

Garbled Row Reduction [NaorPinkasSumner99]

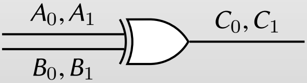


- Instead of choosing all wire labels uniformly ...
- ... choose so that first ciphertext is 0^n
- No need to include 1st ciphertext in garbled gate
- To evaluate, just imagine ciphertext 0^n if you have label combination ●●.

Scoreboard

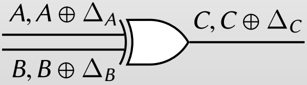
	size ($\times\lambda$)	garble cost	eval cost
Classical [Yao86,GMW87]	8	4	2.5
P&P [BeaverMicaliRogaway90]	4	4	1
GRR3 [NaorPinkasSumner99]	**3**	**4**	**1**

Free XOR [KolesnikovSchneider08]



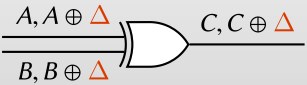
- Define **offset of a wire** ≡ XOR of its two labels

Free XOR [KolesnikovSchneider08]



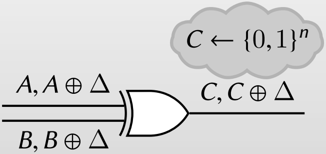
- Define **offset of a wire** ≡ XOR of its two labels

Free XOR [KolesnikovSchneider08]



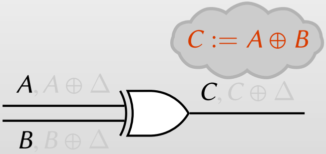
- Define **offset of a wire** ≡ XOR of its two labels
- Choose all wires in circuit to have same (secret) offset Δ

Free XOR [KolesnikovSchneider08]



- Define **offset of a wire** ≡ XOR of its two labels
- Choose all wires in circuit to have same (secret) offset Δ

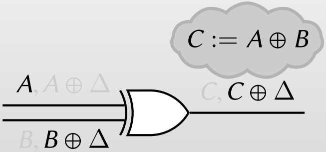
Free XOR [KolesnikovSchneider08]



$$\underbrace{A}_{\text{FALSE}} \oplus \underbrace{B}_{\text{FALSE}} = \underbrace{A \oplus B}_{\text{FALSE}}$$

- Define **offset of a wire** ≡ XOR of its two labels
- Choose all wires in circuit to have same (secret) offset Δ
- Choose FALSE output = FALSE input ⊕ FALSE input

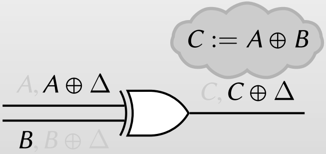
Free XOR [KolesnikovSchneider08]



$$\underbrace{A}_{\text{FALSE}} \oplus \underbrace{B \oplus \Delta}_{\text{TRUE}} = \underbrace{A \oplus B \oplus \Delta}_{\text{TRUE}}$$

- Define **offset of a wire** ≡ XOR of its two labels
- Choose all wires in circuit to have same (secret) offset Δ
- Choose FALSE output = FALSE input $\oplus$ FALSE input
- Evaluate by XORing input wire labels (no crypto)

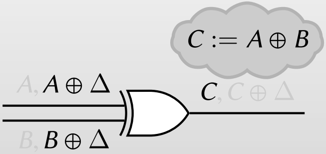
Free XOR [KolesnikovSchneider08]



$$\underbrace{A \oplus \Delta}_{\text{TRUE}} \oplus \underbrace{B \quad}_{\text{FALSE}} = \underbrace{A \oplus B \oplus \Delta}_{\text{TRUE}}$$

- Define **offset of a wire** ≡ XOR of its two labels
- Choose all wires in circuit to have same (secret) offset Δ
- Choose FALSE output = FALSE input $\oplus$ FALSE input
- Evaluate by XORing input wire labels (no crypto)

Free XOR [KolesnikovSchneider08]



$$\underbrace{A \oplus \Delta}_{\text{TRUE}} \oplus \underbrace{B \oplus \Delta}_{\text{TRUE}} = \underbrace{A \oplus B \quad}_{\text{FALSE}}$$

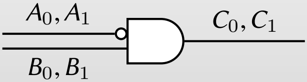
- Define **offset of a wire** ≡ XOR of its two labels
- Choose all wires in circuit to have same (secret) offset Δ
- Choose FALSE output = FALSE input $\oplus$ FALSE input
- Evaluate by XORing input wire labels (no crypto)

Scoreboard

	size ($\times\lambda$)		garble cost		eval cost	
	XOR	AND	XOR	AND	XOR	AND
Classical [Yao86,GMW87]	8	8	4	4	2.5	2.5
P&P [BeaverMicaliRogaway90]	4	4	4	4	1	1
GRR3 [NaorPinkasSumner99]	3	3	4	4	1	1
Free XOR [KolesnikovSchneider08]	**0**	**3**	**0**	**4**	**0**	**1**

Row reduction ×2 [PinkasSchneiderSmartWilliams09]

Garble a gate @ 2 ciphertexts per gate:

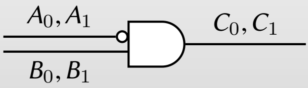


Row reduction ×2 [PinkasSchneiderSmartWilliams09]

Garble a gate @ 2 ciphertexts per gate:

Idea: Evaluator can know exactly one of:

$$K_1 = \mathbb{E}^{-1}_{A_0,B_0}(0^n)$$
$$K_2 = \mathbb{E}^{-1}_{A_0,B_1}(0^n)$$
$$K_3 = \mathbb{E}^{-1}_{A_1,B_0}(0^n)$$
$$K_4 = \mathbb{E}^{-1}_{A_1,B_1}(0^n)$$

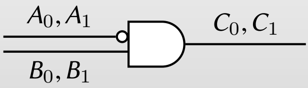


Row reduction $\times 2$ [PinkasSchneiderSmartWilliams09]

Garble a gate @ 2 ciphertexts per gate:

Idea: Evaluator can know exactly one of:

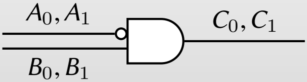
$$K_1 = \mathbb{E}^{-1}_{A_0,B_0}(0^n) \leadsto \text{learn } C_0$$
$$K_2 = \mathbb{E}^{-1}_{A_0,B_1}(0^n) \leadsto \text{learn } C_1$$
$$K_3 = \mathbb{E}^{-1}_{A_1,B_0}(0^n) \leadsto \text{learn } C_0$$
$$K_4 = \mathbb{E}^{-1}_{A_1,B_1}(0^n) \leadsto \text{learn } C_0$$



Row reduction $\times 2$ [PinkasSchneiderSmartWilliams09]

Garble a gate @ 2 ciphertexts per gate:

Idea: Evaluator can know exactly one of:

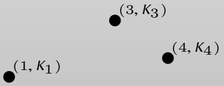


$$K_1 = \mathbb{E}^{-1}_{A_0,B_0}(0^n) \rightsquigarrow \text{learn } C_0$$

$$K_2 = \mathbb{E}^{-1}_{A_0,B_1}(0^n) \rightsquigarrow \text{learn } C_1$$

$$K_3 = \mathbb{E}^{-1}_{A_1,B_0}(0^n) \rightsquigarrow \text{learn } C_0$$

$$K_4 = \mathbb{E}^{-1}_{A_1,B_1}(0^n) \rightsquigarrow \text{learn } C_0$$



$$(1, K_1), (3, K_3), (4, K_4)$$

Row reduction ×2 [PinkasSchneiderSmartWilliams09]

Garble a gate @ 2 ciphertexts per gate:

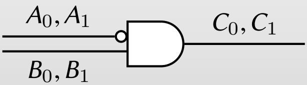
Idea: Evaluator can know exactly one of:

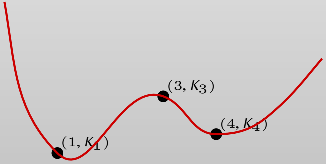
$K_1 = \mathbb{E}^{-1}_{A_0,B_0}(0^n) \rightsquigarrow$ learn C_0

$K_2 = \mathbb{E}^{-1}_{A_0,B_1}(0^n) \rightsquigarrow$ learn C_1

$K_3 = \mathbb{E}^{-1}_{A_1,B_0}(0^n) \rightsquigarrow$ learn C_0

$K_4 = \mathbb{E}^{-1}_{A_1,B_1}(0^n) \rightsquigarrow$ learn C_0





$P =$ uniq deg-2 poly thru $(1, K_1), (3, K_3), (4, K_4)$

Row reduction ×2 [PinkasSchneiderSmartWilliams09]

Garble a gate @ 2 ciphertexts per gate:

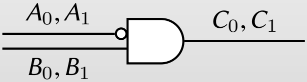
Idea: Evaluator can know exactly one of:

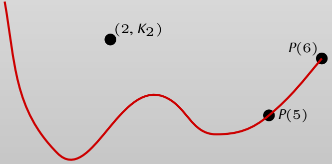
$K_1 = \mathbb{E}^{-1}_{A_0,B_0}(0^n) \rightsquigarrow$ learn C_0

$K_2 = \mathbb{E}^{-1}_{A_0,B_1}(0^n) \rightsquigarrow$ learn C_1

$K_3 = \mathbb{E}^{-1}_{A_1,B_0}(0^n) \rightsquigarrow$ learn C_0

$K_4 = \mathbb{E}^{-1}_{A_1,B_1}(0^n) \rightsquigarrow$ learn C_0





$P =$ uniq deg-2 poly thru

$(1, K_1), (3, K_3), (4, K_4)$

$(2, K_2), (5, P(5)), (6, P(6))$

Row reduction $\times 2$ [PinkasSchneiderSmartWilliams09]

Garble a gate @ 2 ciphertexts per gate:

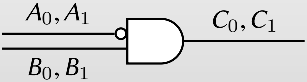
Idea: Evaluator can know exactly one of:

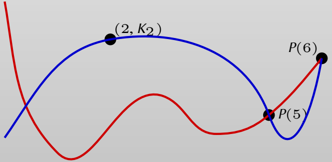
$K_1 = \mathbb{E}^{-1}_{A_0,B_0}(0^n) \rightsquigarrow$ learn C_0

$K_2 = \mathbb{E}^{-1}_{A_0,B_1}(0^n) \rightsquigarrow$ learn C_1

$K_3 = \mathbb{E}^{-1}_{A_1,B_0}(0^n) \rightsquigarrow$ learn C_0

$K_4 = \mathbb{E}^{-1}_{A_1,B_1}(0^n) \rightsquigarrow$ learn C_0





$P =$ uniq deg-2 poly thru $(1, K_1), (3, K_3), (4, K_4)$

$Q =$ uniq deg-2 poly thru $(2, K_2), (5, P(5)), (6, P(6))$

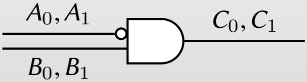
Row reduction ×2 [PinkasSchneiderSmartWilliams09]

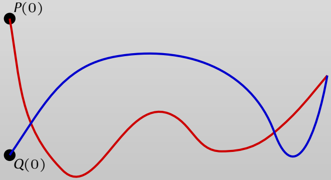
Garble a gate @ 2 ciphertexts per gate:

Idea: Evaluator can know exactly one of:

$$K_1 = \mathbb{E}^{-1}_{A_0,B_0}(0^n) \rightsquigarrow \text{learn } C_0$$
$$K_2 = \mathbb{E}^{-1}_{A_0,B_1}(0^n) \rightsquigarrow \text{learn } C_1$$
$$K_3 = \mathbb{E}^{-1}_{A_1,B_0}(0^n) \rightsquigarrow \text{learn } C_0$$
$$K_4 = \mathbb{E}^{-1}_{A_1,B_1}(0^n) \rightsquigarrow \text{learn } C_0$$

$C_0 = P(0);\ C_1 = Q(0)$





$P =$ uniq deg-2 poly thru $(1, K_1), (3, K_3), (4, K_4)$

$Q =$ uniq deg-2 poly thru $(2, K_2), (5, P(5)), (6, P(6))$

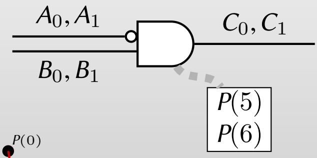
Row reduction ×2 [PinkasSchneiderSmartWilliams09]

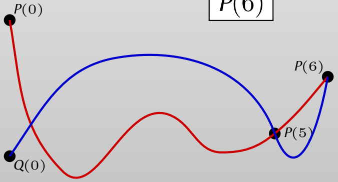
Garble a gate @ 2 ciphertexts per gate:

Idea: Evaluator can know exactly one of:

$$K_1 = \mathbb{E}^{-1}_{A_0,B_0}(0^n) \rightsquigarrow \text{learn } C_0$$
$$K_2 = \mathbb{E}^{-1}_{A_0,B_1}(0^n) \rightsquigarrow \text{learn } C_1$$
$$K_3 = \mathbb{E}^{-1}_{A_1,B_0}(0^n) \rightsquigarrow \text{learn } C_0$$
$$K_4 = \mathbb{E}^{-1}_{A_1,B_1}(0^n) \rightsquigarrow \text{learn } C_0$$

$C_0 = P(0); C_1 = Q(0)$





$P =$ uniq deg-2 poly thru $(1, K_1), (3, K_3), (4, K_4)$

$Q =$ uniq deg-2 poly thru $(2, K_2), (5, P(5)), (6, P(6))$

Row reduction $\times 2$ [PinkasSchneiderSmartWilliams09]

Garble a gate @ 2 ciphertexts per gate:

Idea: Evaluator can know exactly one of:

$$K_1 = \mathbb{E}^{-1}_{A_0,B_0}(0^n) \rightsquigarrow \text{learn } C_0$$
$$K_2 = \mathbb{E}^{-1}_{A_0,B_1}(0^n) \rightsquigarrow \text{learn } C_1$$
$$K_3 = \mathbb{E}^{-1}_{A_1,B_0}(0^n) \rightsquigarrow \text{learn } C_0$$
$$K_4 = \mathbb{E}^{-1}_{A_1,B_1}(0^n) \rightsquigarrow \text{learn } C_0$$

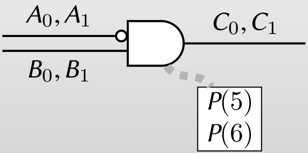
To evaluate a gate:

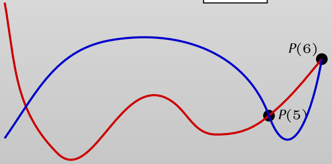
- Compute relevant K_i & interpolate:

$$(i, K_i), (5, P(5)), (6, P(6))$$

- Evaluate polynomial at zero

$C_0 = P(0); C_1 = Q(0)$





$P =$ uniq deg-2 poly thru $(1, K_1), (3, K_3), (4, K_4)$

$Q =$ uniq deg-2 poly thru $(2, K_2), (5, P(5)), (6, P(6))$

Row reduction ×2 [PinkasSchneiderSmartWilliams09]

Garble a gate @ 2 ciphertexts per gate:

Idea: Evaluator can know exactly one of:

$$K_1 = \mathbb{E}^{-1}_{A_0,B_0}(0^n) \rightsquigarrow \text{learn } C_0$$
$$K_2 = \mathbb{E}^{-1}_{A_0,B_1}(0^n) \rightsquigarrow \text{learn } C_1$$
$$K_3 = \mathbb{E}^{-1}_{A_1,B_0}(0^n) \rightsquigarrow \text{learn } C_0$$
$$K_4 = \mathbb{E}^{-1}_{A_1,B_1}(0^n) \rightsquigarrow \text{learn } C_0$$

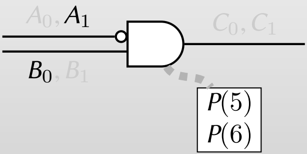
To evaluate a gate:

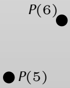
- Compute relevant K_i & interpolate:

$$(i, K_i), (5, P(5)), (6, P(6))$$

- Evaluate polynomial at zero

$C_0 = P(0);\ C_1 = Q(0)$





$P =$ uniq deg-2 poly thru $(1, K_1), (3, K_3), (4, K_4)$

$Q =$ uniq deg-2 poly thru $(2, K_2), (5, P(5)), (6, P(6))$

Row reduction ×2 [PinkasSchneiderSmartWilliams09]

Garble a gate @ 2 ciphertexts per gate:

Idea: Evaluator can know exactly one of:

$K_1 = \mathbb{E}^{-1}_{A_0,B_0}(0^n) \rightsquigarrow$ learn C_0

$K_2 = \mathbb{E}^{-1}_{A_0,B_1}(0^n) \rightsquigarrow$ learn C_1

$K_3 = \mathbb{E}^{-1}_{A_1,B_0}(0^n) \rightsquigarrow$ learn C_0

$K_4 = \mathbb{E}^{-1}_{A_1,B_1}(0^n) \rightsquigarrow$ learn C_0

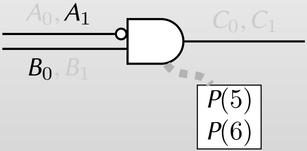
To evaluate a gate:

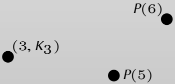
- Compute relevant K_i & interpolate:

$$(i, K_i), (5, P(5)), (6, P(6))$$

- Evaluate polynomial at zero

$C_0 = P(0); C_1 = Q(0)$





$P =$ uniq deg-2 poly thru $(1, K_1), (3, K_3), (4, K_4)$

$Q =$ uniq deg-2 poly thru $(2, K_2), (5, P(5)), (6, P(6))$

Row reduction ×2 [PinkasSchneiderSmartWilliams09]

Garble a gate @ 2 ciphertexts per gate:

Idea: Evaluator can know exactly one of:

$$K_1 = \mathbb{E}^{-1}_{A_0,B_0}(0^n) \rightsquigarrow \text{learn } C_0$$

$$K_2 = \mathbb{E}^{-1}_{A_0,B_1}(0^n) \rightsquigarrow \text{learn } C_1$$

$$K_3 = \mathbb{E}^{-1}_{A_1,B_0}(0^n) \rightsquigarrow \text{learn } C_0$$

$$K_4 = \mathbb{E}^{-1}_{A_1,B_1}(0^n) \rightsquigarrow \text{learn } C_0$$

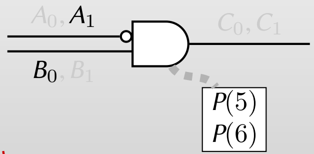
To evaluate a gate:

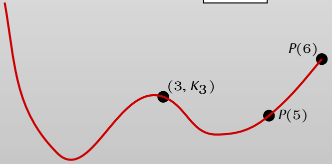
- Compute relevant K_i & interpolate:

$$(i, K_i), (5, P(5)), (6, P(6))$$

- Evaluate polynomial at zero

$$C_0 = P(0); C_1 = Q(0)$$





$P =$ uniq deg-2 poly thru $(1, K_1), (3, K_3), (4, K_4)$

$Q =$ uniq deg-2 poly thru $(2, K_2), (5, P(5)), (6, P(6))$

Row reduction ×2 [PinkasSchneiderSmartWilliams09]

Garble a gate @ 2 ciphertexts per gate:

Idea: Evaluator can know exactly one of:

$$K_1 = \mathbb{E}^{-1}_{A_0,B_0}(0^n) \rightsquigarrow \text{learn } C_0$$

$$K_2 = \mathbb{E}^{-1}_{A_0,B_1}(0^n) \rightsquigarrow \text{learn } C_1$$

$$K_3 = \mathbb{E}^{-1}_{A_1,B_0}(0^n) \rightsquigarrow \text{learn } C_0$$

$$K_4 = \mathbb{E}^{-1}_{A_1,B_1}(0^n) \rightsquigarrow \text{learn } C_0$$

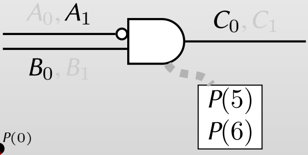
To evaluate a gate:

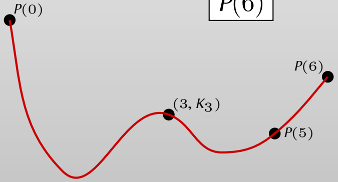
- Compute relevant K_i & interpolate:

$$(i, K_i), (5, P(5)), (6, P(6))$$

- Evaluate polynomial at zero

$C_0 = P(0); C_1 = Q(0)$





$P =$ uniq deg-2 poly thru $(1, K_1), (3, K_3), (4, K_4)$

$Q =$ uniq deg-2 poly thru $(2, K_2), (5, P(5)), (6, P(6))$

Row reduction ×2 [PinkasSchneiderSmartWilliams09]

Garble a gate @ 2 ciphertexts per gate:

Idea: Evaluator can know exactly one of:

$K_1 = \mathbb{E}^{-1}_{A_0,B_0}(0^n) \rightsquigarrow$ learn C_0

$K_2 = \mathbb{E}^{-1}_{A_0,B_1}(0^n) \rightsquigarrow$ learn C_1

$K_3 = \mathbb{E}^{-1}_{A_1,B_0}(0^n) \rightsquigarrow$ learn C_0

$K_4 = \mathbb{E}^{-1}_{A_1,B_1}(0^n) \rightsquigarrow$ learn C_0

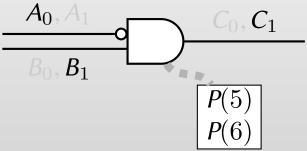
To evaluate a gate:

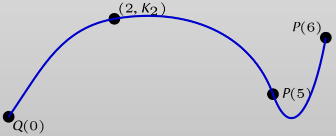
- Compute relevant K_i & interpolate:

$$(i, K_i), (5, P(5)), (6, P(6))$$

- Evaluate polynomial at zero

$C_0 = P(0); C_1 = Q(0)$



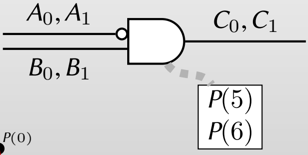


$P =$ uniq deg-2 poly thru $(1, K_1), (3, K_3), (4, K_4)$

$Q =$ uniq deg-2 poly thru $(2, K_2), (5, P(5)), (6, P(6))$

Row reduction ×2 [PinkasSchneiderSmartWilliams09]

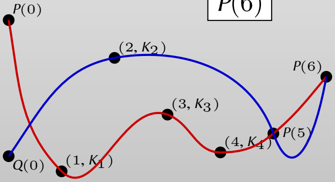
$C_0 = P(0); C_1 = Q(0)$



Garble a gate @ 2 ciphertexts per gate:

Idea: Evaluator can know exactly one of:

$$K_1 = \mathbb{E}^{-1}_{A_0,B_0}(0^n) \rightsquigarrow \text{learn } C_0$$
$$K_2 = \mathbb{E}^{-1}_{A_0,B_1}(0^n) \rightsquigarrow \text{learn } C_1$$
$$K_3 = \mathbb{E}^{-1}_{A_1,B_0}(0^n) \rightsquigarrow \text{learn } C_0$$
$$K_4 = \mathbb{E}^{-1}_{A_1,B_1}(0^n) \rightsquigarrow \text{learn } C_0$$



To evaluate a gate:

- Compute relevant K_i & interpolate:

$$(i, K_i), (5, P(5)), (6, P(6))$$

- Evaluate polynomial at zero

$P =$ uniq deg-2 poly thru $(1, K_1), (3, K_3), (4, K_4)$

$Q =$ uniq deg-2 poly thru $(2, K_2), (5, P(5)), (6, P(6))$

Scoreboard

	size ($\times\lambda$)		garble cost		eval cost	
	XOR	AND	XOR	AND	XOR	AND
Classical [Yao86,GMW87]	8	8	4	4	2.5	2.5
P&P [BeaverMicaliRogaway90]	4	4	4	4	1	1
GRR3 [NaorPinkasSumner99]	3	3	4	4	1	1
Free XOR [KolesnikovSchneider08]	0	3	0	4	0	1
GRR2 [PinkasSchneiderSmartWilliams09]	**2**	**2**	**2**	**2**	**1**	**1**

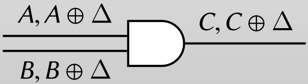
Scoreboard

	size ($\times\lambda$)		garble cost		eval cost	
	XOR	AND	XOR	AND	XOR	AND
Classical [Yao86,GMW87]	8	8	4	4	2.5	2.5
P&P [BeaverMicaliRogaway90]	4	4	4	4	1	1
GRR3 [NaorPinkasSumner99]	3	3	4	4	1	1
Free XOR [KolesnikovSchneider08]	0	3	0	4	0	1
GRR2 [PinkasSchneiderSmartWilliams09]	**2**	**2**	**2**	**2**	**1**	**1**

- Depending on circuit, either Free-XOR or GRR2 may be better
- Two techniques are **incompatible!** (can't guarantee $C_0 \oplus C_1 = \Delta$)

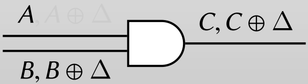
Half Gates [ZahurRosulekEvans15]

What if garbler knows in advance the truth value on one input wire?



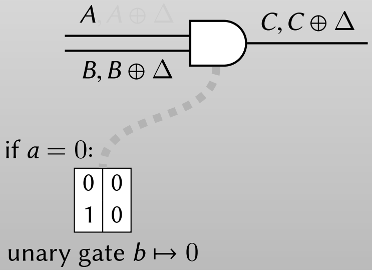
Half Gates [ZahurRosulekEvans15]

What if garbler knows in advance the truth value on one input wire?



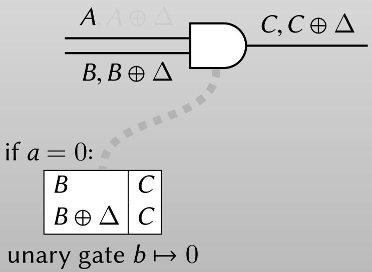
Half Gates [ZahurRosulekEvans15]

What if garbler knows in advance the truth value on one input wire?



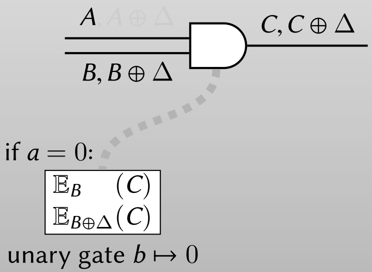
Half Gates [ZahurRosulekEvans15]

What if garbler knows in advance the truth value on one input wire?



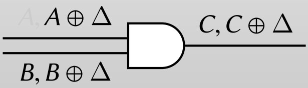
Half Gates [ZahurRosulekEvans15]

What if garbler knows in advance the truth value on one input wire?



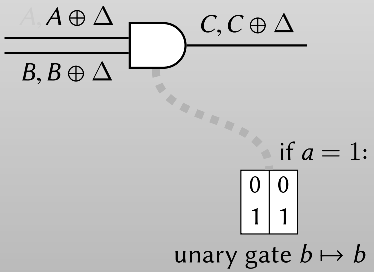
Half Gates [ZahurRosulekEvans15]

What if garbler knows in advance the truth value on one input wire?



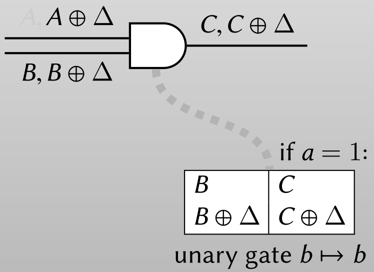
Half Gates [ZahurRosulekEvans15]

What if garbler knows in advance the truth value on one input wire?



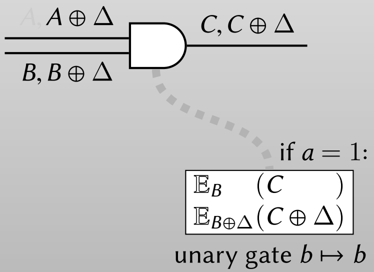
Half Gates [ZahurRosulekEvans15]

What if garbler knows in advance the truth value on one input wire?



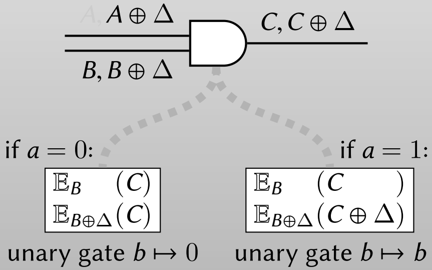
Half Gates [ZahurRosulekEvans15]

What if garbler knows in advance the truth value on one input wire?



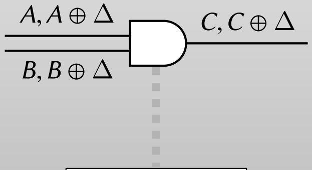
Half Gates [ZahurRosulekEvans15]

What if garbler knows in advance the truth value on one input wire?



Half Gates [ZahurRosulekEvans15]

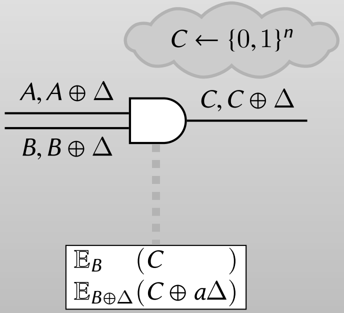
What if garbler knows in advance the truth value on one input wire?



$$\begin{array}{l} \mathbb{E}_B(C) \\ \mathbb{E}_{B\oplus\Delta}(C \oplus a\Delta) \end{array}$$

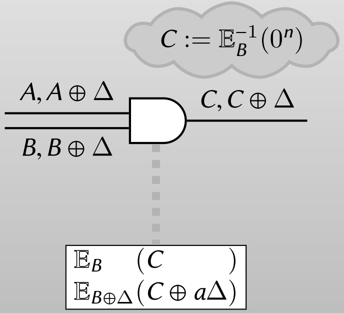
Half Gates [ZahurRosulekEvans15]

What if garbler knows in advance the truth value on one input wire?



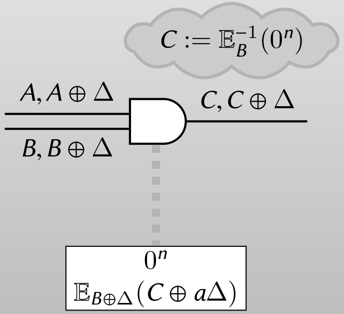
Half Gates [ZahurRosulekEvans15]

What if garbler knows in advance the truth value on one input wire?



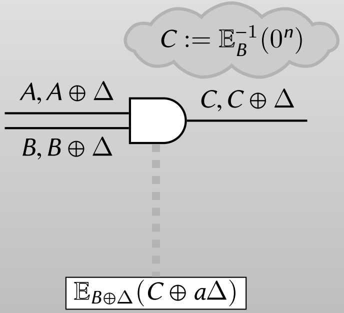
Half Gates [ZahurRosulekEvans15]

What if garbler knows in advance the truth value on one input wire?



Half Gates [ZahurRosulekEvans15]

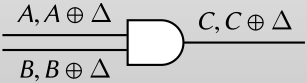
What if garbler knows in advance the truth value on one input wire?



Fine print: permute ciphertexts with permute-and-point.

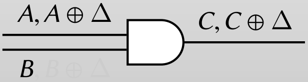
Half Gates [ZahurRosulekEvans15]

What if evaluator knows in advance the truth value on one input wire?



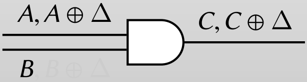
Half Gates [ZahurRosulekEvans15]

What if evaluator knows in advance the truth value on one input wire?



Half Gates [ZahurRosulekEvans15]

What if evaluator knows in advance the truth value on one input wire?

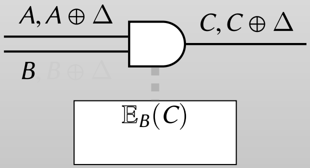


Evaluator has B (knows FALSE):

$\Rightarrow$ should obtain C (FALSE)

Half Gates [ZahurRosulekEvans15]

What if evaluator knows in advance the truth value on one input wire?

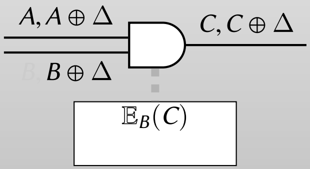


Evaluator has B (knows FALSE):

$\Rightarrow$ should obtain C (FALSE)

Half Gates [ZahurRosulekEvans15]

What if evaluator knows in advance the truth value on one input wire?



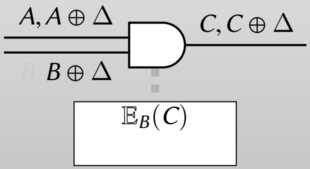
Evaluator has B (knows FALSE):

$\Rightarrow$ should obtain C (FALSE)

Evaluator has $B \oplus \Delta$ (knows TRUE):

Half Gates [ZahurRosulekEvans15]

What if evaluator knows in advance the truth value on one input wire?



Evaluator has B (knows FALSE):

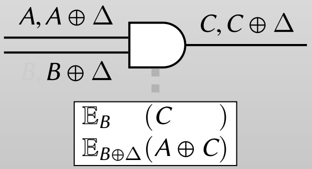
$\Rightarrow$ should obtain C (FALSE)

Evaluator has $B \oplus \Delta$ (knows TRUE):

$\Rightarrow$ should be able to *transfer* truth value from "a" wire to "c" wire

Half Gates [ZahurRosulekEvans15]

What if evaluator knows in advance the truth value on one input wire?



Evaluator has B (knows FALSE):

$\Rightarrow$ should obtain C (FALSE)

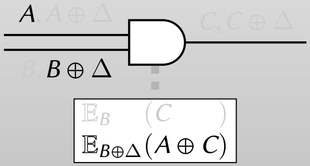
Evaluator has $B \oplus \Delta$ (knows TRUE):

$\Rightarrow$ should be able to *transfer* truth value from "a" wire to "c" wire

- Suffices to learn $A \oplus C$

Half Gates [ZahurRosulekEvans15]

What if evaluator knows in advance the truth value on one input wire?



Evaluator has B (knows FALSE):

$\Rightarrow$ should obtain C (FALSE)

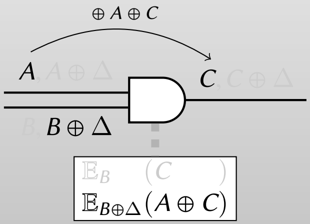
Evaluator has $B \oplus \Delta$ (knows TRUE):

$\Rightarrow$ should be able to *transfer* truth value from "a" wire to "c" wire

- Suffices to learn $A \oplus C$

Half Gates [ZahurRosulekEvans15]

What if evaluator knows in advance the truth value on one input wire?



Evaluator has B (knows FALSE):

$\Rightarrow$ should obtain C (FALSE)

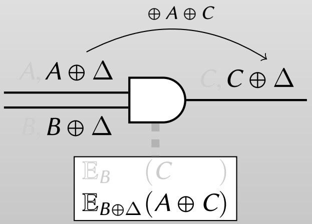
Evaluator has $B \oplus \Delta$ (knows TRUE):

$\Rightarrow$ should be able to *transfer* truth value from "a" wire to "c" wire

- Suffices to learn $A \oplus C$

Half Gates [ZahurRosulekEvans15]

What if evaluator knows in advance the truth value on one input wire?



Evaluator has B (knows FALSE):

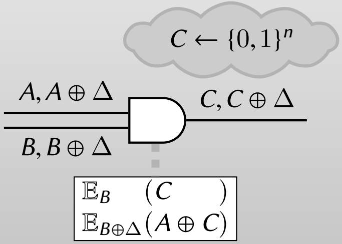
- $\Rightarrow$ should obtain C (FALSE)

Evaluator has $B \oplus \Delta$ (knows TRUE):

- $\Rightarrow$ should be able to *transfer* truth value from "a" wire to "c" wire
  - Suffices to learn $A \oplus C$

Half Gates [ZahurRosulekEvans15]

What if evaluator knows in advance the truth value on one input wire?



Evaluator has B (knows FALSE):

$\Rightarrow$ should obtain C (FALSE)

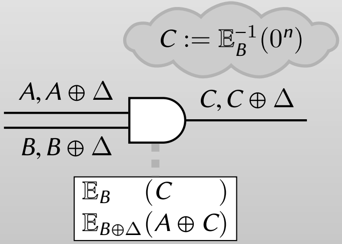
Evaluator has $B \oplus \Delta$ (knows TRUE):

$\Rightarrow$ should be able to *transfer* truth value from "a" wire to "c" wire

- Suffices to learn $A \oplus C$

Half Gates [ZahurRosulekEvans15]

What if evaluator knows in advance the truth value on one input wire?



Evaluator has B (knows FALSE):

$\Rightarrow$ should obtain C (FALSE)

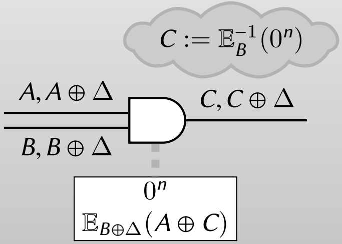
Evaluator has $B \oplus \Delta$ (knows TRUE):

$\Rightarrow$ should be able to *transfer* truth value from "a" wire to "c" wire

- Suffices to learn $A \oplus C$

Half Gates [ZahurRosulekEvans15]

What if evaluator knows in advance the truth value on one input wire?



Evaluator has B (knows FALSE):

$\Rightarrow$ should obtain C (FALSE)

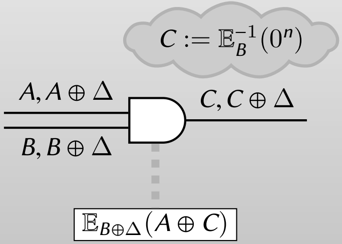
Evaluator has $B \oplus \Delta$ (knows TRUE):

$\Rightarrow$ should be able to *transfer* truth value from "a" wire to "c" wire

- Suffices to learn $A \oplus C$

Half Gates [ZahurRosulekEvans15]

What if evaluator knows in advance the truth value on one input wire?



Evaluator has B (knows FALSE):

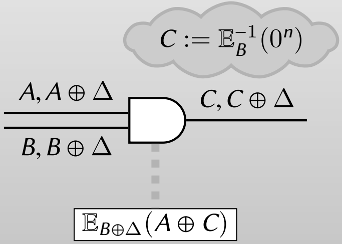
- ⇒ should obtain C (FALSE)

Evaluator has $B \oplus \Delta$ (knows TRUE):

- ⇒ should be able to *transfer* truth value from "a" wire to "c" wire
  - Suffices to learn $A \oplus C$

Half Gates [ZahurRosulekEvans15]

What if evaluator knows in advance the truth value on one input wire?



Evaluator has B (knows FALSE):

$\Rightarrow$ should obtain C (FALSE)

Evaluator has $B \oplus \Delta$ (knows TRUE):

$\Rightarrow$ should be able to *transfer* truth value from "a" wire to "c" wire

- Suffices to learn $A \oplus C$

Fine print: no need for permute-and-point here

Two halves make a whole!

$$a \wedge b$$

Two halves make a whole!

$$a \wedge b = (a \oplus r \oplus r) \wedge b$$

- Garbler chooses random bit r

Two halves make a whole!

$$\begin{aligned} a \wedge b &= (a \oplus r \oplus r) \wedge b \\ &= [(a \oplus r) \wedge b] \oplus [r \wedge b] \end{aligned}$$

- Garbler chooses random bit r

Two halves make a whole!

$$\begin{aligned} a \wedge b &= (a \oplus r \oplus r) \wedge b \\ &= [(a \oplus r) \wedge b] \oplus [r \wedge b] \end{aligned}$$

- Garbler chooses random bit r
- Arrange for evaluator to learn $a \oplus r$ in the clear

Two halves make a whole!

$$\begin{aligned} a \wedge b &= (a \oplus r \oplus r) \wedge b \\ &= \underbrace{[(a \oplus r) \wedge b]}_{\text{one input known to evaluator}} \oplus [r \wedge b] \end{aligned}$$

- Garbler chooses random bit r
- Arrange for evaluator to learn $a \oplus r$ in the clear

Two halves make a whole!

$$\begin{aligned} a \wedge b &= (a \oplus r \oplus r) \wedge b \\ &= [(a \oplus r) \wedge b] \oplus \underbrace{[r \wedge b]}_{\text{one input known to garbler}} \end{aligned}$$

- Garbler chooses random bit r
- Arrange for evaluator to learn $a \oplus r$ in the clear

Two halves make a whole!

$$\begin{aligned} a \wedge b &= (a \oplus r \oplus r) \wedge b \\ &= [(a \oplus r) \wedge b] \oplus \underbrace{[r \wedge b]}_{\text{one input known to garbler}} \end{aligned}$$

- Garbler chooses random bit r
- Arrange for evaluator to learn $a \oplus r$ in the clear
- Total cost = 2 "half gates" + 1 XOR gate = 2 ciphertexts

Two halves make a whole!

$$\begin{aligned} a \wedge b &= (a \oplus r \oplus r) \wedge b \\ &= \big[(a \oplus r) \wedge b\big] \oplus \underbrace{\big[r \wedge b\big]}_{\text{one input known to garbler}} \end{aligned}$$

- Garbler chooses random bit r
  - r = color bit of FALSE wire label A
- Arrange for evaluator to learn $a \oplus r$ in the clear
  - $a \oplus r$ = color bit of wire label evaluator gets (A or $A \oplus \Delta$)
- Total cost = 2 "half gates" + 1 XOR gate = 2 ciphertexts

Scoreboard

	size ($\times\lambda$)		garble cost		eval cost	
	XOR	AND	XOR	AND	XOR	AND
Classical [Yao86,GMW87]	8	8	4	4	2.5	2.5
P&P [BeaverMicaliRogaway90]	4	4	4	4	1	1
GRR3 [NaorPinkasSumner99]	3	3	4	4	1	1
Free XOR [KolesnikovSchneider08]	0	3	0	4	0	1
GRR2 [PinkasSchneiderSmartWilliams09]	2	2	2	2	1	1
Half gates [ZahurRosulekEvans15]	**0**	**2**	**0**	**4**	**0**	**2**

Open Question

Can we do better than half-gates?

NO

[ZahurRosulekEvans15]

Can't garble an AND gate with < 2 ciphertexts

Open Question

Can we do better than half-gates?

NO

[ZahurRosulekEvans15]

Can't garble an AND gate with < 2 ciphertexts

YES

[BallMalkinRosulek16, KempkaKikuchiSuzuki16]

Can garble an AND gate with 1 ciphertext

Open Question

Can we do better than half-gates?

NO

[ZahurRosulekEvans15]

Can't garble an AND gate with < 2 ciphertexts. . .

. . . using "standard techniques" (details to follow)

YES

[BallMalkinRosulek16, KempkaKikuchiSuzuki16]

Can garble an AND gate with 1 ciphertext

Open Question

Can we do better than half-gates?

NO

[ZahurRosulekEvans15]

Can't garble an AND gate with < 2 ciphertexts...

... using "standard techniques" (details to follow)

YES

[BallMalkinRosulek16, KempkaKikuchiSuzuki16]

Can garble an AND gate with 1 ciphertext...

... but the construction doesn't compose with itself or Free-XOR

Open Question

Can we do better than half-gates?

in any useful way?

NO

[ZahurRosulekEvans15]

Can't garble an AND gate with < 2 ciphertexts...

... using "standard techniques" (details to follow)

YES

[BallMalkinRosulek16, KempkaKikuchiSuzuki16]

Can garble an AND gate with 1 ciphertext...

... but the construction doesn't compose with itself or Free-XOR

Lower bound in more detail

Theorem [ZahurRosulekEvans15]

A garbled AND gate requires 2 ciphertexts using "standard techniques."

Lower bound in more detail

Theorem [ZahurRosulekEvans15]

A garbled AND gate requires 2 ciphertexts using "standard techniques."

Standard techniques: *point-permute + calls to random oracle + everything else linear.*

Important: only the constructions, not adversary, must be linear!

Linear (gate) garbling: details

$\$_{\mathbb{F}}$

Garbler:

- samples field elements

Linear (gate) garbling: details

Garbler:

- samples field elements
- calls random oracle

Linear (gate) garbling: details

$\$_{\mathbb{F}}$

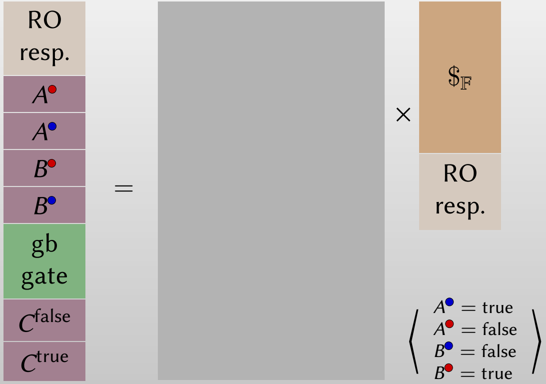
RO resp.

$$\left\langle \begin{array}{l} A^{\color{blue}\bullet} = \text{true} \\ A^{\color{red}\bullet} = \text{false} \\ B^{\color{blue}\bullet} = \text{false} \\ B^{\color{red}\bullet} = \text{true} \end{array} \right\rangle$$

Garbler:

- samples field elements
- calls random oracle
- picks secret "color mapping"

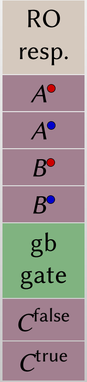
Linear (gate) garbling: details



Garbler:

- samples field elements
- calls random oracle
- picks secret “color mapping”
- applies linear combination

Linear (gate) garbling: details

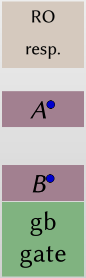


Evaluator:

Garbler:

- samples field elements
- calls random oracle
- picks secret "color mapping"
- applies linear combination

Linear (gate) garbling: details



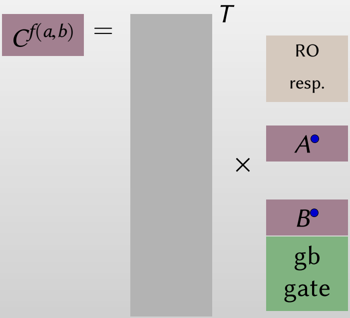
Evaluator:

- knows subset of garbler's values (and knows colors)

Garbler:

- samples field elements
- calls random oracle
- picks secret "color mapping"
- applies linear combination

Linear (gate) garbling: details



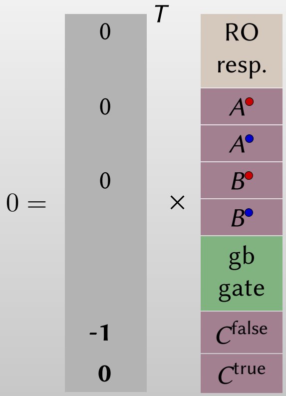
Evaluator:

- knows subset of garbler's values (and knows colors)
- applies linear combination
- result is output label

Garbler:

- samples field elements
- calls random oracle
- picks secret "color mapping"
- applies linear combination

Linear (gate) garbling: details



Evaluator:

- knows subset of garbler's values (and knows colors)
- applies linear combination
- result is output label

Garbler:

- samples field elements
- calls random oracle
- picks secret "color mapping"
- applies linear combination

Linear (gate) garbling: details

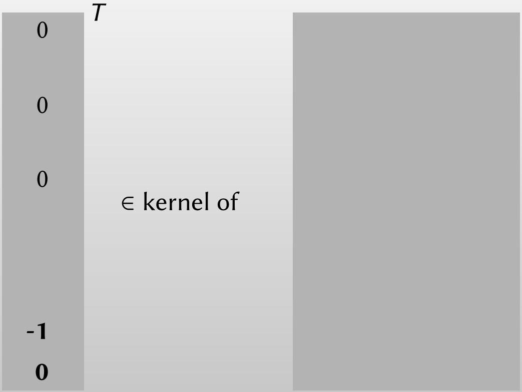
Evaluator:

- knows subset of garbler's values (and knows colors)
- applies linear combination
- result is output label

Garbler:

- samples field elements
- calls random oracle
- picks secret "color mapping"
- applies linear combination

Linear (gate) garbling: details



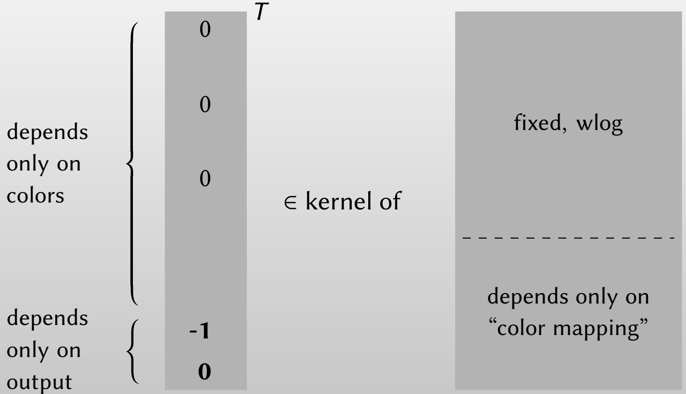
Evaluator:

- knows subset of garbler's values (and knows colors)
- applies linear combination
- result is output label

Garbler:

- samples field elements
- calls random oracle
- picks secret "color mapping"
- applies linear combination

Linear (gate) garbling: details



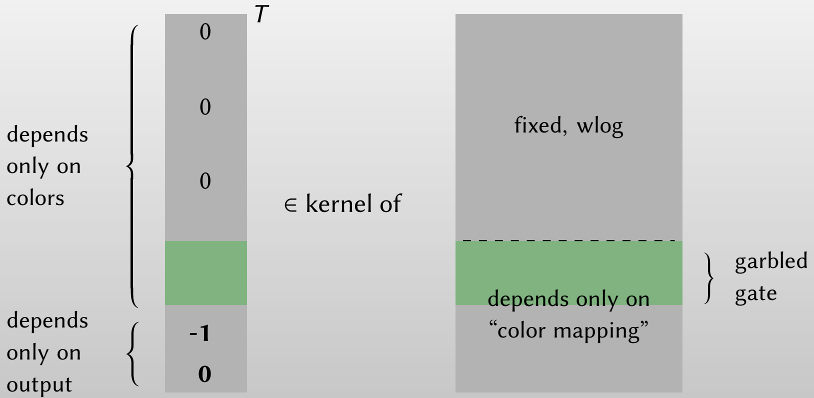
Evaluator:

- knows subset of garbler’s values (and knows colors)
- applies linear combination
- result is output label

Garbler:

- samples field elements
- calls random oracle
- picks secret “color mapping”
- applies linear combination

Linear (gate) garbling: details



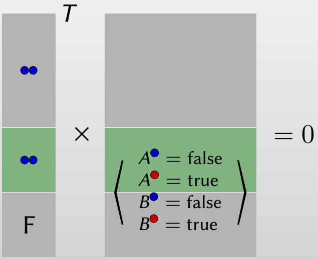
Evaluator:

- knows subset of garbler's values (and knows colors)
- applies linear combination
- result is output label

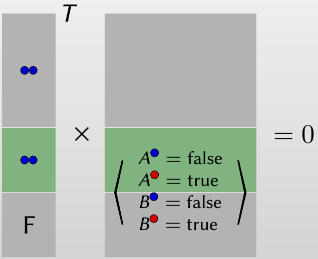
Garbler:

- samples field elements
- calls random oracle
- picks secret "color mapping"
- applies linear combination

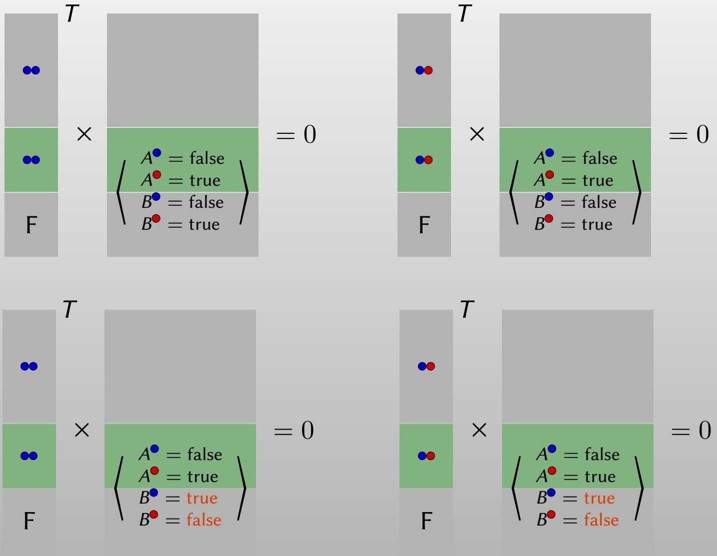
Lower bound (part 1)



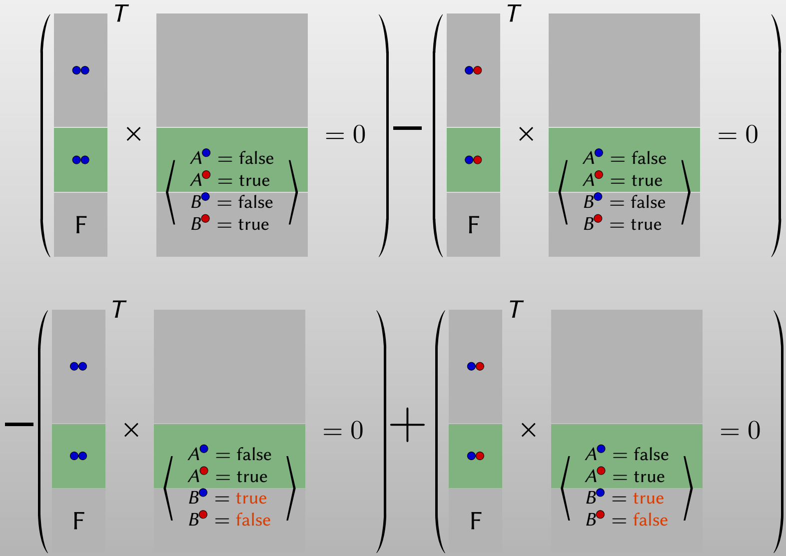
Lower bound (part 1)



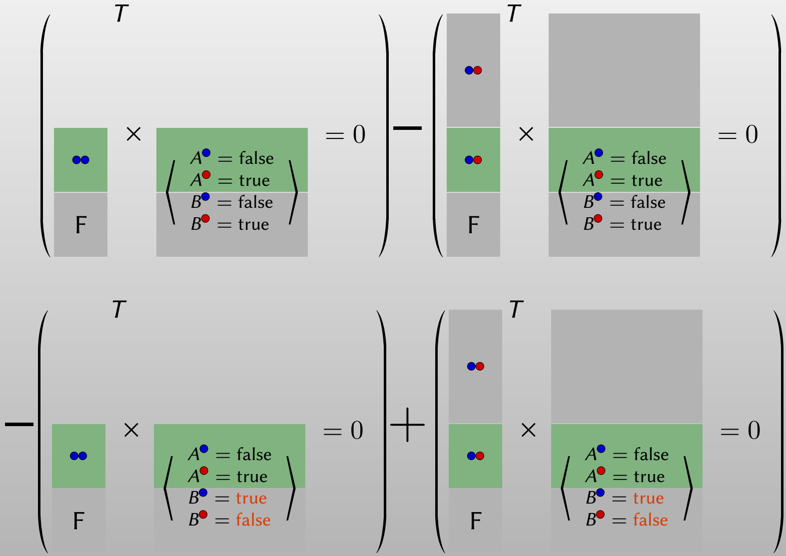
Lower bound (part 1)



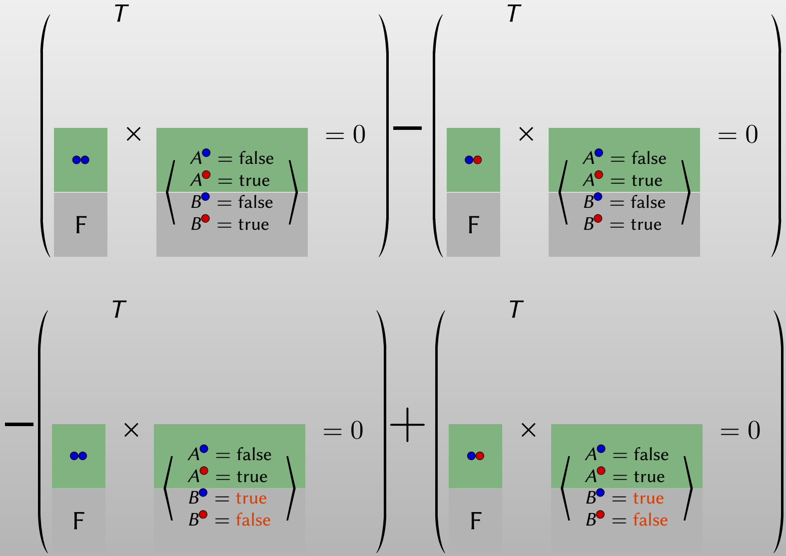
Lower bound (part 1)



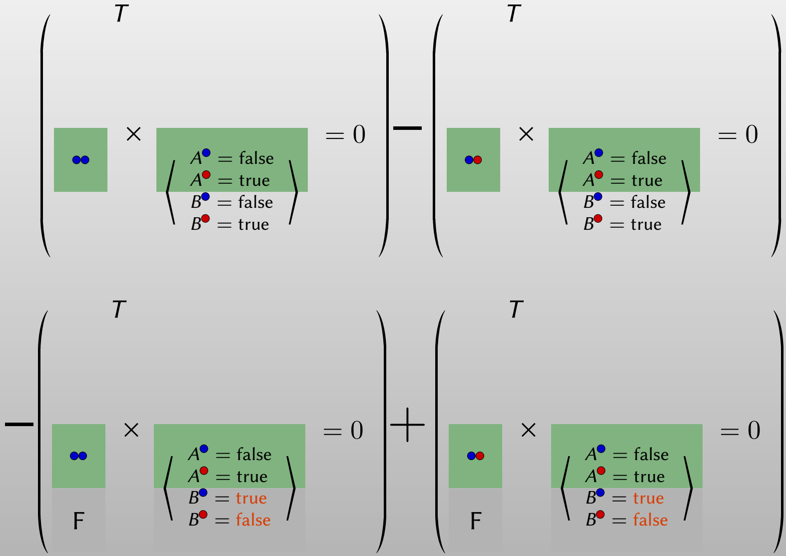
Lower bound (part 1)



Lower bound (part 1)



Lower bound (part 1)



Lower bound (part 1)

$$\left(\boxed{{\color{blue}\bullet}{\color{blue}\bullet}} \times \boxed{\left\langle \begin{array}{l} A^{\color{blue}\bullet} = \text{false} \\ A^{\color{red}\bullet} = \text{true} \\ B^{\color{blue}\bullet} = \text{false} \\ B^{\color{red}\bullet} = \text{true} \end{array} \right\rangle} = 0 \right)^T - \left(\boxed{{\color{blue}\bullet}{\color{red}\bullet}} \times \boxed{\left\langle \begin{array}{l} A^{\color{blue}\bullet} = \text{false} \\ A^{\color{red}\bullet} = \text{true} \\ B^{\color{blue}\bullet} = \text{false} \\ B^{\color{red}\bullet} = \text{true} \end{array} \right\rangle} = 0 \right)^T$$

$$- \left(\boxed{{\color{blue}\bullet}{\color{blue}\bullet}} \times \boxed{\left\langle \begin{array}{l} A^{\color{blue}\bullet} = \text{false} \\ A^{\color{red}\bullet} = \text{true} \\ B^{\color{blue}\bullet} = {\color{orangered}\text{true}} \\ B^{\color{red}\bullet} = {\color{orangered}\text{false}} \end{array} \right\rangle} = 0 \right)^T + \left(\boxed{{\color{blue}\bullet}{\color{red}\bullet}} \times \boxed{\left\langle \begin{array}{l} A^{\color{blue}\bullet} = \text{false} \\ A^{\color{red}\bullet} = \text{true} \\ B^{\color{blue}\bullet} = {\color{orangered}\text{true}} \\ B^{\color{red}\bullet} = {\color{orangered}\text{false}} \end{array} \right\rangle} = 0 \right)^T$$

Lower bound (part 2)

$$(\boldsymbol{v} - \boldsymbol{v}')(\boldsymbol{M} - \boldsymbol{M}') = 0$$

Lower bound (part 2)

$$(\boldsymbol{v} - \boldsymbol{v}')(\boldsymbol{M} - \boldsymbol{M}') = 0$$

Rest of proof:

- v, v' distinct (otherwise privacy can be violated)

$\Rightarrow$ $\dim(\ker(M - M')) \geq 1$

Lower bound (part 2)

$$(\boldsymbol{v} - \boldsymbol{v}')(\boldsymbol{M} - \boldsymbol{M}') = 0$$

Rest of proof:

- v, v' distinct (otherwise privacy can be violated)

$\Rightarrow$ $\dim(\ker(M - M')) \geq 1$

- M, M' distinct (otherwise correctness is violated)

$\Rightarrow$ $\dim(\mathrm{rowspace}(M - M')) \geq 1$

Lower bound (part 2)

$$(\boldsymbol{v} - \boldsymbol{v}')(\boldsymbol{M} - \boldsymbol{M}') = 0$$

Rest of proof:

- v, v' distinct (otherwise privacy can be violated)

$\Rightarrow$ $\dim(\ker(M - M')) \geq 1$

- M, M' distinct (otherwise correctness is violated)

$\Rightarrow$ $\dim(\text{rowspace}(M - M')) \geq 1$

$\Rightarrow$ M, M' have at least 2 rows

$\Rightarrow$ **garbled gate has at least 2λ bits**

Limitations of model

Limitation:
point-and-permute is sole source of non-linearity

Opportunity:
smaller GC by using alternatives to point-and-permute?
[BallMalkinRosulek16,KempkaKikuchiSuzuki16]

Limitations of model

Limitation:
point-and-permute is sole source of non-linearity

Opportunity:
smaller GC by using alternatives to point-and-permute?
[BallMalkinRosulek16,KempkaKikuchiSuzuki16]

optimal for gate-by-gate garbling, in {AND, XOR, NOT} basis

smaller GC by garbling larger "chunks" of a circuit at a time?
[MalkinPastroShelat16,BallMalkinRosulek16]

Limitations of model

Limitation:	**Opportunity:**
point-and-permute is sole source of non-linearity	smaller GC by using alternatives to point-and-permute? [BallMalkinRosulek16,KempkaKikuchiSuzuki16]
optimal for gate-by-gate garbling, in {AND, XOR, NOT} basis	smaller GC by garbling larger "chunks" of a circuit at a time? [MalkinPastroShelat16,BallMalkinRosulek16]
model doesn't allow nested calls to random oracle	maybe nesting leads to smaller GC? e.g.: $H(H(A) \oplus B)$

Limitations of model

Limitation:	**Opportunity:**
point-and-permute is sole source of non-linearity	smaller GC by using alternatives to point-and-permute? [BallMalkinRosulek16,KempkaKikuchiSuzuki16]
optimal for gate-by-gate garbling, in {AND, XOR, NOT} basis	smaller GC by garbling larger "chunks" of a circuit at a time? [MalkinPastroShelat16,BallMalkinRosulek16]
model doesn't allow nested calls to random oracle	maybe nesting leads to smaller GC? e.g.: $H(H(A) \oplus B)$
linear!	maybe non-linear (but still practical) techniques lead to smaller GC? e.g., compute $GF(2^\lambda)$-inverse of a wire label

How to circumvent lower bound

[KempkaKikuchiSuzuki16]: indirection between color bits & non-linearity

- Based on color bits, evaluator decrypts appropriate **1-bit ciphertext**
- 1-bit payload determines **choice of linear combination**

[BallMalkinRosulek16]: more non-linearity than 1 bit / wire

- Instead of color *bit*, wire labels have color **ternary value**
- (details follow)

How to circumvent lower bound

[KempkaKikuchiSuzuki16]: indirection between color bits & non-linearity

- Based on color bits, evaluator decrypts appropriate **1-bit ciphertext**
- 1-bit payload determines **choice of linear combination**

[BallMalkinRosulek16]: more non-linearity than 1 bit / wire

- Instead of color *bit*, wire labels have color **ternary value**
- (details follow)

Neither construction is **self-composable**:

- Input wire labels must have special form: $(A, A+\Delta), (B, B+\Delta)$
- Garbled gate produces output labels that don't have special form

Summary

	size ($\times\lambda$)		garble cost		eval cost	
	XOR	AND	XOR	AND	XOR	AND
Classical [Yao86,GMW87]	8	8	4	4	2.5	2.5
P&P [BeaverMicaliRogaway90]	4	4	4	4	1	1
GRR3 [NaorPinkasSumner99]	3	3	4	4	1	1
Free XOR [KolesnikovSchneider08]	0	3	0	4	0	1
GRR2 [PinkasSchneiderSmartWilliams09]	2	2	2	2	1	1
Half gates [ZahurRosulekEvans15]	0	2	0	4	0	2

Summary

	size ($\times\lambda$)		garble cost		eval cost	
	XOR	AND	XOR	AND	XOR	AND
Classical [Yao86,GMW87]	8	8	4	4	2.5	2.5
P&P [BeaverMicaliRogaway90]	4	4	4	4	1	1
GRR3 [NaorPinkasSumner99]	3	3	4	4	1	1
Free XOR [KolesnikovSchneider08]	0	3	0	4	0	1
GRR2 [PinkasSchneiderSmartWilliams09]	2	2	2	2	1	1
Half gates [ZahurRosulekEvans15]	0	2	0	4	0	2
Your next great idea?	0	1				

Can we do better than half-gates in any useful way?

Roadmap

1 **Standard garbled circuits:** core concepts & constructions

- Yao's construction, security definitions, optimized constructions (row reduction, free XOR, half-gates)

2 **New directions** beyond boolean circuits

- Garbled arithmetic circuits & RAM programs

Beyond Boolean Circuits

What about all the interesting things that are clunky to write as a boolean circuit?

[BallMalkinRosulek16]

Beyond Boolean Circuits

What about all the interesting things that are clunky to write as a boolean circuit?

[BallMalkinRosulek16]

- New generalizations of garbled circuit constructions
- Improved garbling for arithmetic computations
- Improved garbling for high-fan-in boolean computations

Generalized Free XOR

Free XOR:

Wire carries a *truth value* from $\{0,1\}$

Wire labels are bit strings $\{0,1\}^\lambda$.

Global wire-label-offset $\Delta \in \{0,1\}^\lambda$

FALSE wire label is A
TRUE wire label is $A \oplus \Delta$

Generalized Free XOR

Free XOR:

Wire carries a *truth value* from $\{0,1\}$

Wire labels are bit strings $\{0,1\}^\lambda$.

Global wire-label-offset $\Delta \in \{0,1\}^\lambda$

FALSE wire label is A
TRUE wire label is $A \oplus \Delta$

Generalized Free XOR:

Wire carries a *truth value* from $\mathbb{Z}_m$

Generalized Free XOR

Free XOR:

Wire carries a *truth value* from $\{0,1\}$

Wire labels are bit strings $\{0,1\}^\lambda$.

Global wire-label-offset $\Delta \in \{0,1\}^\lambda$

FALSE wire label is A
TRUE wire label is $A \oplus \Delta$

Generalized Free XOR:

Wire carries a *truth value* from $\mathbb{Z}_m$

Wire labels are tuples $(\mathbb{Z}_m)^\lambda$.

Global wire-label-offset $\Delta \in (\mathbb{Z}_m)^\lambda$

Generalized Free XOR

Free XOR:

Wire carries a *truth value* from $\{0,1\}$

Wire labels are bit strings $\{0,1\}^\lambda$.

Global wire-label-offset $\Delta \in \{0,1\}^\lambda$

FALSE wire label is A
TRUE wire label is $A \oplus \Delta$

Generalized Free XOR:

Wire carries a *truth value* from $\mathbb{Z}_m$

Wire labels are tuples $(\mathbb{Z}_m)^\lambda$.

Global wire-label-offset $\Delta \in (\mathbb{Z}_m)^\lambda$

Wire label encoding truth value
$a \in \mathbb{Z}_m$ is $A + a\Delta$

Generalized Free XOR

Free XOR:

Wire carries a *truth value* from $\{0,1\}$

Wire labels are bit strings $\{0,1\}^\lambda$.

Global wire-label-offset $\Delta \in \{0,1\}^\lambda$

FALSE wire label is A
TRUE wire label is $A \oplus \Delta$

$\oplus$ is componentwise addition mod 2

Generalized Free XOR:

Wire carries a *truth value* from $\mathbb{Z}_m$

Wire labels are tuples $(\mathbb{Z}_m)^\lambda$.

Global wire-label-offset $\Delta \in (\mathbb{Z}_m)^\lambda$

Wire label encoding truth value
$a \in \mathbb{Z}_m$ is $A + a\Delta$

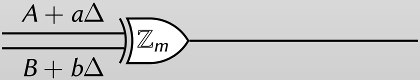
$+$ is componentwise addition mod m

Generalized Free XOR

Idea: Truth value $a \in \mathbb{Z}_m$ encoded by wire label $\underline{A + a\Delta} \in (\mathbb{Z}_m)^\lambda$

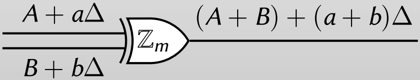
Generalized Free XOR

Idea: Truth value $a \in \mathbb{Z}_m$ encoded by wire label $\underline{A + a\Delta} \in (\mathbb{Z}_m)^\lambda$



Generalized Free XOR

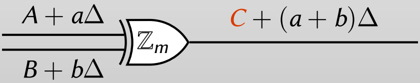
Idea: Truth value $a \in \mathbb{Z}_m$ encoded by wire label $\underline{A + a\Delta} \in (\mathbb{Z}_m)^\lambda$



Evaulator can simply add wire labels

Generalized Free XOR

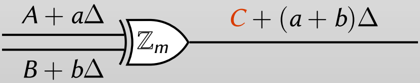
Idea: Truth value $a \in \mathbb{Z}_m$ encoded by wire label $\underline{A + a\Delta} \in (\mathbb{Z}_m)^\lambda$



Evaulator can simply add wire labels $\Rightarrow$ free garbled addition mod m

Generalized Free XOR

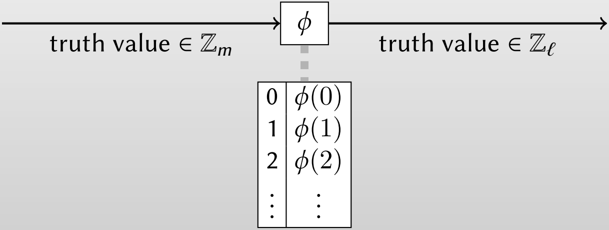
Idea: Truth value $a \in \mathbb{Z}_m$ encoded by wire label $\underline{A + a\Delta} \in (\mathbb{Z}_m)^\lambda$



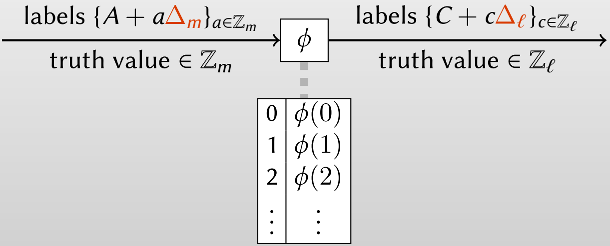
Evaulator can simply add wire labels $\Rightarrow$ free garbled addition mod m

- Free multiplication by public constant c, if $\gcd(c, m) = 1$

Garbling unary gates

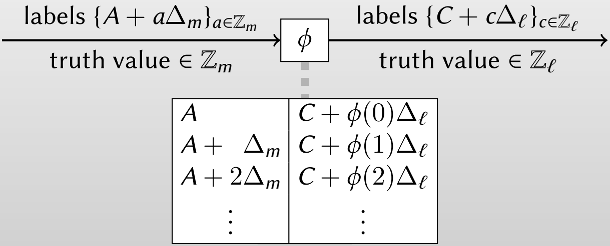


Garbling unary gates



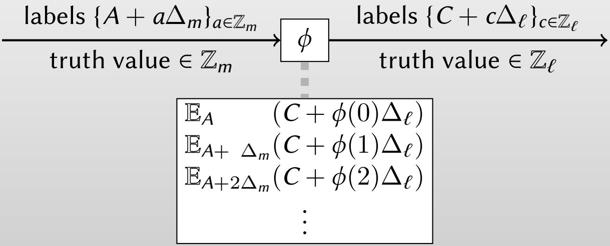
- Different "preferred modulus" on each wire $\Rightarrow$ different offsets Δ

Garbling unary gates



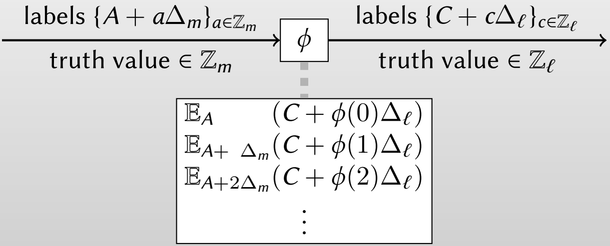
- Different “preferred modulus” on each wire $\Rightarrow$ different offsets Δ

Garbling unary gates



- Different "preferred modulus" on each wire $\Rightarrow$ different offsets Δ
- Cost: m ciphertexts
- Generalized point-and-permute: "color bit" from $\mathbb{Z}_m$

Garbling unary gates



- Different "preferred modulus" on each wire $\Rightarrow$ different offsets Δ
- Cost: m ciphertexts
- Generalized point-and-permute: "color bit" from $\mathbb{Z}_m$
- $m - 1$ using standard row reduction technique

Generalized garbling tools

We can efficiently garble any computation/circuit where:

- Each wire has a preferred modulus $\mathbb{Z}_m$
  - $\Rightarrow$ Wire-label-offset Δ_m global to all $\mathbb{Z}_m$-wires
- Addition gates: all wires touching gate have same modulus
  - $\Rightarrow$ Garbling cost: **free**
- Mult-by-constant gates: input/output wires have same modulus
  - $\Rightarrow$ Garbling cost: **free**
- Unary gates: $\mathbb{Z}_m$ input and $\mathbb{Z}_\ell$ output
  - $\Rightarrow$ Garbling cost: $m - 1$ ciphertexts

Generalized garbling tools

We can efficiently garble any computation/circuit where:

- Each wire has a preferred modulus $\mathbb{Z}_m$
  - $\Rightarrow$ Wire-label-offset Δ_m global to all $\mathbb{Z}_m$-wires
- Addition gates: all wires touching gate have same modulus
  - $\Rightarrow$ Garbling cost: **free**
- Mult-by-constant gates: input/output wires have same modulus
  - $\Rightarrow$ Garbling cost: **free**
- Unary gates: $\mathbb{Z}_m$ input and $\mathbb{Z}_\ell$ output
  - $\Rightarrow$ Garbling cost: $m - 1$ ciphertexts

Better basis for many computations than traditional boolean circuits!

Arithmetic computations

Example Scenario

Securely compute linear optimization problem on 32-bit values.

⇒ Almost all operations are **addition**, **multiplication**, etc

Arithmetic computations

Example Scenario

Securely compute linear optimization problem on 32-bit values.

$\Rightarrow$ Almost all operations are **addition**, **multiplication**, etc

“Standard approach”

- Represent 32-bit integers in binary
- Build circuit from boolean addition/multiplication subcircuits
- Garble with half-gates (AND costs 2, XOR costs 0)

Arithmetic computations

Example Scenario

Securely compute linear optimization problem on 32-bit values.

$\Rightarrow$ Almost all operations are **addition**, **multiplication**, etc

"Standard approach"

- Represent 32-bit integers in binary
- Build circuit from boolean addition/multiplication subcircuits
- Garble with half-gates (AND costs 2, XOR costs 0)

	cost (# ciphertexts)
addition	62
multiplication by public constant	758
multiplication	1200
squaring, cubing, etc	1864

Arithmetic computations

Using generalized garbling techniques:

- Think of arithmetic circuit: wires carry values in $\mathbb{Z}_{2^{32}}$

Arithmetic computations

Using generalized garbling techniques:

- Think of arithmetic circuit: wires carry values in $\mathbb{Z}_{2^{32}}$
- Garbled addition, multiplication by constant is **free**

	standard	ours
addition	62	0
multiplication by public constant	758	0
multiplication	1200	
squaring, cubing, etc	1864	

Arithmetic computations

Using generalized garbling techniques:

- Think of arithmetic circuit: wires carry values in $\mathbb{Z}_{2^{32}}$
- Garbled addition, multiplication by constant is **free**
- Multiplication mod m costs $O(m)$ ciphertexts!

	standard	ours
addition	62	0
multiplication by public constant	758	0
multiplication	1200	8589934590
squaring, cubing, etc	1864	4294967295

Arithmetic computations

$$\text{instead of } \mathbb{Z}_{4294967296}$$

$$\downarrow$$

$$\text{use } \mathbb{Z}_{6469693230}$$

Arithmetic computations

$$\text{instead of } \mathbb{Z}_{4294967296} = \mathbb{Z}_{2^{32}}$$

$$\downarrow$$

$$\text{use } \mathbb{Z}_{6469693230}$$

Arithmetic computations

$$\text{instead of } \mathbb{Z}_{4294967296} = \mathbb{Z}_{2^{32}}$$

$$\downarrow$$

$$\text{use } \mathbb{Z}_{6469693230} = \mathbb{Z}_{2 \cdot 3 \cdot 5 \cdot 7 \cdots 29}$$

Arithmetic computations

CRT residue number system!

- Generalized garbling scheme supports many moduli in same circuit
- Represent 32-bit integer x as $x \% 2,\ x \% 3,\ x \% 5, \ldots,\ x \% 29$
- Do all arithmetic in each residue (each with small modulus)

	standard	madness
addition	62	0
mult by public constant	758	0
multiplication	1200	25769803776
squaring, cubing, etc	1864	4294967296

Arithmetic computations

CRT residue number system!

- Generalized garbling scheme supports many moduli in same circuit
- Represent 32-bit integer x as $x \% 2,\ x \% 3,\ x \% 5, \ldots,\ x \% 29$
- Do all arithmetic in each residue (each with small modulus)

	standard	madness	**CRT**
addition	62	0	**0**
mult by public constant	758	0	**0**
multiplication	1200	25769803776	**238** $\approx 2(2+3+5+$
squaring, cubing, etc	1864	4294967296	**119**

Arithmetic computations

CRT residue number system!

- Generalized garbling scheme supports many moduli in same circuit
- Represent 32-bit integer x as $x \,\%\, 2,\ x \,\%\, 3,\ x \,\%\, 5, \ldots,\ x \,\%\, 29$
- Do all arithmetic in each residue (each with small modulus)

	standard	madness	**CRT**
addition	62	0	**0**
mult by public constant	758	0	**0**
multiplication	1200	25769803776	**238** $\approx 2(2+3+5+$
squaring, cubing, etc	1864	4294967296	**119**

works well for arithmetic on $\mathbb{Z}$, unless you like working mod 6469693230

High fan-in computations

Example Scenario

Securely compute boolean circuit with **high-fan-in threshold gates.**

- fan-in-100: AND, OR, majority, threshold, etc.

High fan-in computations

Example Scenario

Securely compute boolean circuit with **high-fan-in threshold gates.**

- fan-in-100: AND, OR, majority, threshold, etc.

"Standard approach"

- Express threshold gates in terms of fan-in-2 gates
- Garble with half-gates (AND costs 2, XOR costs 0)

High fan-in computations

Example Scenario

Securely compute boolean circuit with **high-fan-in threshold gates.**

- fan-in-100: AND, OR, majority, threshold, etc.

"Standard approach"

- Express threshold gates in terms of fan-in-2 gates
- Garble with half-gates (AND costs 2, XOR costs 0)

	cost (# ciphertexts)
AND/OR	198
majority	948

High fan-in computations

Idea: AND is unary function of a sum

$$\text{AND}(x_1, \ldots, x_{100}) = [\sum x_i \stackrel{?}{=} 100]$$

High fan-in computations

Idea: AND is unary function of a sum

$$\mathsf{AND}(x_1, \ldots, x_{100}) = [\textstyle\sum x_i \stackrel{?}{=} 100] = [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod{101}]$$

High fan-in computations

Idea: AND is **unary function of a sum**

$$\text{AND}(x_1,\ldots,x_{100}) = [\sum x_i \stackrel{?}{=} 100] = [\sum x_i \stackrel{?}{\equiv} 100 \pmod{101}]$$

- Represent each bit on a $\mathbb{Z}_{101}$-wire
- Cost to garble sum in $\mathbb{Z}_{101}$ is **free**
- "$v \stackrel{?}{=} 100$" is simple $\mathbb{Z}_{101}$-unary gate $\Rightarrow$ cost = 100

High fan-in computations

Idea: AND is unary function of a sum

$$\text{AND}(x_1,\ldots,x_{100}) = [\textstyle\sum x_i \stackrel{?}{=} 100] = [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod{101}]$$

- Represent each bit on a $\mathbb{Z}_{101}$-wire
- Cost to garble sum in $\mathbb{Z}_{101}$ is **free**
- "$v \stackrel{?}{=} 100$" is simple $\mathbb{Z}_{101}$-unary gate $\Rightarrow$ cost = 100

	standard	better
AND/OR	198	100
majority	948	

High fan-in computations

Idea: AND is **unary function of a sum**

$$\text{AND}(x_1, \ldots, x_{100}) = [\textstyle\sum x_i \stackrel{?}{=} 100] = [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod{101}]$$

- Represent each bit on a $\mathbb{Z}_{101}$-wire
- Cost to garble sum in $\mathbb{Z}_{101}$ is **free**
- "$v \stackrel{?}{=} 100$" is simple $\mathbb{Z}_{101}$-unary gate $\Rightarrow$ cost = 100

	standard	better
AND/OR	198	100
majority	948	100

Same logic for $\text{MAJ}(x_1, \ldots, x_{100}) = [\sum_i x_i \stackrel{?}{>} 50]$

High fan-in computations

instead of $\mathbb{Z}_{101}$

$\downarrow$

use $\mathbb{Z}_{210}$

High fan-in computations

$$\text{instead of } \mathbb{Z}_{101}$$

$$\downarrow$$

$$\text{use } \mathbb{Z}_{210} = \mathbb{Z}_{2\cdot 3\cdot 5\cdot 7}$$

High fan-in computations

Insight: take advantage of multiple moduli

$$\mathsf{AND}(x_1,\ldots,x_{100}) = [\sum x_i \stackrel{?}{=} 100] = [\sum x_i \stackrel{?}{\equiv} 100 \pmod{210}]$$

High fan-in computations

Insight: take advantage of multiple moduli

$$\mathrm{AND}(x_1,\ldots,x_{100}) = [\textstyle\sum x_i \stackrel{?}{=} 100] = [\sum x_i \stackrel{?}{\equiv} 100 \pmod{210}]$$
$$= \mathrm{AND}\Big([\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod{2}], [\sum x_i \stackrel{?}{\equiv} 100 \pmod{3}], \ldots\Big)$$

High fan-in computations

Insight: take advantage of **multiple moduli**

$$\mathrm{AND}(x_1, \ldots, x_{100}) = [\textstyle\sum x_i \stackrel{?}{=} 100] = [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod{210}]$$

$$= \mathrm{AND}\Big([\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod 2], [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod 3], \ldots\Big)$$

Residue number system approach

- Represent each bit redundantly mod 2, mod 3, mod 5, . . .

High fan-in computations

Insight: take advantage of **multiple moduli**

$$\mathrm{AND}(x_1, \ldots, x_{100}) = [\textstyle\sum x_i \stackrel{?}{=} 100] = [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod{210}]$$
$$= \mathrm{AND}\Big([\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod 2], [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod 3], \ldots\Big)$$

Residue number system approach

- Represent each bit redundantly mod 2, mod 3, mod 5, ...
- Compute summations mod 2, 3, 5, 7 ⇒ cost = **free**
- Compute equality tests mod 2, 3, 5, 7 ⇒ cost = $2 + 3 + 5 + 7$

High fan-in computations

Insight: take advantage of **multiple moduli**

$$\text{AND}(x_1, \ldots, x_{100}) = [\textstyle\sum x_i \stackrel{?}{=} 100] = [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod{210}]$$
$$= \text{AND}\Big([\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod{2}], [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod{3}], \ldots\Big)$$

Residue number system approach

- Represent each bit redundantly mod 2, mod 3, mod 5, . . .
- Compute summations mod 2, 3, 5, 7 ⇒ cost = **free**
- Compute equality tests mod 2, 3, 5, 7 ⇒ cost = $2 + 3 + 5 + 7$
- Compute AND of 4 equality test results ⇒ cost = 4 (via prev method)

High fan-in computations

Insight: take advantage of **multiple moduli**

$$\mathrm{AND}(x_1,\ldots,x_{100}) = [\textstyle\sum x_i \stackrel{?}{=} 100] = [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod{210}]$$

$$= \mathrm{AND}\Big([\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod 2], [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod 3], \ldots\Big)$$

Residue number system approach

- Represent each bit redundantly mod 2, mod 3, mod 5, . . .
- Compute summations mod 2, 3, 5, 7 $\Rightarrow$ cost = **free**
- Compute equality tests mod 2, 3, 5, 7 $\Rightarrow$ cost = $2 + 3 + 5 + 7$
- Compute AND of 4 equality test results $\Rightarrow$ cost = 4 (via prev method)

	standard	better	**CRT**
AND/OR	198	100	**21**
majority	948	100	

High fan-in computations

Insight: take advantage of **multiple moduli**

$$\mathrm{AND}(x_1,\ldots,x_{100}) = [\textstyle\sum x_i \stackrel{?}{=} 100] = [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod{210}]$$
$$= \mathrm{AND}\Big([\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod 2], [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod 3], \ldots\Big)$$

Residue number system approach

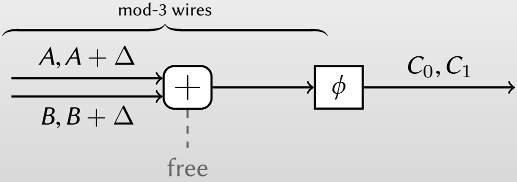
- Represent each bit redundantly mod 2, mod 3, mod 5, ...
- Compute summations mod 2, 3, 5, 7 ⇒ cost = **free**
- Compute equality tests mod 2, 3, 5, 7 ⇒ cost = $2+3+5+7$
- Compute AND of 4 equality test results ⇒ cost = 4 (via prev method)

	standard	better	**CRT**
AND/OR	198	100	**21**
majority	948	100	137

Circumventing lower bound

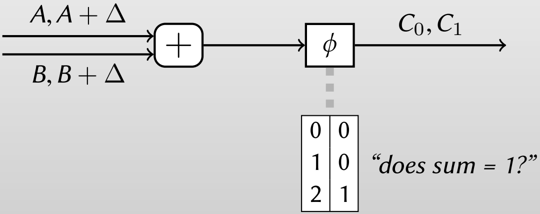
Circumventing lower bound

What about case of fan-in 2 AND gate?



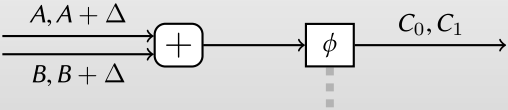
Circumventing lower bound

What about case of fan-in 2 AND gate?



Circumventing lower bound

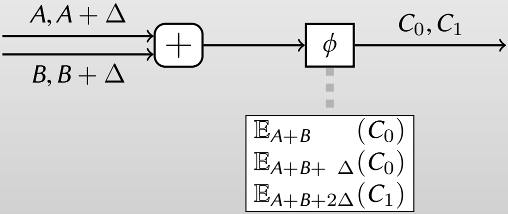
What about case of fan-in 2 AND gate?



$A + B$	C_0
$A + B + \Delta$	C_0
$A + B + 2\Delta$	C_1

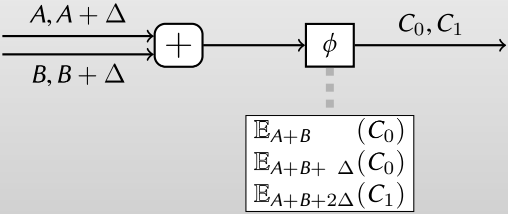
Circumventing lower bound

What about case of fan-in 2 AND gate?



Circumventing lower bound

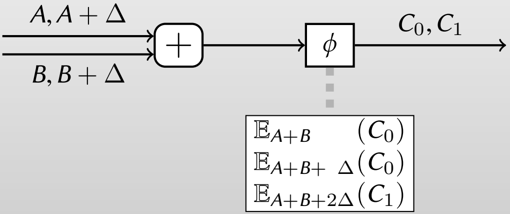
What about case of fan-in 2 AND gate?



- **Row-reduction:** choose C_0 to make 1st ciphertext zero
  - $\Rightarrow$ doesn't need to be sent

Circumventing lower bound

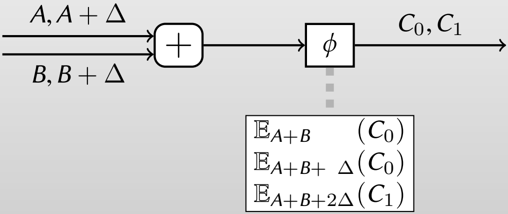
What about case of fan-in 2 AND gate?



- **Row-reduction:** choose C_0 to make 1st ciphertext zero
  - $\Rightarrow$ doesn't need to be sent
- Stopping there allows **composability**: $C_1 = C_0 + \Delta$
- Instead, further choose C_1 so that remaining 2 ciphertexts are equal
  - $\Rightarrow$ don't need to send both [PinkasSchneiderSmartWilliams09,GueronLindellNofPinkas15]

Circumventing lower bound

What about case of fan-in 2 AND gate? Can garble with 1 ciphertext!



- **Row-reduction:** choose C_0 to make 1st ciphertext zero
  - $\Rightarrow$ doesn't need to be sent
- Stopping there allows **composability**: $C_1 = C_0 + \Delta$
- Instead, further choose C_1 so that remaining 2 ciphertexts are equal
  - $\Rightarrow$ don't need to send both
    [PinkasSchneiderSmartWilliams09,GueronLindellNofPinkas15]

Challenges:

State of the art:

"If values are represented in CRT form then garbled operations are cheap."

Challenges:

State of the art:

"If values are represented in CRT form then garbled operations are cheap."

But doesn't it cost something to get values into CRT form??

Dealing with CRT

Claim:

It's **not hard** to convert into CRT representation $\mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2} \times \cdots \times \mathbb{Z}_{p_k}$

Dealing with CRT

Claim:

It's **not hard** to convert into CRT representation $\mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2} \times \cdots \times \mathbb{Z}_{p_k}$

From binary $b_n b_{n-1} \cdots b_1 b_0$:

- For all i, j, use unary gate $b_i \mapsto b_i \pmod{p_j}$ (1 ciphertext each)
- For all j, add to obtain $\sum_i b_i 2^i \pmod{p_j}$ (**free**)
- Total cost = (# primes) × (# bits) (e.g., 320 ciphertexts for 32 bits)

Dealing with CRT

Claim:

It's **not hard** to convert into CRT representation $\mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2} \times \cdots \times \mathbb{Z}_{p_k}$

From binary $b_n b_{n-1} \cdots b_1 b_0$:

- For all i, j, use unary gate $b_i \mapsto b_i \pmod{p_j}$ (1 ciphertext each)
- For all j, add to obtain $\sum_i b_i 2^i \pmod{p_j}$ (**free**)
- Total cost = (# primes) × (# bits) (e.g., 320 ciphertexts for 32 bits)

At the input level (e.g., OTs in Yao): (similar to [Gilboa99,KellerOrsiniScholl16])

- Outside of the circuit, convert plaintext input into CRT form
- Convert $\mathbb{Z}_{p_j}$-residue to binary, and transfer it using $\lceil \log p_j \rceil$ OTs
- Total cost: $\sum_j \log p_j$ OTs (e.g., 37 OTs for 32-bit values)

Open Problems

Improve the cost of any of these:

- **Comparing** two CRT-encoded values
- Converting CRT representation to binary
- Integer division
- Modular reduction different than the CRT composite modulus (e.g., garbled RSA)

Open Problems

Improve the cost of any of these:

- **Comparing** two CRT-encoded values
- Converting CRT representation to binary
- Integer division
- Modular reduction different than the CRT composite modulus (e.g., garbled RSA)

Comparing CRT values

CRT view of $\mathbb{Z}_{2\cdot 3\cdot 5\cdot 7}$:

0 0 0 0	0
1 1 1 1	1
2 2 2 0	2
3 3 0 1	3
4 4 1 0	4
5 0 2 1	5
6 1 0 0	6
0 2 1 1	7
$\vdots$	$\vdots$
1 4 2 1	29
2 0 0 0	30
$\vdots$	$\vdots$

Comparing CRT values

CRT view of $\mathbb{Z}_{2\cdot 3\cdot 5\cdot 7}$:

0 0 0 0	0
1 1 1 1	1
2 2 2 0	2
3 3 0 1	3
4 4 1 0	4
5 0 2 1	5
6 1 0 0	6
0 2 1 1	7
⋮	⋮
1 4 2 1	29
2 0 0 0	30
⋮	⋮

Theorem

CRT representation sucks for comparisons!

Comparing CRT values

CRT view of $\mathbb{Z}_{2\cdot 3\cdot 5\cdot 7}$:

$$\begin{array}{c|l} 0\ 0\ 0\ 0 & 0 \\ 1\ 1\ 1\ 1 & 1 \\ 2\ 2\ 2\ 0 & 2 \\ 3\ 3\ 0\ 1 & 3 \\ 4\ 4\ 1\ 0 & 4 \\ 5\ 0\ 2\ 1 & 5 \\ 6\ 1\ 0\ 0 & 6 \\ 0\ 2\ 1\ 1 & 7 \\ \vdots & \vdots \\ 1\ 4\ 2\ 1 & 29 \\ 2\ 0\ 0\ 0 & 30 \\ \vdots & \vdots \end{array}$$

$$\begin{array}{r|l} 0 & 0 \\ 1 & 1 \\ 1\ 0 & 2 \\ 1\ 1 & 3 \\ 2\ 0 & 4 \\ 2\ 1 & 5 \\ 1\ 0\ 0 & 6 \\ 1\ 0\ 1 & 7 \\ \vdots & \vdots \\ 4\ 2\ 1 & 29 \\ 1\ 0\ 0\ 0 & 30 \\ \vdots & \vdots \end{array}$$

Comparing CRT values

CRT view of $\mathbb{Z}_{2\cdot 3\cdot 5\cdot 7}$:

0 0 0 0	0
1 1 1 1	1
2 2 2 0	2
3 3 0 1	3
4 4 1 0	4
5 0 2 1	5
6 1 0 0	6
0 2 1 1	7
⋮	⋮
1 4 2 1	29
2 0 0 0	30
⋮	⋮

Primorial Mixed Radix (PMR)

0	0
1	1
1 0	2
1 1	3
2 0	4
2 1	5
1 0 0	6
1 0 1	7
⋮	⋮
4 2 1	29
1 0 0 0	30
⋮	⋮

Approach for comparisons

CRT values given

↓

Convert both CRT values to PMR

↓

Compare PMR (simple L→R scan)

Approach for comparisons

CRT values given

$\downarrow$

Convert both CRT values to PMR

PMR representation of x:

$$\ldots, \quad \left\lfloor \frac{x}{2 \cdot 3 \cdot 5} \right\rfloor \% \, 7, \quad \left\lfloor \frac{x}{2 \cdot 3} \right\rfloor \% \, 5, \quad \left\lfloor \frac{x}{2} \right\rfloor \% \, 3, \quad \lfloor x \rfloor \% \, 2$$

$\downarrow$

Compare PMR (simple L→R scan)

Approach for comparisons

CRT values given

$\downarrow$

Convert both CRT values to PMR

Simple building block:

$$(x\%p,\ \ x\%q) \mapsto \left\lfloor \frac{x}{p} \right\rfloor \%q$$

allows you to compute PMR representation of x:

$$\ldots,\ \ \left\lfloor \frac{x}{2\cdot 3\cdot 5} \right\rfloor \%\, 7,\ \ \left\lfloor \frac{x}{2\cdot 3} \right\rfloor \%\, 5,\ \ \left\lfloor \frac{x}{2} \right\rfloor \%\, 3,\ \ \lfloor x \rfloor \%\, 2$$

$\downarrow$

Compare PMR (simple L→R scan)

$(x\%p,\ x\%q) \mapsto \lfloor x/p \rfloor \% q$

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$x \% 3$	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
$x \% 5$	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4

$\lfloor x/3 \rfloor \% 5$	0	0	0	1	1	1	2	2	2	3	3	3	4	4	4

$(x\%p,\ x\%q) \mapsto \lfloor x/p \rfloor \%q$

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$x \% 3$	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
$x \% 5$	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
$x\%3 - x\%5$	0	0	0	-3	-3	2	-1	-1	-1	-4	1	1	-2	-2	-2

$\lfloor x/3 \rfloor \% 5$	0	0	0	1	1	1	2	2	2	3	3	3	4	4	4

1. Subtract $x\%3 - x\%5$

$(x\%p,\ x\%q) \mapsto \lfloor x/p \rfloor \% q$

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$x \% 3$	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
$x \% 5$	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
$x\%3 - x\%5$	0	0	0	-3	-3	2	-1	-1	-1	-4	1	1	-2	-2	-2

$\lfloor x/3 \rfloor \% 5$	0	0	0	1	1	1	2	2	2	3	3	3	4	4	4

1. Subtract $x\%3 - x\%5$

2. Result has the same “constant segments” as what we want

$(x\%p,\ x\%q) \mapsto \lfloor x/p \rfloor \%q$

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$x \% 3$	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
$x \% 5$	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
$x\%3 - x\%5$	0	0	0	-3	-3	2	-1	-1	-1	-4	1	1	-2	-2	-2
$(x\%3 - x\%5)\%7$	0	0	0	4	4	2	6	6	6	3	1	1	5	5	5
$\lfloor x/3 \rfloor \% 5$	0	0	0	1	1	1	2	2	2	3	3	3	4	4	4

1. Subtract $x\%3 - x\%5$ (mod 7 is fine)

2. Result has the same "constant segments" as what we want

$(x\%p,\ x\%q) \mapsto \lfloor x/p \rfloor \%q$

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$x \% 3$	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
$x \% 5$	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
$x\%3 - x\%5$	0	0	0	-3	-3	2	-1	-1	-1	-4	1	1	-2	-2	-2
$(x\%3 - x\%5)\%7$	0	0	0	4	4	2	6	6	6	3	1	1	5	5	5
$\lfloor x/3 \rfloor \% 5$	0	0	0	1	1	1	2	2	2	3	3	3	4	4	4

1. Subtract $x\%3 - x\%5$ (mod 7 is fine)
    - "Project" $x\%3$ and $x\%5$ to $\mathbb{Z}_7$ wires
    - Subtract mod 7 for free
2. Result has the same "constant segments" as what we want

$(x\%p,\ x\%q) \mapsto \lfloor x/p \rfloor \%q$

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$x \% 3$	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
$x \% 5$	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
$x\%3 - x\%5$	0	0	0	-3	-3	2	-1	-1	-1	-4	1	1	-2	-2	-2
$(x\%3 - x\%5)\%7$	0	0	0	4	4	2	6	6	6	3	1	1	5	5	5
$\lfloor x/3 \rfloor \% 5$	0	0	0	1	1	1	2	2	2	3	3	3	4	4	4

1. Subtract $x\%3 - x\%5$ (mod 7 is fine)
   - "Project" $x\%3$ and $x\%5$ to $\mathbb{Z}_7$ wires
   - Subtract mod 7 for free
2. Result has the same "constant segments" as what we want
   - Apply unary projection:

$$0 \mapsto 0 \quad 2 \mapsto 1 \quad 4 \mapsto 1 \quad 6 \mapsto 2$$
$$1 \mapsto 3 \quad 3 \mapsto 3 \quad 5 \mapsto 4$$

Approach for comparisons

1. General $\boxed{(x\%p,\ x\%q) \mapsto \lfloor x/p \rfloor \%q}$ gadget costs $\sim 2p + 2q$ ciphertexts
2. PMR conversion requires this gadget between *all pairs* of primes
3. Total cost $O(k^3)$ for k-bit integers

Approach for comparisons

1. General $\boxed{(x\%p,\ x\%q) \mapsto \lfloor x/p \rfloor \%q}$ gadget costs $\sim 2p + 2q$ ciphertexts
2. PMR conversion requires this gadget between *all pairs* of primes
3. Total cost $O(k^3)$ for k-bit integers

Operations on 32-bit integers:

	boolean	CRT
addition	62	0
multiplication by public constant	758	0
multiplication	1200	238
squaring, cubing, etc	1864	119

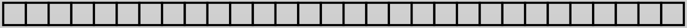
Approach for comparisons

1. General $\boxed{(x\%p,\ x\%q) \mapsto \lfloor x/p \rfloor \ \%q}$ gadget costs $\sim 2p + 2q$ ciphertexts
2. PMR conversion requires this gadget between *all pairs* of primes
3. Total cost $O(k^3)$ for k-bit integers

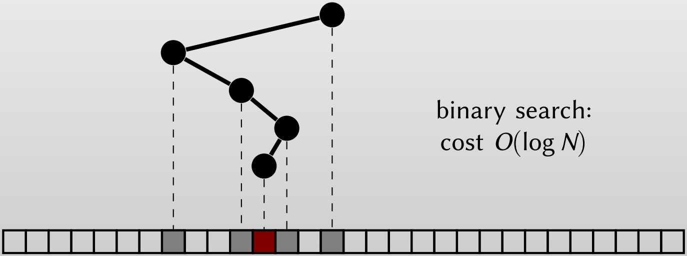
Operations on 32-bit integers:

	boolean	CRT
addition	62	0
multiplication by public constant	758	0
multiplication	1200	238
squaring, cubing, etc	1864	119
comparison	**64**	**2541**

Beyond circuits altogether

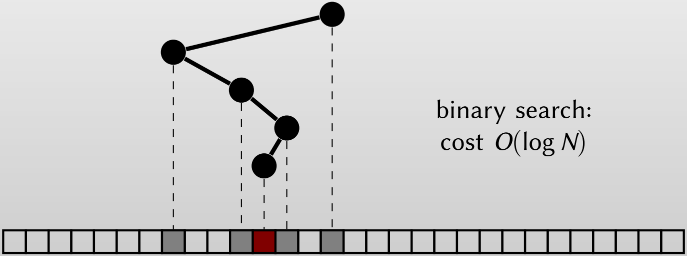


Beyond circuits altogether



binary search:
cost $O(\log N)$

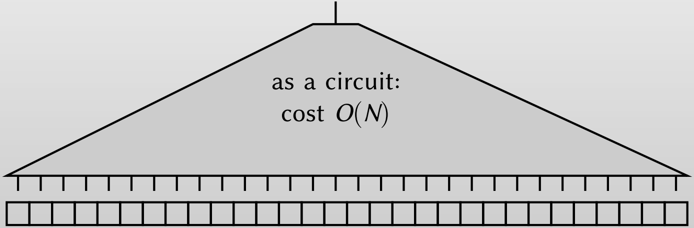
Beyond circuits altogether



binary search:
cost $O(\log N)$

"first express f as a boolean circuit . . ."

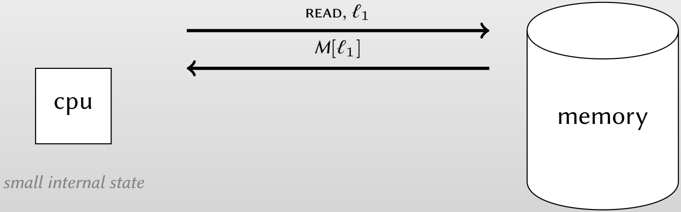
Beyond circuits altogether



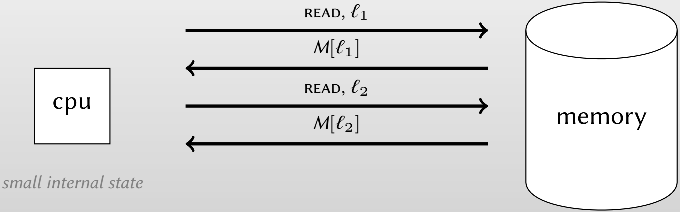
“first express f as a boolean circuit . . .”

RAM programs

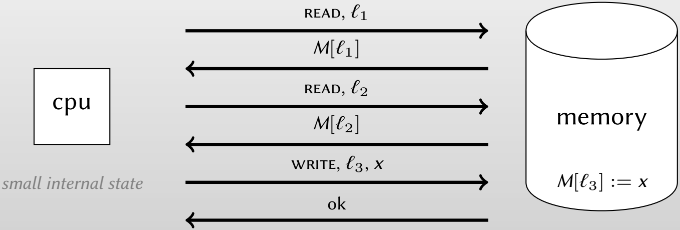
RAM programs



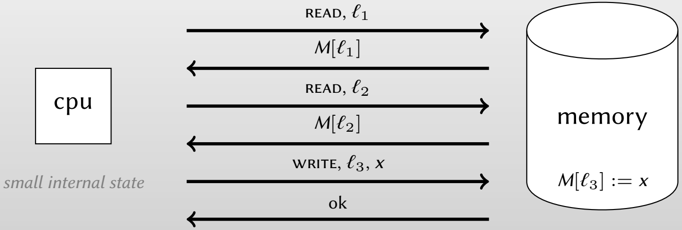
RAM programs



RAM programs

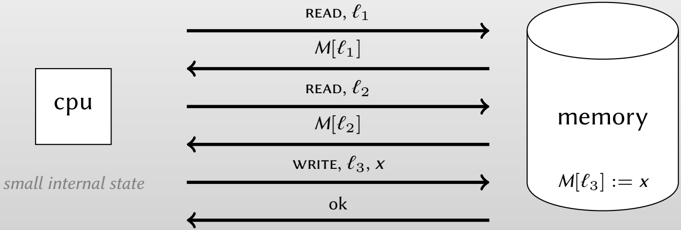


RAM programs



How to garble a RAM computation?

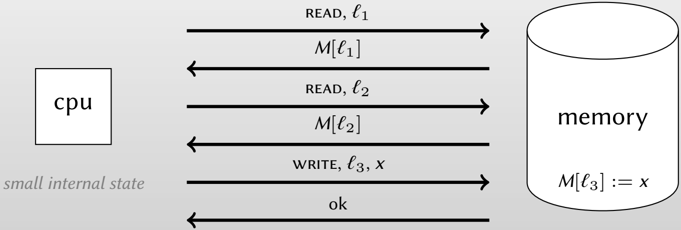
RAM programs



How to garble a RAM computation with cost

$\sim |mem| + |cpu| \cdot$ [# mem accesses] ?

RAM programs



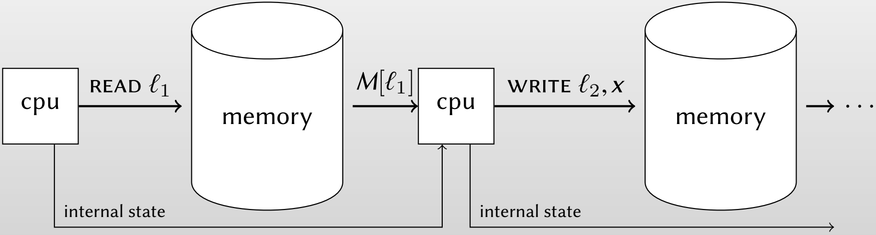
How to garble a RAM computation with cost

$\sim |mem| + |cpu| \cdot$ [# mem accesses] ?

Example: t binary searches

- Naïve cost: $\sim N \cdot t$
- Desired cost: $\sim N + t \cdot \log N$

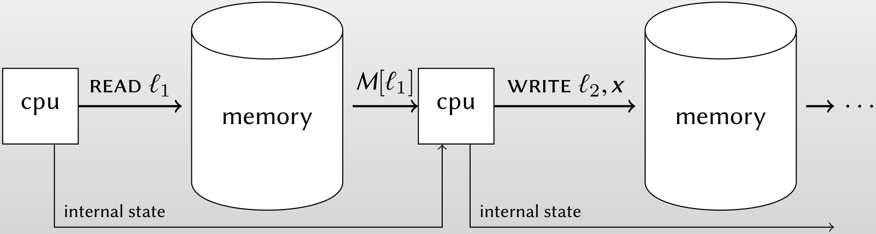
Challenges for Garbled RAM



Garbled circuits are single-use!

$\Rightarrow$ garble *t* separate copies of CPU circuit

Challenges for Garbled RAM



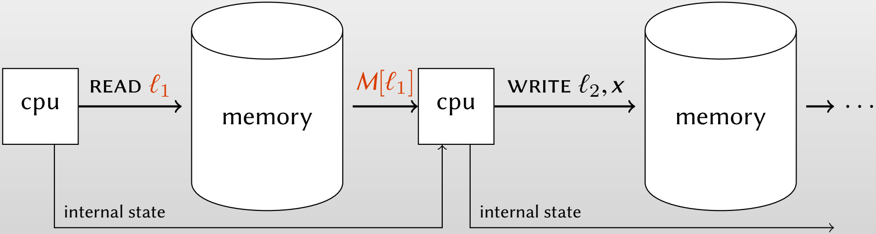
Garbled circuits are single-use!

⇒ garble t separate copies of CPU circuit

Memory accesses can depend on private data!

⇒ Compile to **oblivious RAM** (ORAM) [GoldreichOstrovsky96]

Challenges for Garbled RAM



Garbled circuits are single-use!

⇒ garble t separate copies of CPU circuit

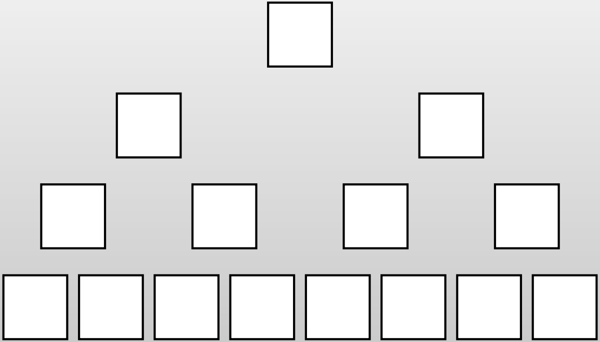
Memory accesses can depend on private data!

⇒ Compile to **oblivious RAM** (ORAM) [GoldreichOstrovsky96]

Memory accesses determined **only at runtime!**

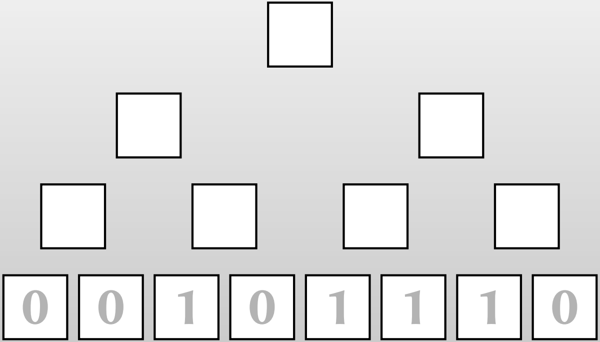
?? How do we get **wire labels** encoding $M[\ell_1]$ when ℓ_1 determined at run-time?

Garbled Memory [GargLuOstrovsky15]



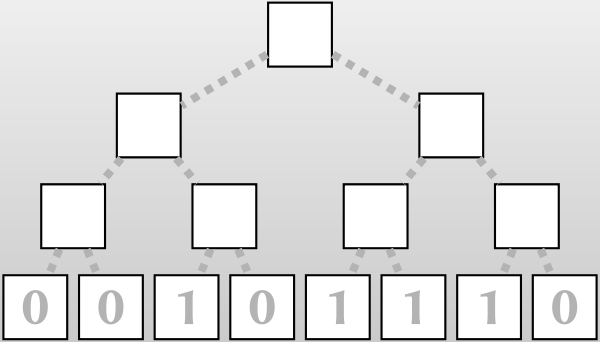
- Binary tree of little **garbled circuits**

Garbled Memory [GargLuOstrovsky15]



- Binary tree of little **garbled circuits**
- Leaf circuits have 1 bit of RAM memory hard-coded

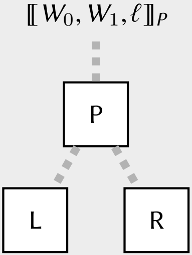
Garbled Memory [GargLuOstrovsky15]



- Binary tree of little **garbled circuits**
- Leaf circuits have 1 bit of RAM memory hard-coded
- Internal circuits have **input wire labels** of children hard-coded (can pass secret information to a child)

Garbled Memory [GargLuOstrovsky15]

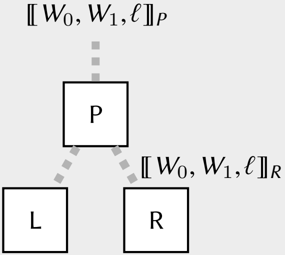
Internal node circuits:



Given (garbled input) W_0, W_1, ℓ

Garbled Memory [GargLuOstrovsky15]

Internal node circuits:

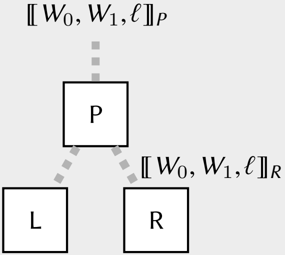


Given (garbled input) W_0, W_1, ℓ

Translate to child's garbled encoding, based on appropriate bit of ℓ

Garbled Memory [GargLuOstrovsky15]

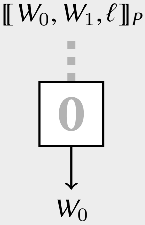
Internal node circuits:



Given (garbled input) W_0, W_1, ℓ

Translate to child's garbled encoding, based on appropriate bit of ℓ

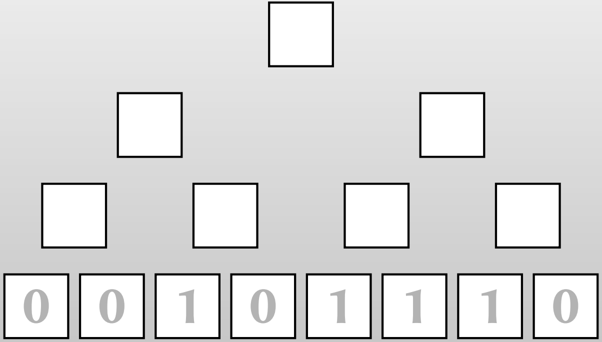
Leaf node circuits:



Given (garbled input) W_0, W_1, ℓ

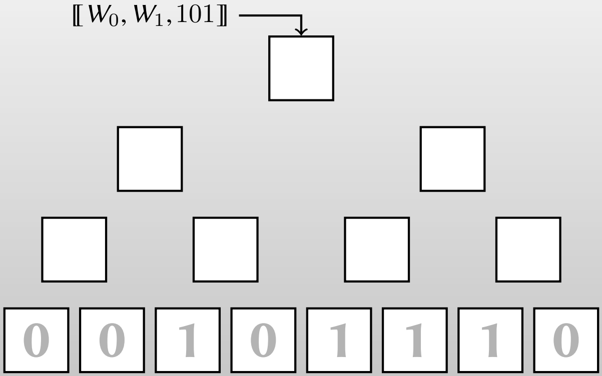
Output W_0 or W_1 **in the clear**, based on hard-coded bit

Garbled Memory [GargLuOstrovsky15]



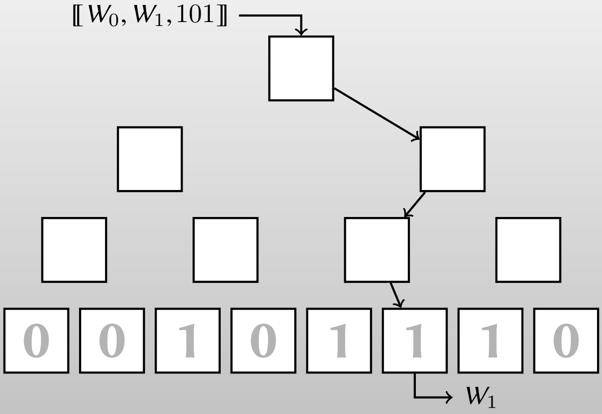
- This collection of circuits hard-codes $M[1 \ldots N]$

Garbled Memory [GargLuOstrovsky15]



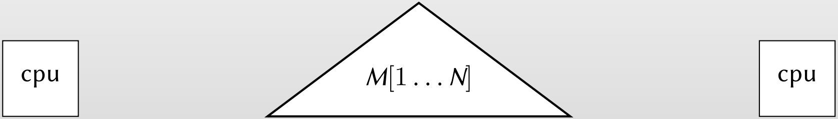
- This collection of circuits hard-codes $M[1 \ldots N]$
- Give it (garbled) $[\![W_0, W_1, \ell]\!]$

Garbled Memory [GargLuOstrovsky15]



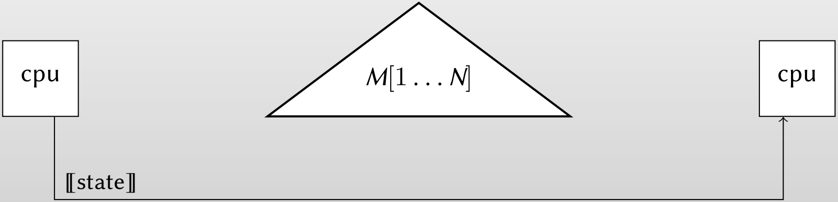
- This collection of circuits hard-codes $M[1 \ldots N]$
- Give it (garbled) $[\![W_0, W_1, \ell]\!] \Rightarrow$ it will give you $W_{M[\ell]}$ (in the clear)
- *(evaluator must know ℓ to evaluate correct subcircuits)*

Garbled RAM main idea



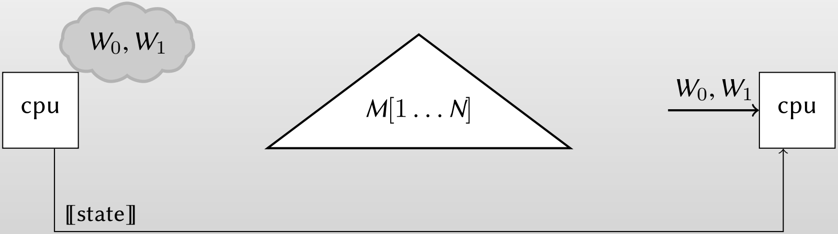
- Garble two copies of the RAM CPU circuit

Garbled RAM main idea



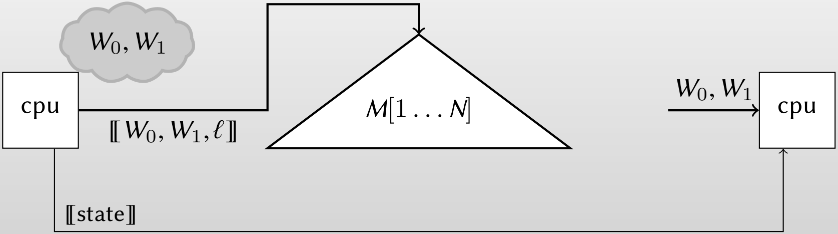
- Garble two copies of the RAM CPU circuit
- Use same wire labels for outgoing state / incoming state ("same wire")

Garbled RAM main idea



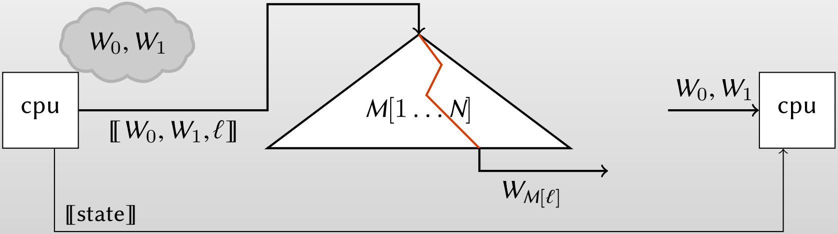
- Garble two copies of the RAM CPU circuit
- Use same wire labels for outgoing state / incoming state ("same wire")
- First circuit has input labels of second circuit hard-coded

Garbled RAM main idea



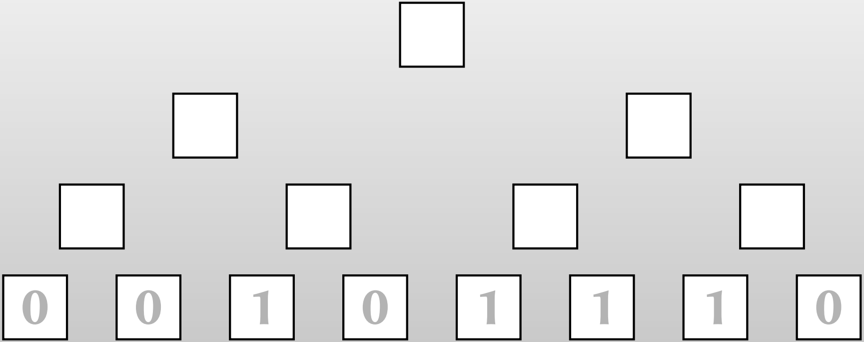
- Garble two copies of the RAM CPU circuit
- Use same wire labels for outgoing state / incoming state ("same wire")
- First circuit has input labels of second circuit hard-coded
- On $(\textsc{read}, \ell)$ operation, first circuit:
  - outputs ℓ in clear
  - passes **garbled** W_0, W_1, ℓ to garbled memory

Garbled RAM main idea

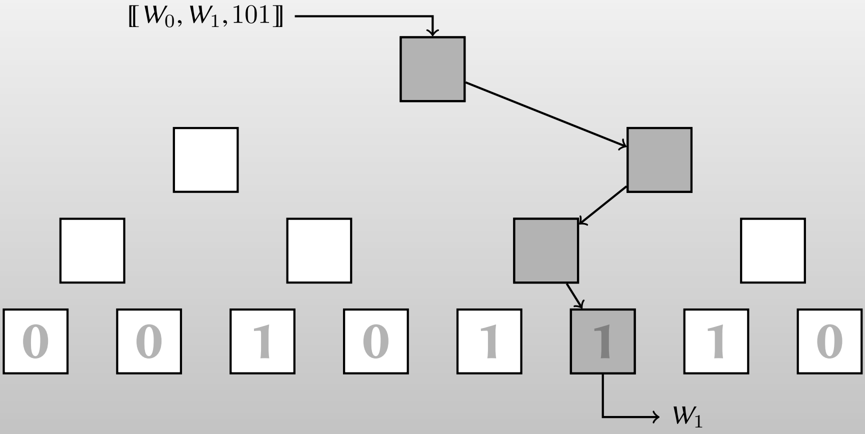


- Garble two copies of the RAM CPU circuit
- Use same wire labels for outgoing state / incoming state ("same wire")
- First circuit has input labels of second circuit hard-coded
- On $(\textsc{read}, \ell)$ operation, first circuit:
  - outputs ℓ in clear
  - passes **garbled** W_0, W_1, ℓ to garbled memory
- Garbled memory outputs correct wire label $W_{M[\ell]}$ in the clear

Garbled Memory Fine Print

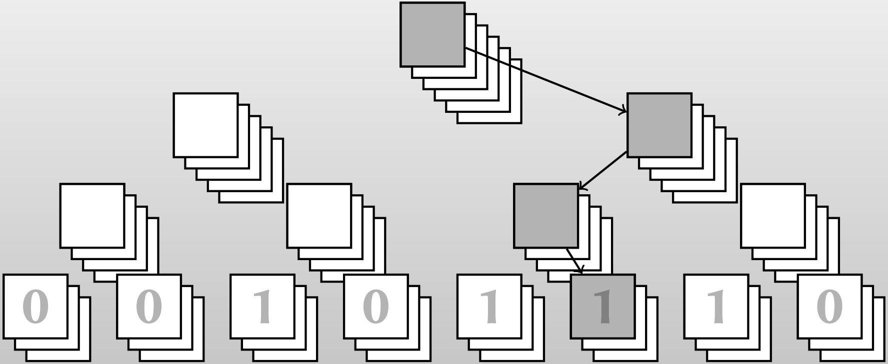


Garbled Memory Fine Print



- Each little garbled circuit is **single-use!**

Garbled Memory Fine Print



- Each little garbled circuit is **single-use!**
- Solution: each node contains **queue** of circuits
  - Chernoff bound determines # of circuits in each queue
  - Circuits pass hard-coded information to successor in queue

Summary / Open Problems

Garbling beyond boolean circuits

New techniques for garbled **arithmetic circuits**:

- Comparing two CRT-encoded values?
- Converting CRT representation to binary?
- Integer division?
- Modular arithmetic (different than CRT primorial modulus)?

New techniques for garbled **RAM programs**:

- How do these constructions perform in real world?
- How to reduce “slack” introduced by Chernoff bound?

the end.

any questions?