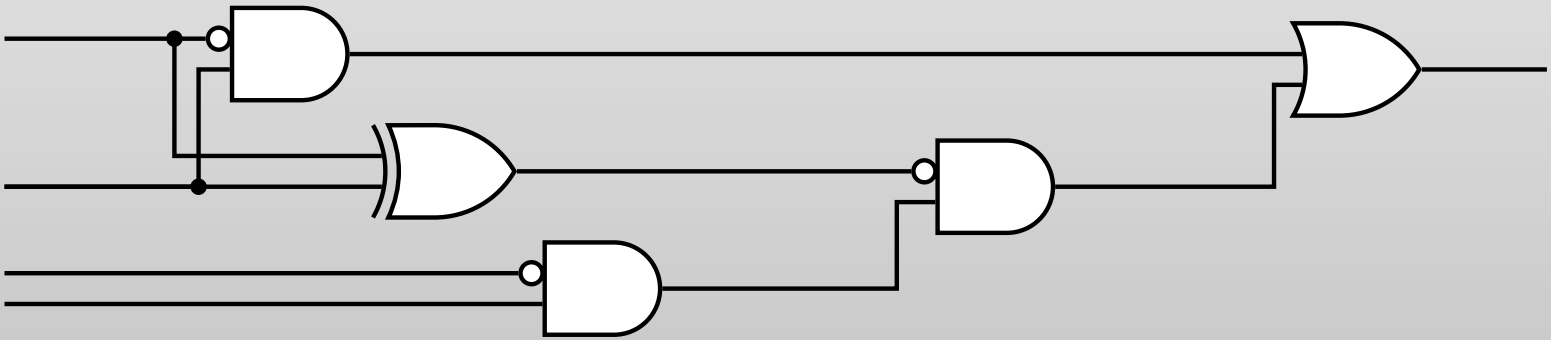
Garbled Circuits: New Results for Arithmetic and High Fan-In Computations

Mike Rosulek

Oregon State University OSU

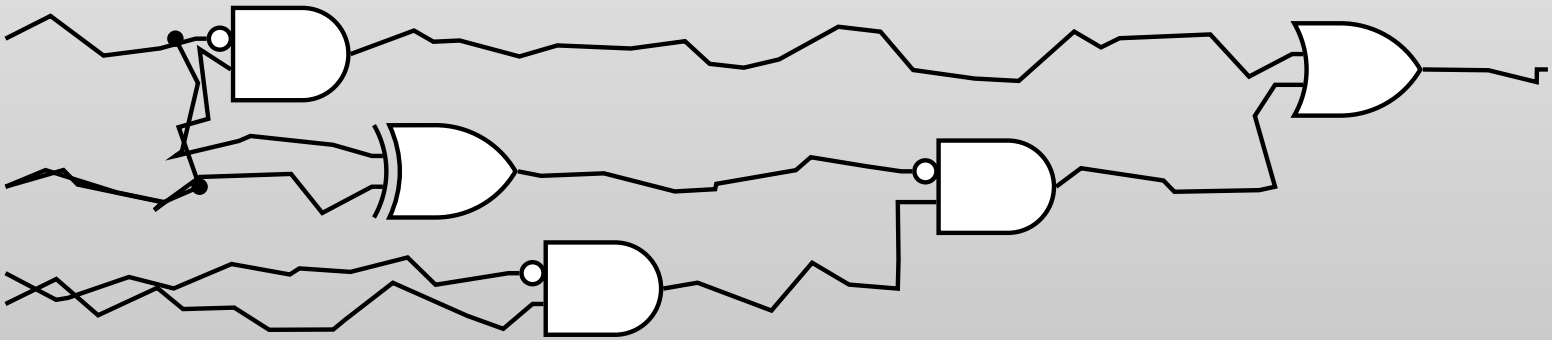


Joint work with Marshall Ball & Tal Malkin

Garbled Circuits: New Results for Arithmetic and High Fan-In Computations

Mike Rosulek

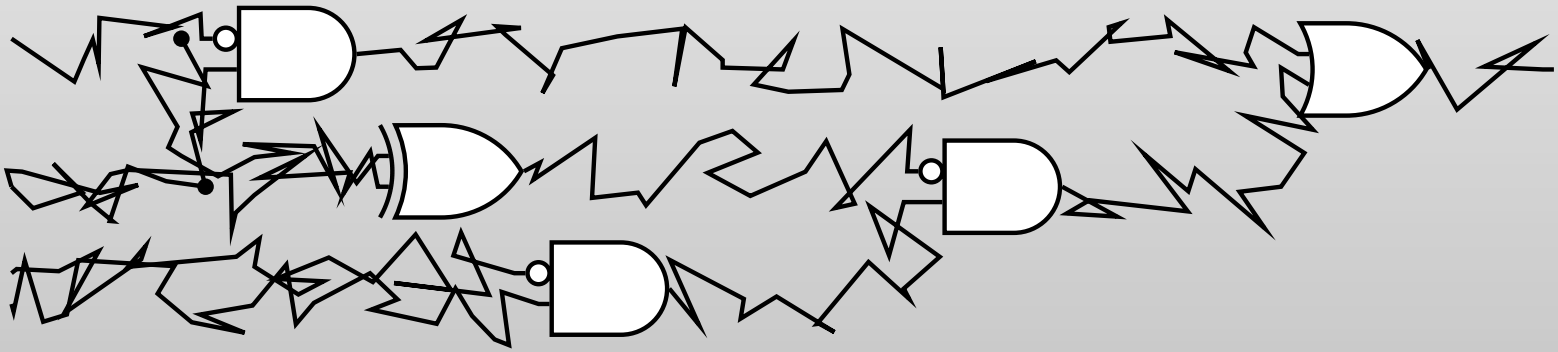
Oregon State University OSU



Joint work with Marshall Ball & Tal Malkin

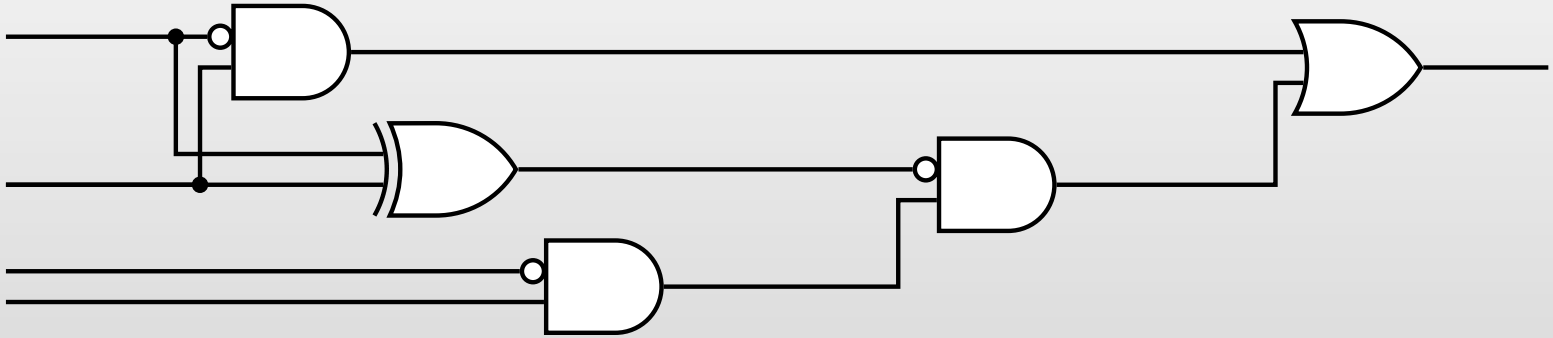
Garbled Circuits: New Results for Arithmetic and High Fan-In Computations

Mike Rosulek
Oregon State University OSU

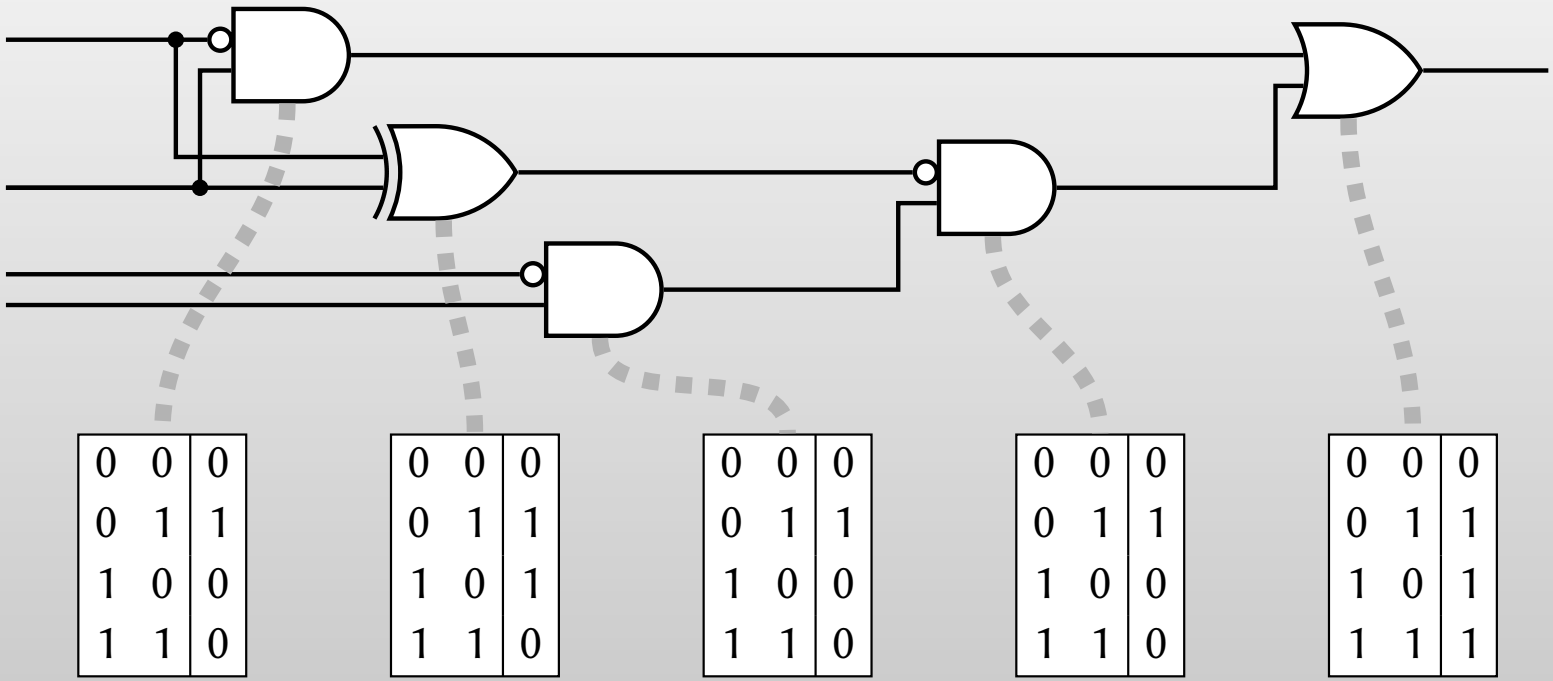


Joint work with Marshall Ball & Tal Malkin

Garbled circuit framework [Yao86]



Garbled circuit framework [Yao86]



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

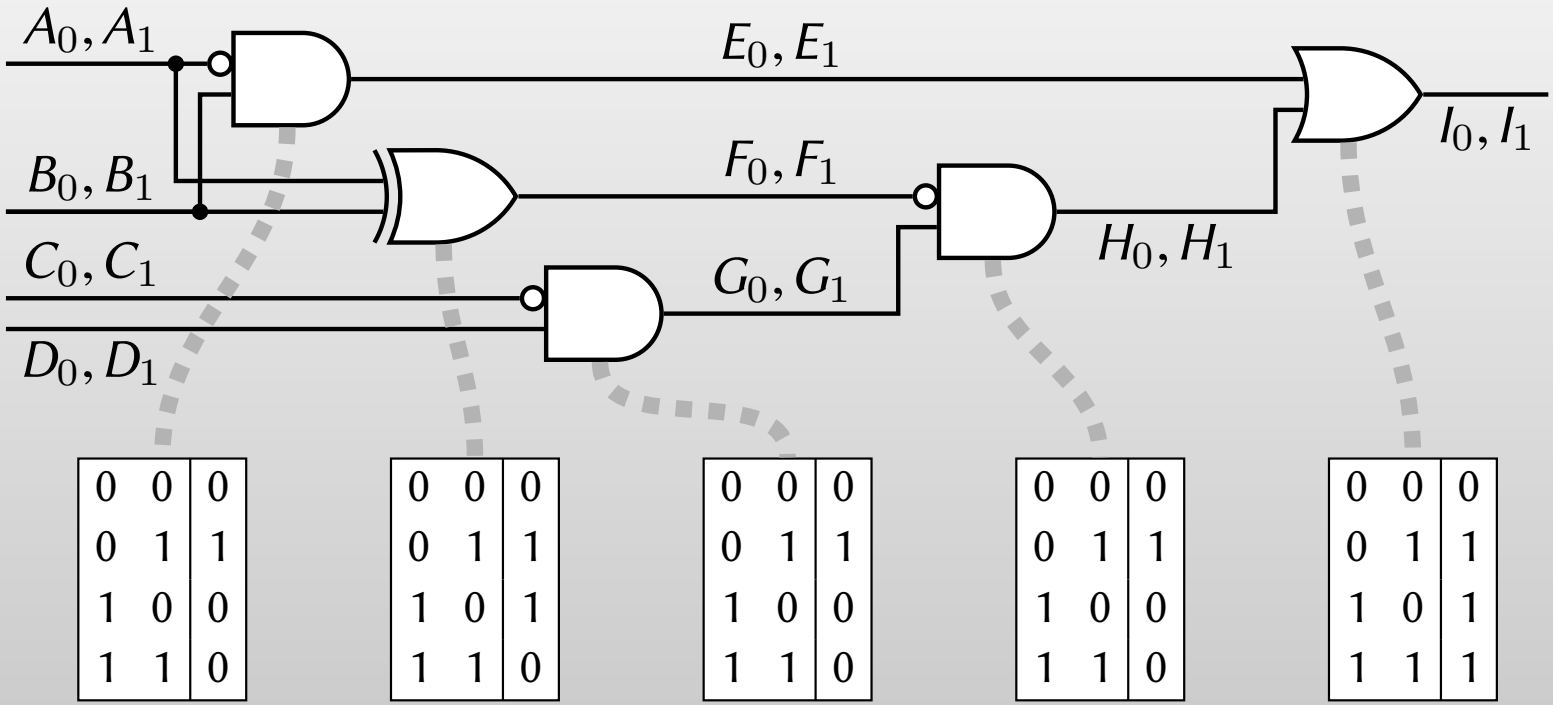
0	0	0
0	1	1
1	0	1
1	1	0

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

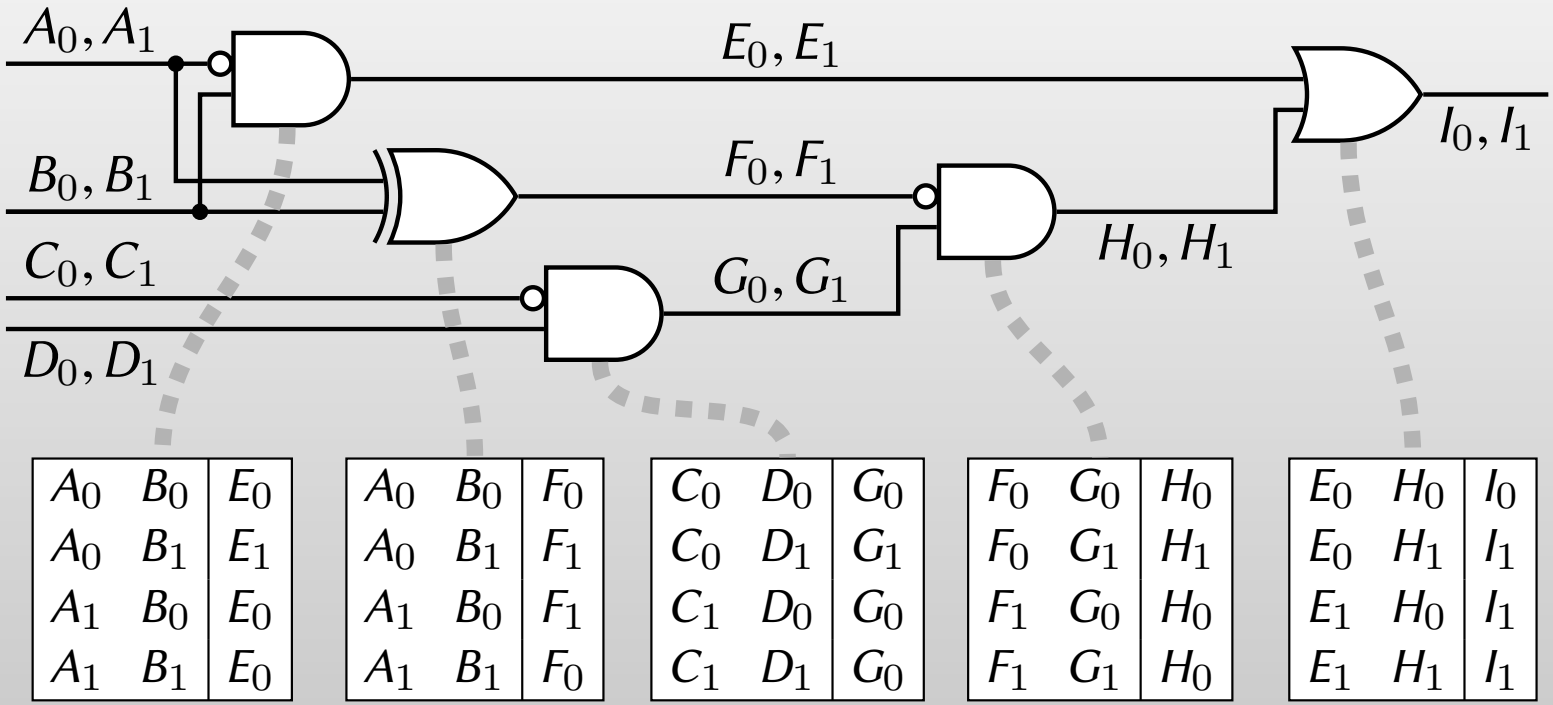
Garbled circuit framework [Yao86]



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

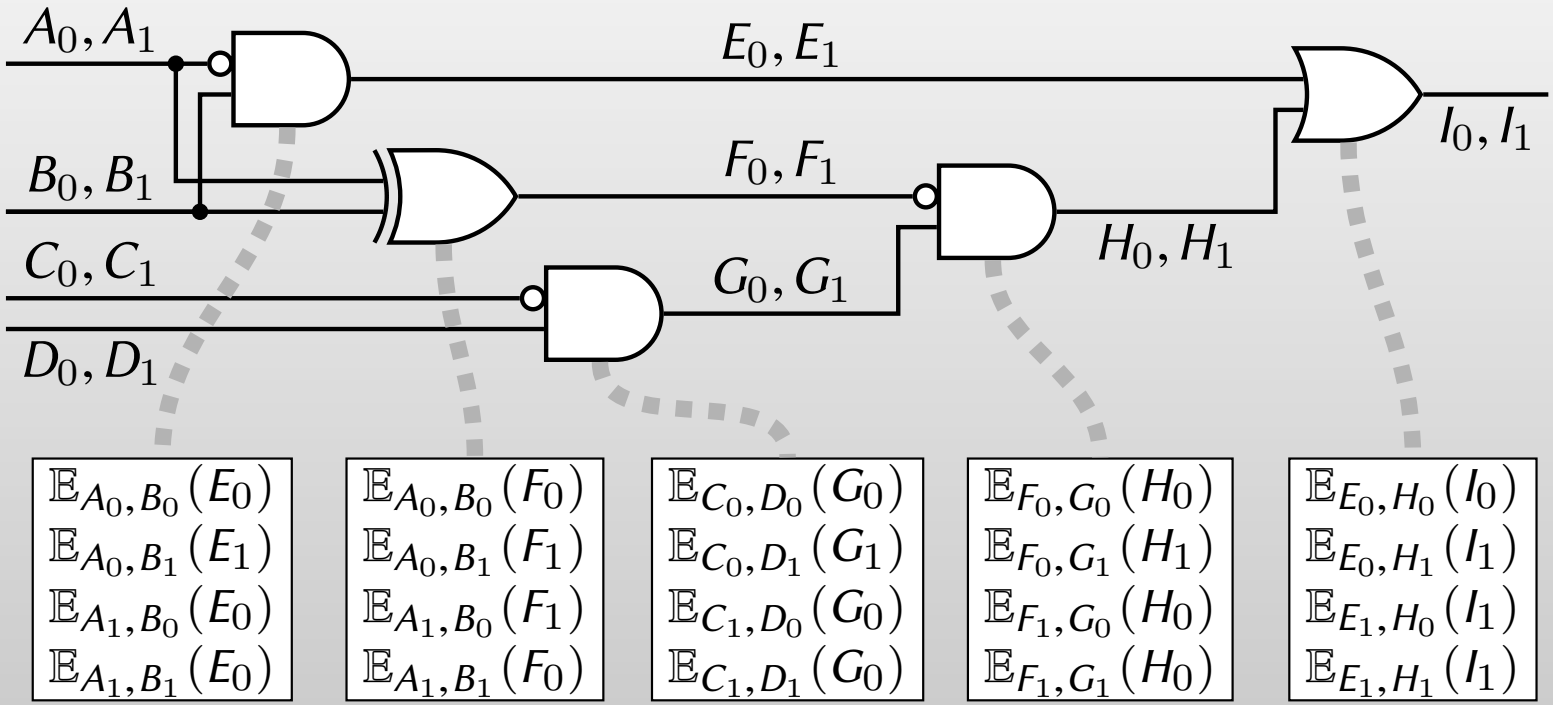
F_0	G_0	H_0
F_0	G_1	H_1
F_1	G_0	H_0
F_1	G_1	H_0

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

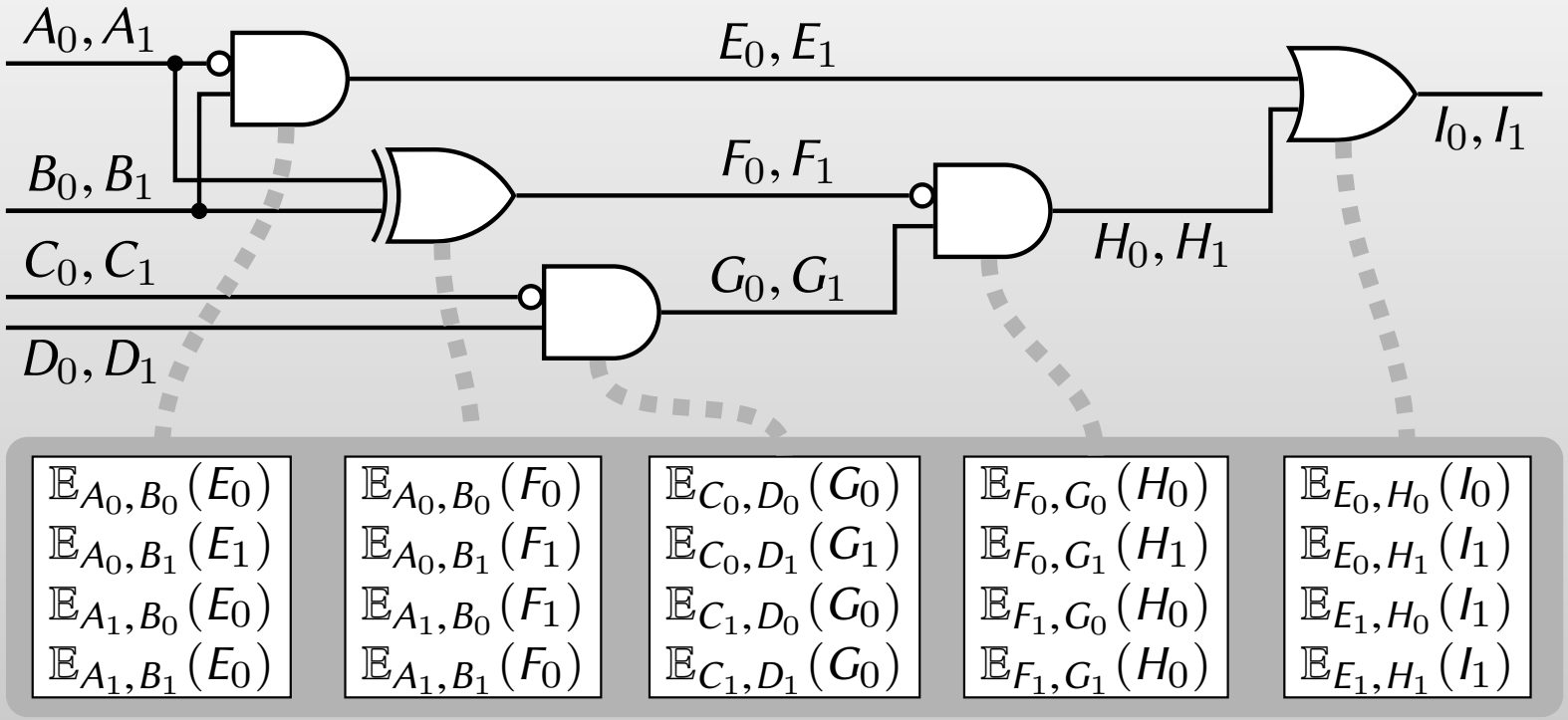
Garbled circuit framework [Yao86]



Garbling a circuit:

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

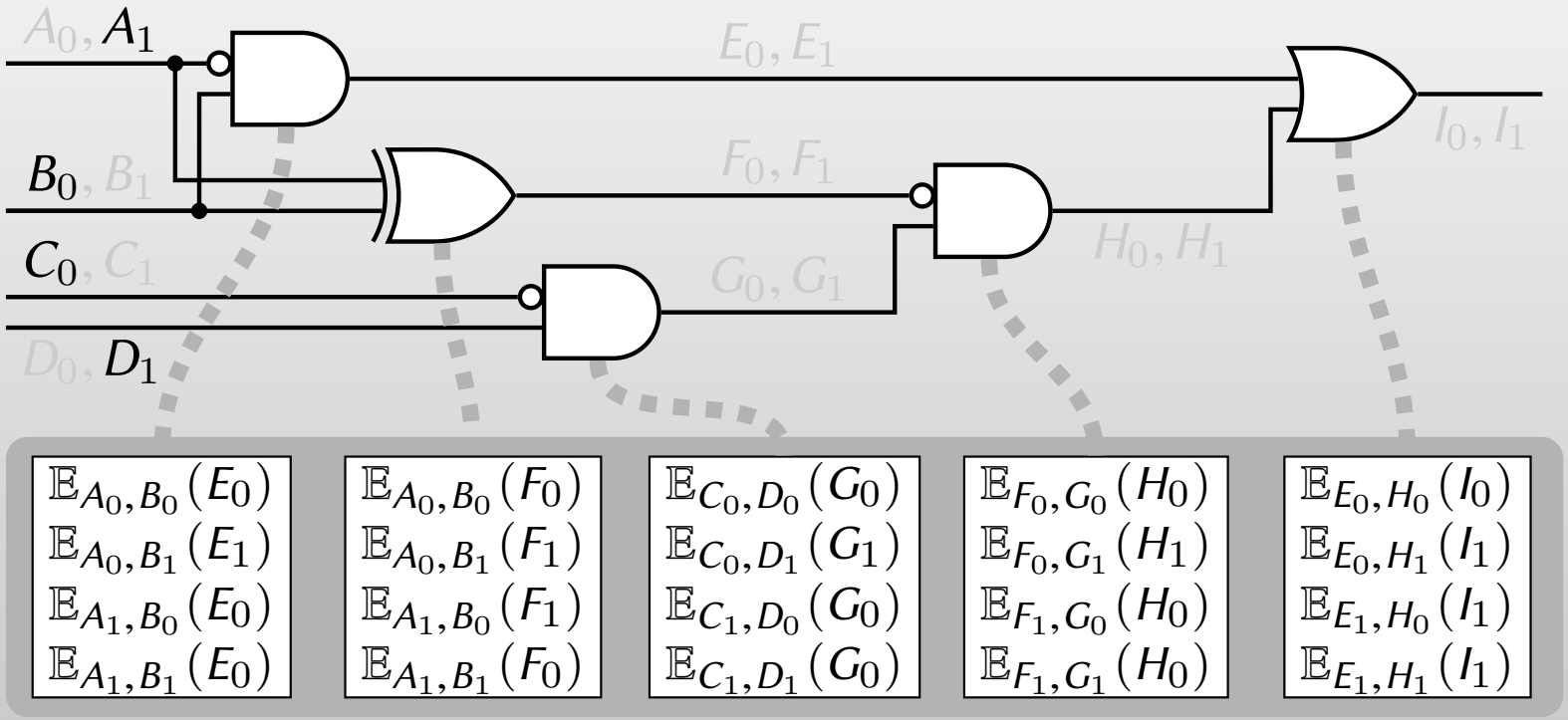
Garbled circuit framework [Yao86]



Garbling a circuit:

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

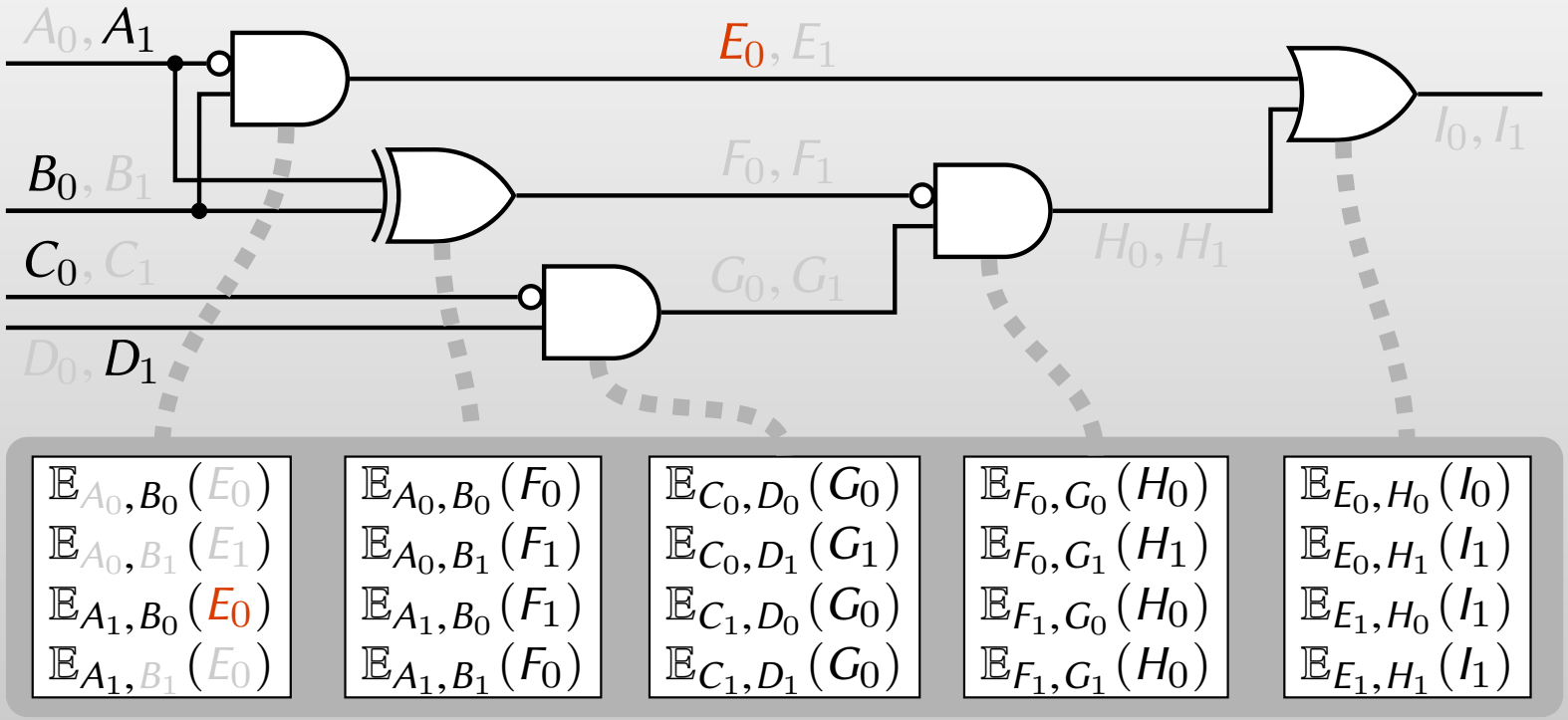
Garbled circuit framework [Yao86]



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]



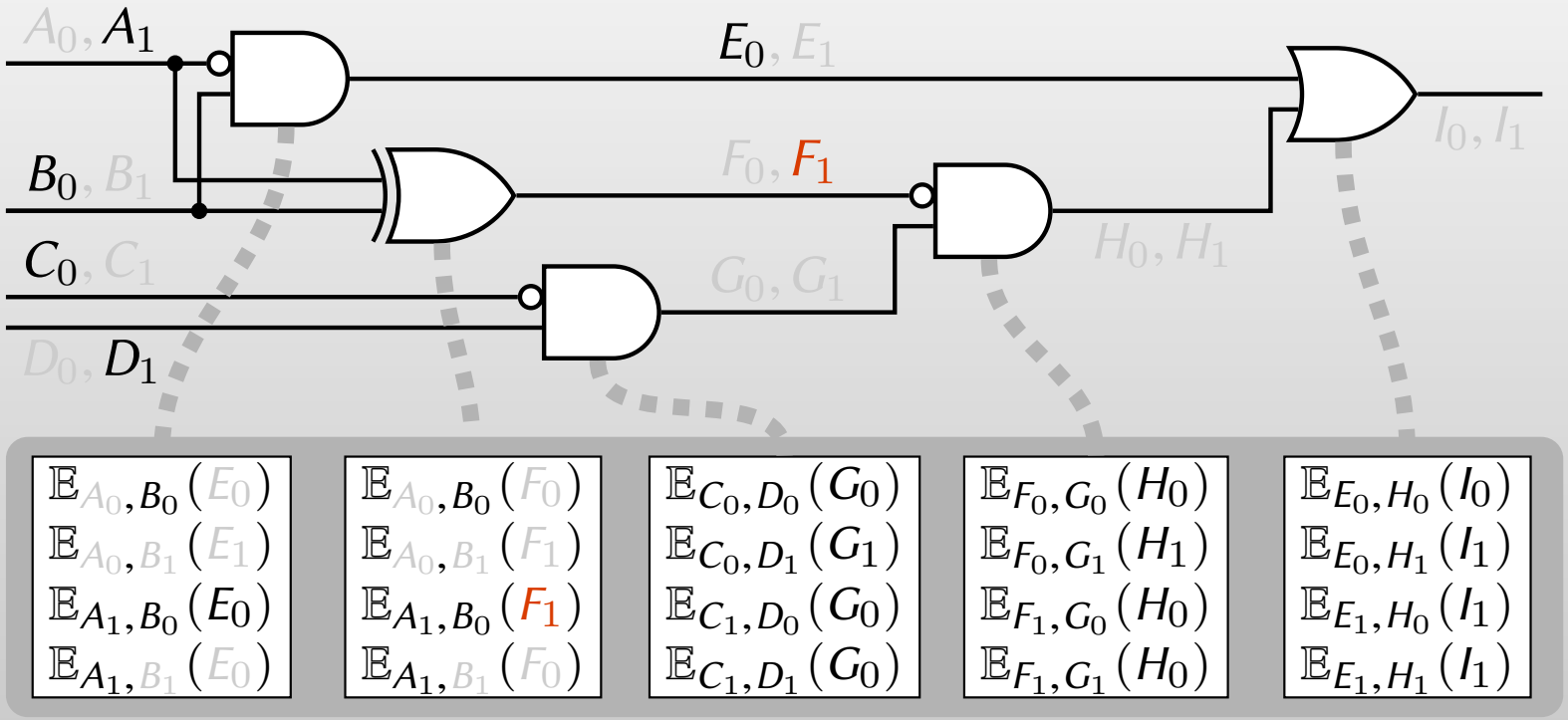
Garbling a circuit:

- 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]



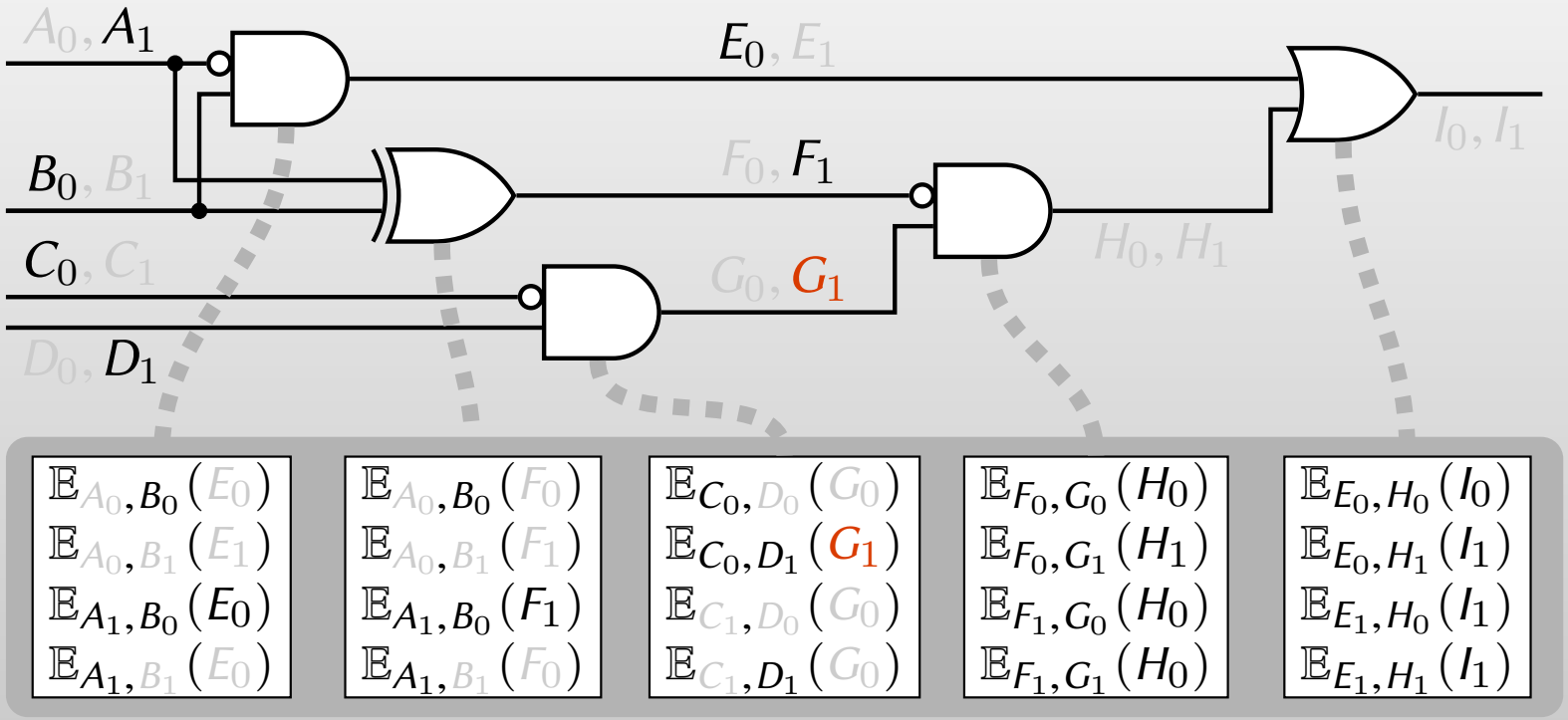
Garbling a circuit:

- 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]



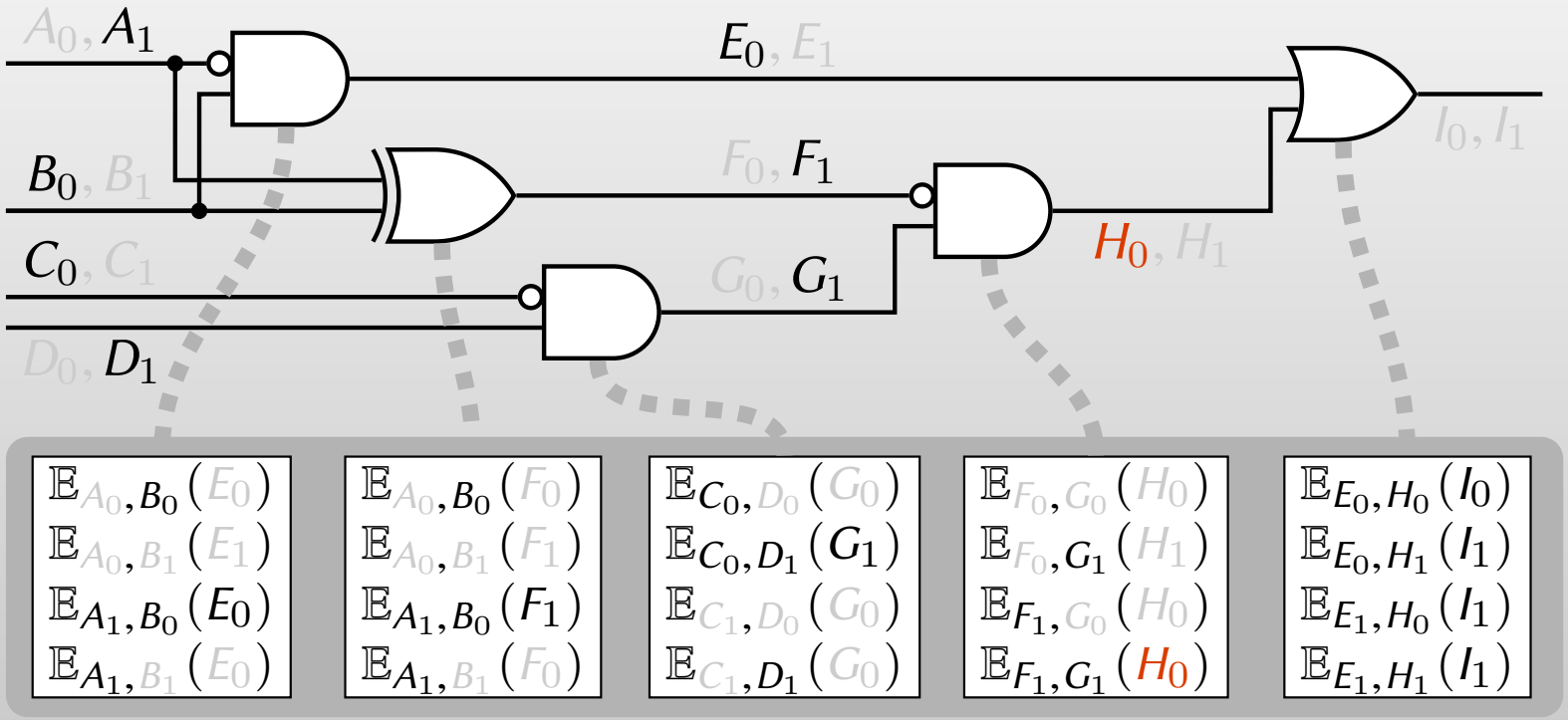
Garbling a circuit:

- 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]



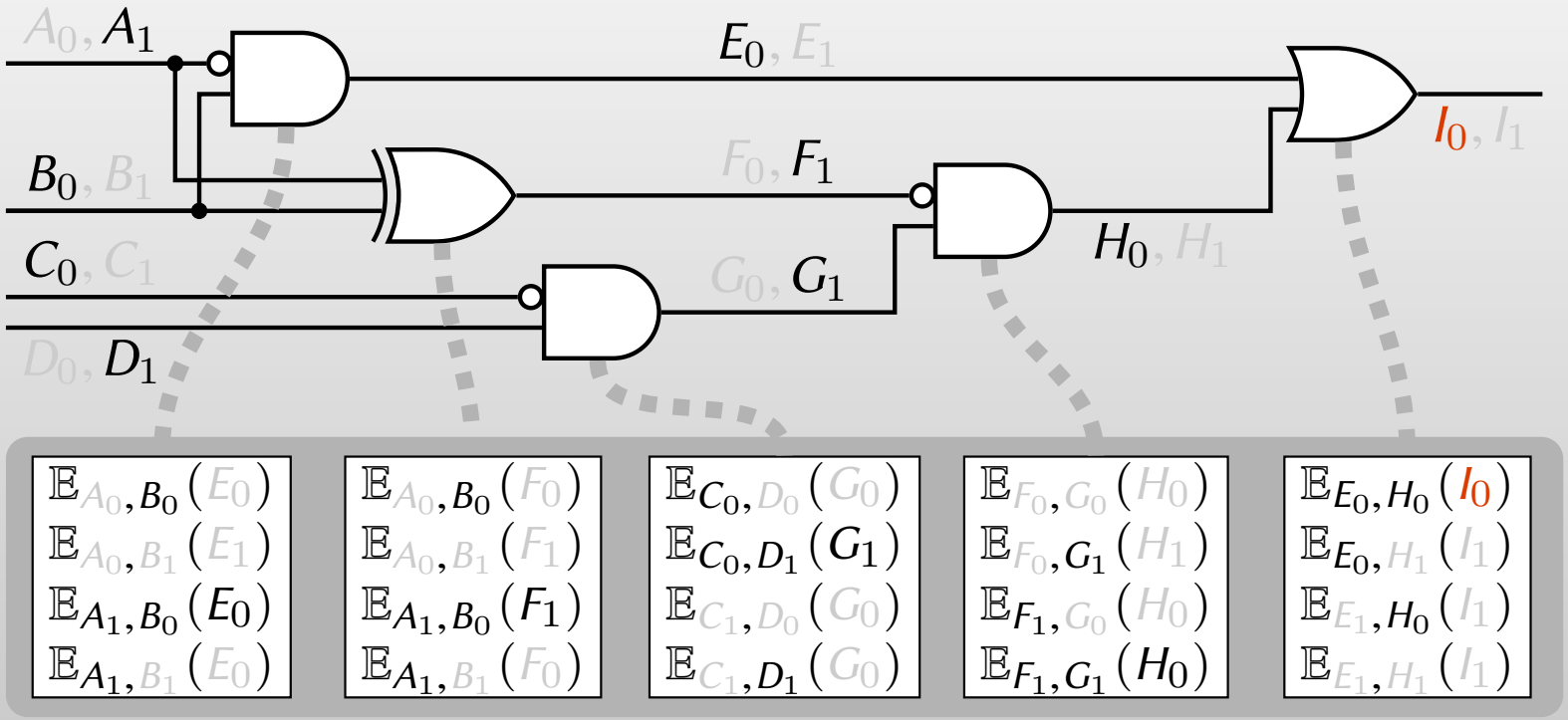
Garbling a circuit:

- 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:

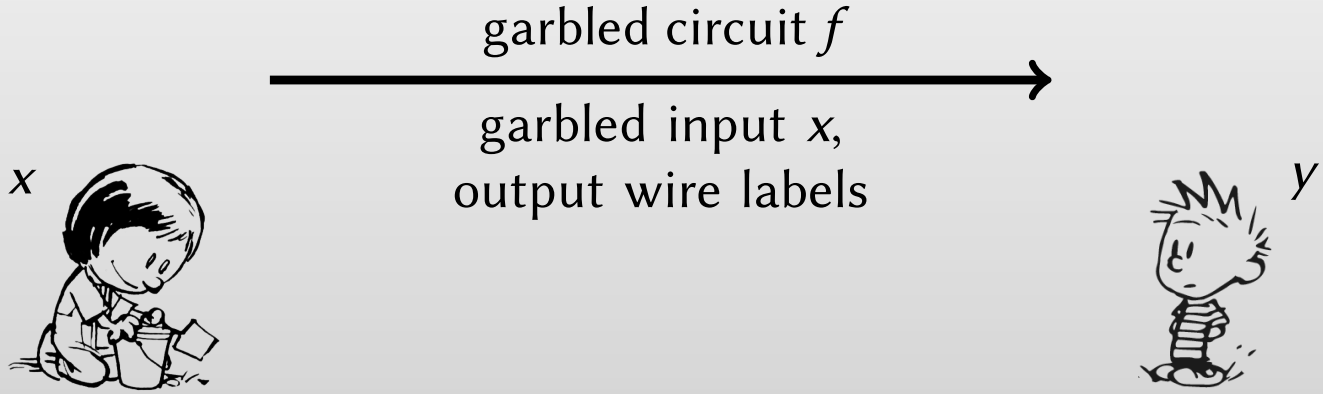
- Pick random **labels** W_0, W_1 on each wire
- "Encrypt" truth table of each gate
- **Garbled circuit** ≡ all encrypted gates
- **Garbled encoding** ≡ one label per wire

Garbled evaluation:

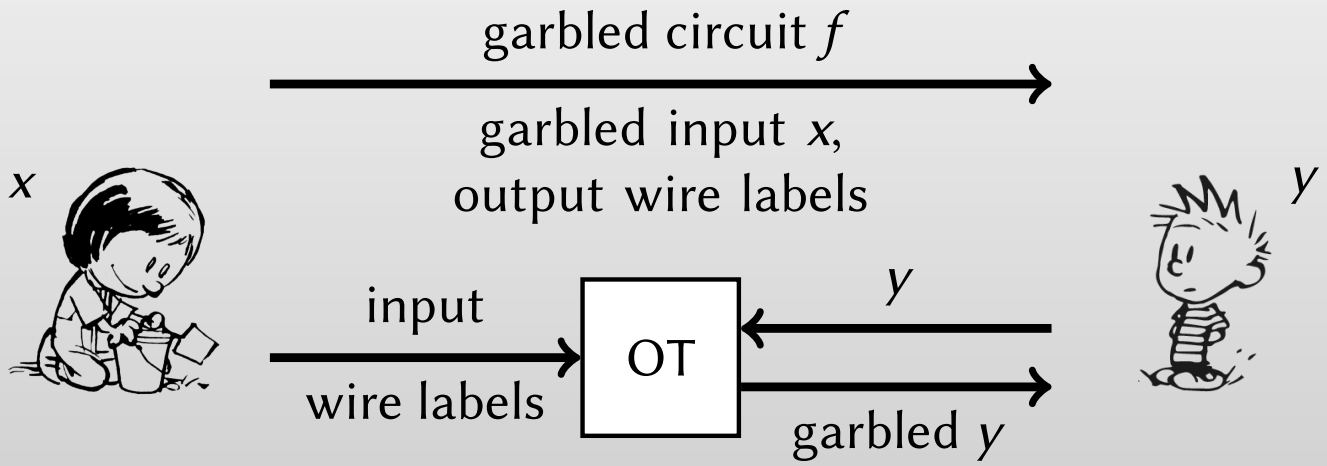
- Only one ciphertext per gate is decryptable
- Result of decryption = value on outgoing wire

Applications: 2PC and more

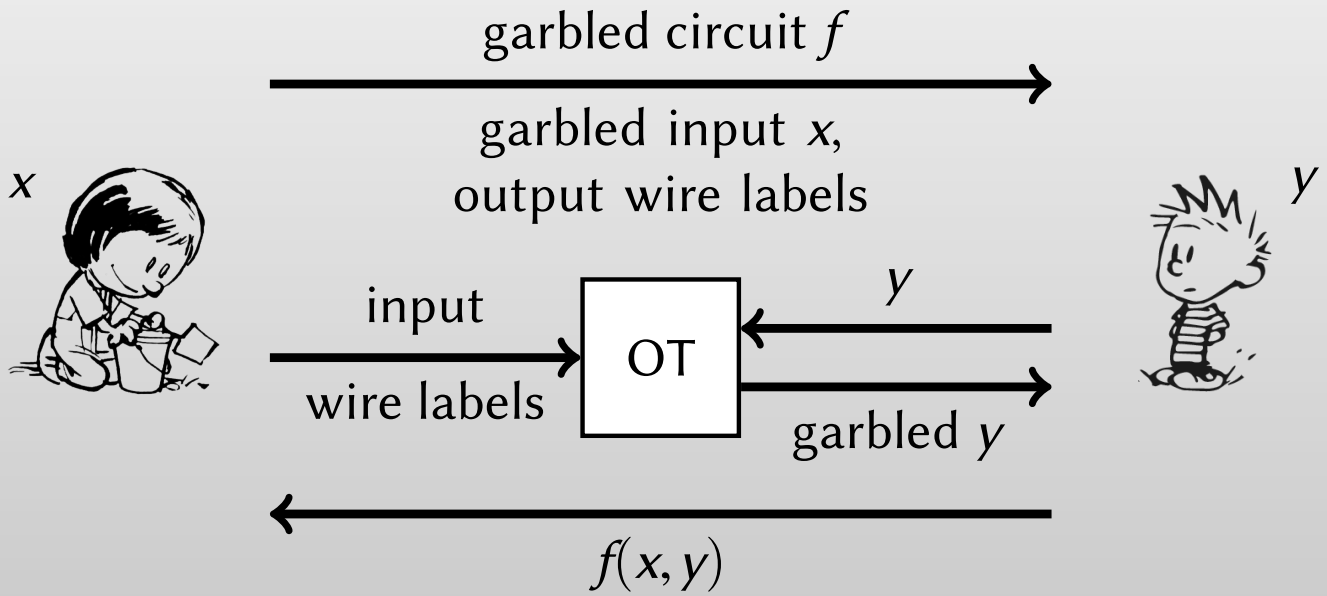
Applications: 2PC and more



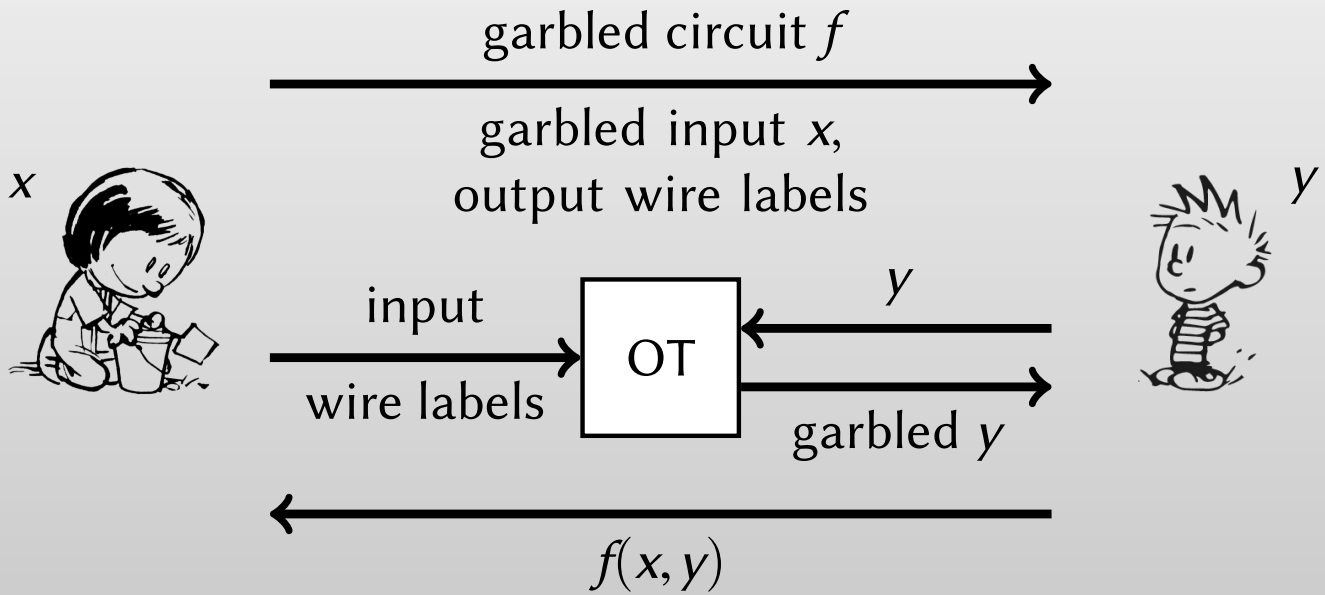
Applications: 2PC and more



Applications: 2PC and more

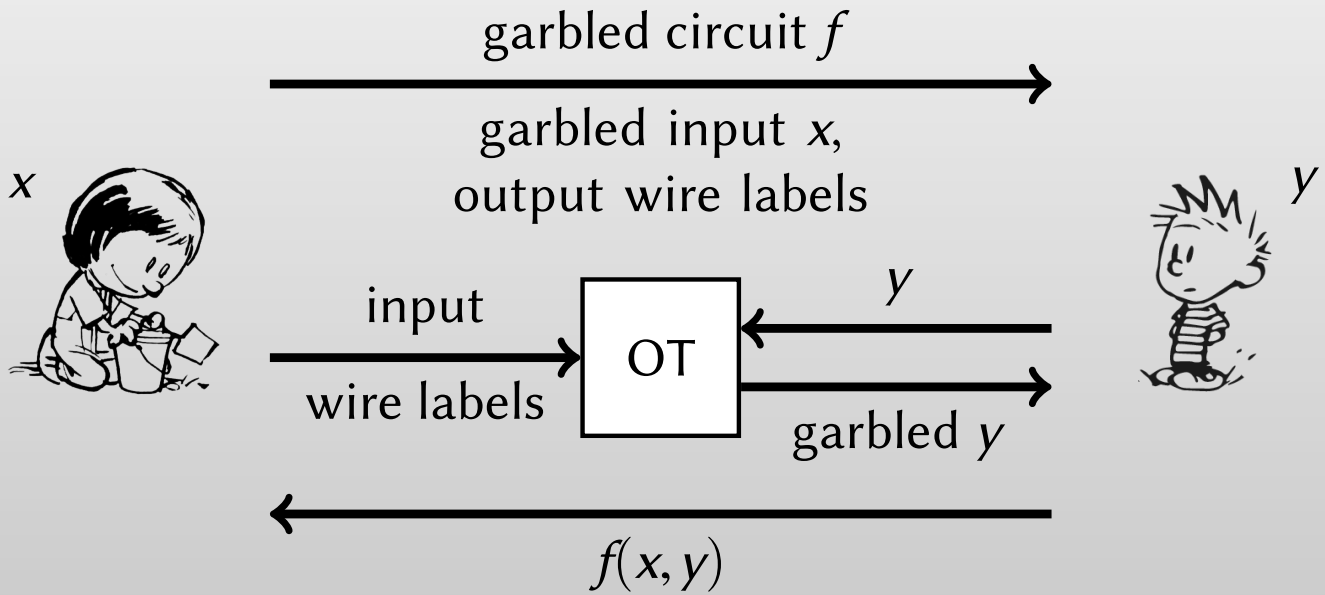


Applications: 2PC and more



Other applications: Private function evaluation, zero-knowledge proofs, encryption with key-dependent message security, randomized encodings, secure outsourcing, one-time programs, . . .

Applications: 2PC and more

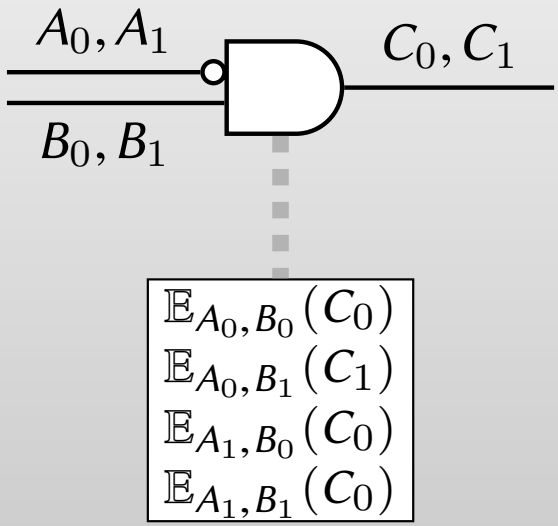


Other applications: Private function evaluation, zero-knowledge proofs, encryption with key-dependent message security, randomized encodings, secure outsourcing, one-time programs, . . .

Consistent bottleneck: **size** of garbled circuits

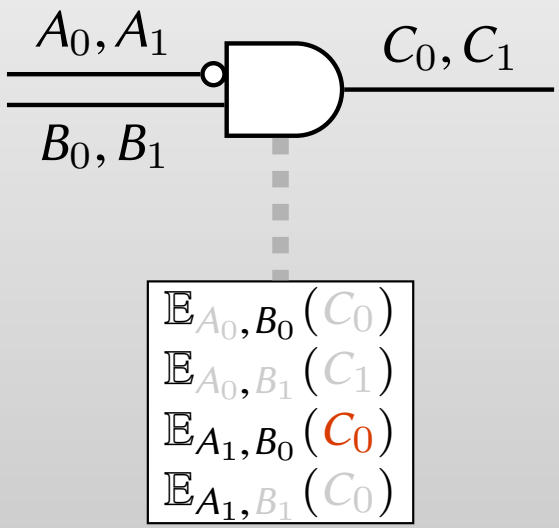
Making garbled circuits smaller

Ciphertext expansion [Yao86]



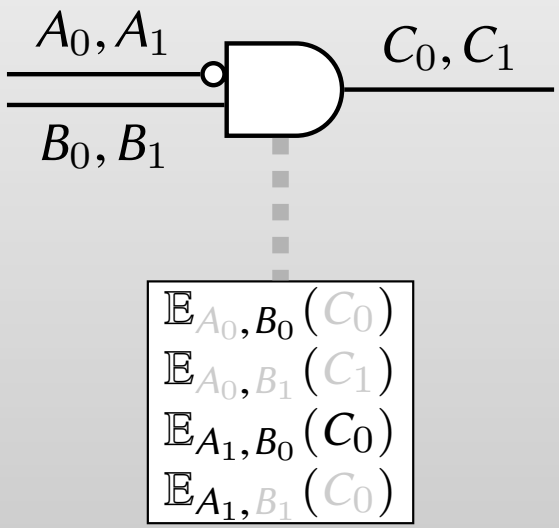
Position in this list leaks semantic value!

Ciphertext expansion [Yao86]



Position in this list leaks semantic value!

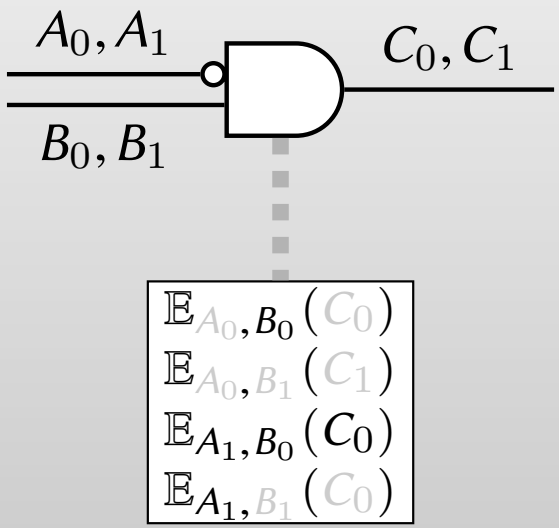
Ciphertext expansion [Yao86]



Position in this list leaks semantic value!

$\Rightarrow$ Need to randomly permute ciphertexts

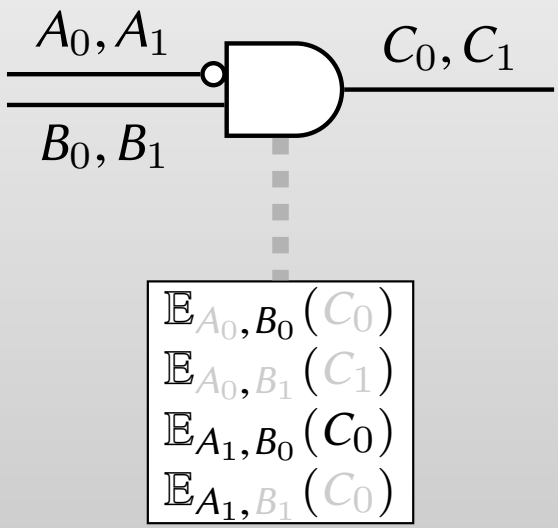
Ciphertext expansion [Yao86]



Position in this list leaks semantic value!

- $\Rightarrow$ Need to randomly permute ciphertexts
- $\Rightarrow$ Need to **detect** [in]correct decryption

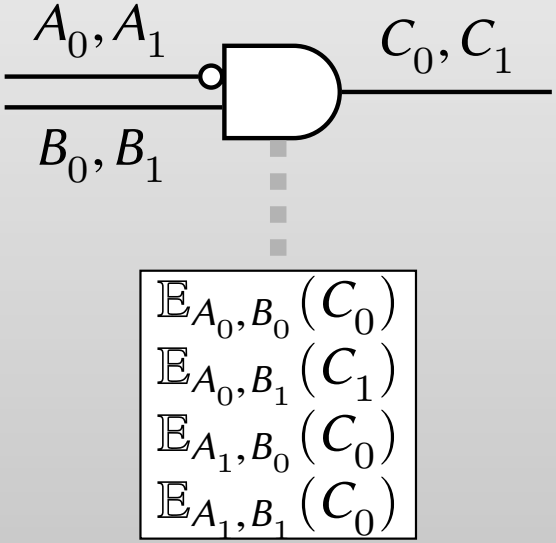
Ciphertext expansion [Yao86]



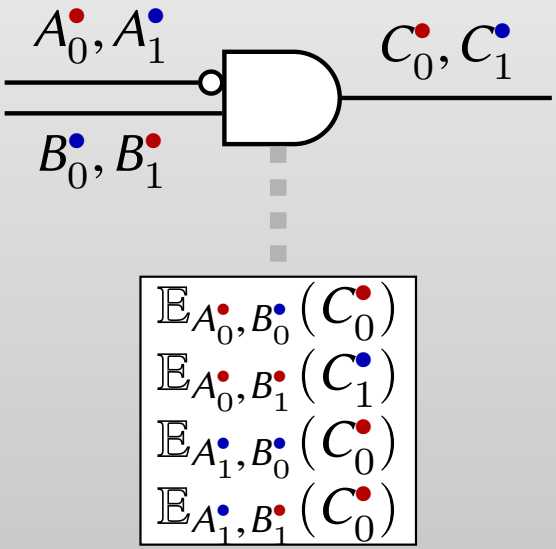
Position in this list leaks semantic value!

- ⇒ Need to randomly permute ciphertexts
- ⇒ Need to **detect** [in]correct decryption
- ⇒ Need encryption scheme with *ciphertext expansion* (size doubles)

Permute-and-Point [BeaverMicaliRogaway90]

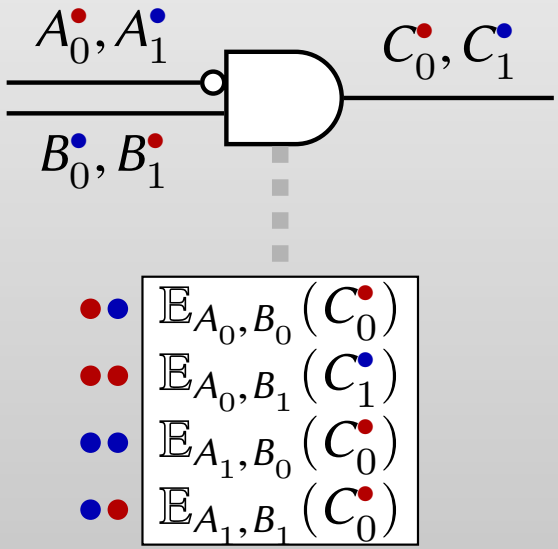


Permute-and-Point [BeaverMicaliRogaway90]



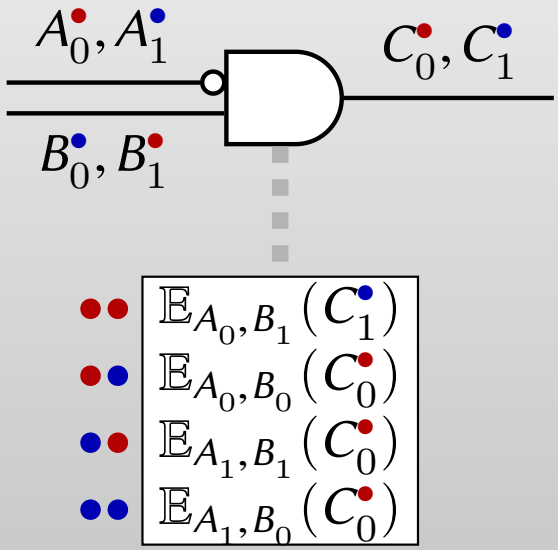
- Randomly assign $(\bullet, \bullet)$ or $(\bullet, \bullet)$ to each pair of wire labels
- Include color in the wire label (e.g., as last bit)

Permute-and-Point [BeaverMicaliRogaway90]



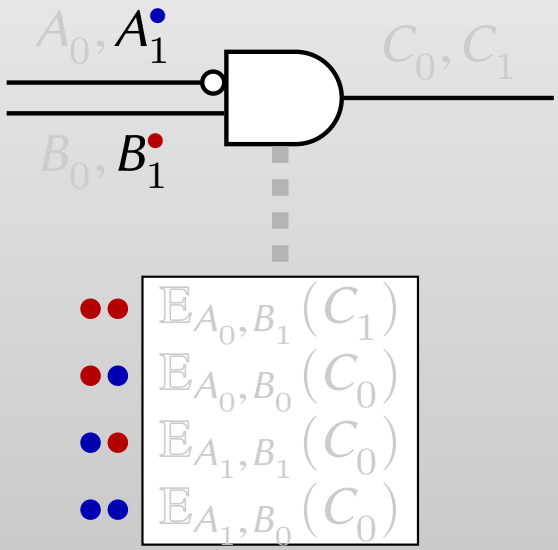
- Randomly assign $(\bullet, \bullet)$ or $(\bullet, \bullet)$ to each pair of wire labels
- Include color in the wire label (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys

Permute-and-Point [BeaverMicaliRogaway90]



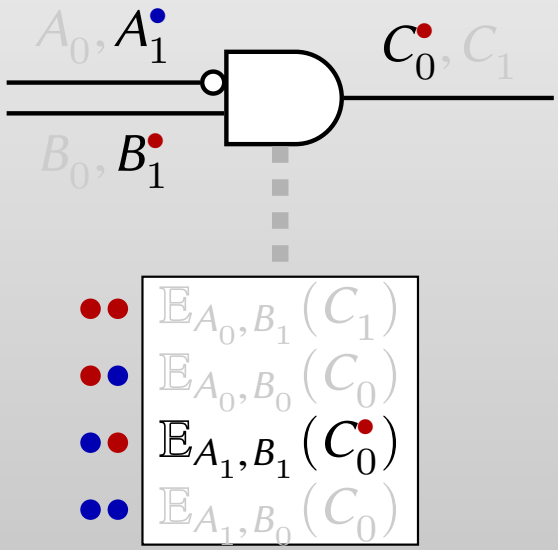
- Randomly assign $(\bullet, \bullet)$ or $(\bullet, \bullet)$ to each pair of wire labels
- Include color in the wire label (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys

Permute-and-Point [BeaverMicaliRogaway90]



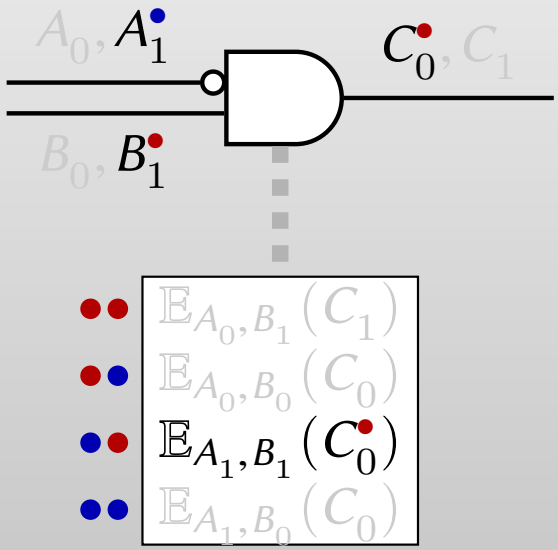
- Randomly assign $(\bullet, \bullet)$ or $(\bullet, \bullet)$ to each pair of wire labels
- Include color in the wire label (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys
- Evaluate by decrypting ciphertext indexed by your colors

Permute-and-Point [BeaverMicaliRogaway90]



- Randomly assign (●,●) or (●,●) to each pair of wire labels
- Include color in the wire label (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys
- Evaluate by decrypting ciphertext indexed by your colors

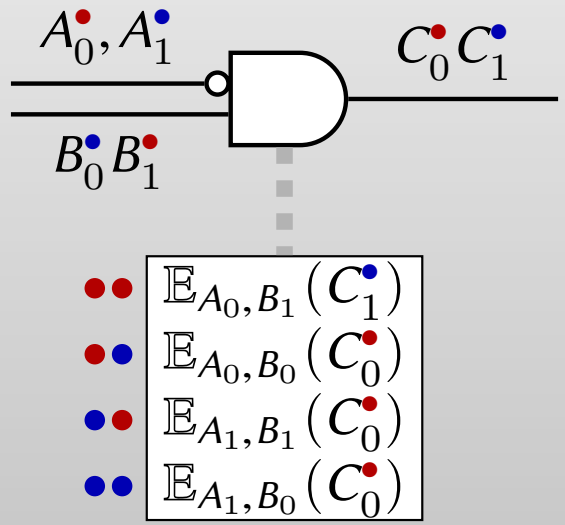
Permute-and-Point [BeaverMicaliRogaway90]



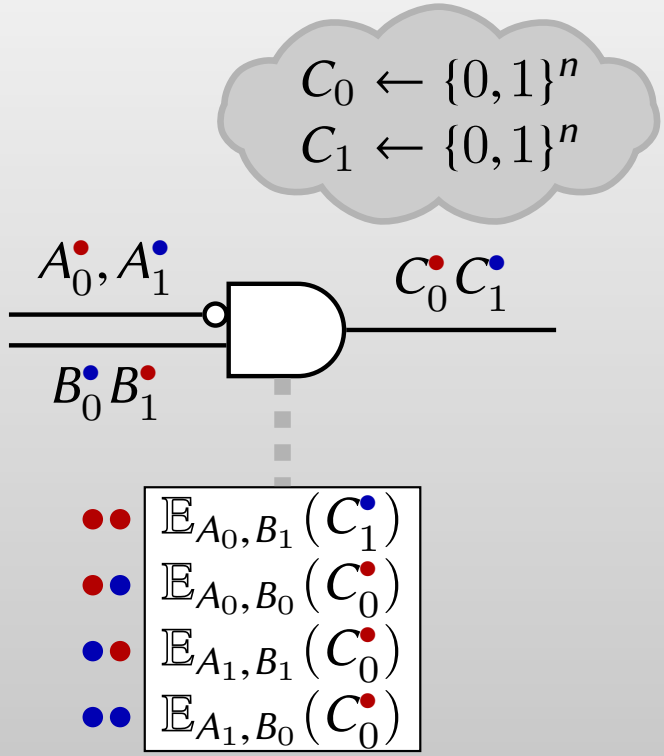
- Randomly assign (●,●) or (●,●) to each pair of wire labels
- Include color in the wire label (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys
- Evaluate by decrypting ciphertext indexed by your colors

Can use **one-time-secure** encryption (no ciphertext expansion!)

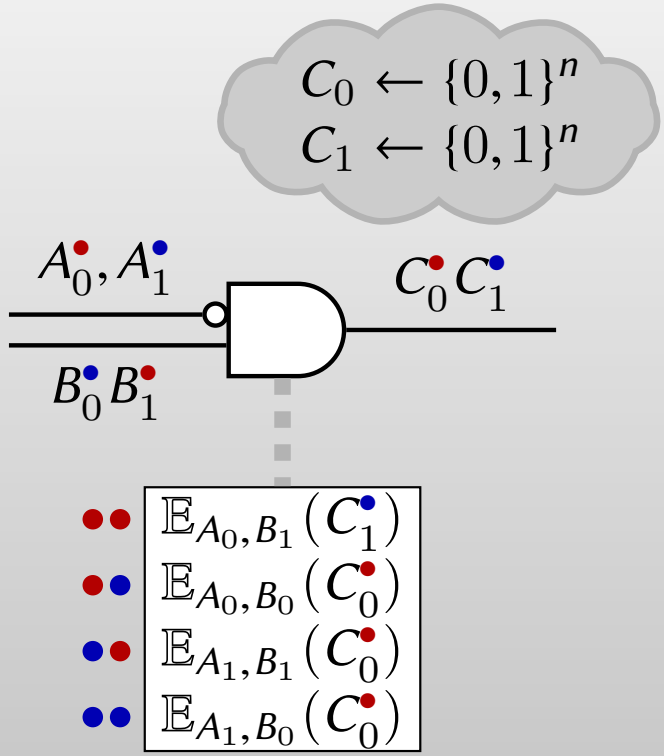
Garbled Row Reduction [NaorPinkasSumner99]



Garbled Row Reduction [NaorPinkasSumner99]

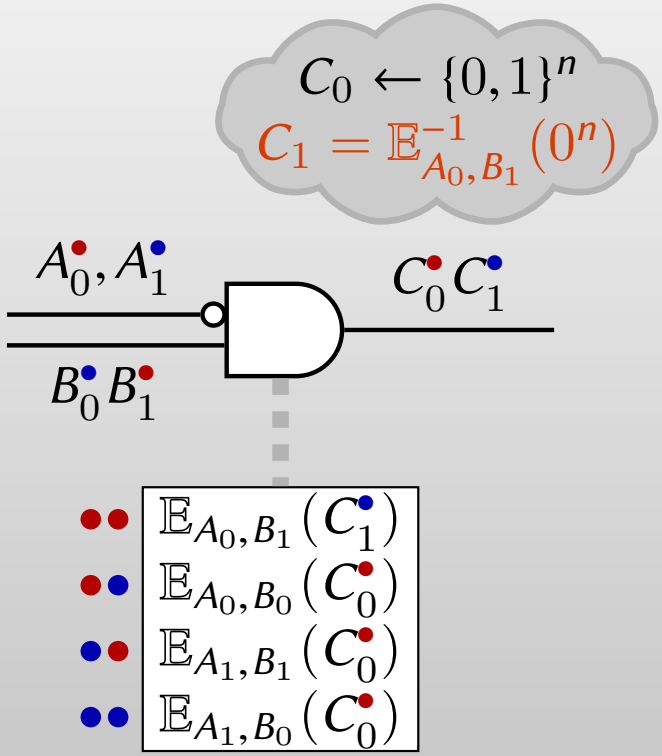


Garbled Row Reduction [NaorPinkasSumner99]



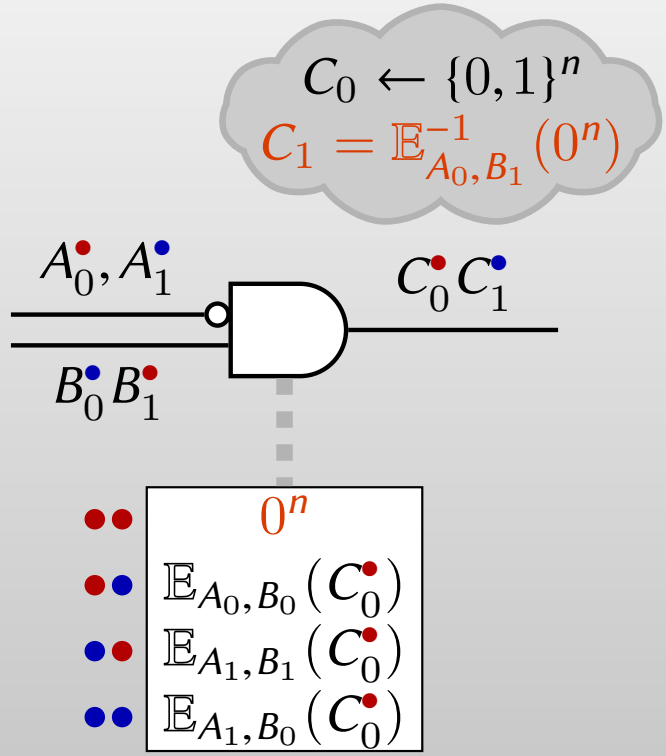
- What wire label will be payload of 1st (●●) ciphertext?

Garbled Row Reduction [NaorPinkasSumner99]



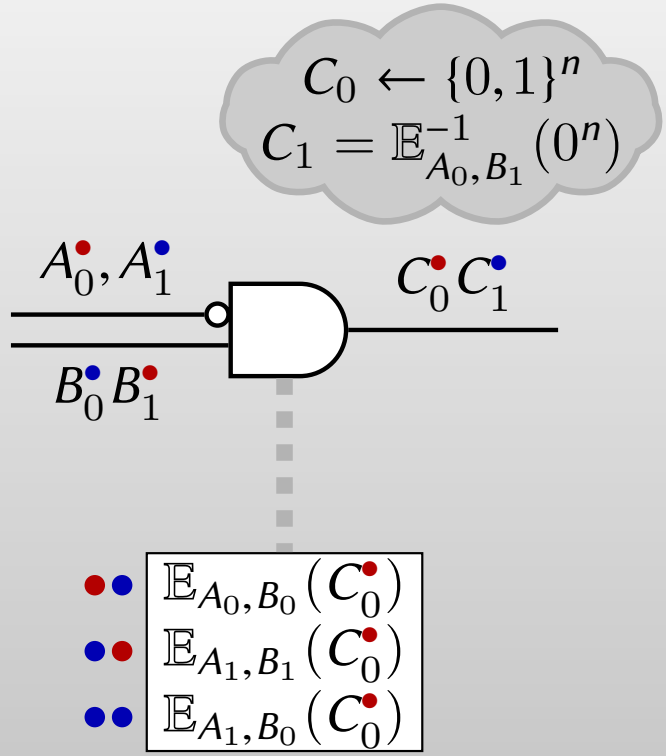
- What wire label will be payload of 1st (●●) ciphertext?
- Choose that label so that 1st ciphertext is 0^n

Garbled Row Reduction [NaorPinkasSumner99]



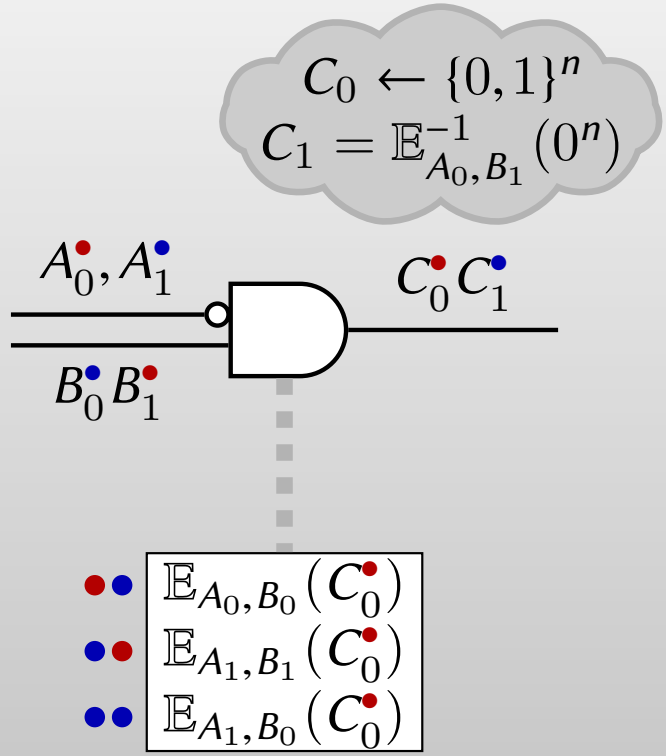
- What wire label will be payload of 1st (●●) ciphertext?
- Choose that label so that 1st ciphertext is 0^n

Garbled Row Reduction [NaorPinkasSumner99]



- What wire label will be payload of 1st (●●) ciphertext?
- Choose that label so that 1st ciphertext is 0^n
- No need to include 1st ciphertext in garbled gate

Garbled Row Reduction [NaorPinkasSumner99]

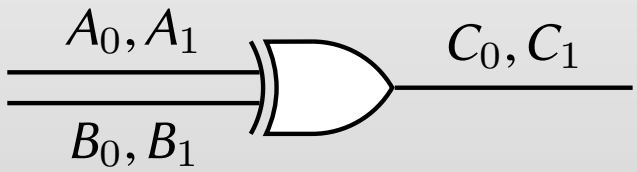


- What wire label will be payload of 1st (●●) ciphertext?
- Choose that label so that 1st ciphertext is 0^n
- No need to include 1st ciphertext in garbled gate
- Evaluate as before, but imagine ciphertext 0^n if you got ●●.

Scoreboard

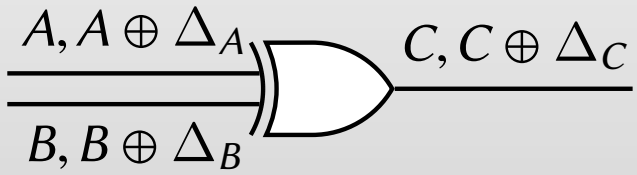
	size ($\times\lambda$)	garble cost	eval cost
Classical [Yao86,GMW87]	8	4	2.5
P&P [BeaverMicaliRogaway90]	4	4	1
GRR3 [NaorPinkasSumner99]	**3**	**4**	**1**

Free XOR [KolesnikovSchneider08]



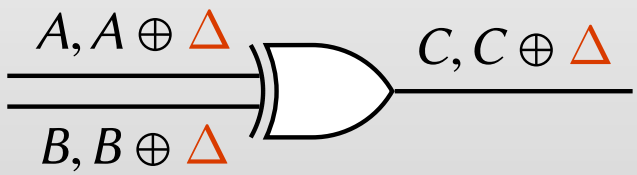
- Wire's **offset** ≡ XOR of its two labels

Free XOR [KolesnikovSchneider08]



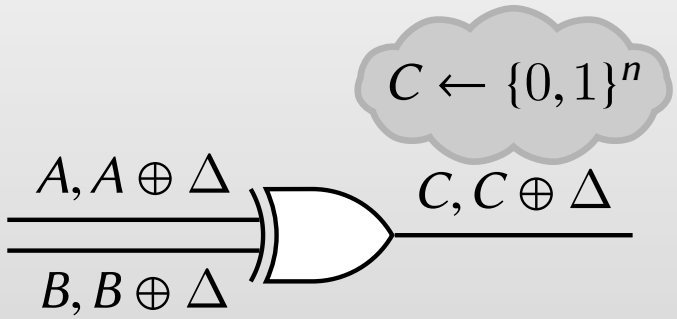
- Wire's **offset** ≡ XOR of its two labels

Free XOR [KolesnikovSchneider08]



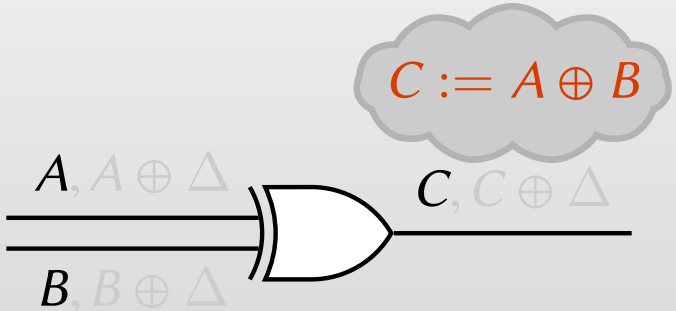
- Wire's **offset** ≡ XOR of its two labels
- Choose all wires in circuit to have same (secret) offset Δ

Free XOR [KolesnikovSchneider08]



- Wire's **offset** ≡ XOR of its two labels
- Choose all wires in circuit to have same (secret) offset Δ

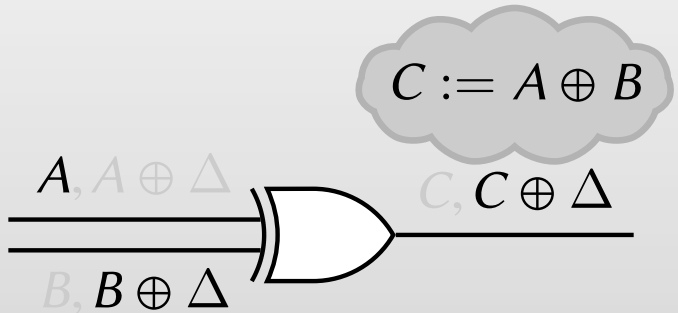
Free XOR [KolesnikovSchneider08]



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

- Wire's **offset** ≡ XOR of its two labels
- Choose all wires in circuit to have same (secret) offset Δ
- Choose FALSE output = FALSE input $\oplus$ FALSE input

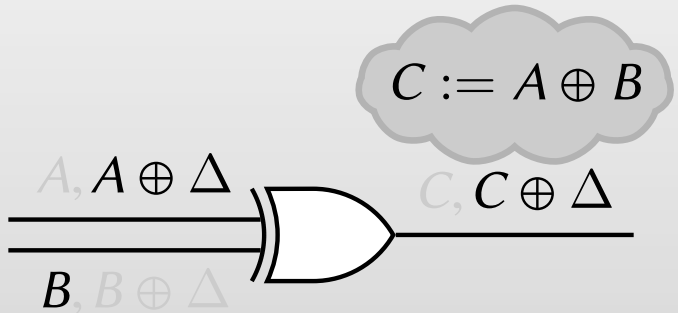
Free XOR [KolesnikovSchneider08]



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

- Wire's **offset** ≡ 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)

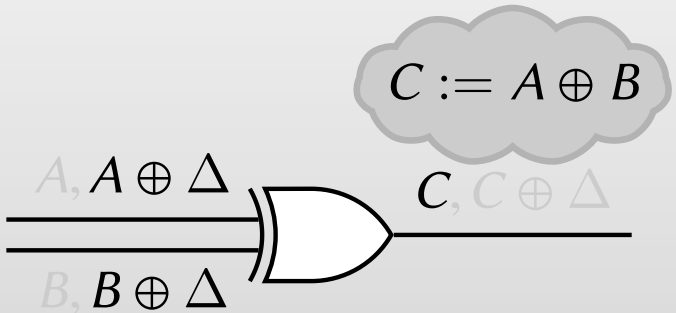
Free XOR [KolesnikovSchneider08]



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

- Wire's **offset** ≡ 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]



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

- Wire's **offset** ≡ 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**

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**
"Half gates" [ZahurRosulekEvans15]	0	2	0	4	0	2

Beyond Boolean Circuits

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

Garbling Gadgets

Garbling Gadgets for Boolean and Arithmetic Circuits (ACM CCS 2016)

- Marshall Ball, Tal Malkin, Mike Rosulek

1. New (simple!) garbled circuit building blocks
2. Applications to arithmetic computations
3. Applications to 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 $A + a\Delta$

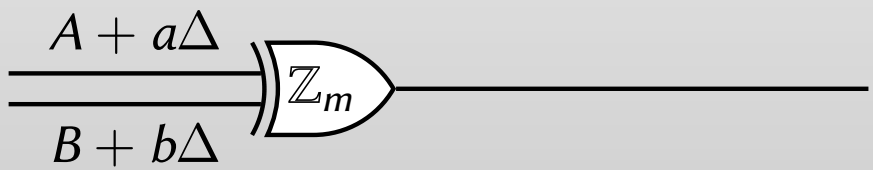
$+$ is componentwise addition mod m

Generalized Free XOR

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

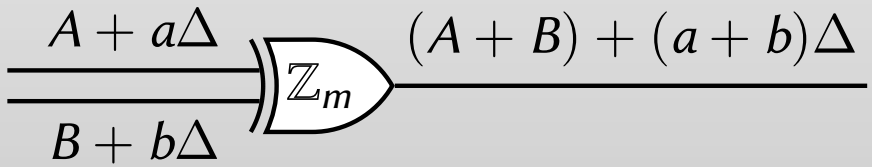
Generalized Free XOR

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



Generalized Free XOR

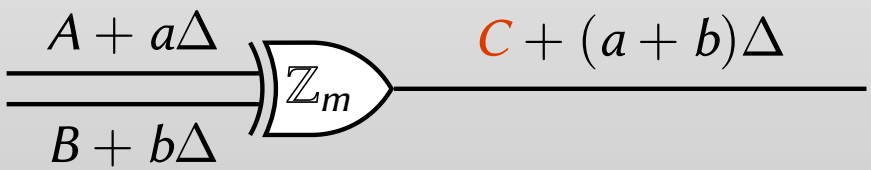
Idea: Truth value $a \in \mathbb{Z}_m$ corresponds to wire label $\underline{A + a\Delta} \in (\mathbb{Z}_m)^\lambda$



Evaulator can simply add wire labels

Generalized Free XOR

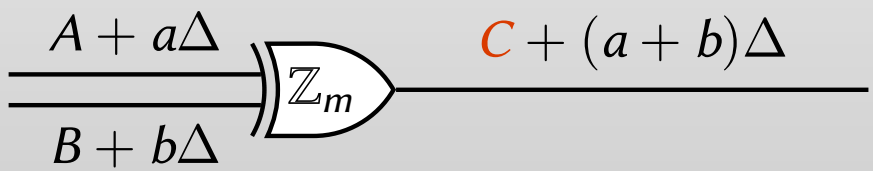
Idea: Truth value $a \in \mathbb{Z}_m$ corresponds to wire label $\underline{A + a\Delta} \in (\mathbb{Z}_m)^\lambda$



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

Generalized Free XOR

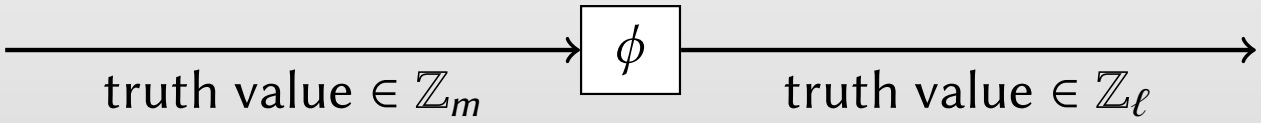
Idea: Truth value $a \in \mathbb{Z}_m$ corresponds to wire label $\underline{A + a\Delta} \in (\mathbb{Z}_m)^\lambda$



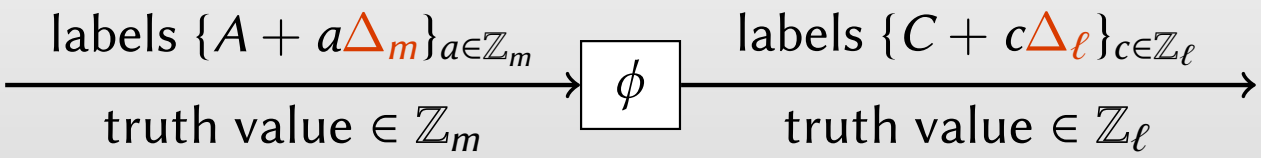
Evaulator can simply add wire labels $\Rightarrow$ free garbled addition mod m

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

Unary gates

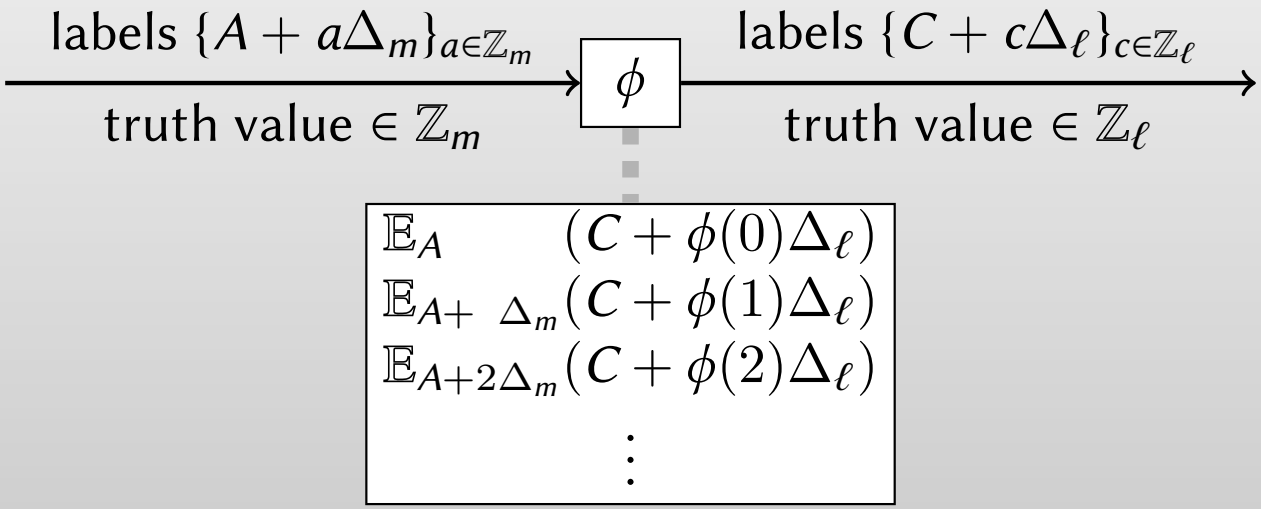


Unary gates



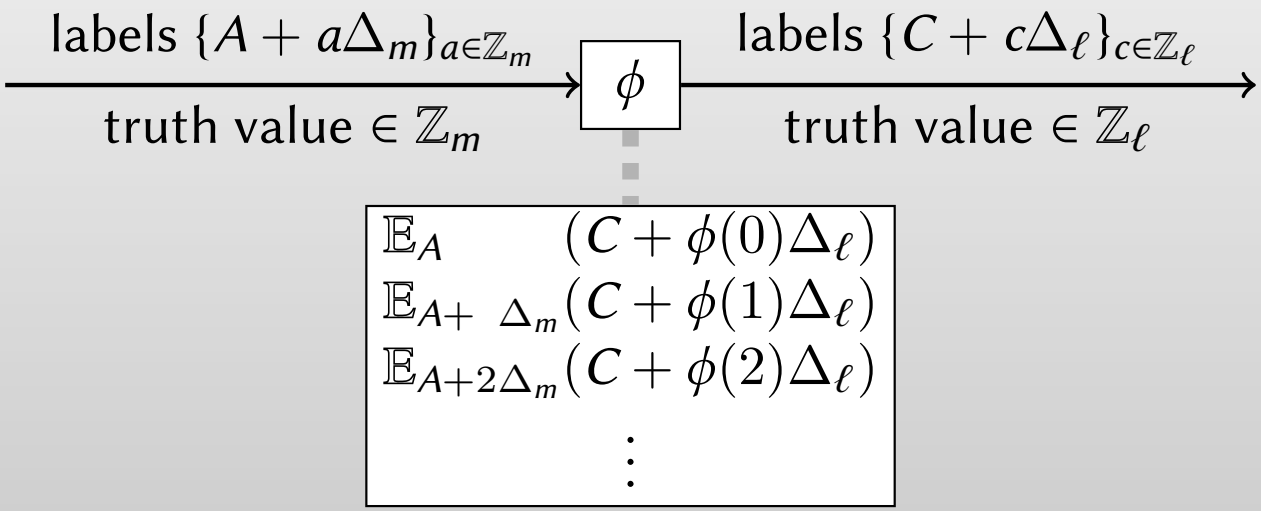
- Different "preferred modulus" on each wire $\Rightarrow$ different offsets Δ

Unary gates



- Different "preferred modulus" on each wire $\Rightarrow$ different offsets Δ
- Cost: m ciphertexts

Unary gates



- Different "preferred modulus" on each wire $\Rightarrow$ different offsets Δ
- Cost: m ciphertexts
- $m - 1$ using simple generalization of permute-and-point technique

Summary of 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

Arithmetic computations

Scenario

Securely compute linear optimization problem on 32-bit values.

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

Arithmetic computations

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 the resulting circuit (AND costs 2, XOR costs 0)

Arithmetic computations

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 the resulting circuit (AND costs 2, XOR costs 0)

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

Arithmetic computations

Insight: take advantage of **free addition**

- Represent 32-bit integers directly in $\mathbb{Z}_{2^{32}}$
- Do all arithmetic modulo 2^{32}
- In our new scheme, garbled addition is **free**

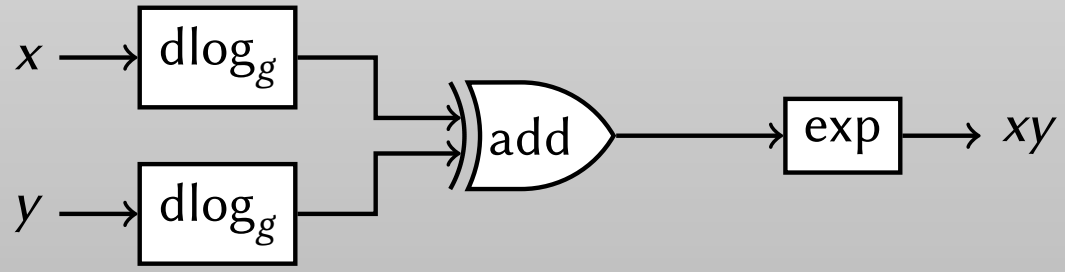
What about multiplication?

Arithmetic computations

Insight: take advantage of **free addition**

- Represent 32-bit integers directly in $\mathbb{Z}_{2^{32}}$
- Do all arithmetic modulo 2^{32}
- In our new scheme, garbled addition is **free**

What about multiplication? Our approach: **log-transform**



(plus extra edge cases when $0 \in \{x, y\}$*)*

Total cost: $\sim 6m$ for garbled multiplication mod m

Arithmetic computations

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

Arithmetic computations

$$\mathbb{Z}_{4294967296}$$

$$\downarrow$$

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

Arithmetic computations

$$\mathbb{Z}_{4294967296} = \mathbb{Z}_{2^{32}}$$
$$\downarrow$$
$$\mathbb{Z}_{6469693230}$$

Arithmetic computations

Insight: take advantage of **many moduli** in circuit

Arithmetic computations

Insight: take advantage of **many moduli** in circuit

- Represent integers via **Chinese remainder**: $\mathbb{Z}_2 \times \mathbb{Z}_3 \times \mathbb{Z}_5 \times \cdots$
- For each *logical* value, circuit includes mod-2 wire, mod-3 wire, ...
- For 32-bit integers, first 10 primes suffice: $2 \cdot 3 \cdot 5 \cdots 29 > 2^{32}$

Arithmetic computations

Insight: take advantage of **many moduli** in circuit

- Represent integers via **Chinese remainder**: $\mathbb{Z}_2 \times \mathbb{Z}_3 \times \mathbb{Z}_5 \times \cdots$
- For each *logical* value, circuit includes mod-2 wire, mod-3 wire, ...
- For 32-bit integers, first 10 primes suffice: $2 \cdot 3 \cdot 5 \cdots 29 > 2^{32}$

Costs:

- To add mod $2 \cdot 3 \cdots 29$, just add each CRT residue
- To multiply by constant, just multiply in each CRT residue

  $\Rightarrow$ garbling cost = **free**

Arithmetic computations

Insight: take advantage of **many moduli** in circuit

- Represent integers via **Chinese remainder**: $\mathbb{Z}_2 \times \mathbb{Z}_3 \times \mathbb{Z}_5 \times \cdots$
- For each *logical* value, circuit includes mod-2 wire, mod-3 wire, ...
- For 32-bit integers, first 10 primes suffice: $2 \cdot 3 \cdot 5 \cdots 29 > 2^{32}$

Costs:

- To add mod $2 \cdot 3 \cdots 29$, just add each CRT residue
- To multiply by constant, just multiply in each CRT residue
  $\Rightarrow$ garbling cost = **free**
- To multiply mod $2 \cdot 3 \cdots 29$, just multiply each CRT residue
  $\Rightarrow$ garbling cost = $6(2 + 3 + 5 + \cdots + 29)$

Arithmetic computations

Insight: take advantage of **many moduli** in circuit

- Represent integers via **Chinese remainder**: $\mathbb{Z}_2 \times \mathbb{Z}_3 \times \mathbb{Z}_5 \times \cdots$
- For each *logical* value, circuit includes mod-2 wire, mod-3 wire, ...
- For 32-bit integers, first 10 primes suffice: $2 \cdot 3 \cdot 5 \cdots 29 > 2^{32}$

Costs:

- To add mod $2 \cdot 3 \cdots 29$, just add each CRT residue
- To multiply by constant, just multiply in each CRT residue
  - $\Rightarrow$ garbling cost = **free**
- To multiply mod $2 \cdot 3 \cdots 29$, just multiply each CRT residue
  - $\Rightarrow$ garbling cost = $6(2 + 3 + 5 + \cdots + 29)$
- To raise to public power, use unary gate $x \mapsto x^c$ in each CRT residue
  - $\Rightarrow$ garbling cost = $(2 - 1) + (3 - 1) + (5 - 1) + \cdots + (29 - 1)$

Arithmetic computations

	standard	awful	**CRT**
addition	62	0	**0**
multiplication by public constant	758	0	**0**
multiplication	1200	25769803776	**724**
squaring, cubing, etc	1864	4294967296	**119**

Arithmetic computations

	standard	awful	**CRT**
addition	62	0	**0**
multiplication by public constant	758	0	**0**
multiplication	1200	25769803776	**724 (238)**
squaring, cubing, etc	1864	4294967296	**119**

High fan-in computations

Scenario

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

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

High fan-in computations

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 the resulting circuit (AND costs 2, XOR costs 0)

High fan-in computations

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 the resulting circuit (AND costs 2, XOR costs 0)

	cost (# ciphertexts)
AND/OR	198
majority	948

High fan-in computations

Insight: AND is unary function of a sum

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

High fan-in computations

Insight: AND is unary function of a sum

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

High fan-in computations

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

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

- Represent each bit in $\mathbb{Z}_{101}$
- Cost to garble sum in $\mathbb{Z}_{101}$ is **free**
- Equality comparison is simple $\mathbb{Z}_{101}$-unary gate $\Rightarrow$ cost = 100

High fan-in computations

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

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

- Represent each bit in $\mathbb{Z}_{101}$
- Cost to garble sum in $\mathbb{Z}_{101}$ is **free**
- Equality comparison is simple $\mathbb{Z}_{101}$-unary gate $\Rightarrow$ cost = 100

	standard	better
AND/OR	198	100
majority	948	

High fan-in computations

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

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

- Represent each bit in $\mathbb{Z}_{101}$
- Cost to garble sum in $\mathbb{Z}_{101}$ is **free**
- Equality comparison is simple $\mathbb{Z}_{101}$-unary gate $\Rightarrow$ cost = 100

	standard	better
AND/OR	198	100
majority	948	100

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

High fan-in computations

$$\mathbb{Z}_{101}$$
$$\downarrow$$
$$\mathbb{Z}_{210}$$

High fan-in computations

$$\mathbb{Z}_{101}$$
$$\downarrow$$
$$\mathbb{Z}_{210} = \mathbb{Z}_{2 \cdot 3 \cdot 5 \cdot 7}$$

High fan-in computations

Insight: take advantage of **multiple moduli**

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

High fan-in computations

Insight: take advantage of **multiple moduli**

$$\begin{aligned}\mathsf{AND}(x_1,\ldots,x_{100}) &= [\textstyle\sum x_i \stackrel{?}{=} 100] = [\sum x_i \stackrel{?}{\equiv} 100 \pmod{210}] \\ &= [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod{p}\ \forall p \in \{2,3,5,7\}]\end{aligned}$$

High fan-in computations

Insight: take advantage of **multiple moduli**

$$\begin{aligned}\mathrm{AND}(x_1,\ldots,x_{100}) &= [\textstyle\sum x_i \stackrel{?}{=} 100] = [\sum x_i \stackrel{?}{\equiv} 100 \pmod{210}] \\ &= [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod{p}\ \forall p \in \{2,3,5,7\}]\end{aligned}$$

Our approach

- Represent each bit via mod-2 wire, mod-3 wire, mod-5 wire, ...

High fan-in computations

Insight: take advantage of **multiple moduli**

$$\begin{aligned}\mathrm{AND}(x_1,\ldots,x_{100}) &= [\textstyle\sum x_i \stackrel{?}{=} 100] = [\sum x_i \stackrel{?}{\equiv} 100 \pmod{210}] \\ &= [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod{p}\ \forall p \in \{2,3,5,7\}]\end{aligned}$$

Our approach

- Represent each bit via mod-2 wire, mod-3 wire, mod-5 wire, ...
- Each summation mod p is **free**
- Cost of equality tests = $2+3+5+7$

High fan-in computations

Insight: take advantage of **multiple moduli**

$$\begin{aligned}\text{AND}(x_1,\ldots,x_{100}) &= [\textstyle\sum x_i \stackrel{?}{=} 100] = [\sum x_i \stackrel{?}{\equiv} 100 \pmod{210}] \\ &= [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod{p}\ \forall p \in \{2,3,5,7\}]\end{aligned}$$

Our approach

- Represent each bit via mod-2 wire, mod-3 wire, mod-5 wire, ...
- Each summation mod p is **free**
- Cost of equality tests = $2+3+5+7$
- Compute AND of 4 equality test results $\Rightarrow$ cost = 4 (previous method)

High fan-in computations

Insight: take advantage of **multiple moduli**

$$\begin{aligned}\text{AND}(x_1,\ldots,x_{100}) &= [\textstyle\sum x_i \stackrel{?}{=} 100] = [\sum x_i \stackrel{?}{\equiv} 100 \pmod{210}] \\ &= [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod{p}\ \forall p \in \{2,3,5,7\}]\end{aligned}$$

Our approach

- Represent each bit via mod-2 wire, mod-3 wire, mod-5 wire, ...
- Each summation mod p is **free**
- Cost of equality tests = $2+3+5+7$
- Compute AND of 4 equality test results $\Rightarrow$ cost = 4 (previous method)

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

High fan-in computations

Insight: take advantage of **multiple moduli**

$$\begin{aligned}\text{AND}(x_1,\ldots,x_{100}) &= [\textstyle\sum x_i \stackrel{?}{=} 100] = [\sum x_i \stackrel{?}{\equiv} 100 \pmod{210}] \\ &= [\textstyle\sum x_i \stackrel{?}{\equiv} 100 \pmod{p}\ \forall p \in \{2,3,5,7\}]\end{aligned}$$

Our approach

- Represent each bit via mod-2 wire, mod-3 wire, mod-5 wire, . . .
- Each summation mod p is **free**
- Cost of equality tests = $2+3+5+7$
- Compute AND of 4 equality test results $\Rightarrow$ cost = 4 (previous method)

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

Summarizing

For **arithmetic operations on bounded integers:**

- Represent in primorial modulus + CRT (10-20 primes)
- **Free** addition & multiplication by constant!
- Multiplication concretely better than boolean
- Exponentiation concretely+asymptotically better

Summarizing

For **arithmetic operations on bounded integers:**

- Represent in primorial modulus + CRT (10-20 primes)
- **Free** addition & multiplication by constant!
- Multiplication concretely better than boolean
- Exponentiation concretely+asymptotically better

For **high-fan-in computations on bits:**

- Represent bits in primorial modulus + CRT (3-5 primes)
- Threshold gates (incl. AND, OR) **exponentially better** than boolean

Challenges

How to compute $x < y$ for 32-bit numbers?

Our approach: represent x, y in $\mathbb{Z}_2 \times \mathbb{Z}_3 \times \cdots \times \mathbb{Z}_{29}$

- Our best cost for garbled comparison = 2541

Challenges

How to compute $x < y$ for 32-bit numbers?

Our approach: represent x, y in $\mathbb{Z}_2 \times \mathbb{Z}_3 \times \cdots \times \mathbb{Z}_{29}$

- Our best cost for garbled comparison = 2541

Stndard approach: represent x, y in binary:

- Boolean circuit is simple $\Rightarrow$ garbled cost = 64

Challenges

How to compute $x < y$ for 32-bit numbers?

Our approach: represent x, y in $\mathbb{Z}_2 \times \mathbb{Z}_3 \times \cdots \times \mathbb{Z}_{29}$

- Our best cost for garbled comparison = 2541

Stndard approach: represent x, y in binary:

- Boolean circuit is simple $\Rightarrow$ garbled cost = 64

How to efficiently convert CRT $\leftrightarrow$ binary?

Example: perform arithmetic operations, then compute a **cryptographic hash** of result

Challenges

"Free" stuff is not free!

In our garbling scheme:

- Addition gates treat wire labels as elements of $(\mathbb{Z}_p)^\lambda$
- Projection gates treat wire labels as **bits** (used as encryption scheme)

Challenges

"Free" stuff is not free!

In our garbling scheme:

- Addition gates treat wire labels as elements of $(\mathbb{Z}_p)^\lambda$
- Projection gates treat wire labels as **bits** (used as encryption scheme)
- Conversion contributes significantly to cost in practice!

the end!