New Results for Garbling Arithmetic and High Fan-In Computations

Mike Rosulek

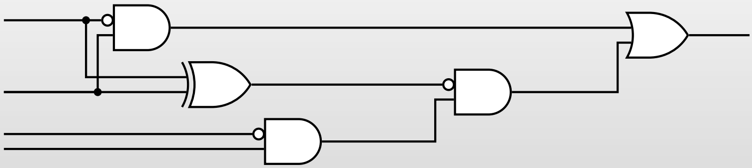
Oregon State University OSU

Garbling Gadgets for Boolean and Arithmetic Circuits

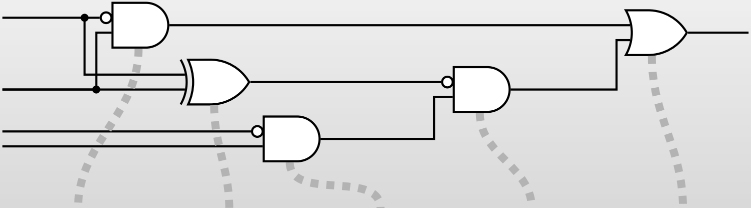
Marshall Ball, Tal Malkin, Mike Rosulek

CCS 2016; eprint.iacr.org/2016/969

Garbled circuit framework [Yao86]



Garbled circuit framework [Yao86]



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

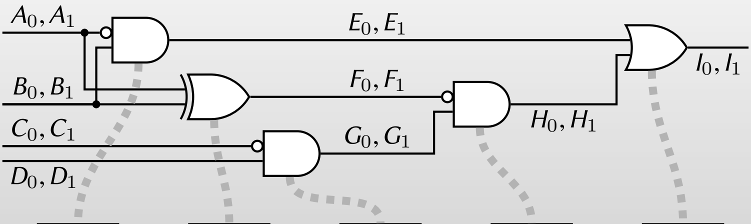
0	0	0
0	1	1
1	0	1
1	1	0

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

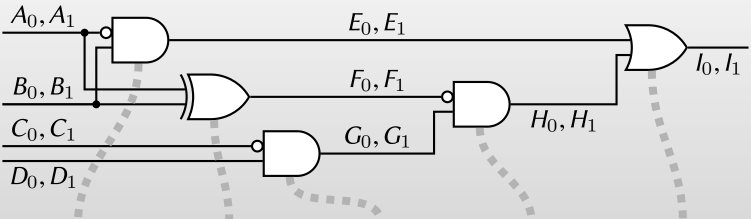
0	0	0
0	1	1
1	0	0
1	1	0

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

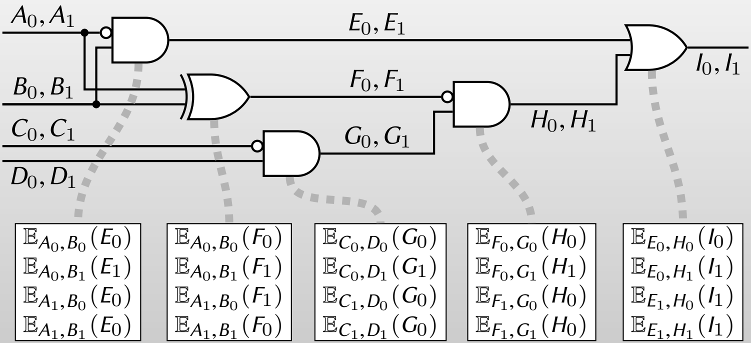
F_0	G_0	H_0
F_0	G_1	H_1
F_1	G_0	H_0
F_1	G_1	H_0

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

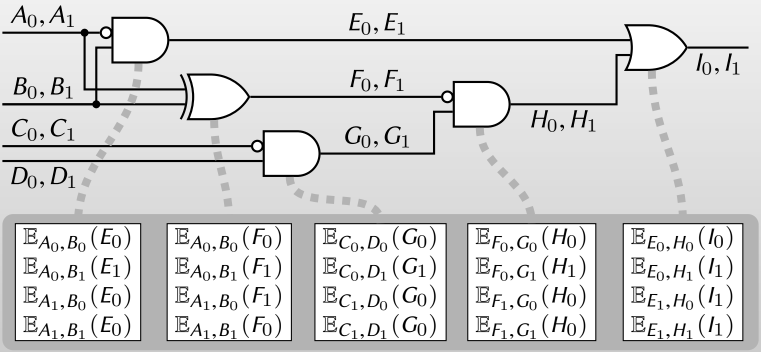
Garbled circuit framework [Yao86]



Garbling a circuit:

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

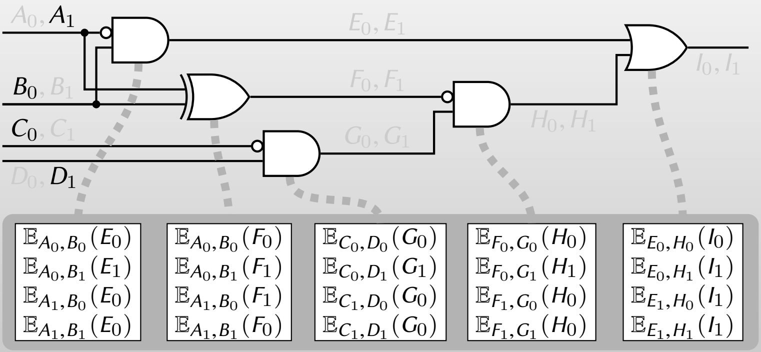
Garbled circuit framework [Yao86]



Garbling a circuit:

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

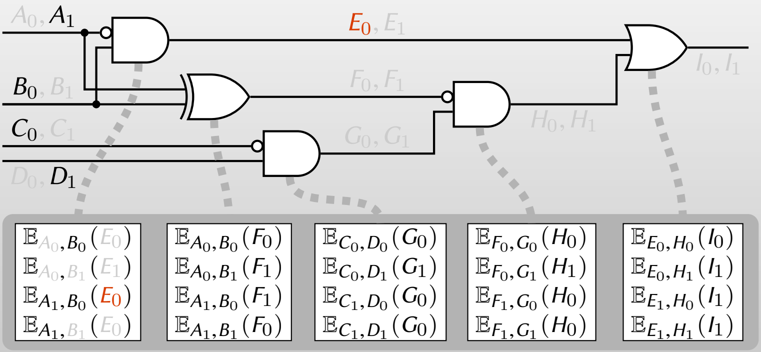
Garbled circuit framework [Yao86]



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]



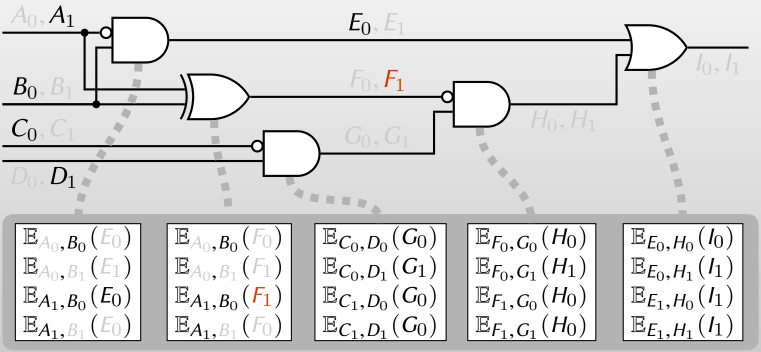
Garbling a circuit:

- 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]



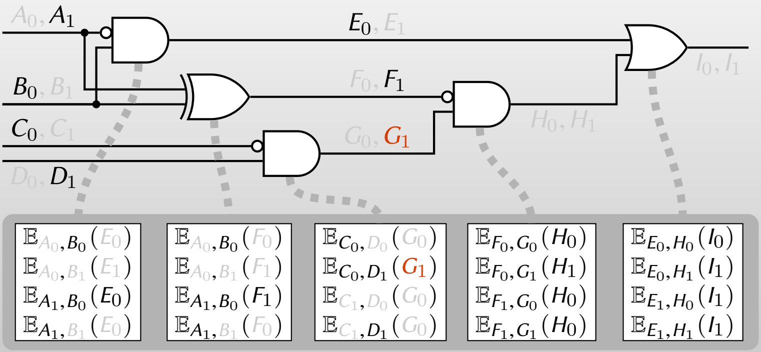
Garbling a circuit:

- 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]



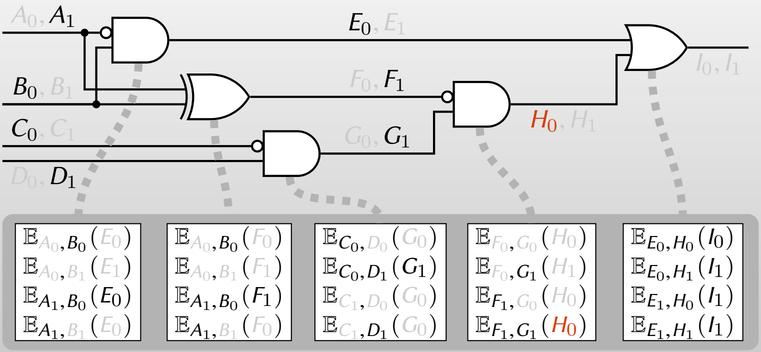
Garbling a circuit:

- 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]



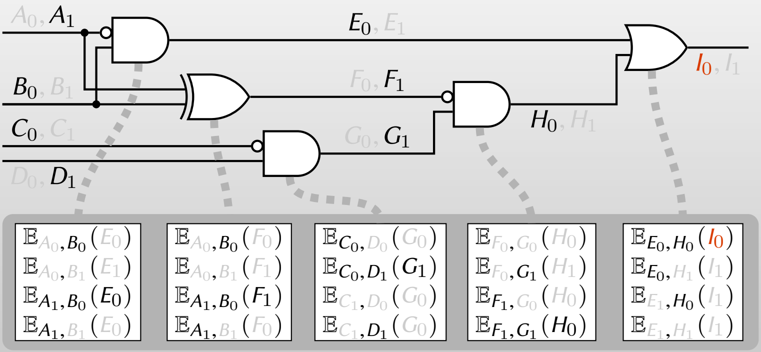
Garbling a circuit:

- 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

Garbled circuits = **boolean** *circuits*

Size of Garbled Circuits

		gc size ($\times\lambda$ bits)	
		XOR	AND
Textbook + P&P	[Yao86 + BeaverMicaliRogaway90]	4	4
GRR3	[NaorPinkasSumner99]	3	3
Free XOR	[KolesnikovSchneider08]	0	3
GRR2	[PinkasSchneiderSmartWilliams09]	2	2
FleXOR	[KolesnikovMohasselRosulek14]	{0,1,2}	2
Half gates	[ZahurRosulekEvans15]	**0**	**2**

"traditional" GC only: ignoring privacy-free, gate-private, formulas-only, etc.

Size of Garbled Circuits

		gc size (×λ bits) XOR	AND
Textbook + P&P	[Yao86 + BeaverMicaliRogaway90]	4	4
GRR3	[NaorPinkasSumner99]	3	3
Free XOR	[KolesnikovSchneider08]	0	3
GRR2	[PinkasSchneiderSmartWilliams09]	2	2
FleXOR	[KolesnikovMohasselRosulek14]	{0,1,2}	2
Half gates	[ZahurRosulekEvans15]	**0**	**2** ★

"traditional" GC only: ignoring privacy-free, gate-private, formulas-only, etc.

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

Garbling Gadgets for Boolean and Arithmetic Circuits

Marshall Ball, Tal Malkin, Mike Rosulek

CCS 2016

---

1. New garbled circuit building blocks
2. Applications to arithmetic computations
3. Applications to high-fan-in boolean computations
4. Other challenges

Garbling Gadgets for Boolean and Arithmetic Circuits

Marshall Ball, Tal Malkin, Mike Rosulek

CCS 2016

1. New garbled circuit building blocks
   - Generalized Free-XOR (very simple)
2. Applications to arithmetic computations
3. Applications to high-fan-in boolean computations
4. Other challenges

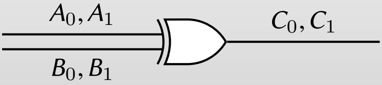
Garbling Gadgets for Boolean and Arithmetic Circuits

Marshall Ball, Tal Malkin, Mike Rosulek

CCS 2016

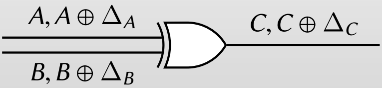
1. New garbled circuit building blocks
   - Generalized Free-XOR (very simple)
2. Applications to arithmetic computations
   - Encoding under many moduli (also very simple)
3. Applications to high-fan-in boolean computations
4. Other challenges

Free XOR [KolesnikovSchneider08]



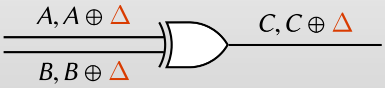
- Wire's **offset** ≡ XOR of its two labels

Free XOR [KolesnikovSchneider08]



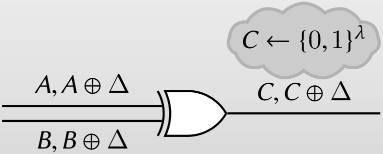
- Wire's **offset** ≡ XOR of its two labels

Free XOR [KolesnikovSchneider08]



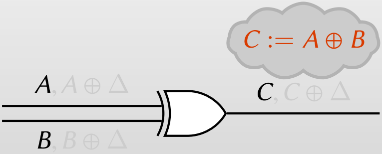
- Wire's **offset** ≡ XOR of its two labels
- Choose all wires in circuit to have same (secret) offset Δ

Free XOR [KolesnikovSchneider08]



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

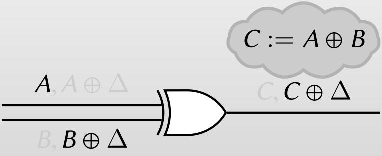
Free XOR [KolesnikovSchneider08]



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

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

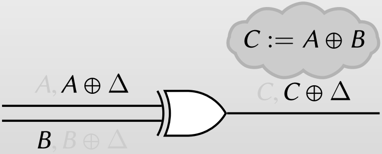
Free XOR [KolesnikovSchneider08]



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

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

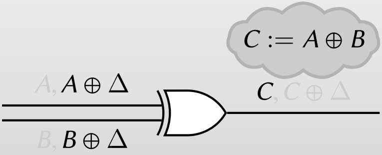
Free XOR [KolesnikovSchneider08]



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

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

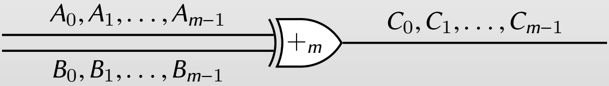
Free XOR [KolesnikovSchneider08]



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

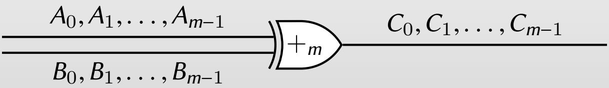
- Wire's **offset** ≡ XOR of its two labels
- Choose all wires in circuit to have same (secret) offset Δ
- Choose FALSE output label = FALSE input ⊕ FALSE input
- Evaluate by XORing input wire labels (no crypto)

Generalized Free XOR [BallMalkinRosulek16]



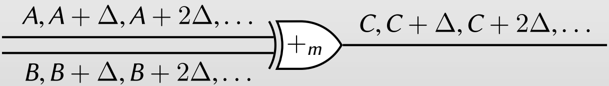
- Wires carry semantic values in $\mathbb{Z}_m$ ($m = 2$ is Free-XOR)

Generalized Free XOR [BallMalkinRosulek16]



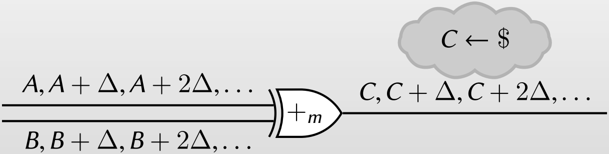
- Wires carry semantic values in $\mathbb{Z}_m$ ($m = 2$ is Free-XOR)
- Wire labels interpreted as elements of $(\mathbb{Z}_m)^\lambda$

Generalized Free XOR [BallMalkinRosulek16]



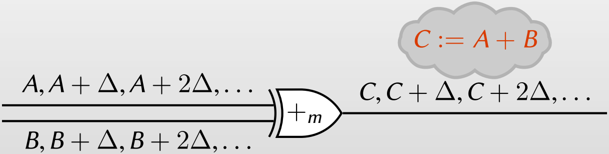
- Wires carry semantic values in $\mathbb{Z}_m$ ($m = 2$ is Free-XOR)
- Wire labels interpreted as elements of $(\mathbb{Z}_m)^\lambda$
- Wire label encoding $a \in \mathbb{Z}_m$ is $A + a\Delta$ for global $\Delta \in (\mathbb{Z}_m)^\lambda$

Generalized Free XOR [BallMalkinRosulek16]



- Wires carry semantic values in $\mathbb{Z}_m$ ($m = 2$ is Free-XOR)
- Wire labels interpreted as elements of $(\mathbb{Z}_m)^\lambda$
- Wire label encoding $a \in \mathbb{Z}_m$ is $A + a\Delta$ for global $\Delta \in (\mathbb{Z}_m)^\lambda$

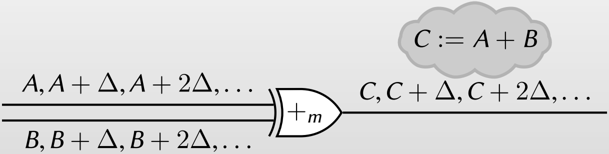
Generalized Free XOR [BallMalkinRosulek16]



$$\underbrace{A}_{\text{ZERO}} + \underbrace{B}_{\text{ZERO}} = \underbrace{A + B}_{\text{ZERO}}$$

- Wires carry semantic values in $\mathbb{Z}_m$ ($m = 2$ is Free-XOR)
- Wire labels interpreted as elements of $(\mathbb{Z}_m)^\lambda$
- Wire label encoding $a \in \mathbb{Z}_m$ is $A + a\Delta$ for global $\Delta \in (\mathbb{Z}_m)^\lambda$
- Choose ZERO output label = ZERO input $+$ ZERO input

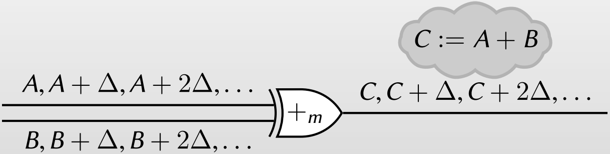
Generalized Free XOR [BallMalkinRosulek16]



$$\underbrace{A + a\Delta}_{a} + \underbrace{B + b\Delta}_{b} = \underbrace{A + B + (a+b)\Delta}_{a+b \bmod m}$$

- Wires carry semantic values in $\mathbb{Z}_m$ ($m = 2$ is Free-XOR)
- Wire labels interpreted as elements of $(\mathbb{Z}_m)^\lambda$
- Wire label encoding $a \in \mathbb{Z}_m$ is $A + a\Delta$ for global $\Delta \in (\mathbb{Z}_m)^\lambda$
- Choose ZERO output label = ZERO input + ZERO input
- **Free addition** mod m: add labels componentwise mod m

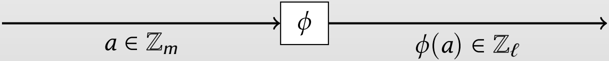
Generalized Free XOR [BallMalkinRosulek16]



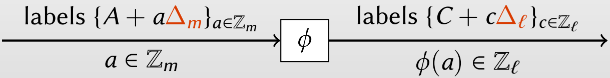
$$\underbrace{A + a\Delta}_{a} + \underbrace{B + b\Delta}_{b} = \underbrace{A + B + (a+b)\Delta}_{a+b \bmod m}$$

- Wires carry semantic values in $\mathbb{Z}_m$ ($m = 2$ is Free-XOR)
- Wire labels interpreted as elements of $(\mathbb{Z}_m)^\lambda$
- Wire label encoding $a \in \mathbb{Z}_m$ is $A + a\Delta$ for global $\Delta \in (\mathbb{Z}_m)^\lambda$
- Choose ZERO output label = ZERO input + ZERO input
- **Free addition** mod m: add labels componentwise mod m
- Free **multiplication by public constant** c, if $\gcd(c, m) = 1$

Unary gates

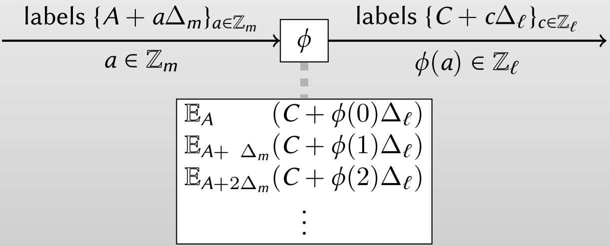


Unary gates



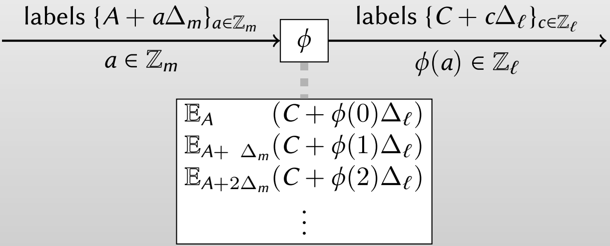
- Different “preferred modulus” on each wire $\Rightarrow$ different offsets Δ

Unary gates



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

Unary gates



- Different “preferred modulus” on each wire ⇒ different offsets Δ
- **Cost:** m ciphertexts
- $m - 1$ using simple generalization of GRR3 technique

Summary of basic garbling

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

"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

"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

"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

Note: prior work on garbling arithmetic circuits [ApplebaumIshaiKushilevitz11]

✓ *free addition, good native support for large integers*

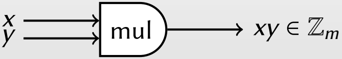
✗ *based on LWE (vs AES for us)*

Arithmetic computations

Our scheme supports free addition. What about multiplication?

Multiplication

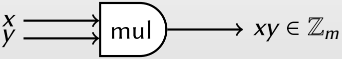
- Straight-forward approach:



Cost: m^2 ciphertexts (textbook Yao)

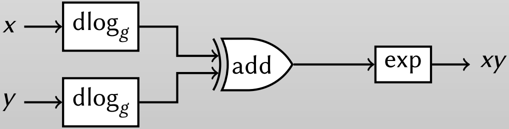
Multiplication

- Straight-forward approach:



Cost: m^2 ciphertexts (textbook Yao)

- Better approach: **log-transform** (works when m small)

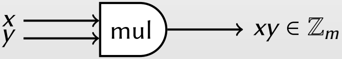


(plus extra edge cases when $0 \in \{x, y\}$)

Cost: $\sim 6m$ ciphertexts

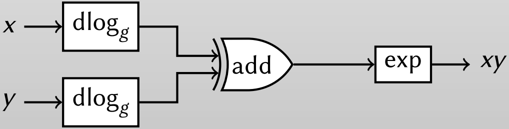
Multiplication

- Straight-forward approach:



Cost: m^2 ciphertexts (textbook Yao)

- Better approach: **log-transform** (works when m small)



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

Cost: $\sim 6m$ ciphertexts

- Best approach: **generalize half-gates** (works when m is prime)
  Cost: $2m$ ciphertexts (also in [MalkinPastroShelat])

Arithmetic computation costs

First Approach:

- Use our new garbling scheme
- Every wire is a $\mathbb{Z}_{2^{32}}$ wire

Arithmetic computation costs

First Approach:

- Use our new garbling scheme
- Every wire is a $\mathbb{Z}_{2^{32}}$ wire

	standard	this
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

$$\mathbb{Z}_{4294967296} = \mathbb{Z}_{2^{32}}$$
$$\downarrow$$
$$\mathbb{Z}_{6469693230} = \mathbb{Z}_{2\cdot 3\cdot 5\cdots}$$

Arithmetic computations

Idea: Use **many moduli** in single circuit

Arithmetic computations

Idea: Use **many moduli** in single circuit

- Represent large ints 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

Idea: Use **many moduli** in single circuit

- Represent large ints 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 each CRT residue
  - $\Rightarrow$ garbling cost = **free**

Arithmetic computations

Idea: Use **many moduli** in single circuit

- Represent large ints 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 each CRT residue
  - $\Rightarrow$ garbling cost = **free**
- To multiply mod $2 \cdot 3 \cdots 29$, just multiply each CRT residue
  - $\Rightarrow$ garbling cost $\sim 2(2 + 3 + 5 + \cdots + 29)$

Arithmetic computations

Idea: Use **many moduli** in single circuit

- Represent large ints 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 each CRT residue
  $\Rightarrow$ garbling cost = **free**
- To multiply mod $2 \cdot 3 \cdots 29$, just multiply each CRT residue
  $\Rightarrow$ garbling cost $\sim 2(2 + 3 + 5 + \cdots + 29)$
- To raise to public power, use **unary gate** $x \mapsto x^c$ on 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

High fan-in computations

Scenario

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

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

$$\mathrm{NC}^0 \subsetneq \mathrm{AC}^0 \subsetneq \mathrm{TC}^0$$

High fan-in computations

Scenario

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

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

$$\mathrm{NC}^0 \subsetneq \mathrm{AC}^0 \subsetneq \mathrm{TC}^0$$

"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.

$$\mathrm{NC}^0 \subsetneq \mathrm{AC}^0 \subsetneq \mathrm{TC}^0$$

"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

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

High fan-in computations

Insight: AND is unary function of a sum

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

High fan-in computations

Insight: AND is unary function of a sum

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

- 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**

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

- 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**

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

- 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 ⇒ 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

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

$$\downarrow$$

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

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

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

High fan-in computations

Insight: take advantage of **multiple moduli**

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

High fan-in computations

Insight: take advantage of **multiple moduli**

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

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**

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

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**

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

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 ⇒ cost = 4 (previous method)

High fan-in computations

Insight: take advantage of **multiple moduli**

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

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 ⇒ cost = 4 (previous method)

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

Summarizing the Gadgets

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 the Gadgets

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

Complications, challenges

Dealing with CRT representation

Our gadgets:

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

Dealing with CRT representation

Our gadgets:

"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 representation

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 representation

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 representation

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)

Challenges

Challenges

Problems that still **seem very hard**:

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

Challenges

Problems that still **seem very hard**:

- **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 $\mathbb{Z}_{2\cdot 3\cdot 5\cdot 7}$:

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

Comparing CRT values

CRT view of $\mathbb{Z}_{2\cdot3\cdot5\cdot7}$:

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

Theorem

CRT representation sucks for comparisons!

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
⋮	⋮

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
⋮	⋮

Comparing CRT values

CRT view of $\mathbb{Z}_{2\cdot3\cdot5\cdot7}$:

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

↓

Convert both CRT values to PMR

PMR representation of x:

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

↓

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$\rightarrow$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:

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

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:

	standard	ours
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:

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

Future Directions

1: Better comparison, conversion, division, modular reduction

2: New circuit ideas for “CRT architecture”

3: Implementation, applications

the end!