Garbled Circuits

Mike Rosulek

crypt@b-it 2018
Garbled circuits (recap)

Key idea: Given garbled gate + one wire label per input wire:
... can learn only one output label (authenticity)
... cannot learn truth value of labels (privacy)
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)
Today’s Agenda:

1. **Optimizations:** How did garbled boolean circuits get so small?

   - Beaver, Micali, Rogaway
   - Naor, Pinkas, Sumner
   - Kolesnikov, Mohassel, Rosulek
   - Kolesnikov, Schneider
   - Pinkas, Schneider, Smart, Williams
   - Zahur, Rosulek, Evans

   ![Graph showing bits per gate over years from 1986 to 2015](image)

2. **New frontiers:** How to garble arithmetic circuits

   - DES
   - SHA256
   - SHA1
   - AES
Ciphertext expansion [Yao86]

Position in this list leaks semantic value!
Ciphertext expansion [Yao86]

Position in this list leaks semantic value!
Ciphertext expansion \[\text{[Yao86]}\]

\[A_0, A_1; B_0, B_1; C_0, C_1\]

\[E_{A_0, B_0}(C_0) \quad E_{A_0, B_1}(C_1) \quad E_{A_1, B_0}(C_0) \quad E_{A_1, B_1}(C_0)\]

Position in this list leaks semantic value!

\[\Rightarrow\text{ Need to randomly permute ciphertexts}\]
Ciphertext expansion \[\text{[Yao86]}\]

\[
\begin{align*}
A_0, A_1 & \quad C_0, C_1 \\
B_0, B_1 & \\
\end{align*}
\]

\[
\begin{array}{c}
E_{A_0, B_0}(C_0) \\
E_{A_0, B_1}(C_1) \\
E_{A_1, B_0}(C_0) \\
E_{A_1, B_1}(C_0) \\
\end{array}
\]

Position in this list leaks semantic value!

⇒ Need to randomly permute ciphertexts

⇒ Need to **detect** \([\text{in}]\)correct decryption
Ciphertext expansion \[^{[Yao86]}\]

Position in this list leaks semantic value!

\[\Rightarrow\] Need to randomly permute ciphertexts

\[\Rightarrow\] Need to \textbf{detect} [in]correct decryption

\[\Rightarrow\] Need encryption scheme with \textit{ciphertext expansion} (size doubles)
Point-and-permute [BeaverMicaliRogaway90]

\[ E_{A_0, B_0}(C_0) \]
\[ E_{A_0, B_1}(C_1) \]
\[ E_{A_1, B_0}(C_0) \]
\[ E_{A_1, B_1}(C_0) \]

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!)
Can use simple one-time encryption
\[ E_{A_0, B_0}(C_0) \]
\[ E_{A_0, B_1}(C_1) \]
\[ E_{A_1, B_0}(C_0) \]
\[ E_{A_1, B_1}(C_0) \]

\( H = \) random oracle (in practice: 1 call to AES)
Point-and-permute \cite{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 \cite{BeaverMicaliRogaway90}

- Assign color bits $\bullet$ & $\bullet$ to wire labels
- Association between $(\bullet, \bullet) \leftrightarrow (T, F)$ 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 \( \bullet \) & \( \bullet \) to wire labels
- Association between \((\bullet, \bullet) \leftrightarrow (T, F)\) 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 $\bullet$ & $\bullet$ to wire labels
- Association between $(\bullet, \bullet) \leftrightarrow (T, F)$ 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
- Evaluate by decrypting ciphertext indexed by your colors

No need for trial decryption $\Rightarrow$ no need for ciphertext expansion!

- Can use simple one-time encryption $E_{A,B}(C) = H(A, B) \oplus C$
- $H = \text{random oracle (in practice: 1 call to AES)}$
Scoreboard

<table>
<thead>
<tr>
<th>Method</th>
<th>size ($\times \lambda$)</th>
<th>garble cost</th>
<th>eval cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical</td>
<td>[Yao86,GMW87]</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>P&amp;P</td>
<td>[BeaverMicaliRogaway90]</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
Garbled Row Reduction [NaorPinkasSumner99]

Instead of choosing output wire labels uniformly, choose so that first ciphertext is 0n (depends on colors & gate function). No need to include 1st ciphertext. Evaluator can "reconstruct" missing ciphertext and do the usual thing.
Garbled Row Reduction [NaorPinkasSumner99]

Instead of choosing output wire labels uniformly . . .

\[ C_0 \leftarrow \{0, 1\}^n \]
\[ C_1 \leftarrow \{0, 1\}^n \]

Instead of choosing output wire labels uniformly . . .

\[ C_0 \leftarrow \{0, 1\}^n \]
\[ C_1 \leftarrow \{0, 1\}^n \]
Garbled Row Reduction \cite{NaorPinkasSumner99}

Instead of choosing output wire labels uniformly \ldots

\ldots choose so that first ciphertext is $0^n$

(depending on colors & gate function)
Garbled Row Reduction [NaorPinkasSumner99]

Instead of choosing output wire labels uniformly . . .

. . . choose so that **first ciphertext is** $0^n$

(dependents on colors & gate function)
Garbled Row Reduction [NaorPinkasSumner99]

Instead of choosing output wire labels uniformly . . .

. . . choose so that first ciphertext is $0^n$

(depending on colors & gate function)

No need to include 1st ciphertext:
Garbled Row Reduction [NaorPinkasSumner99]

Instead of choosing output wire labels uniformly...

... choose so that first ciphertext is $0^n$

(dependents on colors & gate function)

No need to include 1st ciphertext:

- Evaluator can “reconstruct” missing ciphertext and do the usual thing:
## Scoreboard

<table>
<thead>
<tr>
<th>Method</th>
<th>size ($\times \lambda$)</th>
<th>garble cost</th>
<th>eval cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical [Yao86, GMW87]</td>
<td>8</td>
<td>4</td>
<td>2.5</td>
</tr>
<tr>
<td>P&amp;P [BeaverMicaliRogaway90]</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>GRR3 [NaorPinkasSumner99]</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
Free XOR [KolesnikovSchneider08]

- Define **offset of a wire** $\equiv$ XOR of its two labels
Free XOR [KolesnikovSchneider08]

- Define **offset of a wire** ≡ XOR of its two labels
Free XOR \cite{KolesnikovSchneider08}

- Define **offset of a wire** \(\equiv\) XOR of its two labels
- Choose all wires in circuit to have same (secret) offset \(\Delta\)
Free XOR  [KolesnikovSchneider08]

- Define **offset of a wire** ≡ XOR of its two labels
- Choose all wires in circuit to have same (secret) offset \( \Delta \)
Free XOR \[\text{[KolesnikovSchneider08]}\]

\[C := A \oplus B\]

- Define offset of a wire \(\equiv\) XOR of its two labels
- Choose all wires in circuit to have same (secret) offset \(\Delta\)
- Choose \text{FALSE} output = \text{FALSE} input \(\oplus\) \text{FALSE} input
Free XOR \[ \text{[KolesnikovSchneider08]} \]

\[
C := A \oplus B
\]

\[
\begin{align*}
A, & \quad A \oplus \Delta \\
B, & \quad B \oplus \Delta
\end{align*}
\]

\[
\begin{align*}
C, & \quad C \oplus \Delta
\end{align*}
\]

- Define **offset of a wire** \( \equiv \) XOR of its two labels
- Choose all wires in circuit to have same (secret) offset \( \Delta \)
- Choose **FALSE** output = **FALSE** input \( \oplus \) **FALSE** input
- Evaluate by **xor**ing input wire labels (no crypto)
Free XOR \cite{KolesnikovSchneider08}

\[ C := A \oplus B \]

\[ A \oplus \Delta \oplus B = A \oplus B \oplus \Delta \]

\[ \begin{array}{c}
\text{TRUE} \\
\text{FALSE} \\
\text{TRUE}
\end{array} = \begin{array}{c}
A \oplus \Delta \\
B \oplus \Delta \\
C \oplus \Delta
\end{array} \]

- Define **offset of a wire** \( \equiv \) XOR of its two labels
- Choose all wires in circuit to have same (secret) offset \( \Delta \)
- Choose **FALSE** output = **FALSE** input \( \oplus \) **FALSE** input
- Evaluate by **XORing** input wire labels (no crypto)
Free XOR \[\text{[KolesnikovSchneider08]}\]

\[C := A \oplus B\]

\[
\begin{align*}
A, & \ A \oplus \Delta \\
B, & \ B \oplus \Delta \\
\hline
C, & \ C \oplus \Delta
\end{align*}
\]

\[
A \oplus \Delta \oplus B \oplus \Delta = A \oplus B
\]
\[
\begin{align*}
\text{TRUE} & \quad \text{TRUE} & \quad \text{FALSE}
\end{align*}
\]

- Define **offset of a wire** \(\equiv\) XOR of its two labels
- Choose all wires in circuit to have same (secret) offset \(\Delta\)
- Choose **false** output = **false** input \(\oplus\) **false** input
- Evaluate by xorring input wire labels (no crypto)
## Scoreboard

<table>
<thead>
<tr>
<th></th>
<th>size ($\times \lambda$)</th>
<th>garble cost</th>
<th>eval cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XOR</td>
<td>AND</td>
<td>XOR</td>
</tr>
<tr>
<td>Classical</td>
<td>8</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>P&amp;P</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>GRR3</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td><strong>Free XOR</strong></td>
<td><strong>0</strong></td>
<td><strong>3</strong></td>
<td><strong>0</strong></td>
</tr>
</tbody>
</table>

References:
- [Yao86,GMW87]
- [BeaverMicaliRogaway90]
- [NaorPinkasSumner99]
- [KolesnikovSchneider08]
Row reduction $\times 2$ [GueronLindellNofPinkas15]

Instead of choosing output wire labels uniformly, choose them so that . . .

Note: (More complicated) 2-ctxt AND first appeared in [PinkasSchneiderSmartWilliams09].
Row reduction $\times 2$ \cite{GueronLindellNofPinkas15}

Instead of choosing output wire labels uniformly, choose them so that . . .

\[
C_0 \leftarrow \{0, 1\}^n \\
C_1 \leftarrow \{0, 1\}^n
\]

Note: (More complicated) 2-ctxt AND first appeared in \cite{PinkasSchneiderSmartWilliams09}. 

\[
\begin{align*}
H(A_0, B_1) \oplus C_1^* \\
H(A_0, B_0) \oplus C_0^* \\
H(A_1, B_1) \oplus C_0^* \\
H(A_1, B_0) \oplus C_0^*
\end{align*}
\]
Row reduction $\times 2$ [GueronLindellNofPinkas15]

Instead of choosing output wire labels uniformly, choose them so that . . .

. . . first ciphertext is $0^n$

\[
C_0 \leftarrow \{0, 1\}^n
\]
\[
C_1 = H(A_0, B_1)
\]

\[
\begin{align*}
H(A_0, B_1) \oplus C_1 &\leftarrow 0^\lambda \\
H(A_0, B_0) \oplus C_0 &
\end{align*}
\]

\[
\begin{align*}
H(A_1, B_1) \oplus C_0 &
\end{align*}
\]

\[
\begin{align*}
H(A_1, B_0) \oplus C_0 &
\end{align*}
\]

Note: (More complicated) 2-ctxt AND first appeared in [PinkasSchneiderSmartWilliams09].
Instead of choosing output wire labels uniformly, choose them so that:

- First ciphertext is $0^n$.
- XOR of other ciphertexts is $0^n$.

Note: (More complicated) 2-ctxt AND first appeared in [PinkasSchneiderSmartWilliams09].
Row reduction $\times 2$  

Instead of choosing output wire labels uniformly, choose them so that . . .

. . . first ciphertext is $0^n$

. . . XOR of other ciphertexts is $0^n$

First 2 ciphertexts don’t need to be sent!

Note: (More complicated) 2-ctxt AND first appeared in [PinkasSchneiderSmartWilliams09].
# Scoreboard

<table>
<thead>
<tr>
<th></th>
<th>size ($\times \lambda$)</th>
<th>garble cost</th>
<th>eval cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XOR</td>
<td>AND</td>
<td>XOR</td>
</tr>
<tr>
<td>Classical [Yao86,GMW87]</td>
<td>8</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>P&amp;P [BeaverMicaliRogaway90]</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>GRR3 [NaorPinkasSumner99]</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Free XOR [KolesnikovSchneider08]</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>GRR2 [PinkasSchneiderSmartWilliams09]</td>
<td><strong>2</strong></td>
<td><strong>2</strong></td>
<td><strong>2</strong></td>
</tr>
</tbody>
</table>
## Scoreboard

<table>
<thead>
<tr>
<th>Method</th>
<th>size (×(\lambda))</th>
<th>garble cost</th>
<th>eval cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XOR</td>
<td>AND</td>
<td>XOR</td>
</tr>
<tr>
<td>Classical [Yao86,GMW87]</td>
<td>8</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>P&amp;P [BeaverMicaliRogaway90]</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>GRR3 [NaorPinkasSumner99]</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Free XOR [KolesnikovSchneider08]</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>GRR2 [PinkasSchneiderSmartWilliams09]</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

- Depending on circuit, either Free-XOR or GRR2 may be better.
- Two techniques are **incompatible**! (can’t guarantee \(C_0 \oplus C_1 = \Delta\))
Samee Zahur, Mike Rosulek, David Evans: 
Two Halves Make a Whole: Reducing Data Transfer in Garbled Circuits using Half Gates. Eurocrypt 2015

Best of both worlds: Free-XOR + 2-ciphertext AND
What if garbler knows in advance the truth value on one input wire?
What if garbler knows in advance the truth value on one input wire?
What if garbler knows in advance the truth value on one input wire?

If $a = 0$:

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

unary gate $b \mapsto 0$
What if garbler knows in advance the truth value on one input wire?

if $a = 0$:

\[
\begin{array}{c|c}
B & C \\
B \oplus \Delta & C \\
\end{array}
\]

unary gate $b \mapsto 0$
What if garbler knows in advance the truth value on one input wire?

If $a = 0$:

- $H(B) \oplus C$
- $H(B \oplus \Delta) \oplus C$

Unary gate $b \mapsto 0$
What if garbler knows in advance the truth value on one input wire?

A \oplus \Delta

B, B \oplus \Delta

C, C \oplus \Delta
What if garbler knows in advance the truth value on one input wire?

If $a = 1$:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Unary gate $b \mapsto b$
What if garbler knows in advance the truth value on one input wire?

A. \( A \oplus \Delta \)  

B. \( B \oplus \Delta \)  

C. \( C \oplus \Delta \)  

if \( a = 1 \): 

\[ \begin{array}{cc} 
B & C \\
B \oplus \Delta & C \oplus \Delta 
\end{array} \]

unary gate \( b \mapsto b \)
What if garbler knows in advance the truth value on one input wire?

A, $A \oplus \Delta$

B, $B \oplus \Delta$

C, $C \oplus \Delta$

If $a = 1$:

$H(B) \oplus C$

$H(B \oplus \Delta) \oplus C \oplus \Delta$

Unary gate $b \mapsto b$
What if garbler knows in advance the truth value on one input wire?

If $a = 0$:
- $H(B) \oplus C$
- $H(B \oplus \Delta) \oplus C$

Unary gate $b \mapsto 0$

If $a = 1$:
- $H(B) \oplus C$
- $H(B \oplus \Delta) \oplus (C \oplus \Delta)$

Unary gate $b \mapsto b$
What if garbler knows in advance the truth value on one input wire?

A, A ⊕ Δ
B, B ⊕ Δ

C, C ⊕ Δ

H(B) ⊕ C
H(B ⊕ Δ) ⊕ C ⊕ aΔ
What if garbler knows in advance the truth value on one input wire?

\[ C \leftarrow \{0, 1\}^n \]

\[ H(B) \oplus C \]

\[ H(B \oplus \Delta) \oplus C \oplus a\Delta \]
Half Gates [ZahurRosulekEvans15]

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

\[
C := H(B)
\]

\[
\begin{align*}
A, A & \oplus \triangle \\
B, B & \oplus \triangle
\end{align*}
\]

\[
\begin{align*}
H(B) \oplus C \\
H(B \oplus \triangle) \oplus C & \oplus a\triangle
\end{align*}
\]

Fine print: permute ciphertexts with permute-and-point.
What if garbler knows in advance the truth value on one input wire?

\[ C := H(B) \]

\[ A, A \oplus \Delta \]

\[ B, B \oplus \Delta \]

\[ C, C \oplus \Delta \]

\[ 0^n \]

\[ H(B \oplus \Delta) \oplus C \oplus a\Delta \]
What if garbler knows in advance the truth value on one input wire?

\[ C := H(B) \]

\[ H(B \oplus \Delta) \oplus C \oplus a\Delta \]

Fine print: permute ciphertexts with permute-and-point.
What if \textbf{evaluator} knows in advance the truth value on one input wire?
What if \textbf{evaluator} knows in advance the truth value on one input wire?

\[ A, A \oplus \Delta \]
\[ B, B \oplus \Delta \]
\[ C, C \oplus \Delta \]
What if **evaluator** knows in advance the truth value on one input wire?

Evaluator has $B$ (knows **FALSE**):

$\Rightarrow$ should obtain $C$ (**FALSE**)
What if **evaluator** knows in advance the truth value on one input wire?

Evaluator has $B$ (knows FALSE):

$\Rightarrow$ should obtain $C$ (FALSE)
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 $B \oplus \triangle$ (knows TRUE):

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

Suffices to learn $A$ and $C$.
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
What if evaluator knows in advance the truth value on one input wire?

Evaluator has $B$ (knows FALSE):

$⇒$ should obtain $C$ (FALSE)

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

$⇒$ should be able to transfer truth value from “$a$” wire to “$c$” wire

$⇒$ Suffices to learn $A ⊕ C$
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 $B \oplus \triangle$ (knows TRUE):

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

$\triangleleft$ Suffices to learn $A \oplus C$
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 $A \oplus C$
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 $A \oplus C$
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 $A \oplus C$
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 $A \oplus C$
What if evaluator knows in advance the truth value on one input wire?

Evaluator has $B$ (knows FALSE):

$\implies$ should obtain $C$ (FALSE)

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

$\implies$ should be able to transfer truth value from “$a$” wire to “$c$” wire

$\quad$ Suffices to learn $A \oplus C$

$C := H(B)$
What if **evaluator** knows in advance the truth value on one input wire?

Evaluator has $B$ (knows FALSE):

$\implies$ should obtain $C$ (FALSE)

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

$\implies$ 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

$\quad$ Suffices to learn $A \oplus C$

Fine print: no need for permute-and-point here
Two halves make a whole!

\[ a \land b \]
Two halves make a whole!

\[ a \land b = (a \oplus r \oplus r) \land b \]

- Garbler chooses random bit \( r \)
Two halves make a whole!

\[ a \land b = (a \oplus r \oplus r) \land b \]

\[ = [(a \oplus r) \land b] \oplus [r \land b] \]

- Garbler chooses random bit \( r \)
Two halves make a whole!

\[ a \land b = (a \oplus r \oplus r) \land b = [(a \oplus r) \land b] \oplus [r \land b] \]

- Garbler chooses random bit \( r \)
- Arrange for evaluator to learn \( a \oplus r \) in the clear
Two halves make a whole!

\[
a \land b = (a \oplus r \oplus r) \land b \\
= \left[ (a \oplus r) \land b \right] \oplus [r \land b]
\]

one input known to evaluator

- Garbler chooses random bit \( r \)
- Arrange for evaluator to learn \( a \oplus r \) in the clear
Two halves make a whole!

\[
a \land b = (a \oplus r \oplus r) \land b \\
= [(a \oplus r) \land b] \oplus [r \land b]
\]

one input known to garbler

- Garbler chooses random bit \( r \)
- Arrange for evaluator to learn \( a \oplus r \) in the clear
Two halves make a whole!

\[ a \land b = (a \oplus r \oplus r) \land b \]

\[ = [(a \oplus r) \land b] \oplus [r \land b] \]

one input known to garbler

- 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!

\[ a \land b = (a \oplus r \oplus r) \land b \]
\[ = [(a \oplus r) \land b] \oplus [r \land b] \]

one input known to garbler

- 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

<table>
<thead>
<tr>
<th>Method</th>
<th>size ($\times \lambda$)</th>
<th>garble cost</th>
<th>eval cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical [Yao86,GMW87]</td>
<td>8 8</td>
<td>4 4</td>
<td>2.5 2.5</td>
</tr>
<tr>
<td>P&amp;P [BeaverMicaliRogaway90]</td>
<td>4 4</td>
<td>4 4</td>
<td>1 1</td>
</tr>
<tr>
<td>GRR3 [NaorPinkasSumner99]</td>
<td>3 3</td>
<td>4 4</td>
<td>1 1</td>
</tr>
<tr>
<td>Free XOR [KolesnikovSchneider08]</td>
<td>0 3</td>
<td>0 4</td>
<td>0 1</td>
</tr>
<tr>
<td>GRR2 [PinkasSchneiderSmartWilliams09]</td>
<td>2 2</td>
<td>2 2</td>
<td>1 1</td>
</tr>
<tr>
<td>Half gates [ZahurRosulekEvans15]</td>
<td>0 2</td>
<td>0 4</td>
<td>0 2</td>
</tr>
</tbody>
</table>
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
Can we do better than half-gates?

**NO**  
[ZahurRosulekEvans15]

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

... using “standard techniques”

**YES**  
[BallMalkinRosulek16, KempkaKikuchiSuzuki16]

Can garble an AND gate with 1 ciphertext
Can we do better than half-gates?

**NO**

[ZahurRosulekEvans15]

Can’t garble an AND gate with $< 2$ ciphertexts. ...

... using “standard techniques”

---

**YES**

[BallMalkinRosulek16, KempkaKikuchiSuzuki16]

Can garble an AND gate with 1 ciphertext. ...

... but not in context of a larger circuit 😞
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”

YES
[BallMalkinRosulek16, KempkaKikuchiSuzuki16]

Can garble an AND gate with 1 ciphertext... 

... but not in context of a larger circuit 😞
Roadmap

1

**Optimizations**: How did garbled boolean circuits get so small?

2

**New frontiers**: How to garble *arithmetic* circuits
Roadmap

1. **Optimizations:** How did garbled boolean circuits get so small?

2. **New frontiers:** How to garble *arithmetic* circuits
   
   [BallMalkinRosulek16]
Generalized Free XOR [BallMalkinRosulek16]

**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 [BallMalkinRosulek16]

**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 label encoding truth value \( a \in \mathbb{Z}_m \) is
\( a + \Delta \mod m \).
Generalized Free XOR [BallMalkinRosulek16]

Free XOR:
Wire carries a truth value from \(\{0, 1\}\)

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

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

\text{false} wire label is \(A\)
\text{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 [BallMalkinRosulek16]

**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 \( A + a \Delta \in (\mathbb{Z}_m)^\lambda \)
Generalized Free XOR

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

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

![Diagram](attachment:diagram.png)

Evaluator can simply add wire labels
Generalized Free XOR

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

Evaluator 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 $A + a\Delta \in (\mathbb{Z}_m)^\lambda$

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

- Free multiplication by public constant $c$, if $\text{gcd}(c, m) = 1$
Garbling unary gates

![Diagram of unary gates with truth values in \( \mathbb{Z}_m \) and \( \mathbb{Z}_\ell \)]

- Truth value \( \phi \) maps from \( \mathbb{Z}_m \) to \( \mathbb{Z}_\ell \).
  - Table:
    - \( 0 \rightarrow \phi(0) \)
    - \( 1 \rightarrow \phi(1) \)
    - \( 2 \rightarrow \phi(2) \)
    - \( \vdots \)

### Cost:
- \( m \) ciphertexts using standard row reduction.

### Different "preferred modulus" on each wire:
- Different offsets.

### Generalized point-and-permute:
- "Color bit" from \( \mathbb{Z}_m \).
Garbling unary gates

Different “preferred modulus” on each wire ⇒ different offsets $\Delta$
Garbling unary gates

Different “preferred modulus” on each wire ⇒ different offsets $\Delta$
Garbling unary gates

- Different “preferred modulus” on each wire ⇒ different offsets $\Delta$
- Cost: $m$ ciphertexts
- Generalized point-and-permute: “color bit” from $\mathbb{Z}_m$
Garbling unary gates

\[ \text{labels } \{ A + a\Delta_m \}_{a \in \mathbb{Z}_m} \xrightarrow{\phi} \text{labels } \{ C + c\Delta_\ell \}_{c \in \mathbb{Z}_\ell} \]

- Different “preferred modulus” on each wire ⇒ different offsets \( \Delta \)
- Cost: \( m \) ciphertexts (\( m - 1 \) using standard row reduction)
- Generalized point-and-permute: “color bit” from \( \mathbb{Z}_m \)

\[
\begin{align*}
H(A) + C + \phi(0)\Delta_\ell \\
H(A + \Delta_m) + C + \phi(1)\Delta_\ell \\
H(A + 2\Delta_m) + C + \phi(2)\Delta_\ell \\
\vdots
\end{align*}
\]
Generalized garbling tools

We can efficiently garble any computation/circuit where:

- Each wire has a preferred modulus $\mathbb{Z}_m$
  - Wire-label-offset $\Delta_m$ global to all $\mathbb{Z}_m$-wires

- Addition gates: all wires touching gate have same modulus
  - Garbling cost: **free**

- Mult-by-constant gates: input/output wires have same modulus
  - Garbling cost: **free**

- Unary gates: $\mathbb{Z}_m$ input and $\mathbb{Z}_\ell$ output
  - Garbling cost: $m - 1$ ciphertexts
Generalized garbling tools

We can efficiently garble any computation/circuit where:

- Each wire has a preferred modulus $\mathbb{Z}_m$
  - Wire-label-offset $\Delta_m$ global to all $\mathbb{Z}_m$-wires

- Addition gates: all wires touching gate have same modulus
  - Garbling cost: free

- Mult-by-constant gates: input/output wires have same modulus
  - Garbling cost: free

- Unary gates: $\mathbb{Z}_m$ input and $\mathbb{Z}_\ell$ output
  - Garbling cost: $m - 1$ ciphertexts

Better basis for many computations than traditional boolean circuits!
### Example Scenario

Securely compute linear optimization problem on 32-bit values.

Almost all operations are **addition**, **multiplication**, etc.
Example Scenario

Securely compute linear optimization problem on 32-bit values.
⇒ 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.

⇒ 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)

<table>
<thead>
<tr>
<th></th>
<th>cost (# ciphertexts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>addition</td>
<td>62</td>
</tr>
<tr>
<td>multiplication by public constant</td>
<td>758</td>
</tr>
<tr>
<td>multiplication</td>
<td>1200</td>
</tr>
<tr>
<td>squaring, cubing, etc</td>
<td>1864</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th></th>
<th>standard</th>
<th>ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>addition</td>
<td>62</td>
<td>0</td>
</tr>
<tr>
<td>multiplication by public constant</td>
<td>758</td>
<td>0</td>
</tr>
<tr>
<td>multiplication</td>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>squaring, cubing, etc</td>
<td>1864</td>
<td></td>
</tr>
</tbody>
</table>
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 $2m - 2$ ciphertexts (generalization of half-gates)

<table>
<thead>
<tr>
<th></th>
<th>standard</th>
<th>ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>addition</td>
<td>62</td>
<td>0</td>
</tr>
<tr>
<td>multiplication by public constant</td>
<td>758</td>
<td>0</td>
</tr>
<tr>
<td>multiplication</td>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>squaring, cubing, etc</td>
<td>1864</td>
<td></td>
</tr>
</tbody>
</table>
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 $2m - 2$ ciphertexts (generalization of half-gates)

<table>
<thead>
<tr>
<th></th>
<th>standard</th>
<th>ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>addition</td>
<td>62</td>
<td>0</td>
</tr>
<tr>
<td>multiplication by public constant</td>
<td>758</td>
<td>0</td>
</tr>
<tr>
<td>multiplication</td>
<td>1200</td>
<td>8589934590</td>
</tr>
<tr>
<td>squaring, cubing, etc</td>
<td>1864</td>
<td>4294967295</td>
</tr>
</tbody>
</table>
Arithmetic computations

instead of \( \mathbb{Z}_{4294967296} \)

\[ \downarrow \]

use \( \mathbb{Z}_{6469693230} \)
Arithmetic computations

instead of \( \mathbb{Z}_{4294967296} = \mathbb{Z}_{2^{32}} \)

\[ \downarrow \]

use \( \mathbb{Z}_{6469693230} \)
Arithmetic computations

instead of \( \mathbb{Z}_{4294967296} = \mathbb{Z}_{2^{32}} \)

\[ \quad \downarrow \]

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**)

<table>
<thead>
<tr>
<th></th>
<th>standard</th>
<th>madness</th>
</tr>
</thead>
<tbody>
<tr>
<td>addition</td>
<td>62</td>
<td>0</td>
</tr>
<tr>
<td>mult by public constant</td>
<td>758</td>
<td>0</td>
</tr>
<tr>
<td>multiplication</td>
<td>1200</td>
<td>25769803776</td>
</tr>
<tr>
<td>squaring, cubing, etc</td>
<td>1864</td>
<td>4294967296</td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th></th>
<th>standard</th>
<th>madness</th>
<th>CRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>addition</td>
<td>62</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mult by public constant</td>
<td>758</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>multiplication</td>
<td>1200</td>
<td>25769803776</td>
<td>238 ≈ 2(2 + 3 + 5 +</td>
</tr>
</tbody>
</table>
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??

Not so good:
- Converting from binary to CRT
- Getting CRT values into the circuit via OT

Kinda bad: (room for improvement)
- **Comparing** two CRT-encoded values
- Converting from CRT to binary
- Integer division
- Modular reduction different than the CRT composite modulus (e.g., garbled RSA)
Claim:

It’s **not hard** to convert into CRT representation $\mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2} \times \cdots \times \mathbb{Z}_{p_k}$.
Converting to 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) \( \times \) (# bits) (e.g., 320 ciphertexts for 32 bits)
Converting to 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 $b_nb_{n-1} \cdots b_1b_0$:
- For all $i, j$, use unary gate $b_i \mapsto b_i \pmod{p_j}$ (1 ciphertext each)
- For all $j$, add to obtain $\sum_i b_i2^i \pmod{p_j}$ (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)
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 |
| ... | ... |
Comparing CRT values

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

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Theorem**

CRT representation sucks for comparisons!
Comparing CRT values

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

|       | 0 0 0 0 | 0 0 | 1 1 1 1 | 1 1 | 2 2 2 0 | 2 0 2 0 | 3 3 0 1 | 3 1 3 1 | 4 4 1 0 | 4 2 1 0 | 5 0 2 1 | 5 2 1 5 | 6 1 0 0 | 6 1 0 0 | 0 2 1 1 | 7 1 0 1 | 7 1 0 1 | 1 4 2 1 | 29 4 2 1 | 29 1 0 0 | 30 1 0 0 | 30 1 0 0 |
|-------|---------|-----|---------|-----|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|-------------|---------|---------|---------|---------|
|       | 0       | 0   | 1       | 1   | 2       | 2       | 3       | 3       | 4       | 4       | 5       | 5       | 6       | 6       | 7       | 7       | 29      | 29       | 29       | 30       | 30       | 30       |
## Comparing CRT values

**CRT view of \( \mathbb{Z}_{2\cdot3\cdot5\cdot7} \):**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2222</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3301</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4410</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5021</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6100</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0211</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1421</td>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Primorial Mixed Radix (PMR):**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>421</td>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Approach for comparisons

CRT values given

Convert both CRT values to PMR

\[\left\lfloor x_p \right\rfloor \% q; \left\lfloor x_q \right\rfloor \% p\]

allows you to compute PMR representation of \( x \):

\[\left\lfloor x \right\rfloor \% 7; \left\lfloor x \right\rfloor \% 5; \left\lfloor x \right\rfloor \% 3; \left\lfloor x \right\rfloor \% 2\]

Compare PMR (simple L→R scan)
Approach for comparisons

CRT values given

↓

Convert both CRT values to PMR

PMR representation of x:

\[ \ldots, \left\lfloor \frac{x}{2 \cdot 3 \cdot 5} \right\rfloor \mod 7, \left\lfloor \frac{x}{2 \cdot 3} \right\rfloor \mod 5, \left\lfloor \frac{x}{2} \right\rfloor \mod 3, \left\lfloor x \right\rfloor \mod 2 \]

↓

Compare PMR (simple L→R scan)
Approach for comparisons

CRT values given

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 \(x\):

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

Compare PMR (simple L→R scan)
\[(x \% p, x \% q) \mapsto \left\lfloor \frac{x}{p} \right\rfloor \% q\]

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>x % 3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>x % 5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>\left\lfloor \frac{x}{3} \right\rfloor % 5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
\((x \% p, \ x \% q) \mapsto \left\lfloor \frac{x}{p} \right\rfloor \% q\)

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x % 3)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x % 5)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>(x % 3 - x % 5)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-3</td>
<td>-3</td>
<td>2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-4</td>
<td>1</td>
<td>1</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
</tbody>
</table>

\left\lfloor \frac{x}{3} \right\rfloor \% 5

| \left\lfloor \frac{x}{3} \right\rfloor \% 5 | 0  | 0  | 0  | 1  | 1  | 1  | 2  | 2  | 2  | 3  | 3  | 3  | 4  | 4  | 4  |

1. Subtract \(x \% 3 - x \% 5\)
\( (x \% p, x \% q) \mapsto \lfloor x/p \rfloor \% q \)

<table>
<thead>
<tr>
<th>( x )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x % 3 )</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( x % 5 )</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( x % 3 - x % 5 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-3</td>
<td>-3</td>
<td>2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-4</td>
<td>1</td>
<td>1</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
</tbody>
</table>

\[ \lfloor x/3 \rfloor \% 5 \]

| \( \lfloor x/3 \rfloor \% 5 \) | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 | 4 | 4 | 4 |

1. Subtract \( x \% 3 - x \% 5 \)

2. Result has the same “constant segments” as what we want
\[(x \% p, x \% q) \mapsto \lfloor x/p \rfloor \% q\]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(x % 3)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x % 5)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>(x % 3 - x % 5)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-3</td>
<td>-3</td>
<td>2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-4</td>
<td>1</td>
<td>1</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>((x % 3 - x % 5) % 7)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>(\lfloor x/3 \rfloor % 5)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

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

2. Result has the same “constant segments” as what we want
\[(x \% p, \ x \% q) \mapsto \left\lfloor x/p \right\rfloor \% q\]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x % 3)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x % 5)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>((x % 3 - x % 5) % 7)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-3</td>
<td>-3</td>
<td>2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-4</td>
<td>1</td>
<td>1</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>([x/3] % 5)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

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
\[(x \% p, x \% q) \leftrightarrow \left\lfloor \frac{x}{p} \right\rfloor \% q\]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>(x % 3)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x % 5)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>(x % 3 - x % 5)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-3</td>
<td>-3</td>
<td>2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-4</td>
<td>1</td>
<td>1</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>((x % 3 - x % 5) % 7)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>([x/3] % 5)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

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:
     
     \[
     \begin{align*}
     0 & \mapsto 0 \quad 2 \mapsto 1 \quad 4 \mapsto 1 \quad 6 \mapsto 2 \\
     1 & \mapsto 3 \quad 3 \mapsto 3 \quad 5 \mapsto 4
     \end{align*}
     \]
Approach for comparisons

1. General \((x \% p, x \% q) \mapsto \lfloor x/p \rfloor \% q\) 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 \( (x \% p, x \% q) \mapsto \lfloor x/p \rfloor \% q \) 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:

<table>
<thead>
<tr>
<th>Operation</th>
<th>boolean</th>
<th>CRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>addition</td>
<td>62</td>
<td>0</td>
</tr>
<tr>
<td>multiplication by public constant</td>
<td>758</td>
<td>0</td>
</tr>
<tr>
<td>multiplication</td>
<td>1200</td>
<td>238</td>
</tr>
<tr>
<td>squaring, cubing, etc</td>
<td>1864</td>
<td>119</td>
</tr>
<tr>
<td>comparison</td>
<td>64</td>
<td>2541</td>
</tr>
</tbody>
</table>