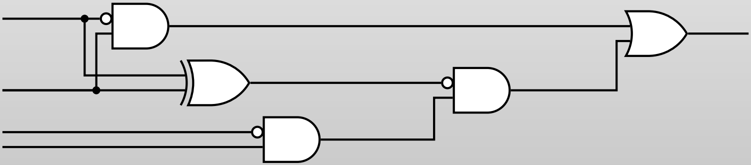
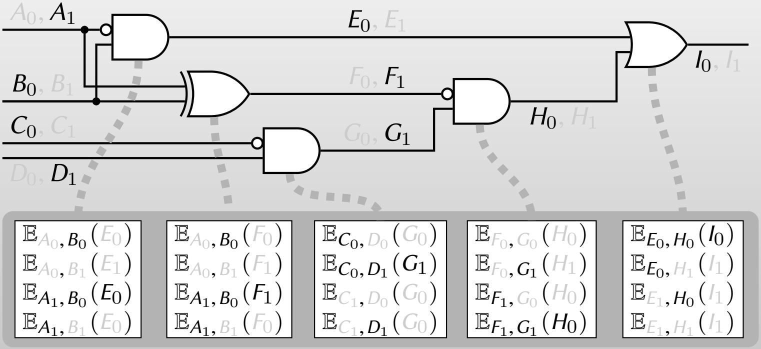
Garbled Circuits

Mike Rosulek
Oregon State University OSU

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 . . .

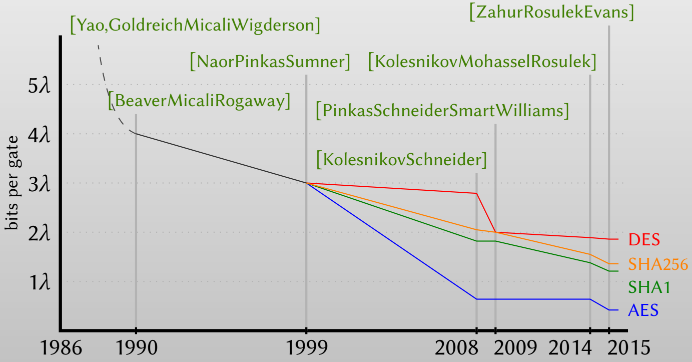
. . . is the most important parameter

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

Today's Agenda:

1

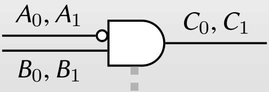
Optimizations: How did garbled boolean circuits get so small?



2

New frontiers: How to garble *arithmetic* circuits

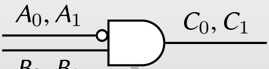
Ciphertext expansion [Yao86]



$$\begin{array}{|l|}\hline \mathbb{E}_{A_0,B_0}(C_0) \\ \mathbb{E}_{A_0,B_1}(C_1) \\ \mathbb{E}_{A_1,B_0}(C_0) \\ \mathbb{E}_{A_1,B_1}(C_0) \\ \hline \end{array}$$

Position in this list leaks semantic value!

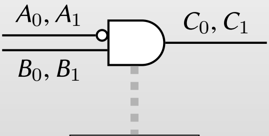
Ciphertext expansion [Yao86]



$$\begin{array}{l} \mathbb{E}_{A_0, B_0}(C_0) \\ \mathbb{E}_{A_0, B_1}(C_1) \\ \mathbb{E}_{A_1, B_0}(C_0) \\ \mathbb{E}_{A_1, B_1}(C_0) \end{array}$$

Position in this list leaks semantic value!

Ciphertext expansion [Yao86]

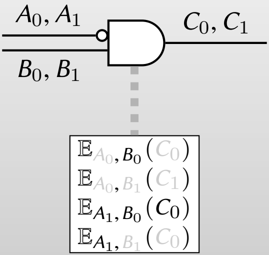


$$
\begin{array}{l}
\mathbb{E}_{A_0, B_0}(C_0) \\
\mathbb{E}_{A_0, B_1}(C_1) \\
\mathbb{E}_{A_1, B_0}(C_0) \\
\mathbb{E}_{A_1, B_1}(C_0)
\end{array}
$$

Position in this list leaks semantic value!

$\Longrightarrow$ Need to randomly permute ciphertexts

Ciphertext expansion [Yao86]

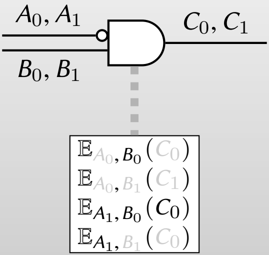


Position in this list leaks semantic value!

$\Rightarrow$ Need to randomly permute ciphertexts

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

Ciphertext expansion [Yao86]



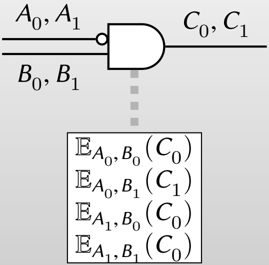
Position in this list leaks semantic value!

$\Rightarrow$ Need to randomly permute ciphertexts

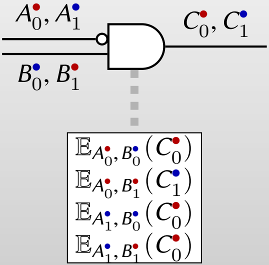
$\Rightarrow$ Need to **detect** [in]correct decryption

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

Point-and-permute [BeaverMicaliRogaway90]

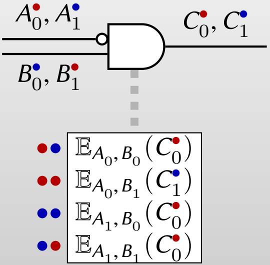


Point-and-permute [BeaverMicaliRogaway90]



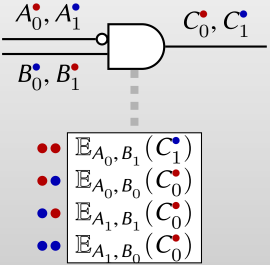
- Assign color bits $\textcolor{red}{\bullet}$ & $\textcolor{blue}{\bullet}$ to wire labels
- Association between $(\textcolor{red}{\bullet}, \textcolor{blue}{\bullet}) \leftrightarrow (T, F)$ is random for each wire
- A wire label reveals its own color (e.g., as last bit)

Point-and-permute [BeaverMicaliRogaway90]



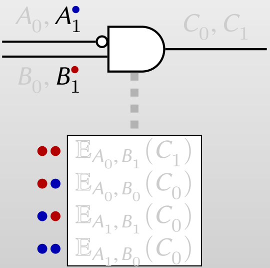
- Assign color bits • & • to wire labels
- Association between (•, •) ↔ (T, F) is random for each wire
- A wire label reveals its own color (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys

Point-and-permute [BeaverMicaliRogaway90]



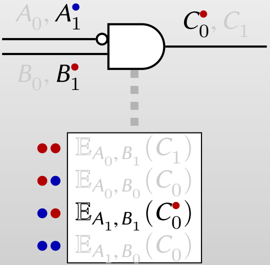
- Assign color bits • & • to wire labels
- Association between (•, •) $\leftrightarrow$ (T, F) is random for each wire
- A wire label reveals its own color (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys

Point-and-permute [BeaverMicaliRogaway90]



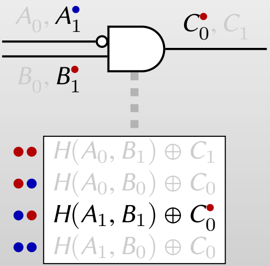
- Assign color bits • & • to wire labels
- Association between (•, •) ↔ (T, F) is random for each wire
- A wire label reveals its own color (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys
- Evaluate by decrypting ciphertext indexed by your colors

Point-and-permute [BeaverMicaliRogaway90]



- Assign color bits • & • to wire labels
- Association between (•, •) $\leftrightarrow$ (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

Point-and-permute [BeaverMicaliRogaway90]



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

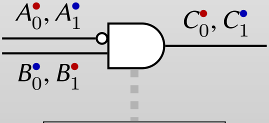
No need for trial decryption $\Rightarrow$ no need for ciphertext expansion!

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

Scoreboard

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

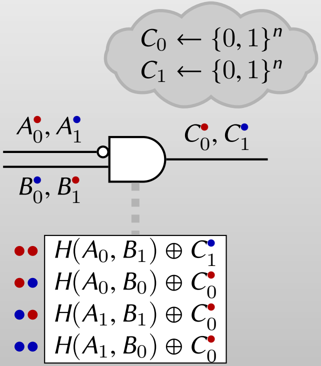
Garbled Row Reduction [NaorPinkasSumner99]



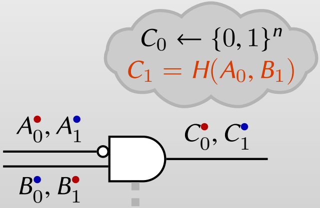
$$H(A_0, B_1) \oplus C_1$$
$$H(A_0, B_0) \oplus C_0$$
$$H(A_1, B_1) \oplus C_0$$
$$H(A_1, B_0) \oplus C_0$$

Garbled Row Reduction [NaorPinkasSumner99]

Instead of choosing output wire labels uniformly . . .



Garbled Row Reduction [NaorPinkasSumner99]



$$\boxed{\begin{array}{l} H(A_0, B_1) \oplus C_1 \\ H(A_0, B_0) \oplus C_0 \\ H(A_1, B_1) \oplus C_0 \\ H(A_1, B_0) \oplus C_0 \end{array}}$$

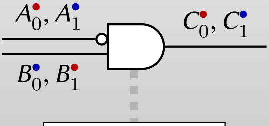
Instead of choosing output wire labels uniformly …

… choose so that first ciphertext is 0^n

(depends on colors & gate function)

Garbled Row Reduction [NaorPinkasSumner99]

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



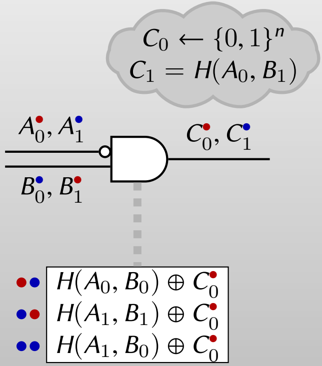
0^n
$H(A_0, B_0) \oplus C_0$
$H(A_1, B_1) \oplus C_0$
$H(A_1, B_0) \oplus C_0$

Instead of choosing output wire labels uniformly …

… choose so that first ciphertext is 0^n

(depends on colors & gate function)

Garbled Row Reduction [NaorPinkasSumner99]



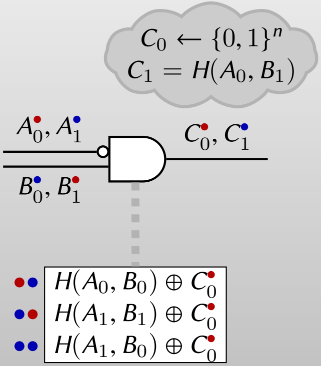
Instead of choosing output wire labels uniformly . . .

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

(depends on colors & gate function)

No need to include 1st ciphertext:

Garbled Row Reduction [NaorPinkasSumner99]



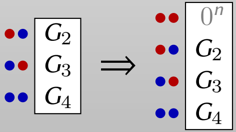
Instead of choosing output wire labels uniformly . . .

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

(depends on colors & gate function)

No need to include 1st ciphertext:

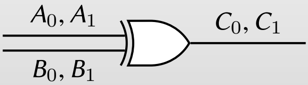
- Evaluator can “reconstruct” missing ciphertext and do the usual thing:



Scoreboard

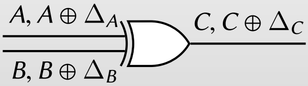
	size ($\times\lambda$)	garble cost	eval cost
Classical [Yao86,GMW87]	8	4	2.5
P&P [BeaverMicaliRogaway90]	4	4	1
GRR3 [NaorPinkasSumner99]	**3**	**4**	**1**

Free XOR [KolesnikovSchneider08]



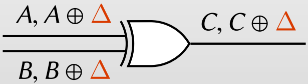
- Define **offset of a wire** ≡ XOR of its two labels

Free XOR [KolesnikovSchneider08]



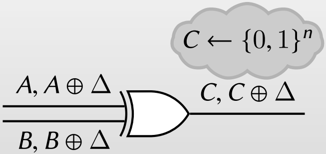
- Define **offset of a wire** ≡ XOR of its two labels

Free XOR [KolesnikovSchneider08]



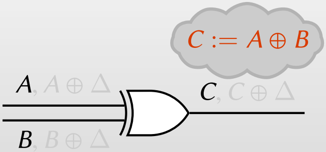
- Define **offset of a wire** ≡ XOR of its two labels
- Choose all wires in circuit to have same (secret) offset Δ

Free XOR [KolesnikovSchneider08]



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

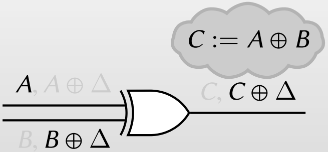
Free XOR [KolesnikovSchneider08]



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

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

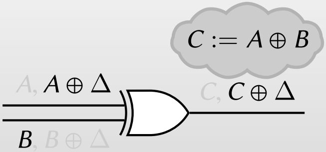
Free XOR [KolesnikovSchneider08]



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

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

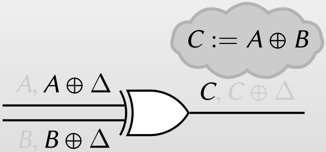
Free XOR [KolesnikovSchneider08]



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

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

Free XOR [KolesnikovSchneider08]



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

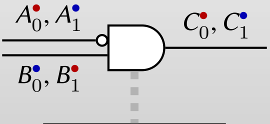
- Define **offset of a wire** ≡ XOR of its two labels
- Choose all wires in circuit to have same (secret) offset Δ
- Choose FALSE output = FALSE input $\oplus$ FALSE input
- Evaluate by XORing input wire labels (no crypto)

Scoreboard

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

Row reduction ×2 [GueronLindellNofPinkas15]

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



$H(A_0, B_1) \oplus C_1$

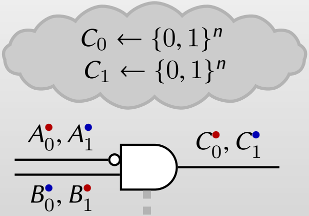
$H(A_0, B_0) \oplus C_0$

$H(A_1, B_1) \oplus C_0$

$H(A_1, B_0) \oplus C_0$

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

Row reduction ×2 [GueronLindellNofPinkas15]



$H(A_0, B_1) \oplus C_1$
$H(A_0, B_0) \oplus C_0$
$H(A_1, B_1) \oplus C_0$
$H(A_1, B_0) \oplus C_0$

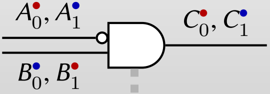
Instead of choosing output wire labels uniformly, choose them so that . . .

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

Row reduction ×2 [GueronLindellNofPinkas15]

$C_0 \leftarrow \{0,1\}^n$

$C_1 = H(A_0, B_1)$



$H(A_0, B_1) \oplus C_1$ $\leftarrow 0^\lambda$
$H(A_0, B_0) \oplus C_0$
$H(A_1, B_1) \oplus C_0$
$H(A_1, B_0) \oplus C_0$

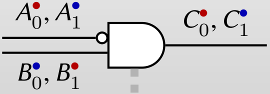
Instead of choosing output wire labels uniformly, choose them so that . . .

. . . first ciphertext is 0^n

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

Row reduction $\times 2$ [GueronLindellNofPinkas15]

$$C_0 = H_{00} \oplus H_{11} \oplus H_{10}$$
$$C_1 = H(A_0, B_1)$$



$$\boxed{\begin{array}{l} H(A_0, B_1) \oplus C_1 \\ H(A_0, B_0) \oplus C_0 \\ H(A_1, B_1) \oplus C_0 \\ H(A_1, B_0) \oplus C_0 \end{array}} \begin{array}{l} \leftarrow 0^\lambda \\ \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \leftarrow \bigoplus = 0^\lambda \end{array}$$

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

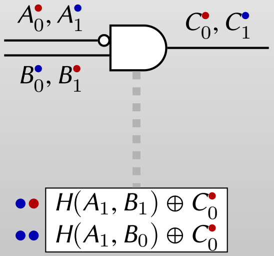
... first ciphertext is 0^n

... XOR of other ciphertexts is 0^n

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

Row reduction ×2 [GueronLindellNofPinkas15]

$$C_0 = H_{00} \oplus H_{11} \oplus H_{10}$$
$$C_1 = H(A_0, B_1)$$

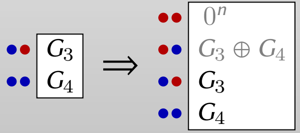


Instead of choosing output wire labels uniformly, choose them so that . . .

. . . 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

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

Scoreboard

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

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

Samee Zahur, Mike Rosulek, David Evans:

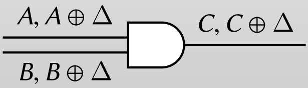
Two Halves Make a Whole: Reducing Data Transfer in Garbled Circuits using Half Gates.

Eurocrypt 2015

Best of both worlds: Free-XOR + 2-ciphertext AND

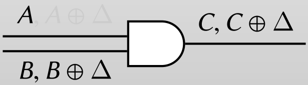
Half Gates [ZahurRosulekEvans15]

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



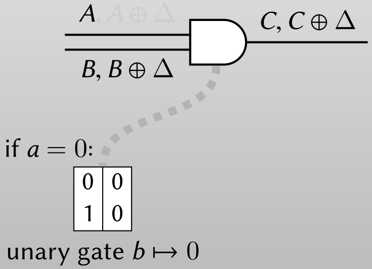
Half Gates [ZahurRosulekEvans15]

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



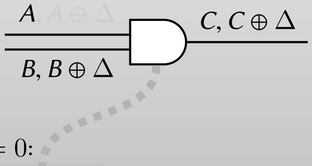
Half Gates [ZahurRosulekEvans15]

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



Half Gates [ZahurRosulekEvans15]

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



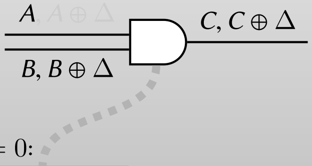
if $a = 0$:

B	C
$B \oplus \Delta$	C

unary gate $b \mapsto 0$

Half Gates [ZahurRosulekEvans15]

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



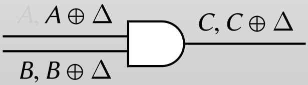
if $a = 0$:

$$\boxed{\begin{array}{l} H(B) \oplus C \\ H(B \oplus \Delta) \oplus C \end{array}}$$

unary gate $b \mapsto 0$

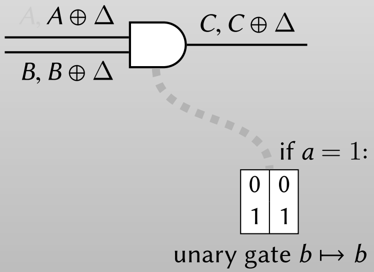
Half Gates [ZahurRosulekEvans15]

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



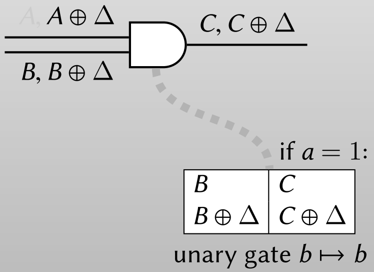
Half Gates [ZahurRosulekEvans15]

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



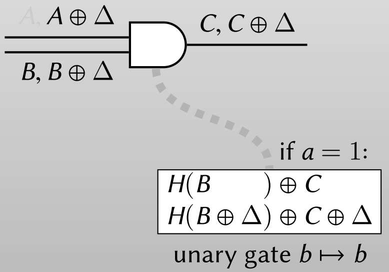
Half Gates [ZahurRosulekEvans15]

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



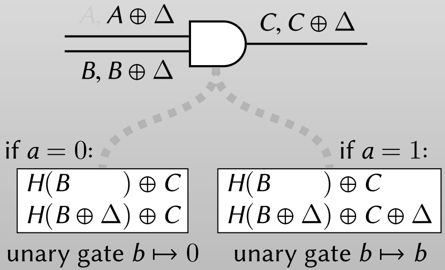
Half Gates [ZahurRosulekEvans15]

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



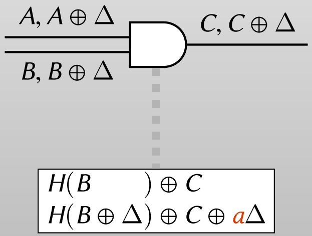
Half Gates [ZahurRosulekEvans15]

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



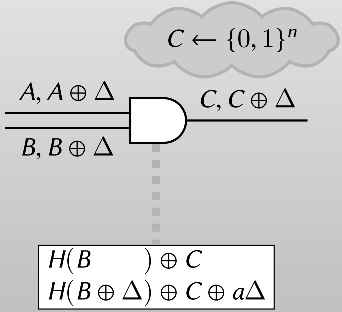
Half Gates [ZahurRosulekEvans15]

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



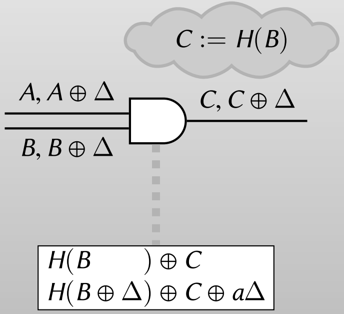
Half Gates [ZahurRosulekEvans15]

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



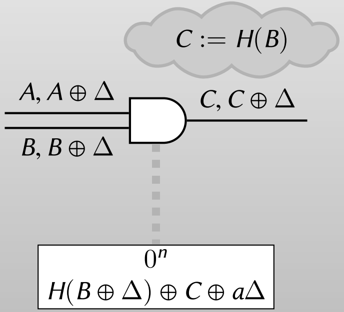
Half Gates [ZahurRosulekEvans15]

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



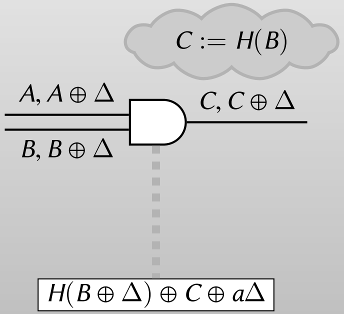
Half Gates [ZahurRosulekEvans15]

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



Half Gates [ZahurRosulekEvans15]

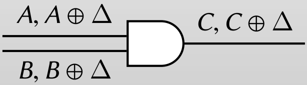
What if garbler knows in advance the truth value on one input wire?



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

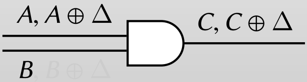
Half Gates [ZahurRosulekEvans15]

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



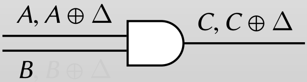
Half Gates [ZahurRosulekEvans15]

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



Half Gates [ZahurRosulekEvans15]

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

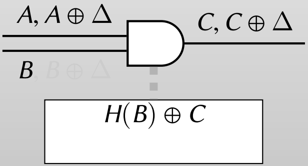


Evaluator has B (knows FALSE):

$\Longrightarrow$ should obtain C (FALSE)

Half Gates [ZahurRosulekEvans15]

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

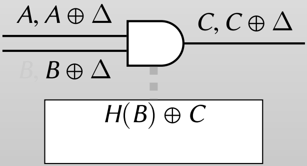


Evaluator has B (knows FALSE):

$\Longrightarrow$ should obtain C (FALSE)

Half Gates [ZahurRosulekEvans15]

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



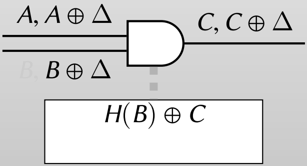
Evaluator has B (knows FALSE):

$\Rightarrow$ should obtain C (FALSE)

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

Half Gates [ZahurRosulekEvans15]

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



Evaluator has B (knows FALSE):

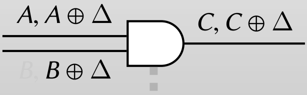
$\Longrightarrow$ should obtain C (FALSE)

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

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

Half Gates [ZahurRosulekEvans15]

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



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

Evaluator has B (knows FALSE):

$\Longrightarrow$ should obtain C (FALSE)

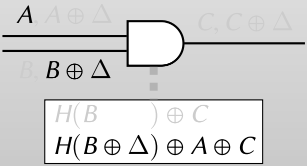
Evaluator has $B \oplus \Delta$ (knows TRUE):

$\Longrightarrow$ 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):

$\Longrightarrow$ should obtain C (FALSE)

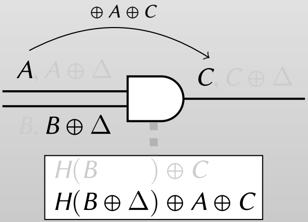
Evaluator has $B \oplus \Delta$ (knows TRUE):

$\Longrightarrow$ 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):

$\Longrightarrow$ should obtain C (FALSE)

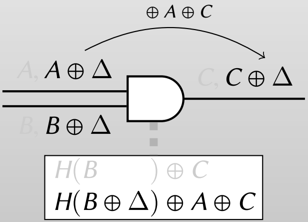
Evaluator has $B \oplus \Delta$ (knows TRUE):

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

- Suffices to learn $A \oplus C$

Half Gates [ZahurRosulekEvans15]

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



Evaluator has B (knows FALSE):

$\Rightarrow$ should obtain C (FALSE)

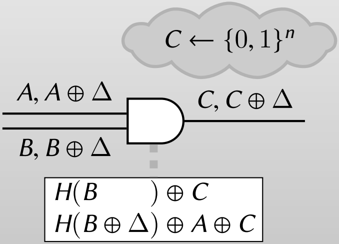
Evaluator has $B \oplus \Delta$ (knows TRUE):

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

- Suffices to learn $A \oplus C$

Half Gates [ZahurRosulekEvans15]

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



Evaluator has B (knows FALSE):

$\Longrightarrow$ should obtain C (FALSE)

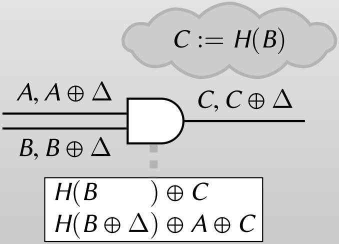
Evaluator has $B \oplus \Delta$ (knows TRUE):

$\Longrightarrow$ 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):

$\Longrightarrow$ should obtain C (FALSE)

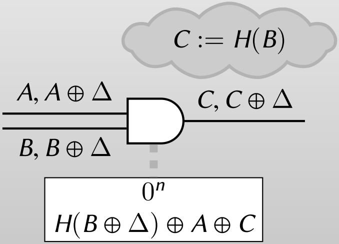
Evaluator has $B \oplus \Delta$ (knows TRUE):

$\Longrightarrow$ 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):

$\Longrightarrow$ should obtain C (FALSE)

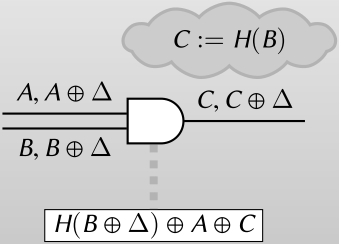
Evaluator has $B \oplus \Delta$ (knows TRUE):

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

- Suffices to learn $A \oplus C$

Half Gates [ZahurRosulekEvans15]

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



Evaluator has B (knows FALSE):

$\Rightarrow$ should obtain C (FALSE)

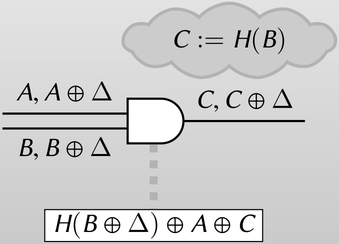
Evaluator has $B \oplus \Delta$ (knows TRUE):

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

- Suffices to learn $A \oplus C$

Half Gates [ZahurRosulekEvans15]

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



Evaluator has B (knows FALSE):

$\Rightarrow$ should obtain C (FALSE)

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

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

- Suffices to learn $A \oplus C$

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

Two halves make a whole!

$$a \wedge b$$

Two halves make a whole!

$$a \wedge b \;= \left(a \oplus r \oplus r\right) \wedge b$$

- Garbler chooses random bit r

Two halves make a whole!

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

- Garbler chooses random bit r

Two halves make a whole!

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

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

Two halves make a whole!

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

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

Two halves make a whole!

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

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

Two halves make a whole!

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

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

Two halves make a whole!

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

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

Scoreboard

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

Open Question

Can we do better than half-gates?

NO

[ZahurRosulekEvans15]

Can't garble an AND gate with < 2 ciphertexts

Open Question

Can we do better than half-gates?

NO

[ZahurRosulekEvans15]

Can't garble an AND gate with < 2 ciphertexts

YES

[BallMalkinRosulek16, KempkaKikuchiSuzuki16]

Can garble an AND gate with 1 ciphertext

Open Question

Can we do better than half-gates?

NO

[ZahurRosulekEvans15]

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

... using "standard techniques"

YES

[BallMalkinRosulek16, KempkaKikuchiSuzuki16]

Can garble an AND gate with 1 ciphertext

Open Question

Can we do better than half-gates?

NO

[ZahurRosulekEvans15]

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

... using "standard techniques"

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 $\mathbb{Z}_m$

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 $\Delta \in (\mathbb{Z}_m)^\lambda$

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 [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 $A \oplus \Delta$

$\oplus$ is componentwise addition mod 2

Generalized Free XOR:

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

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

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

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

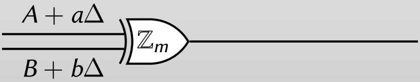
$+$ is componentwise addition mod m

Generalized Free XOR

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

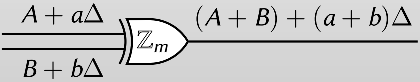
Generalized Free XOR

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



Generalized Free XOR

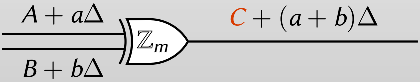
Idea: Truth value $a \in \mathbb{Z}_m$ encoded by wire label $\underline{A + a\Delta} \in (\mathbb{Z}_m)^\lambda$



Evaulator can simply add wire labels

Generalized Free XOR

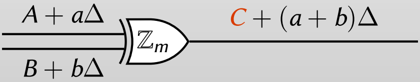
Idea: Truth value $a \in \mathbb{Z}_m$ encoded by wire label $\underline{A + a\Delta} \in (\mathbb{Z}_m)^\lambda$



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

Generalized Free XOR

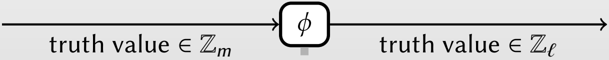
Idea: Truth value $a \in \mathbb{Z}_m$ encoded by wire label $\underline{A + a\Delta} \in (\mathbb{Z}_m)^\lambda$



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

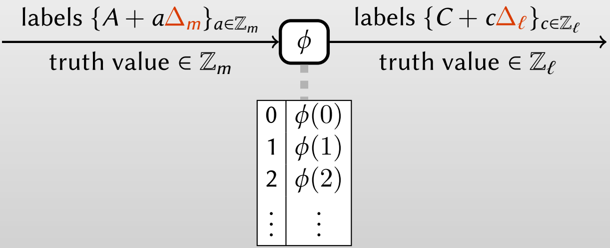
- Free multiplication by public constant c, if $\gcd(c, m) = 1$

Garbling unary gates



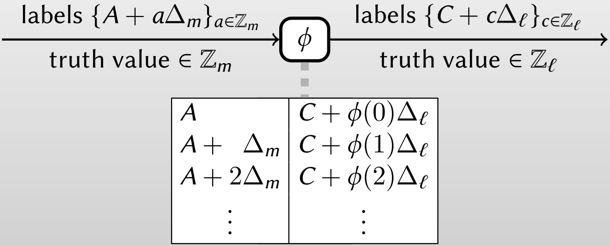
0	$\phi(0)$
1	$\phi(1)$
2	$\phi(2)$
$\vdots$	$\vdots$

Garbling unary gates



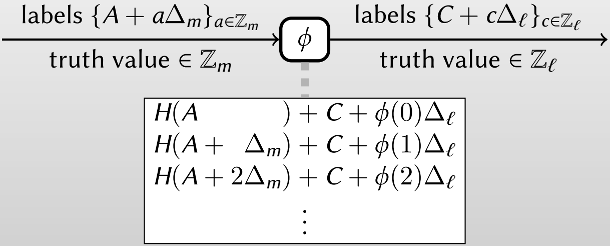
- Different “preferred modulus” on each wire $\Rightarrow$ different offsets Δ

Garbling unary gates



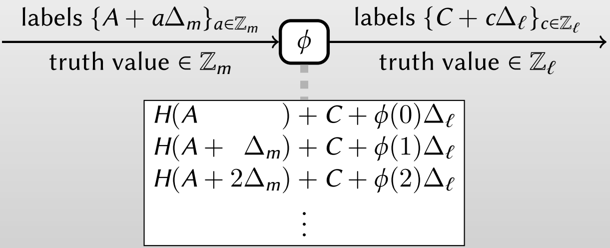
- Different “preferred modulus” on each wire $\Rightarrow$ different offsets Δ

Garbling unary gates



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

Garbling unary gates



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

Generalized garbling tools

We can efficiently garble any computation/circuit where:

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

Generalized garbling tools

We can efficiently garble any computation/circuit where:

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

Better basis for many computations than traditional boolean circuits!

Arithmetic computations

Example Scenario

Securely compute linear optimization problem on 32-bit values.

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

Arithmetic computations

Example Scenario

Securely compute linear optimization problem on 32-bit values.

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

"Standard approach"

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

Arithmetic computations

Example Scenario

Securely compute linear optimization problem on 32-bit values.

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

"Standard approach"

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

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

Arithmetic computations

Using generalized garbling techniques:

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

Arithmetic computations

Using generalized garbling techniques:

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

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

Arithmetic computations

Using generalized garbling techniques:

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

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

Arithmetic computations

Using generalized garbling techniques:

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

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

Arithmetic computations

$$\begin{array}{rl} \text{instead of} & \mathbb{Z}_{4294967296} \\ & \downarrow \\ \text{use} & \mathbb{Z}_{6469693230} \end{array}$$

Arithmetic computations

$$\text{instead of } \mathbb{Z}_{4294967296} = \mathbb{Z}_{2^{32}}$$
$$\downarrow$$
$$\text{use } \mathbb{Z}_{6469693230}$$

Arithmetic computations

$$\text{instead of } \mathbb{Z}_{4294967296} = \mathbb{Z}_{2^{32}}$$
$$\downarrow$$
$$\text{use } \mathbb{Z}_{6469693230} = \mathbb{Z}_{2 \cdot 3 \cdot 5 \cdot 7 \cdots 29}$$

Arithmetic computations

CRT residue number system!

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

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

Arithmetic computations

CRT residue number system!

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

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

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)

Converting to CRT

Claim:

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

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) × (# 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 $b_n b_{n-1} \cdots b_1 b_0$:

- 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)

Comparing CRT values

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

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

Comparing CRT values

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

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

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
$\vdots$	$\vdots$
1 4 2 1	29
2 0 0 0	30
$\vdots$	$\vdots$

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

Comparing CRT values

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

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

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
$\vdots$	$\vdots$
4 2 1	29
1 0 0 0	30
$\vdots$	$\vdots$

Approach for comparisons

CRT values given

$\downarrow$

Convert both CRT values to PMR

$\downarrow$

Compare PMR (simple L$\longrightarrow$R scan)

Approach for comparisons

CRT values given

$\downarrow$

Convert both CRT values to PMR

PMR representation of x:

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

$\downarrow$

Compare PMR (simple L$\longrightarrow$R scan)

Approach for comparisons

CRT values given

$\downarrow$

Convert both CRT values to PMR

Simple building block:

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

allows you to compute PMR representation of x:

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

$\downarrow$

Compare PMR (simple L$\longrightarrow$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:

$$
\begin{array}{cccc}
0 \mapsto 0 & 2 \mapsto 1 & 4 \mapsto 1 & 6 \mapsto 2 \\
1 \mapsto 3 & 3 \mapsto 3 & 5 \mapsto 4 &
\end{array}
$$

Approach for comparisons

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

Approach for comparisons

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

Operations on 32-bit integers:

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