Efficiently Computing with Private Data

David Heath
Collaboration with Vlad Kolesnikov
The “Millionaires’ Problem”
The “Millionaires’ Problem”

I wonder…
Which one of us has more money?
The “Millionaires’ Problem”

I wonder… Which one of us has more money?

Tell me how much money you have and I’ll tell you!
The “Millionaires’ Problem”

I wonder… Which one of us has more money?

Tell me how much money you have and I’ll tell you!

No way! You tell *me* how much money *you* have!
The “Millionaires’ Problem”

I wonder… Which one of us has more money?

Tell me how much money you have and I’ll tell you!

No way! You tell me how much money you have!

No way!
Multiparty Computation
Multiparty Computation

The set of techniques that allows mutually untrusting parties to compute any function over their private data.
Multiparty Computation
Multiparty Computation
Multiparty Computation

\[ f \]

MPC
Multiparty Computation

\[ f(x, y) \]

MPC
Multiparty Computation

$f(x, y)$

$f(x, y)$
Motivation for MPC
Motivation for MPC

• **Medical Studies:** researchers can collect statistics they need without compromising patient data
Motivation for MPC

• **Medical Studies:** researchers can collect statistics they need without compromising patient data

• **Joint Statistics:** companies can work together to build better models without sharing proprietary information
Motivation for MPC

• **Medical Studies:** researchers can collect statistics they need without compromising patient data

• **Joint Statistics:** companies can work together to build better models without sharing proprietary information

• **Private Auctions:** participants can be sure competitors aren’t cheating
Motivation for MPC
Motivation for MPC

• **Location Services:** Companies can provide services without tracking your exact location
Motivation for MPC

• **Location Services:** Companies can provide services without tracking your exact location.

• **Contact sharing:** Find the intersection between two parties friends without giving away entire list.
Motivation for MPC

- **Location Services:** Companies can provide services without tracking your exact location.

- **Contact sharing:** Find the intersection between two parties friends without giving away entire list.

- ???
Motivation for MPC

MPC enables Secure Data Sharing
MPC in Practice?
MPC in Practice?

- MPC does see use
MPC in Practice?

- MPC *does* see use
  - Google “Password Checkup” uses MPC to warn you of unsafe login credentials
MPC in Practice?

- MPC does see use
  - Google “Password Checkup” uses MPC to warn you of unsafe login credentials
  - Partisia commercialized MPC based private auctions
MPC in Practice?

- MPC does see use
  - Google “Password Checkup” uses MPC to warn you of unsafe login credentials
  - Partisia commercialized MPC based private auctions
  - MPC has been used to build privacy preserving databases, e.g. Blind Seer
MPC in Practice?

• MPC does see use
  • Google “Password Checkup” uses MPC to warn you of unsafe login credentials
  • Partisia commercialized MPC based private auctions
  • MPC has been used to build privacy preserving databases, e.g. Blind Seer
  • ...

MPC in Practice?
MPC in Practice?

- MPC is expensive
MPC in Practice?

- MPC is expensive
  - We compute “under encryption”
MPC in Practice?

- MPC is expensive
  - We compute “under encryption”
- Reducing the cost of MPC will enable more applications
This talk...
This talk...

- **Yao’s Garbled Circuit (GC)**
  - Allows secure two party computation for arbitrary functions
  - The *most efficient technique* for secure two party computation
This talk...

- Yao’s Garbled Circuit (GC)
  - Allows secure two party computation for arbitrary functions
  - The most efficient technique for secure two party computation
- How can we make it faster?
  - Communication is the bottleneck: Reduce communication
  - We will take advantage of conditional branching (if, switch program statements)
  - Surprising result, breaks community-held assumptions about GC
Computing Under Encryption

\[ f \]
Computing Under Encryption
Computing Under Encryption

Generator
Computing Under Encryption
Computing Under Encryption
Computing Under Encryption
Computing Under Encryption
Computing Under Encryption
Computing Under Encryption
Computing Under Encryption
Computing Under Encryption
Computing Gate-by-gate Under Encryption
Computing Gate-by-gate Under Encryption
Computing Gate-by-gate Under Encryption
Computing Gate-by-gate Under Encryption

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

Generator
Computing Gate-by-gate Under Encryption

Generator
Computing Gate-by-gate Under Encryption

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

Generator
Computing Gate-by-gate Under Encryption

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Computing Gate-by-gate Under Encryption

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Computing Gate-by-gate Under Encryption

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

Generator
Computing Gate-by-gate Under Encryption

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

Generator
Computing Gate-by-gate Under Encryption

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

Generator
Computing Gate-by-gate Under Encryption

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

Generator
Computing Gate-by-gate Under Encryption

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

Generator
Computing Gate-by-gate Under Encryption
Computing Gate-by-gate Under Encryption
Computing Under Encryption
Computing Under Encryption
Computing Under Encryption
Computing Under Encryption
Computing Under Encryption
Computing Under Encryption
Sending Material is the Bottleneck
Sending Material is the Bottleneck
Sending Material is the Bottleneck
Sending Material is the Bottleneck
Sending Input Keys
Sending Input Keys

Generator

Evaluator
Sending Input Keys
Sending Input Keys
Sending Input Keys
Oblivious Transfer

OT

Generator

Evaluator
Oblivious Transfer

OT

Generator

Evaluator
Oblivious Transfer

Evaluator

"I need key 1"

Generator
Oblivious Transfer

Generators

Evaluator

“I need key 1”
Computing Under Encryption

Evaluator
Computing Under Encryption
Computing Under Encryption
Computing Under Encryption
Computing Under Encryption
Computing Under Encryption
Computing Under Encryption
Computing Gate-by-gate Under Encryption
Computing Gate-by-gate Under Encryption
Computing Gate-by-gate Under Encryption
Computing Gate-by-gate Under Encryption
Computing Gate-by-gate Under Encryption
Computing Gate-by-gate Under Encryption
Computing Gate-by-gate Under Encryption
Computing Under Encryption
Computing Under Encryption
Computing Under Encryption
Computing Under Encryption
Yao’s Garbled Circuit

- The **generator** encrypts the circuit by constructing **material**
- The **evaluator** uses material to evaluate gate-by-gate **under encryption**
- The dominant cost of GC is the **amount of material** the generator must transmit over a network to the evaluator
How can we go faster?
How can we go faster?

- Lots of previous work has focused on improving the efficiency of individual gates
  - GC is efficient for circuits with large numbers of gates
How can we go faster?

- Lots of previous work has focused on improving the efficiency of individual gates
  - GC is efficient for circuits with large numbers of gates
- But when we model programs with **conditional branches** as circuits, we waste work
How can we go faster?

- Lots of previous work has focused on improving the efficiency of individual gates.
  
  - GC is efficient for circuits with large numbers of gates.
  
- But when we model programs with **conditional branches** as circuits, we waste work.

- Our work reduces communication for programs with **conditional branching**.
How can we go faster?

• Since 2004, 10000x overall improvement in speed. Communication improvement much harder to come by
  • 2 encrypted truth table rows for “odd” gates (AND gates)
  • 0 encrypted truth table rows for “even” gates (XOR gates)
• Even small improvements to GC are highly valued
• Our work achieves significant improvement for programs with conditional branching
  • Improvement factor equal to the “branching factor”
### Performance

<table>
<thead>
<tr>
<th></th>
<th>Zero Knowledge</th>
<th>General MPC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Before</td>
<td>After</td>
</tr>
<tr>
<td><strong>Computation</strong></td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><strong>Communication</strong></td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

$n$ is the branching factor
### Performance

<table>
<thead>
<tr>
<th></th>
<th>Zero Knowledge</th>
<th>General MPC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Before</td>
<td>After</td>
</tr>
<tr>
<td>Computation</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Communication</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

Universal Circuits, Comp and Comm: $O(5 \cdot |C| \cdot \log |C|)$

$n$ is the branching factor    
$|C|$ is the total number of gates
Conditional Branching

If (...) {
    ...
}
else {
    ...
}
Conditional Branching

• Traditionally assumed the generator must send material for every program branch separately

If (...) {
...
}
else {
...
}

Conditional Branching

- Traditionally assumed the generator must send material for every program branch separately

- Our work breaks this assumption

```plaintext
If (...) {
    ...
} else {
    ...
}
```
Conditional Branching

- Traditionally assumed the generator must send material for every program branch separately
- Our work breaks this assumption
- Our constructions enough material for the longest branch only

```plaintext
If (...) {
    ...
} else {
    ...
}
```
Naive Conditional Branching

Generator

Evaluator
Naive Conditional Branching

Diagram showing a circuit with labeled components: Generator and Evaluator.
Naive Conditional Branching
Conditional Branching

Generator

Evaluator
Conditional Branching

- “Free If”: What can we do if the generator knows which branch is taken?
Conditional Branching

- **“Free If”:** What can we do if the *generator* knows which branch is taken?

- **“Stacked Garbling for Zero-Knowledge”:** What can we do if the *evaluator* knows which branch is taken?
Conditional Branching

- **“Free If”**: What can we do if the *generator* knows which branch is taken?

- **“Stacked Garbling for Zero-Knowledge”**: What can we do if the *evaluator* knows which branch is taken?

- **“Stacked Garbling”**: What if *no one* knows which branch is taken?
“Free IF” [K18]
“Free IF” [K18]
Key Idea

- Material can be viewed as a long random string

- Evaluator cannot tell which branch a particular string was generated from
Free IF

Generator
Free IF
Free IF
Free I
Free IF
Evaluating with Garbage Material
Evaluating with Garbage Material
Evaluating with Garbage Material
Evaluating with Garbage Material
Evaluating with Garbage Material

Evaluator
Evaluating with Garbage Material
Evaluating with Garbage Material
Evaluating with Garbage Material

Evaluator
Evaluating with Garbage Material
Evaluating with Garbage Material
Evaluating with Garbage Material

Evaluator
Evaluating with Garbage Material
Evaluating with Garbage Material

Evaluator
Evaluating with Garbage Material
Evaluating with Garbage Material
Evaluating with Garbage Material
Eliminating Garbage Output
Eliminating Garbage Output
Eliminating Garbage Output

Output Selection Protocol

Evaluator

Generator
Eliminating Garbage Output

Evaluator

Output Selection Protocol

Generator

“You evaluated Circuit 1”
Eliminating Garbage Output

Evaluator

Generator

Output Selection Protocol

“You evaluated Circuit 1”
“Free If”

• Key Ideas

• Material can be viewed as a random string

• The evaluator cannot tell which branch the material is meant for

• Therefore, evaluator can use same material with multiple branches
“Free If”
“Free If”

• Why has it taken so long for this improvement to be found?
“Free If”

• Why has it taken so long for this improvement to be found?

• “Free If” breaks from tradition
“Free If”

- Why has it taken so long for this improvement to be found?
- “Free If” breaks from tradition
  - **Material, circuit topology** (i.e. the circuit itself), and **wire labels** traditionally considered together
“Free If”

- Why has it taken so long for this improvement to be found?
- “Free If” breaks from tradition
  - **Material, circuit topology** (i.e. the circuit itself), and **wire labels** traditionally considered together
  - This intuition is even **formalized** in many works
“Free If”

- Why has it taken so long for this improvement to be found?

- “Free If” breaks from tradition
  - Material, circuit topology (i.e. the circuit itself), and wire labels traditionally considered together
  - This intuition is even formalized in many works
  - “Free If” encourages a separation of these components
Conditional Branching

• “Free If”: What can we do if the *generator* knows which branch is taken?

• “Stacked Garbling for Zero-Knowledge”: What can we do if the *evaluator* knows which branch is taken?

• “Stacked Garbling”: What if *no one* knows which branch is taken?
Stacked Garbling for ZK
Stacked Garbling for ZK

Generator

Evaluator
Exclusive Or (XOR)

\[ a \oplus a \oplus b = b \]
Expanding a Seed
Expanding a Seed
Expanding a Seed

Generator

Seed
Expanding a Seed

![Diagram showing the process of expanding a seed with binary numbers and logic gates. The seed is connected to a generator, and the output is shown as binary sequences: 01101001, 10010101, and 00010100.](image-url)
Expanding a Seed
Encryption from Seed
Encryption from Seed

• Why can’t we just send seed instead of entire garbling?
Encryption from Seed

- Why can’t we just send seed instead of entire garbling?
- Reveals too much!
Encryption from Seed

• Why can’t we just send seed instead of entire garbling?

• Reveals too much!

  • The evaluator must not know which keys he receives from the generator
Stacked Garbling for ZK
Stacked Garbling for ZK

Seed 1

Generator
Stacked Garbling for ZK

Seed 1

Generator
Stacked Garbling for ZK

Seed 1

Generator
Stacked Garbling for ZK

Seed 1

Seed 2

Generator
Stacked Garbling for ZK

Seed 1

..01101001..

..10010101..

..00010100..

Seed 2

Generator
Stacked Garbling for ZK

Seed 1

Seed 2

Generator
Stacked Garbling for ZK

Seed 1

Seed 2

Generator
Stacked Garbling for ZK

Seed 1

Seed 2

Generator

..01101001..
..10010101..
..00010100..

..11110010..
..10101011..
..00101100..

..11110010..
..10101011..
..00101100..
"Stacking" Material

<table>
<thead>
<tr>
<th>Seed 1</th>
<th>Seed 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>..01101001..</td>
<td>..10101011..</td>
</tr>
<tr>
<td>..1110010..</td>
<td>..00101100..</td>
</tr>
<tr>
<td>..1110010..</td>
<td>..10101011..</td>
</tr>
<tr>
<td>..00010100..</td>
<td>..00101100..</td>
</tr>
</tbody>
</table>
## “Stacking” Material

<table>
<thead>
<tr>
<th>Seed 1</th>
<th>Seed 2</th>
<th>Generator</th>
</tr>
</thead>
<tbody>
<tr>
<td>.01101001..</td>
<td>.10010101..</td>
<td>.00010100..</td>
</tr>
<tr>
<td>.11100101..</td>
<td>.10101011..</td>
<td>.00101100..</td>
</tr>
<tr>
<td>.10011011..</td>
<td>.00111110..</td>
<td>.00111000..</td>
</tr>
<tr>
<td>.01101001..</td>
<td>.00111110..</td>
<td>.00111000..</td>
</tr>
<tr>
<td>.10011011..</td>
<td>.00111110..</td>
<td>.00111000..</td>
</tr>
</tbody>
</table>
“Stacking” Material

Seed 1

Seed 2

Generator

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>( M_1 )</th>
<th></th>
<th></th>
<th>( M_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \oplus )</td>
<td>..11110010..</td>
<td>..10010101..</td>
<td>..00101100..</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \oplus )</td>
<td>..01101001..</td>
<td>..10010101..</td>
<td>..00010100..</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \oplus )</td>
<td>..10011011..</td>
<td>..00111110..</td>
<td>..00111000..</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( M_1 \oplus M_2 \)
“Stacking” Material

$M_1 \oplus M_2$
Sending Seeds

\[ M_1 \oplus M_2 \]
Sending Seeds

Seed 1

Seed 2

Generator

Evaluator

\[ M_1 \oplus M_2 \]
Sending Seeds

Seed 1

Seed 2

Generator

Evaluator

\[ M_1 \oplus M_2 \]
Sending Seeds

Seed 1

Seed 2

Oblivious Transfer

$M_1 \oplus M_2$

Generator

Evaluator
Sending Seeds

Seed 1

Seed 2

Oblivious Transfer

$M_1 \oplus M_2$

Generator

Evaluator
Sending Seeds

Seed 1

Seed 2

Oblivious Transfer

Generator

Evaluator

“I want to evaluate Circuit 1 so I need seed 2”

$M_1 \oplus M_2$
Sending Seeds

Seed 1

Seed 2

Oblivious Transfer

“I want to evaluate Circuit 1 so I need seed 2”

Evaluator

Generator

$M_1 \oplus M_2$
Sending Seeds

“I want to evaluate Circuit 1 so I need seed 2”
Copying the Generator

Evaluator

$M_1 \oplus M_2$

Evaluator

Seed 2
Copying the Generator

Evaluator

\[ M_1 \oplus M_2 \]
Copying the Generator

\[ M_1 \oplus M_2 \]

Seed 2

Evaluator
Copying the Generator

Evaluator

$\mathbf{M}_1 \oplus \mathbf{M}_2$

Seed 2

..1110010..
..10101011..
..00101100..
Copying the Generator

\[ M_1 \oplus M_2 \]

..1110010..
..10101011..
..00101100..
“Unstacking” Material

\[ M_1 \oplus M_2 \]

Seed 2

Evaluator
“Unstacking” Material

\[
M_1 \oplus M_2
\]

\[
\begin{array}{ccc}
..11110010.. & ..10101011.. & ..00101100..

..10011011.. & ..00111110.. & ..00111000..
\end{array}
\]

Seed 2

Evaluator
“Unstacking” Material

\[ M_1 \oplus M_2 \]

Seed 2

Evaluator
Evaluator
Stacked Garbling for ZK

• Key Ideas

• Material can be viewed as a random string

• Material can be reconstructed via a random seed

• Material can be “stacked” via bitwise XOR

• Therefore, the generator can efficiently send material by stacking with XOR and also obliviously transferring seeds

Generator
Conditional Branching

- **“Free If”:** What can we do if the *generator* knows which branch is taken?
- **“Stacked Garbling for Zero-Knowledge”:** What can we do if the *evaluator* knows which branch is taken?
- **“Stacked Garbling”:** What if *no one* knows which branch is taken?
Stacked Garbling
Stacked Garbling

Evaluator

Generator

Evaluator
Stacked Garbling

![Diagram showing two interconnected boxes labeled $C_1$ and $C_2$ with an input from the Generator and an output to the Evaluator.](image-url)
Stacked Garbling

Diagram:

- Generator
- Evaluator
- $C_1$
- $C_2$
Stacked Garbling

C_1

C_2

Generator

Evaluator
Stacked Garbling

$C'$

$C_1$

$C_2$

Generator

Evaluator
Stacked Garbling

"Branch Condition"

C′

C₁

C₂

Generator

Evaluator
Naive

"Branch Condition"

C' -> C1

Evaluator

Generator
"Branch Condition" 

Naive
“Branch Condition”

Evaluator

Generator

Naive

MUX
Evaluator

Generator

Naive

"Branch Condition"

\[ C' \]

\[ C_1 \]

\[ C_2 \]

\[ M' \]

\[ M_1 \]

\[ M_2 \]

\[ M_{mux} \]

Evaluator

Generator
Evaluator

Generator

Naive

"Branch Condition"

C'

M' M_1 M_2 M_{mux}

C_1

C_2

MUX

Evaluator
“Branch Condition”

$C'$

$C_1$

$C_2$

Generator

Evaluator
Stacked Garbling

“Branch Condition”

$C'$

$C_1$

$C_2$

Generator

Evaluator
Stacked Garbling

Generator

\[ C' \]

\[ C_1 \]

\[ C_2 \]
Generator

$C'$

$C_1$

$C_2$
$C'$

$C_1$

$C_2$

$M'$

Generator
Generator

\[ C' \]

\[ M' \]

\[ C_1 \]

\[ C_2 \]

\[ \text{Seed 1} \]

\[ \text{Seed 2} \]
Generator

Seed 1

Seed 2

$C'$

$C_1$

$C_2$

$M_1$

$M'$

$M_2$
Seed 1

Seed 2

$C'$

$C_1$

$M_1$

$C_2$

$M'$

$M_2$

Generator
Generators

Seed 1

Seed 2

\[ \text{Seed 1} \rightarrow \text{Seed 2} \]

\[ \text{Seed 2} \rightarrow \text{Seed 1} \]

\[ C' \]

\[ C_1 \]

\[ M_1 \]

\[ M' \]

\[ M_2 \]

\[ C_2 \]
Seed 1

Seed 2

$M'$

$M_1$

$M_2$

$C'$

$C_1$

$C_2$

Generator
$C'$

$C_1$

$C_2$

$M'$

$M_1 \oplus M_2$

Evaluator

Generator
$M_1 \oplus M_2$

Evaluator

Seed 1

Seed 2

$C'_1$

$C'_2$

$M'$
$M'_1 \oplus M'_2$
Seed 1

Seed 2

\[ C \]

\[ C' \]

\[ M' \]

\[ M_1 \oplus M_2 \]

Evaluator
$M_1 \oplus M_2$
$M_1 \oplus M_2$

Evaluator

$C'$

$C_1$

$C_2$

$M'$

Seed 1
\[ M_1 \oplus M_2 \]
$C' \rightarrow C_1 \oplus C_2$

$M'$

$M_1 \oplus M_2$

$M_1$

Evaluator

Seed 1
$$C'$$

$$C_1$$

$$C_2$$

$$M'$$

$$M_1 \oplus M_2$$

$$M_1$$

Evaluator

Seed 1

$$M_1 \oplus (M_1 \oplus M_2)$$
Evaluator

\[ M_1 \oplus M_2 \]

\[ M_1 \]

\[ M_2 \]

Seed 1

C

C′

C_1

C_2

M'

M′
Evaluator

$C'^{2}$

$C_1 \oplus M_2$

$M_1 \oplus M_2$

Seed 1

$M_1$

$M_2$
$C'$

$C_1$

$C_2$

$M' \otimes M_1 \oplus M_2$

Evaluator

Seed 1
$$C'$$

$$C_1$$

$$C_2$$

$$M_1 \oplus M_2$$

Evaluator

Seed 1

$$M'$$

$$M_2^*$$
$M' = M_2^* \oplus (M_1 \oplus M_2)$
$M_2^* \oplus (M_1 \oplus M_2)$

Evaluator

Seed 1
Evaluator
Eliminating Garbage Output

Evaluator
Eliminating Garbage Output

Evaluator

??

Generator

??

Key

Evaluator

Generator
Eliminating Garbage Output

Output Selection Protocol

Evaluator

Generator
Eliminating Garbage Output

Evaluator

??

Key

Output Selection Protocol

Generator
Eliminating Garbage Output

Evaluator

??

Output Selection Protocol

Generator
Eliminating Garbage Non-interactively
Eliminating Garbage Non-interactively

$C'$

$C_1$

$C_2$

Seed 1

Generator

Evaluator
Eliminating Garbage
Non-interactively
Eliminating Garbage Non-interactively

$C'$

$C_1$

$C_2$

Seed 1

$M_2^*$

Generator

Evaluator
Eliminating Garbage Non-interactively

$C'$

$C_1$

$C_2$

$M_2^*$

Generator

Evaluator
Eliminating Garbage Non-interactively

$C'$

$C_1$

$C_2$

Seed 2

$M_2^*$

Generator

Evaluator
Eliminating Garbage Non-interactively
Eliminating Garbage
Non-interactively

Generator

Evaluator
Eliminating Garbage Non-interactively

Generator

Evaluator

$C'$

$C_1$

$C_2$

$M_1^*$

$M_2^*$
Eliminating Garbage Non-interactively
Eliminating Garbage
Non-interactively

Generator

Evaluator
Eliminating Garbage Non-interactively

Generator

Evaluator
Eliminating Garbage
Non-interactively

Generator

Evaluator
Stacked Garbling

• By combining ideas from previous approaches…

  • Viewing material as a random string

  • Constructing material from seeds

  • Stacking material with XOR

  • We can discard garbage output encryptions non-interactively

• We can compute programs with conditional branching much more efficiently
<table>
<thead>
<tr>
<th></th>
<th>Zero Knowledge</th>
<th>General MPC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Before</td>
<td>After</td>
</tr>
<tr>
<td>Computation</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Communication</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

$n$ is the branching factor