Faster Secure Computation through Automatic Parallelization

Niklas Buescher, Stefan Katzenbeisssser
USENIX Security Symposium`15

\[1 \times \text{CPU} = 1\]
\[2 \times \text{CPU} = 2.2\]
\[4 \times \text{CPU} = 4.3\]
Preliminaries - Secure Two-party Computation (STC)

Privacy-preserving fingerprint matching between mobile and server
Preliminaries - Secure Two-party Computation (STC)

Privacy-preserving fingerprint matching between mobile and server

\[ f(x, y) \]

\[ x \]

\[ f(x, y) \]

\[ y \]
Idea
- Functionalities as Boolean circuits
- Bits → random wire labels
- Gates → garbled truth tables (GTT)

<table>
<thead>
<tr>
<th>x</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y = 1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y = 0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Preliminaries - Yao’s Garbled Circuits
Andrew Yao

Idea
• Functionalities as Boolean circuits
• Bits → random wire labels
• Gates → garbled truth tables (GTT)

<table>
<thead>
<tr>
<th>∧</th>
<th>$x = 1$</th>
<th>$x = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y = 1$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$y = 0$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

| $y = 1$ | $x = 1$ | $E(w_1^1, E(w_2^1, w_3^1))$ | $E(w_1^0, E(w_2^1, w_3^0))$ |
| $y = 0$ | $x = 1$ | $E(w_1^1, E(w_2^0, w_3^0))$ | $E(w_1^0, E(w_2^0, w_3^0))$ |
Preliminaries - Yao’s Garbled Circuits
Andrew Yao

Idea
- Functionalities as Boolean circuits
- Bits → random wire labels
- Gates → garbled truth tables (GTT)

Inputs A Inputs B

Outputs

\[ f \]

\[ w_1^1, w_1^0 \]
\[ w_2^1, w_2^0 \]
\[ w_3^1, w_3^0 \]

\[ \land \]

<table>
<thead>
<tr>
<th>( x )</th>
<th>( x = 1 )</th>
<th>( x = 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y )</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x )</th>
<th>( x = 1 )</th>
<th>( x = 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y )</td>
<td>( E(w_1^1, E(w_2^1, w_3^1)) )</td>
<td>( E(w_1^0, E(w_2^1, w_3^0)) )</td>
</tr>
<tr>
<td>( y )</td>
<td>( E(w_1^1, E(w_2^0, w_3^0)) )</td>
<td>( E(w_1^0, E(w_2^0, w_3^0)) )</td>
</tr>
</tbody>
</table>

Secure in the semi-honest model
Motivation - Practical Yao’s Garbled Circuits

2004 Fairplay [MNP04] – Garbled row reduction (GRR), Point-and-permute
2008 Free-XOR [KS08] – XOR gates require no encryption/communication
2013 Fixed-key Garbling [BHK13] – Garbling from a fixed-key blockcipher with AES-NI

→ Proof: Lower bound on the number of ciphertexts

Still a huge gap between STC and generic computation ➔ This work: parallelization
Parallel Yao Garbled Circuits

Parallel garbling: Independent gates or partitions of gates can be garbled and evaluated by any thread in any order.

No impact on security [LIP09]
Parallelization - Circuit Decomposition
Parallelization - Circuit Decomposition

Fine-grained parallelization (FGP)

- All gates within a level (circuit depth) are independent
- Synchronization is needed after every level
Parallelization - Circuit Decomposition

Fine-grained parallelization (FGP)
- All gates within a level (circuit depth) are independent
- Synchronization is needed after every level

Coarse grained parallelization (CGP)
- Larger coherent partitions
- NP-hard graph partitioning problem

➔ Our approach: Detect parallelism on source code level
void millionaires() {
    int INPUT_A_wealth, INPUT_B_wealth;
    int OUTPUT_res;
    if(INPUT_A_wealth > INPUT_B_wealth)
        OUTPUT_res = 1;
    else
        OUTPUT_res = 0;
}
ParCC - Parallel Circuit Compiler

C Source with input annotations

ParCC source-to-source compiler

Par4all [ACE12]

void millionaires() {
  int INPUT_A_wealth, INPUT_B_wealth;
  int OUTPUT_res;
  if(INPUT_A_wealth > INPUT_B_wealth)
    OUTPUT_res = 1;
  else
    OUTPUT_res = 0;
}
void millionaires() {  
    int INPUT_A_wealth, INPUT_B_wealth;  
    int OUTPUT_res;  
    if(INPUT_A_wealth > INPUT_B_wealth)  
        OUTPUT_res = 1;  
    else  
        OUTPUT_res = 0;  
}
void millionaires() {
    int INPUT_A_wealth, INPUT_B_wealth;
    int OUTPUT_res;
    if (INPUT_A_wealth > INPUT_B_wealth) {
        OUTPUT_res = 1;
    } else {
        OUTPUT_res = 0;
    }
}
void millionaires() {
    int INPUT_A_wealth, INPUT_B_wealth;
    int OUTPUT_res;
    if(INPUT_A_wealth > INPUT_B_wealth)
        OUTPUT_res = 1;
    else
        OUTPUT_res = 0;
}
Evaluation - Example Applications and Circuit Properties

Biometric Matching (BioMatch):

Database with \( n = 512 \) samples of degree \( d = 4 \); 64 bit integers

\[
\min \left( \sum_{i=1}^{d} (s_{1,i} - e_i)^2, \ldots, \sum_{i=1}^{d} (s_{n,i} - e_i)^2 \right)
\]

Parallel Modular Exponentiation (MExp):

32 x 32 bit modular exponentiation

Matrix-Vector Multiplication (MVMul):

16 x 16 Matrix with 64 bit integers

<table>
<thead>
<tr>
<th></th>
<th>BioMatch</th>
<th>MExp</th>
<th>MVMul</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circuit Size</td>
<td>66M</td>
<td>21.5M</td>
<td>3.3M</td>
</tr>
<tr>
<td>%AND</td>
<td>25%</td>
<td>41%</td>
<td>37%</td>
</tr>
<tr>
<td>Inputs bits</td>
<td>512 / 112K</td>
<td>1K / 1K</td>
<td>17K / 1K</td>
</tr>
<tr>
<td>Offline garbling time</td>
<td>2.07s</td>
<td>1.136s</td>
<td>0.154s</td>
</tr>
</tbody>
</table>
Parallel Framework – Implementation and Testbed

UltraSFE:

- Based on: JustGarble, ME_SFE (FastGC)
- Focus: Parallelization, low memory footprint
- Garbled row reduction, pipe-lining, fixed-key garbling with AES-NI, half-gates
- Written in C++, OpenMP and pthreads, SSE4.2

Testbed:

- Amazon EC2 – c3.8xlarge instance (report 16 physical cores)
- Serial garbling speed ~8M AND gates / s
Results – Circuit Garbling (offline)

Speed-up of FGP for a different number of cores.
Results – Circuit Garbling (offline)

Speed-up of FGP and CGP for a different number of cores.

CGP significantly outperforms FGP
The Limits of Fine-Grained Parallelization

“Increasing input sizes should overcome limits of FGP” ➜ Not really:

Distribution of level widths in number of non-linear gates when compiled with CBMC-GC
The Limits of Fine-Grained Parallelization

“Increasing input sizes should overcome limits of FGP” ➔ Not really:

Current high level compiler are optimized for Free-XOR but not width

Examples:
- Array accesses compile to multiplexers with constant width
- Comparisons compile to circuits with constant width

Distribution of level widths in number of non-linear gates when compiled with CBMC-GC
Results - CGP in the Online Setting

Security levels: 80bit and 128bit

Throughput: ~7M AND gates

Bandwidth req.: ~1 – 1.5 Gbit / core
Results - CGP in the Online Setting

Security levels: 80bit and 128bit
Throughput: ~7M AND gates
Bandwidth req.: ~1 – 1.5 Gbit / core

Can we overcome the boundary?
Results - CGP in the Online Setting

Security levels: 80bit and 128bit
Throughput: ~7M AND gates
Bandwidth req.: ~1 – 1.5 Gbit / core

Can we overcome the boundary?

⇒ Yes! See next slides
Inter-Party Parallelization (IPP)

a.) Yao’s Garbled Circuits
Inter-Party Parallelization (IPP)

a.) Yao’s Garbled Circuits

b.) Coarse-grained decomposition
Inter-Party Parallelization (IPP)

a.) Yao’s Garbled Circuits  
b.) Coarse-grained decomposition  
c.) Inter-party parallelization

Similar computation power $\rightarrow$ Speed-up of 1.6 / 1.33
Inter-Party Parallelization (IPP) for Mixed Functionalities

Sequential

Parallel Parallel Parallel Parallel Parallel

Sequential
Inter-Party Parallelization (IPP) for Mixed Functionalities
Inter-Party Parallelization (IPP) for Mixed Functionalities

Transferring roles $\rightarrow$ secure state sharing
### Results – Inter-Party Parallelization

#### High bandwidth environments (10Gbit)

<table>
<thead>
<tr>
<th>Cores</th>
<th>BioMatch</th>
<th>MExp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 CGP</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 IPP</td>
<td>1</td>
<td>1.1</td>
</tr>
<tr>
<td>2 CGP</td>
<td>1.8</td>
<td>1.9</td>
</tr>
<tr>
<td>2 IPP</td>
<td>1.8</td>
<td>2.2</td>
</tr>
<tr>
<td>4 CGP</td>
<td>3.2</td>
<td>3.6</td>
</tr>
<tr>
<td>4 IPP</td>
<td>3.1</td>
<td>4.3</td>
</tr>
</tbody>
</table>

**Speed-up**
## Results – Inter-Party Parallelization

### High bandwidth environments (10Gbit)

<table>
<thead>
<tr>
<th>Cores</th>
<th>BioMatch</th>
<th>MExp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 CGP</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 IPP</td>
<td>1</td>
<td><strong>1.1</strong></td>
</tr>
<tr>
<td>2 CGP</td>
<td>1.8</td>
<td>1.9</td>
</tr>
<tr>
<td>2 IPP</td>
<td>1.8</td>
<td><strong>2.2</strong></td>
</tr>
<tr>
<td>4 CGP</td>
<td>3.2</td>
<td>3.6</td>
</tr>
<tr>
<td>4 IPP</td>
<td>3.1</td>
<td><strong>4.3</strong></td>
</tr>
</tbody>
</table>

**Speed-up**

### Low bandwidth environments (100Mbit)

<table>
<thead>
<tr>
<th></th>
<th>BioMatch</th>
<th>MExp</th>
</tr>
</thead>
<tbody>
<tr>
<td>raw</td>
<td>45.0±0.5s</td>
<td>24.1±0.2s</td>
</tr>
<tr>
<td>IPP</td>
<td>29.9±0.3s</td>
<td>16.1±0.1s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td><strong>1.5</strong></td>
</tr>
</tbody>
</table>

**Time and speed-up**

---

Conclusion & Outlook

- Automatized toolchain for parallel circuits
- Yao’s Garbled Circuits profits from automatic parallelization
- Network boundaries have been reached

Outlook

- Open source releases of ParCC and UltraSFE
- IPP in other STC protocols
Questions and Answers

Thank you!
References


References


