FANTASTIC FOUR

Honest Majority Four-Party Secure Computation with Malicious Security

* Anders Dalskov (Aarhus University & Bitciso, Denmark)
* Daniel Escudero (Aarhus University, Denmark)
* Marcel Keller (CSIRO's Data61, Australia)
Outsourced Secure Computation

Privacy is maintained even if one party is actively corrupt.
Our Contribution

We propose a Four-Party protocol with the following features:

* Suitable for computations over $\mathbb{Z}_{2^k}$
Our Contribution

We propose a Four-Party protocol with the following features:

* Suitable for computations over $\mathbb{Z}_{2^k}$
* Tolerates one active corruption and provides G.O.D.
Our Contribution

We propose a Four-Party protocol with the following features:

* Suitable for computations over \( \mathbb{Z}_{2^k} \)
* Tolerates one active corruption and provides G.O.D.
* Same overall complexity than state-of-the-art protocols

Cross-checking (ASIACRYPT'18)

SWIFT (USENIX'21)

Six ring elements in total per multiplication
Our Contribution

We propose a Four-Party protocol with the following features:

* Suitable for computations over $\mathbb{Z}_{2^k}$
* Tolerates one active corruption and provides G.O.D.
* Same overall complexity than state-of-the-art protocols

- No function-dependent preprocessing!!
- No expensive checks based on large-degree Galois-Ring extensions!!

SIX RING ELEMENTS in total per multiplication
Our Contribution

We propose a Four-Party protocol with the following features:

* Suitable for computations over $\mathbb{Z}_{2^k}$
* Tolerates one active corruption and provides G.O.D.
* Same overall complexity than state-of-the-art protocols

- No function-dependent preprocessing!!
- No expensive checks based on large-degree Galois-Ring extensions!!

**Six Ring Elements in total per multiplication**

* We also introduce a Three-party protocol!!
Private Robustness

* Our protocol satisfies the traditional notion of G.O.D.

If the adversary tries to cheat then at least one honest party is identified, who can finish the computation in the clear.
Private Robustness

* Our protocol satisfies the traditional notion of G.O.D.

If the adversary tries to cheat, then at least one honest party is identified, who can finish the computation in the clear.

This is not realistic! (As identified by Bar Alon et al. CRYPTO’20)

None of the parties were trusted initially to carry the computation on their own...
Private Robustness

Our protocol satisfies the traditional notion of G.O.D.

If the adversary tries to cheat then at least one honest party is identified, who can finish the computation in the clear.

This is not realistic!! (As identified by Bar Alon et al. CRYPTO'20)

None of the parties were trusted initially to carry the computation on their own...

... Suddenly one party is trusted because it “did not cheat”
Private Robustness

Our approach:
Private Robustness

Our approach:

Semi-corrupt pair

1
Private Robustness

Our approach:

1. Semi-Corrupt Pair

2. Actively Secure Three-Party Protocol with Abort

Kick one of the two parties
Private Robustness

Our approach:

1. Semi-crupt Pair

2. Actively secure three-party protocol with abort
   - Kick one of the two parties

3. Passively secure two-party protocol
   - If the protocol aborts

Our approach:

WITH ABORT

R1 → R2

P1 → P2

P4

P1

P2

P3
Applications and Benchmarks

We apply our protocol to the tasks of multiclass deep learning and binary-label logistic regression.
Applications and Benchmarks

We apply our protocol to the tasks of multiclass deep learning and binary-label logistic regression.

* This requires us to introduce different novel primitives

- Probabilistic truncation
- Random bit generation
- Mixed-circuit computation

Making use of edaBits (Escudero et al. CRYPTO'20)
Table 2: Time and accuracy for MNIST with various models and protocols with one corrupted party. “SH 3PC” stands for the semi-honest protocol implemented MP-SPDZ while “Mal. 4PC” and “Mal. 3PC” stand for the protocols with abort presented in this work.

<table>
<thead>
<tr>
<th>No. dense layers</th>
<th>Seconds per epoch</th>
<th>Accuracy after $n$ epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SH 3PC</td>
<td>Mal. 4PC</td>
</tr>
<tr>
<td>1</td>
<td>12.2</td>
<td>22.1</td>
</tr>
<tr>
<td>2</td>
<td>28.2</td>
<td>42.4</td>
</tr>
<tr>
<td>3</td>
<td>33.8</td>
<td>51.1</td>
</tr>
</tbody>
</table>

Table 3: Time and accuracy for MNIST 4/9 distinction with various models and protocols with one corrupted party.

<table>
<thead>
<tr>
<th>No. dense layers</th>
<th>Seconds per epoch</th>
<th>Global comm. per epoch (MB)</th>
<th>Accuracy after $n$ epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SH 3PC</td>
<td>Mal. 4PC</td>
<td>Mal. 3PC</td>
</tr>
<tr>
<td>SWIFT [25]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>103.23</td>
<td>143.22</td>
</tr>
<tr>
<td>Ours</td>
<td>1</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3.7</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4.8</td>
<td>5.5</td>
</tr>
</tbody>
</table>

*AWS c5.9.large (different instances in same location)*
About the interpolation-based check

* In order to check the correctness of several secure multiplications, an interpolation-based check can be performed

Boneh et al. CRYPTO ’19, Boyle et al. CCS ’19

Used in several subsequent protocols: BLAZE SWIFT
About the interpolation-based check

* In order to check the correctness of several secure multiplications, an interpolation-based check can be performed.
  Boneh et al. CRYPTO '19, Boyle et al. CCS '19
  Used in several subsequent protocols: [BLAZE, SWIFT]

* Interpolation over a ring like $\mathbb{Z}_2[x]$ requires working with an extension $R = \mathbb{Z}_2[x]/(f(x))$ of degree in $[46, 72]$
About the interpolation-based check

* In order to check the correctness of several secure multiplications, an interpolation-based check can be performed

  Boneh et al. CRYPTO'19, Boyle et al. CCS'19

  Used in several subsequent protocols: [BLAZE, SWIFT]

* Interpolation over a ring like $\mathbb{Z}_2^k$, requires working with an extension $R = \mathbb{Z}_2^k[x]/(f(x))$ of degree in $[46, 72]$

  $\rightarrow$ less than 22000 multi. in $R$ per second (k=64, degree=46, Single-core, 2.8GHz)


About the interpolation-based check

* In order to check the correctness of several secure multiplications, an interpolation-based check can be performed:
  Boneh et al. CRYPTO’19, Boyle et al. CCS’19
  Used in several subsequent protocols: BLAZE, SWIFT

* Interpolation over a ring like $\mathbb{Z}_2^k$ requires working with an extension $R = \mathbb{Z}_2^k[x]/(f(x))$ of degree in $[46, 72]$,
  \[ \rightarrow \text{less than 22000 mult. in } R \text{ per second} \quad (k = 64, \text{ degree } = 46, \text{ single-core, } 2.8 \text{GHz i7}) \]
  \[ \rightarrow \text{less than 22 secure multiplications per second} \]
About the interpolation-based check

- In order to check the correctness of several secure multiplications, an interpolation-based check can be performed. Boneh et al. CRYPTO'19, Boyle et al. CCS'19. Used in several subsequent protocols: BLAZE, SWIFT.

- Interpolation over a ring like $\mathbb{Z}_2^k$ requires working with an extension $R = \mathbb{Z}_2[x]/(f(x))$ of degree in $[46, 72]$. Less than 22,000 mult. in $R$ per second ($k=64$, degree $=46$, single-core, 2.8GHz i7).
- Less than 22 secure multiplications per second.

Unless computation over $R$ improves, this is NOT practical.
Our Four-Party Protocol
Replicated Secret-Sharing

Given $x \in \mathbb{Z}_{2^k}$:

1. Sample $x = x_1 + x_2 + x_3 + x_4 \pmod{2^k}$
Replicated Secret-Sharing

Given $x \in \mathbb{Z}_{2^k}$:

1. Sample $x = x_1 + x_2 + x_3 + x_4 \pmod{2^k}$

2. Distribute shares:

$(x_2, x_3, x_4)$
Replicated Secret-Sharing

Given $x \in \mathbb{Z}_{2^k}$:

1. Sample $x = x_1 + x_2 + x_3 + x_4 \pmod{2^k}$

2. Distribute shares:

$(x_2, x_3, x_4)$ to $P_1$

$(x_1, x_3, x_4)$ to $P_2$
Replicated Secret-Sharing

\[ \text{Given } x \in \mathbb{Z}_{2^k} : \]

1. Sample \( x = x_1 + x_2 + x_3 + x_4 \pmod{2^k} \)

2. Distribute shares:

\[ (x_2, x_3, x_4), (x_1, x_3, x_4), (x_1, x_2, x_4), (x_1, x_2, x_3) \]
Secure Multiplication

\[ x = x_1 + x_2 + x_3 + x_4 \]

\[ y = y_1 + y_2 + y_3 + y_4 \]
Secure Multiplication

\[ x = x_1 + x_2 + x_3 + x_4 \]
\[ y = y_1 + y_2 + y_3 + y_4 \]

\[ x \cdot y = x_1 \cdot y_1 + x_2 \cdot y_2 + x_3 \cdot y_3 + x_4 \cdot y_4 \]
Goal: Get shares of each one of these summands and add these shares up

\[ \begin{bmatrix} x_1 \cdot y_1 \\ x_2 \cdot y_2 \end{bmatrix} + \begin{bmatrix} x_1 \cdot y_2 + y_1 \cdot x_2 \\ x_2 \cdot y_3 + y_2 \cdot x_3 \end{bmatrix} + \begin{bmatrix} x_1 \cdot y_3 + y_1 \cdot x_3 \\ x_2 \cdot y_4 + y_2 \cdot x_4 \end{bmatrix} + \begin{bmatrix} x_1 \cdot y_4 + y_1 \cdot x_4 \\ x_3 \cdot y_4 + y_3 \cdot x_4 \end{bmatrix} \]

\[ [x \cdot y] = \begin{bmatrix} x_1 \cdot y_1 + x_2 \cdot y_2 + x_3 \cdot y_3 + x_4 \cdot y_4 \end{bmatrix} \]
Getting Shares $[x_1 \cdot y_1]$
Getting Shares $[x_1 \cdot y_1]$ 

$$x_1 \cdot y_1 = x_1 \cdot y_1 + 0 + 0 + 0$$
Getting Shares \[ \[ x_1 \cdot y_1 \] \]

\[ x_1 \cdot y_1 = x_1 \cdot y_1 + 0 + 0 + 0 \]
Getting Shares \[ [x_1 \cdot y_1] \]

\[ x_1 \cdot y_1 = x_1 \cdot y_1 + 0 + 0 + 0 \]

\( (0,0,0) \)  \( (x_1, y_1, 0, 0) \)  \( (x_1 y_1, 0, 0) \)  \( (x_1 y, 0, 0) \)

\( P_1 \)  \( P_2 \)  \( P_3 \)  \( P_4 \)

No Interaction!
Getting Shares $[x_1 \cdot y_2 + y_1 \cdot x_2]$
Getting Shares \[
[z := x_1 \cdot y_2 + y_1 \cdot x_2] \]

known by \{ P_3, P_4 \}
Getting Shares \[ [x_1 \cdot y_2 + y_1 \cdot x_2] \]

\[ z := x_1 \cdot y_2 + y_1 \cdot x_2 \leftarrow \text{known by} \{ P_3, P_4 \} \]

Assume pre-shared keys:

\[ P_1, P_2, P_3, P_4 \]

\[ k, k, k, k \]
Getting Shares \[ x_1 \cdot y_2 + y_1 \cdot x_2 \]\n
\[ z := x_1 \cdot y_2 + y_1 \cdot x_2 \leftarrow \text{known by} \\begin{cases} P_3 \\kern1cm \ k \kern1cm \ k \end{cases} \]

Assume pre-shared keys:

\[ z = x + (2 - x) + 0 + 0 \]

\[ x = \text{PRG}_k(\cdot) \]

\[ (z - x, 0, 0) \quad (x, 0, 0) \quad (x, 2 - x, 0) \quad (x, 2 - x, 0) \]
Getting Shares \( [x_1 \cdot y_2 + y_1 \cdot x_2] \)

\[ z := x_1 \cdot y_2 + y_1 \cdot x_2 \leftarrow \text{known by} \{P_3, P_4\} \]

Assume pre-shared keys:

\[ z = y + (2 - y) + 0 + 0 \]

\[ y = \text{PRG}_k(\cdot) \]

**Problem!!**
$P_1$ complains if the hash does not match by sending the two hashes.
$P_1$ complains if the hash does not match by sending the two hashes

$P_3$ complains if the hash does not match the value $P_3$ sent to $P_1$
The diagram illustrates a network of devices and the interactions between them. There are three pairs of semi-corrupt devices, represented by circles: 

1. \( \{P_3, P_4\} \) is a pair of semi-corrupt devices.
2. \( \{P_3, P_3\} \) is a pair of semi-corrupt devices.
3. \( \{P_4, P_3\} \) is a pair of semi-corrupt devices.

Each pair has a hash function \( H(x) \) applied to their interactions.

- \( P_1 \) complains if the hash does not match by sending the two hashes.
- \( P_3 \) complains if the hash does not match the value \( P_3 \) sent to \( P_1 \).
- \( P_4 \) complains if the hash does not match the value \( P_4 \) sent to \( P_1 \).
We propose a Four-Party protocol with the following features:

- Suitable for computations over $\mathbb{Z}_{2^k}$
- Tolerates one active corruption and provides G.O.D.
- Same overall complexity than state-of-the-art protocols

- No function-dependent preprocessing!!
- No expensive checks based on large-degree Galois-Ring extensions!!

We also introduce a Three-party protocol!!
Thank you!