Introducing TopGear: an efficient and secure Zero-Knowledge proof for multiparty computation

Carsten Baum\textsuperscript{3}, Daniele Cozzo\textsuperscript{1} & Nigel P. Smart\textsuperscript{12}

\textsuperscript{1}KU Leuven
ESAT/imec-COSIC

\textsuperscript{2}University of Bristol
Department of Computer Science

\textsuperscript{3}Aarhus University
Department of Computer Science
Outline

• Multiparty computation
• SPDZ
• Achieving active security with zero-knowledge proofs
• The zero-knowledge proof in SPDZ
• Improving zero-knowledge proof: TopGear
• Experimental results
Multiparty computation

- Secure: privacy, correctness, fairness etc...
SPDZ preprocessing model

Preprocessing

Public key algorithms
Randomness (Beaver triples)

Circuit evaluation

Information theoretic primitives

\[ X \]

\[ + \]
The preprocessing phase

• Multiplications computed using Beaver method
  – Beaver triples: \((a,b,c)\) with \(a,b\) randomly selected from \(R_p\) and \(c = ab\).
  – Triples are authenticated and shared among parties

• This is done by using a distributed (somewhat) homomorphic scheme (BGV)
  – Ring-LWE scheme with plaintext space \(R_p = \mathbb{Z}_p[X] / (X^N+1)\)
Beaver triples

1. $P_i$ samples $a_i, b_i, c_i$

2. $P_i$ computes $\text{BGV.Enc}(a_i), \text{BGV.Enc}(b_i), \text{BGV.Enc}(c_i)$

3. $P_i$ proves knowledge of $\text{BGV.Enc}(a_i), \text{BGV.Enc}(b_i), \text{BGV.Enc}(c_i)$ and broadcasts them

4. Parties compute $ct_c = (\sum_{\text{BGV.Enc}}a_i)(\sum_{\text{BGV.Enc}}b_i) - (\sum_{\text{BGV.Enc}}c_i)$, with $ct_c = \text{BGV.Enc}(\gamma)$ for some $\gamma$.

5. Parties jointly run $\text{BGV.Dec}(ct_c)$ and get $\gamma = \text{BGV.Dec}(ct_c)$

6. $P_1$‘s shared triple is $(a_1, b_1, c_1 + \gamma)$

7. $P_i$‘s shared triple is $(a_i, b_i, c_i)$

8. At the end of the process each party holds a share of the triple $a, b, c$
Preprocessing phase

\[ \text{To avoid fault attacks, need to check each BGV ciphertext is well formed} \]

- Noise must be bounded
- Prove that \( ||x^i||, ||r^i|| < B \)
**Schnorr-like approach**

<table>
<thead>
<tr>
<th>Prover ([i])</th>
<th>Verifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>(y^i, t^i \leftarrow D)</td>
<td></td>
</tr>
<tr>
<td>(\epsilon^i = t^i + er^i)</td>
<td></td>
</tr>
<tr>
<td>(\sigma^i = y^i + e x^i)</td>
<td></td>
</tr>
<tr>
<td>(com_i = ct^i_0 \cdot y^i + t^i)</td>
<td></td>
</tr>
<tr>
<td>(chal_i = e)</td>
<td></td>
</tr>
<tr>
<td>(resp_i = \sigma^i, \epsilon^i)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

One ciphertext per run, no auxiliary ciphertexts

repeat \(n\) times

\(e \in \{0, 1\}\)

\(ct^i_0 \cdot \sigma^i + \epsilon^i = (ct^i_0 \cdot y^i + t^i) + e \cdot ct^i_1\)

\(||\sigma^i||, ||\epsilon^i|| < 2B'\)
Schnorr-like approach

- **Issue 1**
  - Not zero-knowledge unless Prover masks the secret with large values
  - Blows up parameters
  - Introduces the adversarial language
    \[ \mathcal{L'} = \{ (x, r) \text{ s.t. } ||x||, ||r|| \leq 2^{\text{ZK}_\text{sec}B} \} \]
  - As opposite to the honest language
    \[ \mathcal{L} = \{ (x, r) \text{ s.t. } ||x||, ||r|| \leq B \} \]
  - \( \mathcal{L'}/\mathcal{L} = \text{slack} \)
Schnorr-like approach

• Issue 2
  - Binary challenge set
  - Dishonest prover has ½ chances to cheat
  - Need to repeat the protocol \( \text{sec} \) times to achieve \( 2^{-\text{sec}} \) soundness security
  - In general, enlarging the challenge space does not help

we only prove \( \mathcal{D}' = \{ (x, r) \text{ s.t. } \kappa \cdot ||x||, \kappa \cdot ||r|| \leq 2^{2K_{\text{sec}}B} \} \)
SPDZ’12 – amortization

**Prover**[$i$]

\[ y^i, t^i \leftarrow D \]

\[ \epsilon^i = t^i + e r^i \]

\[ \sigma^i = y^i + e x^i \]

**Verifier**

\[ \text{com}_i = c t_0^i \cdot y^i + t^i \]

\[ \text{chal}_i = e \]

\[ \text{resp}_i = \sigma^i, \epsilon^i \]

\[ c t_0^i \cdot \sigma^i + \epsilon^i = (c t_0^i y^i + t^i) + e c t_1^i \]

\[ \|\sigma^i\|, \|\epsilon^i\| < 2 B' \]

\[ U = \text{sec} \] ciphertexts at once, \[ V = 2U - 1 \] auxiliary ciphertexts

Repeat \( n \) times
SPDZ’12 – amortization

- Still binary challenge space
- Slack increases by $2^{\text{sec}}$
- Needs $\text{sec} + U$ ciphertexts for proving $U$ ciphertexts
- Memory consumption roughly $3U = 3\text{sec}$
- Need to keep $U$ small
  - Limits security
Beaver triples

1. $P_i$ samples $a_i$, $b_i$, $c_i$
2. $P_i$ computes $\text{BGV.Enc}(a_i)$, $\text{BGV.Enc}(b_i)$, $\text{BGV.Enc}(c_i)$
3. $P_i$ proves knowledge of $\text{BGV.Enc}(a_i)$, $\text{BGV.Enc}(b_i)$, $\text{BGV.Enc}(c_i)$ and broadcasts them
4. Parties compute $ct_c = (\sum \text{BGV.Enc}(a_i))(\sum \text{BGV.Enc}(b_i)) - (\sum \text{BGV.Enc}(c_i))$, with $ct_c = \text{BGV.Enc}(\gamma)$ for some $\gamma$.
5. Parties jointly run $\text{BGV.Dec}(ct_c)$ and get $\gamma = \text{BGV.Dec}(ct_c)$
6. $P_1$‘s shared triple is $(a_1, b_1, c_1 + \gamma)$
7. $P_i$‘s shared triple is $(a_i, b_i, c_i)$
8. At the end of the process each party holds a share of the triple $a, b, c$
Overdrive’18 – sum of statements

**Prover** \([i]\)

\[ y^i, t^i \leftarrow D \]

\[ \epsilon^i = t^i + e r^i \]

\[ \sigma^i = y^i + e x^i \]

**Verifier**

\[ \text{com}_i = ct^i_0 \cdot y^i + t^i \]

\[ \text{chal} = e \]

\[ \text{resp}_i = \sigma^i, \epsilon^i \]

\[ c t^i_0 \sum_{i=1}^{n} \sigma^i + \sum_{i=1}^{n} \epsilon^i = \sum_{i=1}^{n} (c t^i_0 \cdot y^i + t^i) + e \sum_{i=1}^{n} c t^i_1 \]

\[ \| \sum_i \sigma^i \|, \| \sum_i \epsilon^i \| < 2nB' \]

\[ U = \text{sec ciphertexts at once}, \ V = 2U - 1 \text{ auxiliary ciphertexts} \]

Do it only once
Overdrive – sum of statements

• Do not need to repeat the protocol
• Amortization as in CD09
  - High soundness slack
• Memory consumption is still roughly $3U = 3_{sec}$
• Security not optimal
  - High security parameter means more memory usage
$U = 2V$ ciphertexts at once, $V = \frac{sec}{\log(2N+1)}$

Do it only once
TopGear

• Larger challenge space
  - Use \( \{ \pm X^i \cup \{0\} \}_{i=1,\ldots,2N} \)
    - Extraction as in BBC\(^{+}18\)
    - Uses crucially the fact that in \( R_q \) the element \( 2/(X^i-X^j) \) has norm less than 1
• Better amortization while maintaining the same level of security
  - \( V = \frac{sec}{\log_2(2N+1)}, \ U = 2V \)
  - \( N \) typically of 15 bits
• More triples
TopGear

• We do not care about slack
  - Distributed decryption uses modulo switching operations which reduces the slack
  - Even better with new amortization

• We can fix extraction
  - Proof lies in $\mathcal{D}' = \{ (x, r) \text{ s.t. } 2 \cdot ||x||, 2 \cdot ||r|| \leq 2^{z_{K,sec}B} \}$
  - Scheme is homomorphic hence multiply ciphertexts by 2
Implementation

• SCALE-MAMBA v. 1.4
  - https://github.com/KULeuven-COSIC/SCALE-MAMBA

• TopGear
Memory consumption (2 parties, $\text{ZK\_sec} = \text{sec} = 40$)\(^1\)

<table>
<thead>
<tr>
<th>$t_{\text{Tr}}$</th>
<th>$t_{\text{ZK}}$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>41</td>
<td>68</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>38</td>
<td>68</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>28</td>
<td>49</td>
<td>75</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>52</td>
<td>81</td>
<td>98</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$t_{\text{Tr}}$</th>
<th>$t_{\text{ZK}}$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>9</td>
<td>15</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>10</td>
<td>15</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>12</td>
<td>17</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>14</td>
<td>17</td>
<td>21</td>
<td>33</td>
<td></td>
</tr>
</tbody>
</table>

**Overdrive**

**TopGear**

\(^1\) Tested on i7-7700K CPUs in a LAN setting.
Memory consumption (2 parties, $\text{ZK\_sec} = \text{sec} = 128$)\(^1\)

<table>
<thead>
<tr>
<th>$t_{tr}$</th>
<th>$t_{zk}$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>70</td>
<td>98</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>72</td>
<td>98</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>73</td>
<td>98</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>76</td>
<td>98</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$t_{tr}$</th>
<th>$t_{zk}$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>11</td>
<td>19</td>
<td>33</td>
<td>63</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>12</td>
<td>18</td>
<td>33</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>14</td>
<td>21</td>
<td>34</td>
<td>64</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>16</td>
<td>24</td>
<td>39</td>
<td>70</td>
</tr>
</tbody>
</table>

Overdrive

TopGear

\(^1\) Tested on i7-7700K CPUs in a LAN setting.
Triples per second (2 parties, $\tau_{ZK\_sec} = \tau_{sec} = 40$)\(^1\)

<table>
<thead>
<tr>
<th>$t_{Tr}$</th>
<th>$t_{ZK}$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1503</td>
<td>1602</td>
<td>1562</td>
<td>1335</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>1488</td>
<td>2347</td>
<td>2212</td>
<td>1976</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>1272</td>
<td>1876</td>
<td>2150</td>
<td>1865</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>976</td>
<td>1307</td>
<td>1464</td>
<td>1533</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$t_{Tr}$</th>
<th>$t_{ZK}$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>2806</td>
<td>2829</td>
<td>2846</td>
<td>2752</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>3809</td>
<td>4851</td>
<td>4709</td>
<td>4540</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>5672</td>
<td>6086</td>
<td>6692</td>
<td>6293</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>4666</td>
<td>5635</td>
<td>6084</td>
<td>5636</td>
</tr>
</tbody>
</table>

**Overdrive**

**TopGear**

\(^1\) Tested on i7-7700K CPUs in a LAN setting.
Triples per second (2 parties, $ZK_{\text{sec}} = \text{sec} = 128$)\(^1\)

<table>
<thead>
<tr>
<th>$t_{Tr}$</th>
<th>$t_{ZK}$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1240</td>
<td>1369</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>1426</td>
<td>1834</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>1231</td>
<td>1612</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>940</td>
<td>1129</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$t_{Tr}$</th>
<th>$t_{ZK}$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1743</td>
<td>2775</td>
<td>2692</td>
<td>2569</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>4251</td>
<td>4622</td>
<td>4572</td>
<td>4021</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>3943</td>
<td>3712</td>
<td>4955</td>
<td>5000</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>3265</td>
<td>3272</td>
<td>5254</td>
<td>5041</td>
</tr>
</tbody>
</table>

Overdrive

TopGear

\(^1\) Tested on i7-7700K CPUs in a LAN setting.
Questions?

https://eprint.iacr.org/2019/035