New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning

Ivan Damgård\textsuperscript{1} Daniel Escudero\textsuperscript{1} Tore Frederiksen\textsuperscript{2} Marcel Keller\textsuperscript{3} Peter Scholl\textsuperscript{1} Nikolaj Volgushev\textsuperscript{2}

May 19, 2019

\textsuperscript{1}Aarhus University, Denmark
\textsuperscript{2}Alexandra Institute, Denmark
\textsuperscript{3}Data61, CSIRO, Australia

\textsuperscript{a}This work has been supported by the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme under grant agreements No 669255 (MPCPRO), No 731583 (SODA) and the Danish Independent Research Council under Grant-ID DFF6108-00169 (FoCC).
Introduction
MPC

Alice

Bob

Trusted Party

Charlie

Dave
MPC

Alice $x_1$

Bob $x_2$

Charlie $x_3$

Dave $x_4$

Trusted Party
MPC

Alice  Bob

Charlie  Dave
Many different approaches to MPC

<table>
<thead>
<tr>
<th>Circuits over $\mathbb{F}_2$</th>
<th>Circuits over $\mathbb{F}_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Garbled Circuits</td>
<td>• BGW</td>
</tr>
<tr>
<td>• BMR</td>
<td>• BeDOZa</td>
</tr>
<tr>
<td>• GMW</td>
<td>• SPDZ</td>
</tr>
<tr>
<td>• ...</td>
<td>• MASCOT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Circuits over $\mathbb{Z}_{2^k}$ (dishonest majority and active security)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• SPDZ$_{2^k}$, Cramer et al. CRYPTO’18.</td>
</tr>
</tbody>
</table>
Benefits of $\mathbb{Z}_{2^k}$

(Already conjectured in SPD$\mathbb{Z}_{2^k}$)
Benefits of $\mathbb{Z}_{2^k}$

(Already conjectured in SPD$\mathbb{Z}_{2^k}$)

- Computation modulo $2^{64}$ or $2^{32}$ can be done natively in hardware.
Benefits of $\mathbb{Z}_{2^k}$

(Already conjectured in SPD$\mathbb{Z}_{2^k}$)

- Computation modulo $2^{64}$ or $2^{32}$ can be done natively in hardware.
- Easier compilation of pre-existing programs to MPC programs.
Benefits of $\mathbb{Z}_{2^k}$

(Already conjectured in SPD$\mathbb{Z}_{2^k}$)

- Computation modulo $2^{64}$ or $2^{32}$ can be done natively in hardware.
- Easier compilation of pre-existing programs to MPC programs.
- Computation modulo powers of 2 should be “more compatible” with computation modulo 2.
Our Contribution

New sub-protocols for \( \text{SPD}\mathbb{Z}_{2^k} \)

We expand \( \text{SPD}\mathbb{Z}_{2^k} \) with a series of sub-protocols to enhance the potential range of applications.

- Arithmetic-Binary share conversions
- Random-bit generation
- Bit-decomposition
- Secure truncation, comparison and equality check.
We implement the SPD$\mathbb{Z}_{2^k}$ protocol in Java, as part of the FRamework for Efficient Secure COmputation (FRESCO).

- Our implementation contains several optimizations that can be of independent interest.
- In the microbenchmarks we observe several improvements with respect to other protocols over fields.
Applications to Secure Machine Learning

We illustrate the benefits of our techniques by performing certain ML tasks in $\text{SPD}\mathbb{Z}_{2^k}$ and observe several improvements with respect to other protocols over fields. We consider:

- Secure evaluation of Decision Trees
- Secure evaluation of Support Vector Machines
$\text{SPD}_{\mathbb{Z}_{2^k}}$
SPD$\mathbb{Z}_{2^k}$ in a nutshell

Additive Authenticated Secret-Sharing over $\mathbb{Z}_{2^k}$

$x \in \mathbb{Z}_{2^k}$ is shared, denoted by $[x]_{2^k}$, if

- Each $P_i$ has $x^i, \alpha^i, m^i \in \mathbb{Z}_{2^{k+s}}$
- $\sum x^i \equiv_{k+s} x'$ with $x' \equiv_k x$
- $\sum \alpha^i \equiv_{k+s} \alpha$, where $\alpha \in \mathbb{Z}_{2^s}$ is a random global key
- $\sum m^i \equiv_{k+s} \alpha \cdot x'$

$x \equiv y \mod 2^\ell$ is abbreviated by $x \equiv_{\ell} y$
Secure computation with preprocessing

**Input phase**

\[
[x_i]_{2^k} = (x_i - r_i) + [r_i]_{2^k}
\]

where \(x_i\) are the inputs and \((r_i, [r_i]_{2^k})\) is preprocessed.

**Addition gates**

\[
[x + y]_{2^k} = [x]_{2^k} + [y]_{2^k}
\]

**Multiplication gates**

\[
[x \cdot y]_{2^k} = [c]_{2^k} + (x - a) \cdot [b]_{2^k} + (y - b) \cdot [a]_{2^k} + (x - a)(y - b)
\]

where \(([a]_{2^k}, [b]_{2^k}, [c]_{2^k})\) is preprocessed with \(c = a \cdot b\).
Primitives for MPC Modulo $2^k$
\[ \cdot \rightarrow [\cdot]_{2^k} \]

TinyOT

Triple

\[ \cdot \rightarrow [\cdot]_{2^k} \]

Random Bit

\[ [\cdot]_{2^k} \rightarrow [\cdot]_2 \]

Decision Trees

BitDec

SVM
Generating Random Bits $[b]_{2^k}$ (Intuition)

**Ideal Protocol**

1. Sample $[r]_{2^k}$ at random and let $[a]_{2^k} = [r^2]_{2^k}$.
2. Open $a$. Let $c$ be some square root of $a$.
3. Compute $[d]_{2^k} = c^{-1}[r]_{2^k}$.
   - Now $d$ is a random square root of 1, so $d \in_R \{-1, +1\}$.
4. Output $[b]_{2^k}$, where $b = (d + 1)/2$. 
Generating Random Bits $[b]_{2^k}$ (Intuition)

**Ideal Protocol**

1. Sample $[r]_{2^k}$ at random and let $[a]_{2^k} = [r^2]_{2^k}$.
2. Open $a$. Let $c$ be some square root of $a$.
3. Compute $[d]_{2^k} = c^{-1}[r]_{2^k}$.
   - Now $d$ is a random square root of 1, so $d \in_R \{-1, +1\}$.
4. Output $[b]_{2^k}$, where $b = (d + 1)/2$. 
Generating Random Bits $[b]_{2^k}$ (Intuition)

**Actual Protocol**

1. Sample $[r]_{2^{k+2}}$ at random, where $r$ is odd, and let $[a]_{2^{k+2}} = [r^2]_{2^{k+2}}$.
2. Open $a$. Let $c$ be some square root of $a$.
3. Compute $[d]_{2^{k+2}} = c^{-1}[r]_{2^{k+2}}$
   - Now $d$ is a random square root of $1$ mod $2^{k+2}$, so $d \in R \{-1, +1, -1 + 2^{k+1}, +1 + 2^{k+1}\}$.
4. Output $[b]_{2^k}$, where $b \equiv_k (d + 1)/2$. 
Share Conversions

\([b]_{2^k} \rightarrow [b]_2\)

Local reduction modulo 2.\(^a\)

\(^a\)In fact, it is reduction modulo \(2^{s+1}\) for the extra \(s\) “MAC” bits.

\([b]_2 \rightarrow [b]_{2^k}\)

1. Sample a random bit \([r]_{2^k} (r \in \mathbb{Z}_2)\)
2. Convert \([r]_{2^k}\) to \([r]_2\).
3. Open \([c] = [b]_2 \oplus [r]_2\)
4. Output \([b]_{2^k} = [r]_{2^k} + [c]_{2^k} - 2[r]_{2^k}[c]_{2^k}\)
Bit Decomposition: \([x]_2^k \rightarrow ([x_0]_2^k, \ldots, [x_{k-1}]_2^k)\)

1. Sample random bits \([r_0]_2^k, \ldots, [r_{k-1}]_2^k\) and let \([r]_2^k = \sum_{i=0}^{k-1} 2^i [r_i]_2^k\).
2. Compute \([a]_2^k = [x]_2^k - [r]_2^k\) and open \(a\).
3. Convert \(([r_0]_2^k, \ldots, [r_{k-1}]_2^k)\) to \(([r_0]_2, \ldots, [r_{k-1}]_2)\).
4. Compute the binary circuit
\[
([x_0]_2, \ldots, [x_{k-1}]_2) = \text{ADD} \left( (a_0, \ldots, a_{k-1}), ([r_0]_2, \ldots, [r_{k-1}]_2) \right).
\]
5. Convert the result \(([x_0]_2, \ldots, [x_{k-1}]_2)\) to \(([x_0]_2^k, \ldots, [x_{k-1}]_2^k)\).
Implementation and Benchmarks
Throughput in elements per second for the online phase of micro operations over 1 Gbps network. The factor columns express the runtime improvement factor of SPD$\mathbb{Z}_{2^k}$ over SPDZ in FRESCO.

<table>
<thead>
<tr>
<th></th>
<th>$k = 32$</th>
<th></th>
<th></th>
<th>$k = 64$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SPD$\mathbb{Z}_{2^k}$ ($\sigma = 26$)</td>
<td>SPDZ ($\sigma = 26$)</td>
<td>Factor</td>
<td>SPD$\mathbb{Z}_{2^k}$ ($\sigma = 57$)</td>
<td>SPDZ ($\sigma = 57$)</td>
</tr>
<tr>
<td>Multiplication</td>
<td>687041</td>
<td>141346</td>
<td>4.9x</td>
<td>522258</td>
<td>114071</td>
</tr>
<tr>
<td>Equality</td>
<td>15334</td>
<td>3213</td>
<td>4.8x</td>
<td>6902</td>
<td>1282</td>
</tr>
<tr>
<td>Comparison</td>
<td>9153</td>
<td>1769</td>
<td>5.2x</td>
<td>4514</td>
<td>756</td>
</tr>
</tbody>
</table>
Online phase benchmarking of SVM evaluation over 1 Gbps network. The factor columns express the runtime improvement factor of $\text{SPD}_{2k}$ over SPDZ in FRESCO. Times are in milliseconds per sample.

$$k = 32, \sigma = 26$$

$$k = 64, \sigma = 57$$

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Num. Classes, Features</th>
<th>Batch Size</th>
<th>$\text{SPD}_{2k}$</th>
<th>SPDZ</th>
<th>Factor</th>
<th>$\text{SPD}_{2k}$</th>
<th>SPDZ</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR</td>
<td>10, 2048</td>
<td>1</td>
<td>82 ms</td>
<td>214 ms</td>
<td>2.6x</td>
<td>99 ms</td>
<td>255 ms</td>
<td>2.6x</td>
</tr>
<tr>
<td>MIT</td>
<td>67, 2048</td>
<td>1</td>
<td>379 ms</td>
<td>1318 ms</td>
<td>3.5x</td>
<td>499 ms</td>
<td>1582 ms</td>
<td>3.2x</td>
</tr>
<tr>
<td>ALOI</td>
<td>463, 128</td>
<td>1</td>
<td>242 ms</td>
<td>857 ms</td>
<td>3.5x</td>
<td>362 ms</td>
<td>1312 ms</td>
<td>3.6x</td>
</tr>
<tr>
<td>CIFAR</td>
<td>10, 2048</td>
<td>5</td>
<td>39 ms</td>
<td>168 ms</td>
<td>4.3x</td>
<td>57 ms</td>
<td>209 ms</td>
<td>3.7x</td>
</tr>
<tr>
<td>MIT</td>
<td>67, 2048</td>
<td>5</td>
<td>225 ms</td>
<td>1101 ms</td>
<td>4.9x</td>
<td>294 ms</td>
<td>1428 ms</td>
<td>4.9x</td>
</tr>
<tr>
<td>ALOI</td>
<td>463, 128</td>
<td>5</td>
<td>162 ms</td>
<td>741 ms</td>
<td>4.6x</td>
<td>244 ms</td>
<td>1220 ms</td>
<td>5.0x</td>
</tr>
</tbody>
</table>
Online phase benchmarking of evaluation of decision trees over 1 Gbps network. The factor columns express the runtime improvement factor of $\text{SPDZ}_2^k$ over SPDZ in FRESCO. Times are in milliseconds per sample.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Depth, Num. Features</th>
<th>Batch Size</th>
<th>$k = 32, \sigma = 26$</th>
<th>$k = 64, \sigma = 57$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\text{SPDZ}_2^k$</td>
<td>SPDZ</td>
</tr>
<tr>
<td>Hill Valley</td>
<td>3, 100</td>
<td>1</td>
<td>21 ms</td>
<td>24 ms</td>
</tr>
<tr>
<td>Spambase</td>
<td>6, 57</td>
<td>1</td>
<td>48 ms</td>
<td>104 ms</td>
</tr>
<tr>
<td>Diabetes</td>
<td>9, 8</td>
<td>1</td>
<td>80 ms</td>
<td>215 ms</td>
</tr>
<tr>
<td>Hill Valley</td>
<td>3, 100</td>
<td>5</td>
<td>6 ms</td>
<td>10 ms</td>
</tr>
<tr>
<td>Spambase</td>
<td>6, 57</td>
<td>5</td>
<td>14 ms</td>
<td>40 ms</td>
</tr>
<tr>
<td>Diabetes</td>
<td>9, 8</td>
<td>5</td>
<td>41 ms</td>
<td>185 ms</td>
</tr>
</tbody>
</table>
Triple Generation Throughput

- SPDZ$_{2k}$ ($k = 32$, $\sigma = 26$)
- SPDZ$_{2k}$ ($k = 64$, $\sigma = 57$)
- Mascot (128 bit field)
- Overdrive ($k = 64$ (128 bit field), $\sigma = 57$)
- Overdrive ($k = 32$ (64 bit field), $\sigma = 40$)

(a) WAN (50 Mbps, 100 ms latency)  
(b) LAN (1 Gbps, 0.1 ms latency)  
(c) LAN (10 Gbps, 0.1 ms latency)
Conclusions

• We implemented the SPD\(\mathbb{Z}_{2^k}\) protocol along with practical primitives for MPC mod \(2^k\).
• We saw up to a 5-fold improvement in computation for various tasks, and up to a 85-fold reduction in online communication costs for secure comparison, as compared to the field setting.

Future Work

• Close the gap for the preprocessing.
• Expand the range of applications for computation modulo \(2^k\).
Thank you!