SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority[a]

Ronald Cramer[1] Ivan Damgård[2] Daniel Escudero[2] Peter Scholl[2] Chaoping Xing[3]

August 21, 2018

[1]CWI, Amsterdam

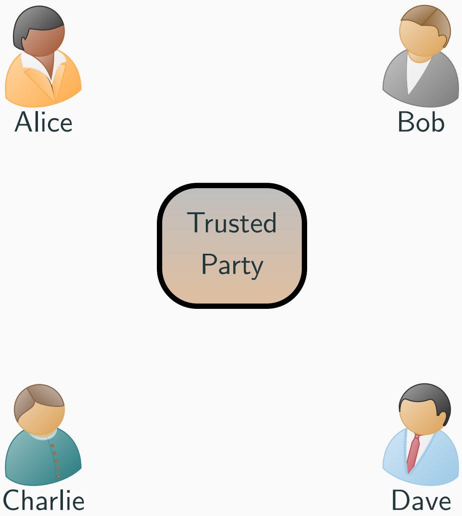
[2]Aarhus University, Denmark

[3]Nanyang Technological University, Singapore

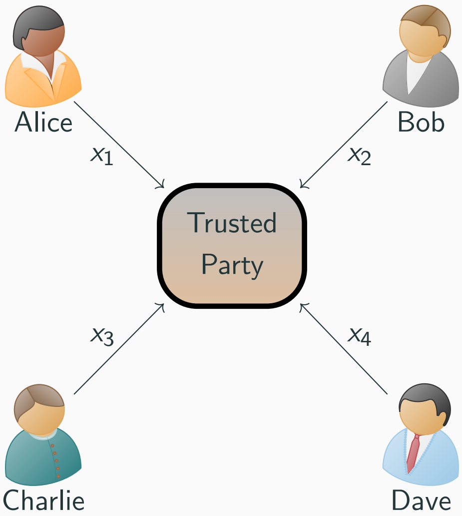
[a]Supported by the European Research Council (ERC); the European Union's Horizon 2020 research and innovation programme; the European Union's Horizon 2020 research and innovation programme and the Danish Independent Research Council.

Introduction

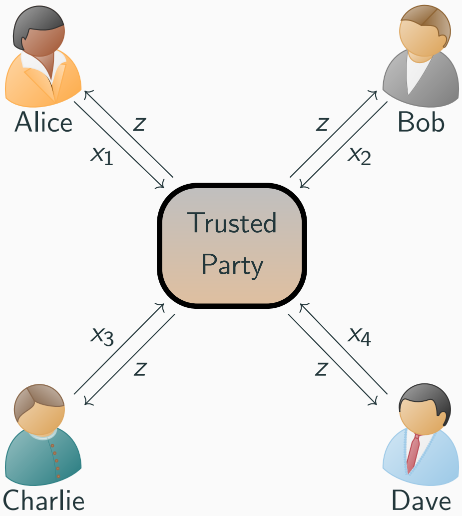
MPC



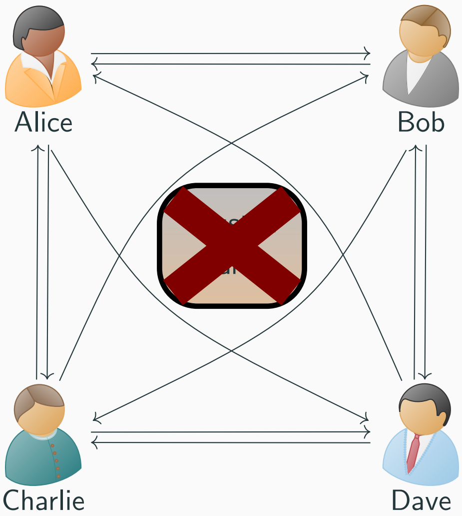
MPC



MPC



MPC



Many different approaches

Circuits over $\mathbb{F}_2$

- Garbled Circuits
- BMR
- GMW
- $\cdots$

Circuits over $\mathbb{F}_p$

- BGW
- BeDOZa
- SPDZ
- MASCOT
- $\cdots$

AARHUS UNIVERSITY
Scalable Oblivious Data Analytics

Many different approaches

Circuits over $\mathbb{F}_2$

- Garbled Circuits
- BMR
- GMW
- $\cdots$

Circuits over $\mathbb{F}_p$

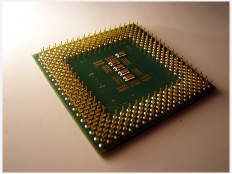
- BGW
- BeDOZa
- SPDZ
- MASCOT
- $\cdots$

Few works address circuits over $\mathbb{Z}_{2^k}$ with active security

Why should we care about computation modulo 2^k?

Closer to standard CPUs

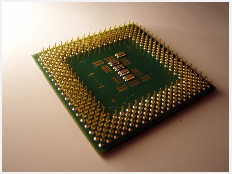
- Efficiency improvement
- Simple compilation of existing 32/64-bit code into arithmetic circuits.
- Simplified implementations



Why should we care about computation modulo 2^k?

Closer to standard CPUs

- Efficiency improvement
- Simple compilation of existing 32/64-bit code into arithmetic circuits.
- Simplified implementations



Completeness result

- Filling a gap in the theory of MPC
- Just for fun!

Some works on this direction

Cramer et al, EUROCRYPT 2003	Actively secure MPC over black-box rings	Mostly a feasibility result, honest majority
Bogdanov et al, ESORICS 2008 (Sharemind); Araki et al, CCS 2016	Computation over $\mathbb{Z}_{2^k}$	Passive security, $n = 3$ and $t = 1$
Damgård, Orlandi, Simkin, CRYPTO 2018	Compiler from passive to active security for arbitrary rings	Small number of corruptions

Why is it so difficult?

Pratical protocols use information-theoretic MACs over finite fields.

Problems with $\mathbb{Z}_{2^k}$

- Zero-divisors!
- Non-invertible elements!
- $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ is not a 2-universal hash function!

Open problem

Design an efficient homomorphic authentication scheme modulo 2^k

Our contributions

1. **A new additively homomorphic authentication scheme over $\mathbb{Z}_{2^k}$**
    - Efficient
    - Number-theoretic tricks
    - Fine-grained analysis of batch-checking

Our contributions

1. **A new additively homomorphic authentication scheme over $\mathbb{Z}_{2^k}$**
    - Efficient
    - Number-theoretic tricks
    - Fine-grained analysis of batch-checking
2. **Triples generation**
    - Communication complexity: $O((k+s)^2)$ bits per multiplication gate.
    - Roughly twice the communication cost of MASCOT

Our contributions

1. **A new additively homomorphic authentication scheme over $\mathbb{Z}_{2^k}$**
   - Efficient
   - Number-theoretic tricks
   - Fine-grained analysis of batch-checking
2. **Triples generation**
   - Communication complexity: $O((k+s)^2)$ bits per multiplication gate.
   - Roughly twice the communication cost of MASCOT
3. **A protocol for MPC over $\mathbb{Z}_{2^k}$**
   - $O(|C|n)$ operations over $\mathbb{Z}_{2^{k+s}}$
   - Amortized communication complexity of online phase: $O(|C|k)$ bits

SPDZ

Additive Secret sharing with MACs

We denote by $[x]$ the following

- $\sum x^i = x$.
- $\sum \alpha^i = \alpha$, where α is a random global key
- $\sum m^i = \alpha \cdot x$.

P_i has x^i, α^i, m^i

Additive Secret sharing with MACs

We denote by $[x]$ the following

- $\sum x^i = x$.
- $\sum \alpha^i = \alpha$, where α is a random global key
- $\sum m^i = \alpha \cdot x$.

P_i has x^i, α^i, m^i

Important!

$[x+y] = [x] + [y]$, $[c \cdot x] = c \cdot [x]$ and $[x+c] = [x] + c$ can be computed locally.

Secure computation with preprocessing

Input phase

$$[x_i] = \underbrace{(x_i - r_i)}_{\text{open}} + [r_i]$$

where x_i are the inputs and $(r_i, [r_i])$ is preprocessed.

Addition gates

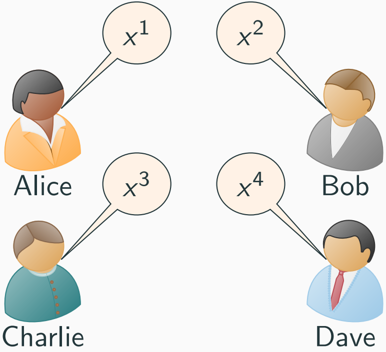
$$[x + y] = [x] + [y]$$

Multiplication gates

$$[x \cdot y] = [c] + \underbrace{(x - a)}_{\text{open}} \cdot [b] + \underbrace{(y - b)}_{\text{open}} \cdot [a] + \underbrace{(x - a)}_{\text{open}} \underbrace{(y - b)}_{\text{open}}$$

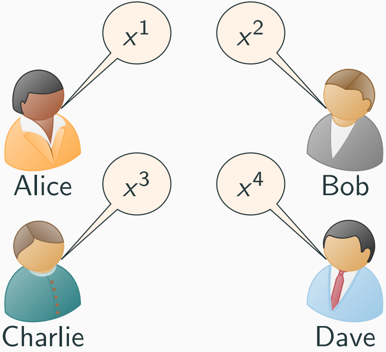
where $([a], [b], [c])$ is preprocessed with $c = a \cdot b$.

Reconstruction of $[x]$

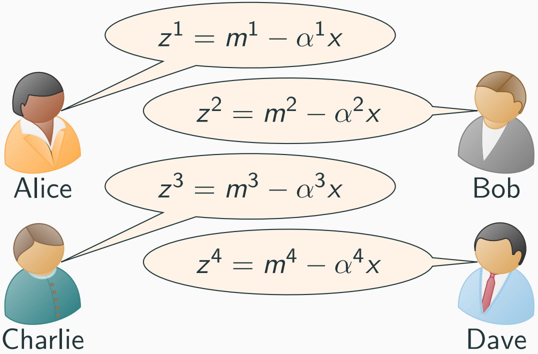


$$\sum_{i=1}^{n} x^i = x$$

Reconstruction of $[x]$



$\sum_{i=1}^{n} x^i = x$



Check that $\sum_{i=1}^{n} z_i = 0$

Security Analysis

Adversarial behavior can cause: $x' = x + \delta$ and $z' = z + \Delta$ with $\delta \neq 0$.

$\Rightarrow$ Adversary knows Δ and δ such that $\delta \cdot \alpha = \Delta$.

$\Rightarrow$ The adversary guesses $\alpha = \delta^{-1} \cdot \Delta$

$\Rightarrow$ Probability at most $1/|\mathbb{F}|$

Security Analysis

Adversarial behavior can cause: $x' = x + \delta$ and $z' = z + \Delta$ with $\delta \neq 0$.

$\Rightarrow$ Adversary knows Δ and δ such that $\delta \cdot \alpha = \Delta$.

$\Rightarrow$ The adversary guesses $\alpha = \delta^{-1} \cdot \Delta$

$\Rightarrow$ Probability at most $1/|\mathbb{F}|$

This does not work modulo 2^k

The equation $\Delta \equiv \alpha \cdot \delta \bmod 2^k$ can be satisfied with high probability

- Main problem: δ may not be invertible modulo 2^k.
- For instance: $\delta = 2^{k-1}$ and $\Delta = 0$

SPD$\mathbb{Z}_{2^k}$

Our solution

The computation is done in $\mathbb{Z}_{2^{k+s}}$ but correctness is only guaranteed modulo 2^k

Our solution

The computation is done in $\mathbb{Z}_{2^{k+s}}$ but correctness is only guaranteed modulo 2^k

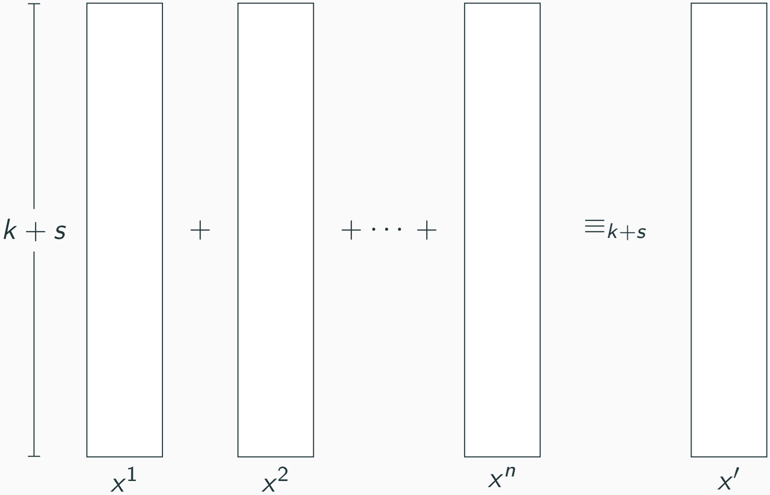
To share $x \in \mathbb{Z}_{2^k}$:

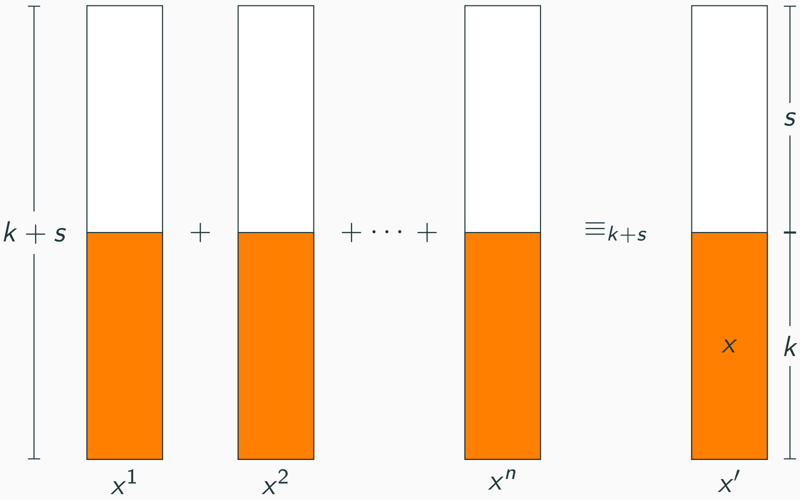
We denote by $[x]$ the following

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

P_i has $x^i, \alpha^i, m^i \in \mathbb{Z}_{2^{k+s}}$

$x \equiv y \bmod 2^\ell$ will be abbreviated by $x \equiv_\ell y$


$k+s$
$x^1 + x^2 + \cdots + x^n \equiv_{k+s} x'$



k + s
+
+ · · · +
≡k+s
s
x
k
x1
x2
xn
x′

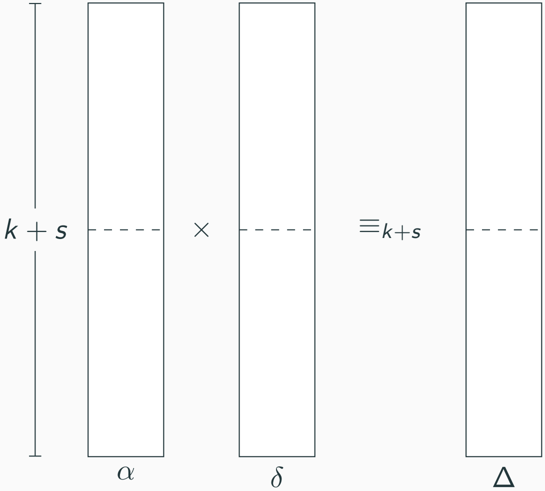
Security Analysis

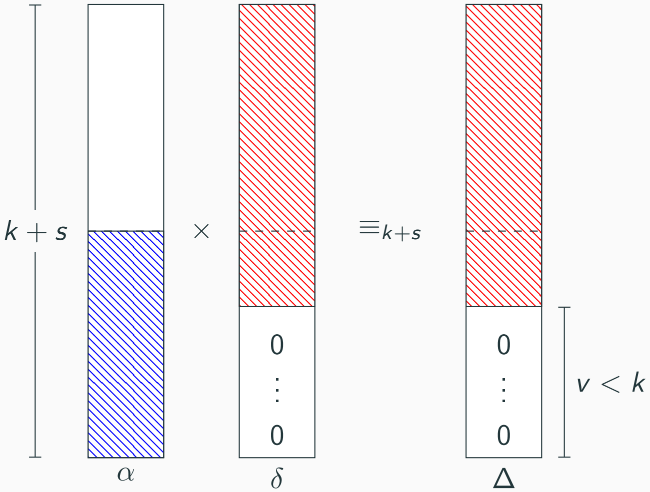
Security Analysis

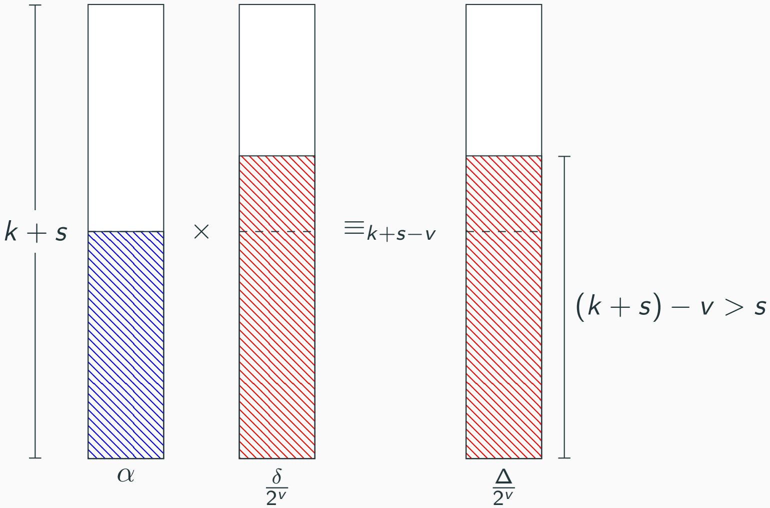
- There is an error $\Leftrightarrow x' = x + \delta$ with $\delta \not\equiv_k 0$

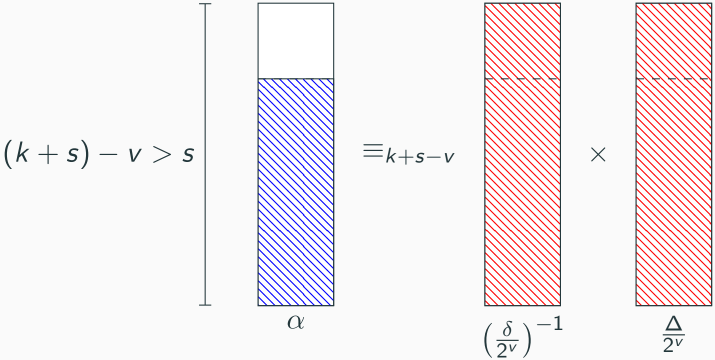
Security Analysis

- There is an error $\Leftrightarrow x' = x + \delta$ with $\delta \not\equiv_k 0$
- The check passes $\Leftrightarrow$ $\alpha \cdot \delta \equiv_{k+s} \Delta$


$k+s$
$\times$
$\equiv_{k+s}$
α
δ
Δ


k + s
×
≡k+s
0
⋮
0
0
⋮
0
v < k
α
δ
Δ


$k + s$
α
$\times$
$\frac{\delta}{2^v}$
$\equiv_{k+s-v}$
$\frac{\Delta}{2^v}$
$(k + s) - v > s$


$(k+s)-v > s$
$\equiv_{k+s-v}$
$\times$
α
$\left(\frac{\delta}{2^v}\right)^{-1}$
$\frac{\Delta}{2^v}$

Security Analysis

- There is an error $\Leftrightarrow x' = x + \delta$ with $\delta \not\equiv_k 0$
- The check passes $\Leftrightarrow \alpha \cdot \delta \equiv_{k+s} \Delta$

Security Analysis

- There is an error $\Leftrightarrow x' = x + \delta$ with $\delta \not\equiv_k 0$
- The check passes $\Leftrightarrow$ $\alpha \cdot \delta \equiv_{k+s} \Delta$

- $\alpha \cdot \frac{\delta}{2^v} \equiv_{k+s-v} \frac{\Delta}{2^v}$ where v is the largest integer such that $2^v | \delta$ (we have that $v < k$)
- But $\delta / 2^v$ is odd! So we can invert: $\alpha \equiv_{k+s-v} \left(\frac{\delta}{2^v}\right)^{-1} \cdot \frac{\Delta}{2^v}$
- Therefore, the adversary knows the last $k + s - v$ bits of α, which happens with probability at most $2^{v-k-s} < 2^{-s}$.

SPD$\mathbb{Z}_{2^k}$: Protocol overview

Offline phase (preprocessing)

1. Random authenticated values
2. Multiplication triples
3. Generate shares of MAC key and shares of MACked values

Online phase

1. Distribute inputs
2. Compute shares of the values on the circuit
3. Check correctness of the opened values using their MACs
    - Checking individual MACs
    - Batch MAC-checking

SPD$\mathbb{Z}_{2^k}$: Protocol overview

Offline phase (preprocessing)

1. Random authenticated values
2. Multiplication triples
3. Generate shares of MAC key and shares of MACked values

Online phase

1. Distribute inputs
2. Compute shares of the values on the circuit
3. Check correctness of the opened values using their MACs
   - Checking individual MACs
   - Batch MAC-checking

Batch MAC-checking

Motivation

Many values are opened... **it is expensive to check each one of them!**

Motivation

Many values are opened... **it is expensive to check each one of them!**

Typical solution over fields

To check correctness of $x_1, \ldots, x_t$, only check correctness of $x = \sum_i r_i \cdot x_i$.

- Individual errors δ_i get aggregated $\delta = \sum_i r_i \cdot \delta_i$
- $\delta_i \neq 0$ **for at least one** i **implies** $\delta \neq 0$ **with high probability**

Motivation

Many values are opened... **it is expensive to check each one of them!**

Typical solution over fields

To check correctness of $x_1, \ldots, x_t$, only check correctness of $x = \sum_i r_i \cdot x_i$.

- Individual errors δ_i get aggregated $\delta = \sum_i r_i \cdot \delta_i$
- $\delta_i \neq 0$ **for at least one** i **implies** $\delta \neq 0$ **with high probability**

Key idea for SPD$\mathbb{Z}_{2^k}$

Do the same! (analysis gets tricky...)

Batch MAC-checking in SPD$\mathbb{Z}_{2^k}$

- Let E be the event: $\delta \cdot \alpha \equiv_{k+s} \Delta$

Naive approach

$$\Pr[E] \leq 2^{-\frac{s}{2}}$$

Batch MAC-checking in SPD$\mathbb{Z}_{2^k}$

- Let E be the event: $\delta \cdot \alpha \equiv_{k+s} \Delta$

Naive approach

$$\Pr[E] \leq 2^{-\frac{s}{2}}$$

Fine-grained analysis

$$\Pr[E] \leq 2^{-s} + 2^{-s-1+\log s}$$

Multiplication Triples

General Idea (high level)

Preprocess triples $([a], [b], [c])$ such that a, b are random and $c \equiv_k a \cdot b$.

Key idea (two parties)

$$(a^1 + a^2) \cdot (b^1 + b^2) = a^1 b^1 + a^2 b^2 + a^1 b^2 + a^2 b^1$$

Share mixed products using OT

Similar to the MASCOT triple generation protocol (Keller et al, CCS 2016). Based on Oblivious Transfer.

General Idea (high level)

1. **OT:** $\boldsymbol{c} \equiv_{k+s} \boldsymbol{a} \cdot b$
2. **Combine:** Take inner product with a random vector: $\langle \boldsymbol{r}, \boldsymbol{c} \rangle \equiv_{k+s} \langle \boldsymbol{r}, \boldsymbol{a} \rangle \cdot b$
   - MASCOT: $\boldsymbol{a}$ is a vector of (field) elements
   - SPD$\mathbb{Z}_{2^k}$: $\boldsymbol{a}$ is a vector of bits
3. **Authenticate:** Shares are authenticated (using a MAC functionality)
4. **Sacrifice:** Check correctness

Conclusions

We develop an efficient dishonest majority MPC protocol for computation over $\mathbb{Z}_{2^k}$.

- New number-theoretic tricks introduced to overcome the difficulties of working over a ring as $\mathbb{Z}_{2^k}$:
  - Zero-divisors!
  - Non-invertible elements!
  - Taking dot product with random vectors is not a 2-universal function!

First efficient, information-theoretic secure, homomorphic authentication scheme modulo 2^k.

Future work

Implementation and performance test

- Preprocessing is theoretically slower than MASCOT
- SPD$\mathbb{Z}_{2^k}$'s online phase is expected to be faster in practice.

Future work

Implementation and performance test

- Preprocessing is theoretically slower than MASCOT
- $SPD\mathbb{Z}_{2^k}$'s online phase is expected to be faster in practice.

Develop sub-protocols for basic primitives

Inequality and equality tests, bit comparisons, bit decomposition, shifting, etc.

- Highly non-trivial! Dividing by 2 is not possible directly.

Future work

Implementation and performance test

- Preprocessing is theoretically slower than MASCOT
- SPD$\mathbb{Z}_{2^k}$'s online phase is expected to be faster in practice.

Develop sub-protocols for basic primitives

Inequality and equality tests, bit comparisons, bit decomposition, shifting, etc.

- Highly non-trivial! Dividing by 2 is not possible directly.

Extending security model

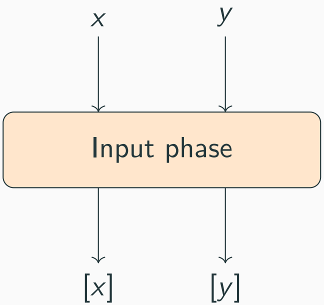
MPC over $\mathbb{Z}_{2^k}$ in the **honest majority** setting.

Thank you!

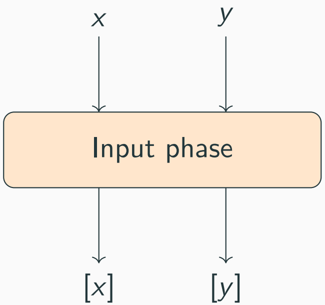
Supplementary Material

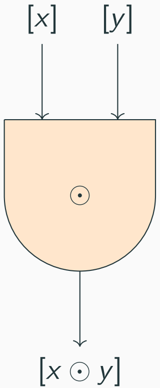
A Secret-sharing-based protocol

A Secret-sharing-based protocol

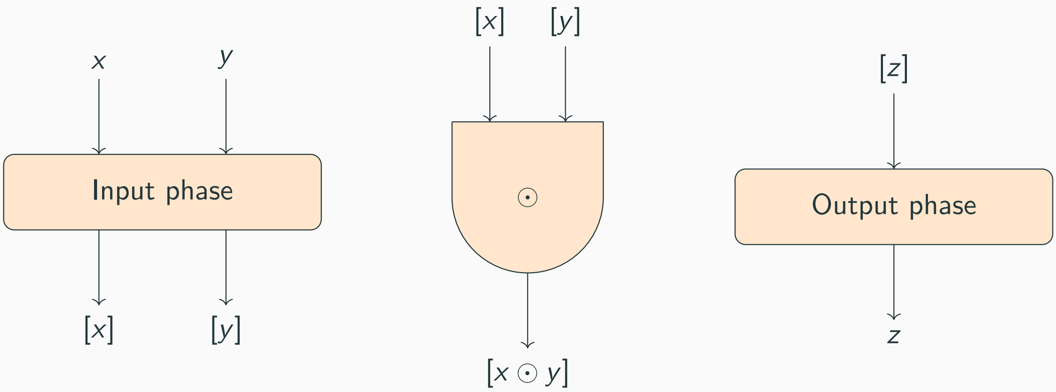


A Secret-sharing-based protocol





A Secret-sharing-based protocol



Batch MAC-checking in SPD$\mathbb{Z}_{2^k}$

- Let E be the event: $\delta \cdot \alpha \equiv_{k+s} \Delta$
- Let w be the largest integer such that 2^w divides δ.

$$\Pr[E] = \overbrace{\Pr[E \mid 0 \le w \le k]}^{\le 2^{-s}} \cdot \overbrace{\Pr[0 \le w \le k]}^{\le 1} + \sum_{c=1}^{s} \underbrace{\Pr[E \mid w = k + c]}_{\le 2^{c-s}} \cdot \underbrace{\Pr[w = k + c]}_{\le 2^{-c-1}} \le \boxed{2^{-s} + 2^{-s-1+\log s}}$$

$\Pr[E \mid 0 \leq w \leq k] \leq 2^{-s}$

We have that

$$\alpha \equiv_{k+s-w} \left(\frac{\delta}{2^w}\right)^{-1} \cdot \frac{\Delta}{2^w}$$

- $\alpha \bmod 2^{k+s-w}$ is fully determined
- This happens with probability at most $2^{w-k-s} \leq 2^{-s}$.

$\Pr[0 \leq w \leq k] \leq 1$

$$\cdots$$

$\Pr[E|w = k + c] \leq 2^{c-s}$, $c \in \{1, \ldots, s\}$

Follows from the first proof (writing $w = k + c$)

$\Pr[w = k + c] \leq 2^{-c-1}$, $c \in \{1, \ldots, s\}$

Since 2^w divides δ, we have that $\delta \equiv_w 0$, which implies

$$\chi_t \cdot \delta_t \equiv_w \underbrace{-\sum_{i=1}^{t-1} \chi_i \cdot \delta_i}_{S'}$$

Let $v \leq k - 1$ be the largest integer such that 2^v divides δ_t, then

$$\chi_t \equiv_{w-v} \left(\frac{\delta_t}{2^v}\right)^{-1} \cdot \frac{S'}{2^v}.$$

Since $\chi_t \bmod 2^{w-v}$ is fully determined, this happens with probability at most $2^{v-w} \leq 2^{-c-1}$.

Batch MAC Checking

Procedure BatchCheck

Procedure for opening and checking the MACs on t shared values $[x_1], \ldots, [x_t]$. Let x_i^j, m_i^j, α^j be P_j's share, MAC share and MAC key share for $[x_i]$.

Open phase:

1. Each party P_j broadcasts for each i the value $\tilde{x}_i^j = x_i^j \mod 2^k$.
2. The parties compute $\tilde{x}_i = \sum_{j=1}^{n} \tilde{x}_i^j \mod 2^{k+s}$.

MAC check phase:

3. The parties call $\mathcal{F}_{\mathsf{Rand}}(\mathbb{Z}_{2^s}^t)$ to sample public random values $\chi_1, \ldots, \chi_t \in \mathbb{Z}_{2^s}$ and then compute $\tilde{y} = \sum_{i=1}^{t} \chi_i \cdot \tilde{x}_i \mod 2^{k+s}$.
4. Each party P_j samples $r^j \leftarrow_R \mathbb{Z}_{2^s}$, and then calls $\mathcal{F}_{\mathsf{MAC}}$ on input $(s, s, r^j, \mathsf{MAC})$ to obtain $[r]$. Denote P_j's MAC share on r by ℓ^j.
5. Each party P_j computes $p^j = \sum_{i=1}^{t} \chi_i \cdot p_i^j \mod 2^s$ where $p_i^j = \frac{x_i^j - \tilde{x}_i^j}{2^k}$ and broadcasts $\tilde{p}^j = p^j + r^j \mod 2^s$.
6. Parties compute $\tilde{p} = \sum_{j=1}^{n} \tilde{p}^j \mod 2^s$.
7. Each party P_j computes $m^j = \sum_{i=1}^{t} \chi_i \cdot m_i^j \mod 2^{k+s}$ and $z^j = m^j - \alpha^j \cdot \tilde{y} - 2^k \cdot \tilde{p} \cdot \alpha^j + 2^k \cdot \ell^j \mod 2^{k+s}$. Then it commits to z^j, and then all parties open their commitments.
8. Finally, the parties verify that $\sum_{j=1}^{n} z^j \equiv_{k+s} 0$. If the check passes then the parties accept the values $\tilde{x}_i \mod 2^k$, otherwise they abort.

Triples - Part 1

Protocol Π_{Triple}

The integer parameter $\tau = 4s + 2k$ specifies the size of the input triple used to generate each output triple.

Multiply:

1. Each party P_i samples $\boldsymbol{a}^i = (a_1^i, \ldots, a_\tau^i) \leftarrow_R (\mathbb{Z}_2)^\tau$, $b^i \leftarrow_R \mathbb{Z}_{2^{k+s}}$
2. Every ordered pair of parties (P_i, P_j) does the following:
   (a) Both parties call $\mathcal{F}_{\mathsf{ROT}}^\tau$ with P_i as the receiver and P_j as the sender. P_i inputs the bits $(a_1^i, \ldots, a_\tau^i) \in (\mathbb{Z}_2)^\tau$.
   (b) P_j receives $q_{0,h}^{j,i}, q_{1,h}^{j,i} \in \mathbb{Z}_{2^{k+s}}$ and P_i receives $s_h^{i,j} = q_{a_h^i,h}^{j,i}$ for $h = 1, \ldots, \tau$.
   (c) P_j sends $d_h^{j,i} = q_{0,h}^{j,i} - q_{1,h}^{j,i} + b^j \mod 2^{k+s}$, for $h = 1, \ldots, \tau$.
   (d) P_i sets $t_h^{i,j} = s_h^{i,j} + a_h^i \cdot d_j^{j,i} \mod 2^{k+s}$ for $h = 1, \ldots, \tau$. In particular

$$\begin{aligned} t_h^{i,j} &\equiv_{k+s} s_h^{i,j} + a_h^i \cdot d_j^{j,i} \\ &\equiv_{k+s} q_{a_h^i,h}^{j,i} + a_h^i \cdot \left(q_{0,h}^{j,i} - q_{1,h}^{j,i} + b^j\right) \\ &\equiv_{k+s} q_{0,h}^{j,i} + a_h^i b^j . \end{aligned}$$

Therefore, the following equation holds modulo 2^{k+s} on each entry

$$\begin{pmatrix} t_1^{i,j} \\ t_2^{i,j} \\ \vdots \\ t_\tau^{i,j} \end{pmatrix} = \begin{pmatrix} q_{0,1}^{j,i} \\ q_{0,2}^{j,i} \\ \vdots \\ q_{0,\tau}^{j,i} \end{pmatrix} + b^j \begin{pmatrix} a_1^i \\ a_2^i \\ \vdots \\ a_\tau^i \end{pmatrix}$$

(e) P_i sets $\boldsymbol{c}^i_{i,j} = \left(t^{i,j}_1, t^{i,j}_2, \ldots, t^{i,j}_\tau\right) \in (\mathbb{Z}_{2^{k+s}})^\tau$.

(f) P_j sets $\boldsymbol{c}^j_{i,j} = -\left(q^{j,i}_{0,1}, q^{j,i}_{0,2}, \ldots, q^{j,i}_{0,\tau}\right) \in (\mathbb{Z}_{2^{k+s}})^\tau$.

(g) The following congruence holds

$$\boldsymbol{c}^i_{i,j} + \boldsymbol{c}^j_{i,j} \equiv_{k+s} \boldsymbol{a}^i \cdot b^j,$$

where the modulo congruence is component-wise.

3. Each party P_i computes:

$$\boldsymbol{c}^i = \boldsymbol{a}^i \cdot b^i + \sum_{j \neq i} (\boldsymbol{c}^i_{i,j} + \boldsymbol{c}^i_{j,i}) \mod 2^{k+s}$$

Triples - Part 3

Protocol Π_{Triple} (continuation)

Combine:

1. Sample $\boldsymbol{r}, \hat{\boldsymbol{r}} \leftarrow_R \mathcal{F}_{\mathsf{Rand}}((\mathbb{Z}_{2^{k+s}})^{\tau})$.
2. Each party P_i sets

$$a^i = \sum_{h=1}^{\tau} r_h \boldsymbol{a}^i[h] \mod 2^{k+s}, \qquad c^i = \sum_{h=1}^{\tau} r_h \boldsymbol{c}^i[h] \mod 2^{k+s} \qquad \text{and}$$

$$\hat{a}^i = \sum_{h=1}^{\tau} \hat{r}_h \boldsymbol{a}^i[h] \mod 2^{k+s}, \qquad \hat{c}^i = \sum_{h=1}^{\tau} \hat{r}_h \boldsymbol{c}^i[h] \mod 2^{k+s}$$

Authenticate: Each party P_i runs $\mathcal{F}_{\mathsf{MAC}}$ on their shares to obtain authenticated shares $[a], [b], [c], [\hat{a}], [\hat{c}]$.

Sacrifice: Check correctness of the triple $([a], [b], [c])$ by sacrificing $[\hat{a}], [\hat{c}]$.

1. Sample $t := \mathcal{F}_{\mathsf{Rand}}(\mathbb{Z}_{2^s})$.
2. Execute the procedure AffineComb to compute $[\rho] = t \cdot [a] - [\hat{a}]$
3. Execute the procedure BatchCheck on $[\rho]$ to obtain ρ.
4. Execute the procedure AffineComb to compute $[\sigma] = t \cdot [c] - [\hat{c}] - [b] \cdot \rho$.
5. Run BatchCheck on $[\sigma]$ to obtain σ, and abort if this value is not zero modulo 2^{k+s}.

Output: Generate using $\mathcal{F}_{\mathsf{MAC}}$ a random value $[r]$ with $r \in \mathbb{Z}_{2^s}$. Output $([a], [b], [c + 2^k r])$ as a valid triple.

Communication

Protocol	Message space	Stat. security	Input cost (kbit)	Triple cost (kbit)
Ours	$\mathbb{Z}_{2^{32}}$	26	3.17	79.87
	$\mathbb{Z}_{2^{64}}$	57	12.48	319.49
	$\mathbb{Z}_{2^{128}}$	57	16.64	557.06
MASCOT	32-bit field	32	1.06	51.20
	64-bit field	64	4.16	139.26
	128-bit field	64	16.51	360.44

Table 1. Communication cost of our protocol and previous protocols for various rings and fields, and statistical security parameters

Performance (1)

Suite	Mult (par)	Mult (seq)	Input-Mult-Output	Input (par)
SPDZ	1148ms	328ms	2118ms	335ms
$SPDZ_{2^k}$	236ms	318ms	674ms	166ms
$SPDZ_{2^k}$ (Optimized)	165ms	-	-	-
Improvement	4.86	1.03	3.14	2.01

Table 1. Primitive non-linear operations.

Performance (2)

Protocol	1 Thread	5 Threads	10 threads	20 threads
Mascot (k = 128)	1031	1551	1862	1952
$SPDZ_{2^k}$ (k = 64, s = 64)	1199	1932	2047	2076
$SPDZ_{2^k}$ (k = 64, s = 96)	-	-	-	-

Table 2. Multiplication triple generation (throughput in triples per second).

We ran triple generation on two t2-medium tier AWS EC2 instances, each instance with 2 vCPUs and 4GB memory, connected over a 800 Mbits/sec link.

We generate 500 elements per thread both for Mascot and $SPDZ_{2^k}$.

Total amount of bits sent per triple, per party in two-party setting: $(k + 2s)(9s + 4k) + 2(k + 2s) = (k + 2s)(9s + 4k + 2)$, where $2(k + 2s)$ comes from the sacrifice step.