High-order Table-based Conversion Algorithms and Masking Lattice-based Encryption

Jean-Sébastien Coron[1], Francois Gérard[1], Simon Montoya[2,3], Rina Zeitoun[2]

[1]University of Luxembourg

[2]IDEMIA, Cryptography & Security Labs

[3]LIX, INRIA, CNRS, École Polytechnique, Institut Polytechnique de Paris

Masking

- Common countermeasure against side-channel attacks
- Boolean masking

$$x = x_1 \oplus \cdots \oplus x_n \in \{0,1\}^k$$

- Arithmetic masking

$$y = y_1 + \cdots + y_n \pmod{q}$$

- Useful for masking lattice-based schemes
  - Kyber: $q = 3329$
  - Saber: $q = 2^k$
- In this paper: generic conversions between different masked forms

Application for masking lattice-based schemes

- Kyber: high-order masking IND-CCA decryption with FO transform, $q = 3329$
  - Masking IND-CPA decryption: A $\longrightarrow$ (1-bit) B
  - Masking binomial sampling in re-encryption: (1-bit) B $\longrightarrow$ A
- Saber:
  - similar with $q = 2^{13}$, $p = 2^{10}$.

Generic high-order table-based conversion

- Consider two additive groups G and H and a function $f : G \to H$.
  - Our algorithm takes as input n shares $x_1, \ldots, x_n \in G$ and outputs n shares $y_1, \ldots, y_n \in H$ such that:

$$y_1 + \cdots + y_n = f(x_1 + \cdots + x_n)$$

Algorithm 1 Convert$_{G,H,f}$

Input: $x_1, \ldots, x_n \in G$
Output: $y_1, \ldots, y_n \in H$ such that $y_1 + \cdots + y_n = f(x_1 + \cdots + x_n)$

1: **for all** $u \in G$ **do** $T(u) \leftarrow \big(f(u), 0, \ldots, 0\big) \in H^n$
2: **for** $i = 1$ to $n - 1$ **do**
3: **for all** $u \in G$ **do** $T'(u) \leftarrow T(u + x_i)$
4: **for all** $u \in G$ **do** $T(u) \leftarrow \text{Refresh}_H\big(T'(u)\big)$
5: $(y_1, \ldots, y_n) \leftarrow \text{Refresh}_H\big(T(x_n)\big)$
6: **return** $y_1, \ldots, y_n$

Arithmetic to Boolean XOR 1

Convert $(\mathbb{Z}_4, +) \rightarrow (\{0,1\}^2, \oplus)$ with $f : x \mapsto x \oplus 1$.

Let $x = x_1 + x_2 + x_3 = 3 + 2 + 1 \equiv 2 \bmod 4$ and thus $f(x) = 3$.

i	y_1	y_2	y_3	$\bigoplus y_i$
0	01	00	00	1
1	00	00	00	0
2	11	00	00	3
3	10	00	00	2

Arithmetic to Boolean XOR 1

Convert $(\mathbb{Z}_4, +) \rightarrow (\{0,1\}^2, \oplus)$ with $f : x \mapsto x \oplus 1$.

Let $x = x_1 + x_2 + x_3 = 3 + 2 + 1 \equiv 2 \bmod 4$ and thus $f(x) = 3$.

i	y_1	y_2	y_3	$\bigoplus y_i$
0	01	01	10	2
1	00	10	11	1
2	11	00	11	0
3	11	11	11	3

Arithmetic to Boolean XOR 1

Convert $(\mathbb{Z}_4, +) \rightarrow (\{0,1\}^2, \oplus)$ with $f : x \mapsto x \oplus 1$.

Let $x = x_1 + x_2 + x_3 = 3 + 2 + 1 \equiv 2 \bmod 4$ and thus $f(x) = 3$.

i	y_1	y_2	y_3	$\bigoplus y_i$
0	10	00	10	0
1	10	00	01	3
2	11	10	11	2
3	01	11	11	1

Boolean to arithmetic

Algorithm 2 BooleanToArithmetic

Input: $k \in \mathbb{Z}$ and $x_1, \ldots, x_n \in \{0,1\}^k$
Output: $y_1, \ldots, y_n \in \mathbb{Z}_q$ such that $y_1 + \cdots + y_n \bmod q = x_1 \oplus \cdots \oplus x_n$

1: **for all** $u \in \{0,1\}^k$ **do** $T(u) \leftarrow \big(u \bmod q, 0, \ldots, 0\big)$
2: **for** $i = 1$ to $n-1$ **do**
3: **for all** $u \in \{0,1\}^k$ **do** $T'(u) \leftarrow T(u \oplus x_i)$
4: **for all** $u \in \{0,1\}^k$ **do** $T(u) \leftarrow \mathsf{Refresh}_{\mathbb{Z}_q}\big(T'(u)\big)$
5: $(y_1, \ldots, y_n) \leftarrow \mathsf{Refresh}_{\mathbb{Z}_q}\big(T(x_n)\big)$
6: **return** $y_1, \ldots, y_n$

Optimized B2A

Naive algorithm is in $\mathcal{O}(2^k \cdot n^2)$.

But if we write $x_i = \bigoplus_{j=0}^{k-1} 2^j \cdot x_i^{(j)}$ where $x_i^{(j)}$ is the j-th bit of x_i, we have

$$x = \bigoplus_{j=0}^{k-1} 2^j \cdot \bigoplus_{i=1}^{n} x_i^{(j)} = \sum_{j=0}^{k-1} 2^j \cdot \bigoplus_{i=1}^{n} x_i^{(j)}$$

We can convert bits one by one and reconstruct x.

1-bit Boolean to Arithmetic

We only need a small table of size 2.

Let $z = z_1 \oplus z_2 \oplus z_3 = 0 \oplus 1 \oplus 1 = 0$ and convert to $\mathbb{Z}_{3329}$

i	c_1	c_2	c_3	$\sum c_i$
0	0	0	0	0
1	1	0	0	1

1-bit Boolean to Arithmetic

We only need a small table of size 2.
Let $z = z_1 \oplus z_2 \oplus z_3 = 0 \oplus 1 \oplus 1 = 0$ and convert to $\mathbb{Z}_{3329}$

i	c_1	c_2	c_3	$\sum c_i$
0	2979	2676	1003	0
1	2279	152	899	1

1-bit Boolean to Arithmetic

We only need a small table of size 2.
Let $z = z_1 \oplus z_2 \oplus z_3 = 0 \oplus 1 \oplus 1 = 0$ and convert to $\mathbb{Z}_{3329}$

i	c_1	c_2	c_3	$\sum c_i$
0	921	2588	3150	1
1	3298	1955	1405	0

Optimized B2A

Algorithm 3 Optimized BooleanToArithmetic

Input: $x_1, \ldots, x_n \in \{0,1\}^k$
Output: $y_1, \ldots, y_n \in \mathbb{Z}_q$ such that $x_1 \oplus \cdots \oplus x_n = y_1 + \cdots + y_n \bmod q$

1: **for** $i = 1$ to n **do** $y_i \leftarrow 0$
2: **for** $j = 0$ to $k - 1$ **do**
3: **for** $i = 1$ to n **do** $z_i \leftarrow (x_i \gg j)\ \&\ 1$
4: $(y_1^{(j)}, \ldots, y_n^{(j)}) \leftarrow$ 1bitB2A$(1, z_1, \ldots, z_n)$
5: **for** $i = 1$ to n **do** $y_i \leftarrow y_i + (y_i^{(j)} \ll j) \bmod q$
6: **return** $y_1, \ldots, y_n$

Arithmetic to Boolean conversion

- For arithmetic to Boolean conversion, we can use the table-based generic algorithm
  - Complexity $\mathcal{O}(q \cdot n^2)$ or $\mathcal{O}(2^k \cdot n^2)$

- Improved algorithm: smaller table and carry propagation
  - Complexity $\mathcal{O}(k \cdot n^2)$ or $\mathcal{O}(k \cdot n^3)$

Masked shift

Goal: compute $f(z) = \lfloor z/2^\ell \rfloor \bmod 2^{k-l}$ for $z = z_1 + \cdots + z_n$.

Writing $z_i = 2^\ell \cdot y_i + x_i$, we have

$$z = 2^\ell \cdot \sum_i y_i + \sum_i x_i$$

and

$$f(z) = \sum_i y_i + f\left(\sum_i x_i \bmod 2^k\right) \bmod 2^{k-l}$$

$$\sum_i x_i \leq n \cdot (2^\ell - 1)$$

Shift by $\ell = 2$ from $\mathbb{Z}_{16}$ to $\mathbb{Z}_4$

Let

$$z = z_1 + z_2 + z_3 = 13 + 6 + 10 \equiv 13 \bmod 16,$$

the low part is

$$1 + 2 + 2$$

.

i	c_1	c_2	c_3	$\sum c_i$
0	...	...	...	0
1	...	...	...	0
2	...	...	...	0
3	...	...	...	0
4	...	...	...	1
5	...	...	...	1
6	...	...	...	1
7	...	...	...	1
8	...	...	...	2
9	...	...	...	2

Shift by $\ell = 2$ from $\mathbb{Z}_{16}$ to $\mathbb{Z}_4$

Let

$$z = z_1 + z_2 + z_3 = 13 + 6 + 10 \equiv 13 \bmod 16,$$

low part is

$$1 + 2 + 2$$

i	c_1	c_2	c_3	$\sum c_i$
0	...	...	...	0
1	...	...	...	0
2	...	...	...	0
3	...	...	...	1
4	...	...	...	1
5	...	...	...	1
6	...	...	...	1
7	...	...	...	2
8	...	...	...	2

Shift by $\ell = 2$ from $\mathbb{Z}_{16}$ to $\mathbb{Z}_4$

Let

$$z = z_1 + z_2 + z_3 = 13 + 6 + 10 \equiv 13 \bmod 16,$$

low part is

$$1 + 2 + 2$$

.

i	c_1	c_2	c_3	$\sum c_i$
0	...	...	...	0
1	...	...	...	1
2	...	...	...	1
3	...	...	...	1
4	...	...	...	1
5	...	...	...	2
6	...	...	...	2

Carry is 1 :

$$\lfloor z/4 \rfloor = \lfloor 13/4 \rfloor + \lfloor 6/4 \rfloor + \lfloor 10/4 \rfloor + 1 = 2 + 1 + 2 + 1 \equiv 2 \bmod 4$$

Arithmetic to Boolean conversion

Starting from

$$z = 2^{\ell} \cdot \lfloor z/2^{\ell} \rfloor + z \bmod 2^{\ell}$$

we can see that

$$\texttt{Bool}(z) = \texttt{Bool}(\lfloor z/2^{\ell} \rfloor) || \texttt{Bool}(z \bmod 2^{\ell})$$

If ℓ is small the right part can be directly converted using a small table and the whole z converted chunk by chunk. Even better, if $\ell = 1$, the right part does not need to be converted.

Arithmetic to Boolean conversion with shift of 1

Algorithm 4 Optimized ArithmeticToBoolean

Input: $k, \ell \in \mathbb{N}^+$ and $z_1, \ldots, z_n \in \mathbb{Z}_{2^k}$

Output: $s_1, \ldots, s_n \in \{0,1\}^k$ such that $s_1 \oplus \cdots \oplus s_n = z_1 + \cdots + z_n \bmod 2^k$

1: **for** $i = 1$ to n **do** $s_i \leftarrow 0$
2: **for** $j = 0$ to $k - 1$ **do**
3: **for** $i = 1$ to n **do** $s_i \leftarrow s_i + ((z_i \,\&\, 1) \ll j)$
4: $(z_1, \ldots, z_n) \leftarrow \mathsf{Shift}(k - j, 1, (z_1, \ldots, z_n))$
5: **return** $s_1, \ldots, s_n$

Arithmetic to 1-bit Boolean conversion: register optimization

When working with small tables with $G = \mathbb{Z}_{2^k}$ and $H = \{0, 1\}$, it is natural to consider storing them in registers.

0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

```
uint16_t t = 0x00FF;
```

Shifting each share of the table is cheap and the whole table is refreshed in parallel with XORs.

Arithmetic to 1-bit Boolean conversion: register optimization

Algorithm 5 ArithmeticToBoolean, register optimization

Input: $x_1, \ldots, x_n \in \mathbb{Z}_{2^k}$

Output: $y_1, \ldots, y_n \in \{0,1\}$ such that $y_1 \oplus \cdots \oplus y_n = f(x_1 + \cdots + x_n \bmod 2^k)$

1: **for all** $u \in \mathbb{Z}_{2^k}$ **do** $R_1[u] \leftarrow f(u)$
2: **for all** $2 \leq j \leq n$ **do** $R_j \leftarrow 0$.
3: **for** $i = 1$ to $n-1$ **do**
4: **for** $j = 1$ to n **do** $R_j \leftarrow \mathrm{ROR}[x_i](R_j)$
5: **for** $j = 1$ to $n-1$ **do**
6: $r \leftarrow \{0,1\}^{2^k}, \; R_j \leftarrow R_j \oplus r, \; R_n \leftarrow R_n \oplus r$
7: $(y_1, \ldots, y_n) \leftarrow \mathrm{Refresh}_{\{0,1\}}(R_1[x_n], \ldots, R_n[x_n])$
8: **return** $y_1, \ldots, y_n$

Comparison with existing work

A mod $2^{32} \rightarrow$ **B**	Security order t								
	1	2	3	4	5	6	8	10	12
Goubin	165								
CGV14 $32 \rightarrow 32$		1 132	2 070	4 030	6 218	8 597	15 053	23 655	33 572
Carry prop. 1		2 496	5 248	9 600	15 936	24 640	50 688	90 816	148 096
Carry prop. 2		3 872	7 520	12 448	18 656	26 144	44 960	68 896	97 952

A mod $2^{6} \rightarrow$ **B**	Security order t								
	1	2	3	4	5	6	8	10	12
Goubin	38								
CGV14 $6 \rightarrow 1$		226	411	786	1 207	1 663	2 895	4 531	6 416
Table with reg.	20	54	104	170	252	350	594	902	1 274

Threshold arithmetic modulo q to 1-bit Boolean

- Threshold function for Kyber IND-CPA decryption: $\mathrm{th} : \mathbb{Z}_q \to \{0,1\}$, with $q = 3329$:

$$\mathrm{th}(x) = \begin{cases} 0 & \text{if } (x \bmod^{\pm} q) \in [-q/4, q/4[\\ 1 & \text{otherwise.} \end{cases}$$

  - Goal: compute 1-bit Boolean shares

$$b_1 \oplus \cdots \oplus b_n = \mathrm{th}(x_1 + \cdots + x_n \bmod q)$$

  - First step: modulus switching with precision $\ell = O(\log n)$

$$y_i = \left\lfloor \frac{x_i \cdot 2^{\ell}}{q} + \frac{1}{2} \right\rfloor = \left\lfloor \frac{x_i \cdot 2^{\ell+1} + q}{2q} \right\rfloor$$

  - Second step: table-based arithmetic modulo 2^{ℓ} to 1-bit Boolean conversion with $f : \mathbb{Z}_{2^{\ell}} \to \{0,1\}$ where

$$f(y) = \begin{cases} 0 & \text{if } (y \bmod^{\pm} 2^{\ell}) \in (-2^{\ell-2}, 2^{\ell-2}) \\ 1 & \text{otherwise.} \end{cases}$$

Practical results

1-bit B → A	Security order t								
	1	2	3	4	5	6	7	8	9
[SPOG19]	61	142	250	408	520	687	882	1 133	1 354
1-bitB2A	59	98	207	347	524	749	962	1 241	1 618

A → A shift, $k = 13$	Security order t								
	1	2	3	4	5	6	7	8	9
[CGV14]	270	1183	1 743	3 176	4 819	6 107	7 610	10 108	12 661
Shift1 $(\ell = 1)$	15	223	429	686	1104	1 567	2 967	3 822	4 879
Shift1 $(\ell = 3)$	16	230	443	2 087	3 339	4 689	9 006	11 613	15 053

Practical results

A $\rightarrow$ **B from** $\mathbb{Z}_{16}$ **to** $\{0,1\}^4$	Security order t								
	1	2	3	4	5	6	7	8	9
[CGV14]	9	158	228	460	688	850	1 066	1 392	1 777
ArithmeticToBoolean	10	32	71	108	159	226	295	390	511

	A mod $q \rightarrow$ **1-bit B**	Security order t								
		1	2	3	4	5	6	7	8	9
Saber	[CGV14]	24	330	529	997	1 565	1 999	2 498	3 460	4 648
	ThresholdAtoB (Saber')	15	59	120	210	827	1 240	1 485	1 957	2 634
Kyber	[BBE+18]	819	2 062	3 298	5 560	8 168	11 137	14 033	17 912	21 752
	ThresholdAtoB	15	69	138	225	852	1 260	1 563	2 007	2 649

Thank you for your attention !