[COSADE 2017] Efficient Conversion Method from Arithmetic to Boolean Masking in Constrained Devices

Yoo-Seung Won[1] and Dong-Guk Han[1,2]
[1]Department of Financial Information Security, Kookmin University, Seoul, Korea
[2]Department of Mathematics, Kookmin University, Seoul, Korea

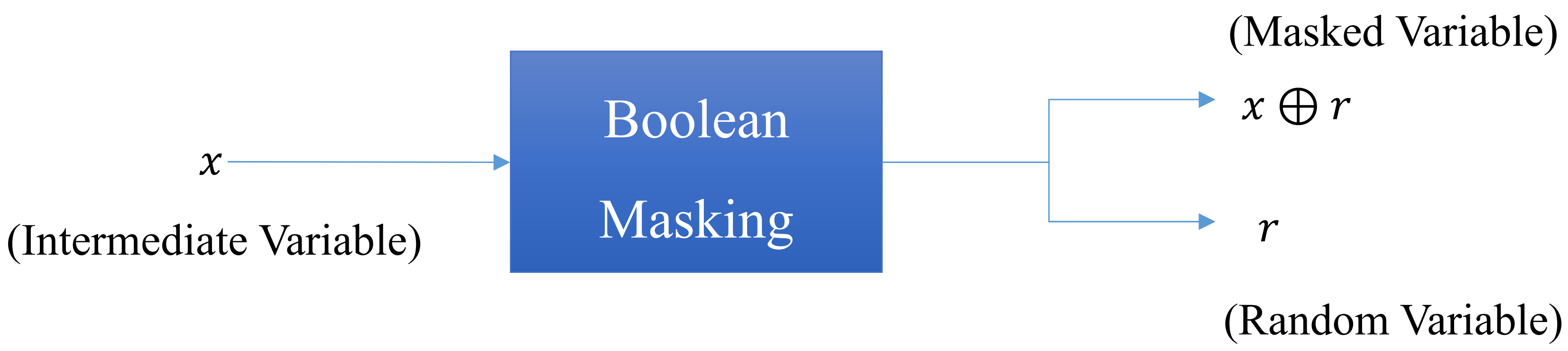
Apr.13.2017

Yoo-Seung Won

Contents

Motivation

- **Software Countermeasure against side channel analysis**
  - **Boolean Masking**
    - ✓ Easily compatible
    - ✓ Low cost



※ SPN : Substitution-Permutation Network

Motivation

- **When Boolean Masking countermeasure is applied to block cipher based on SPN structure**
  - **Nonlinear operation consumes heavily cost to construct countermeasure**

S-box operation

Input	x
Output	$S(x)$

1. return $S(x)$

In order to reduce the cost

Solution : Tablization

Generating Masking S-box

Input	x, m, m'
Output	$MS(x)$

1. for i from 0 to $2^n - 1$
2. $MS(x \oplus m) = S(x) \oplus m'$
3. endfor
4. Return MS

Masked S-box operation

Input	$x'(= x \oplus m)$
Output	$S(x) \oplus m'$

1. return $MS(x')$

※ ARX : Addition-Rotation-Xor

Motivation

- **When Boolean Masking countermeasure is applied to block cipher based on ARX structure**
  - **The bit size of addition : generally 32 or 64 bit**
    - ✓ Totally tablization is impossible because of too large

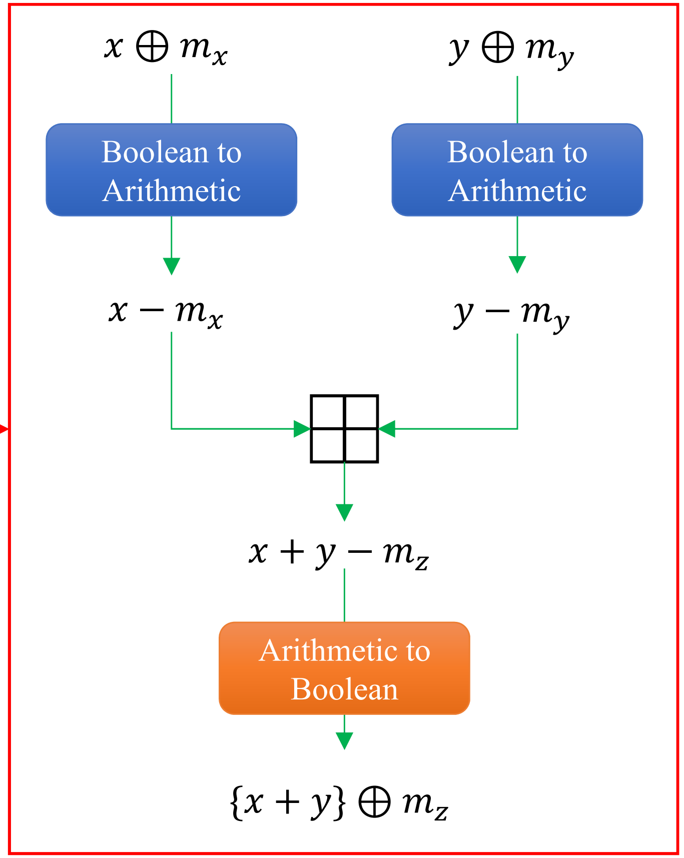
Addition operation

Input	x, y
Output	$z = x + y$

1. return $x + y$

Solution :

BtoA and AtoB



※ BtoA : Boolean to Arithmetic Masking

AtoB : Arithmetic to Boolean Masking

Motivation

History

Scheme	First-Order Countermeasure	Complexity
BtoA & AtoB	[CHES 2001] Goubin Method	$O(1)$ & $O(k)$
AtoB	[CHES 2003] CT Method	Lookup Table (Tablization)
AtoB	[CHES 2004] NP Method	Lookup Table (Tablization)
AtoB	[CHES 2012] Debraize Method	Lookup Table (Tablization)
AtoB	[COSADE 2014] KRJ Method	$O(k)$
AtoB	[*] [FSE 2015] CGTV Method	$\boldsymbol{O(\log k)}$

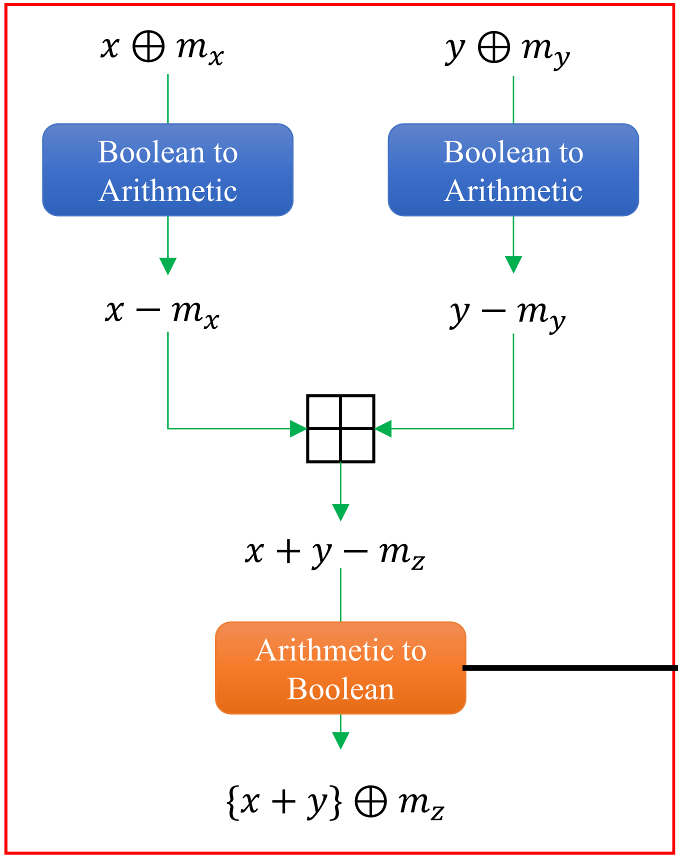
[*] [FSE 2015] Jean-Sébastien Coron, Johann Großschädl, Mehdi Tibouchi, Praveen Kumar Vadnala : Conversion from Arithmetic to Boolean Masking with Logarithmic Complexity

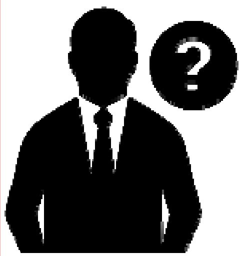
※ CGTV method is based on the principle of Kogge-Stone Adder

※ IoT : Internet of Things

Motivation

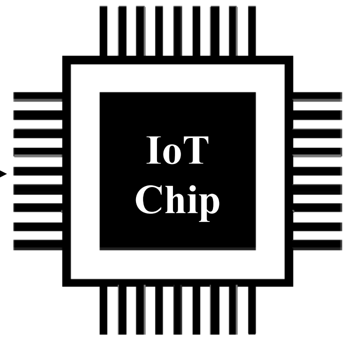
❖ **Kogge-Stone Adder is dependent on the size of addition bit**





The size of register bit : l bit

The size of arithmetic operation bit : k bit



The constrained device

How can I implement the countermeasure when $l < k$?

For example) $l = 4, \ k = 8$

Contents

[FSE 2015] CGTV method

※ Notation

The size of register bit : *l* bit

The size of arithmetic operation bit : *k* bit

- [FSE 2015] CGTV method is based on Kogge-Stone Adder
  - $k = 8,\ n_k = 3$

Algorithm 1 Kogge-Stone Adder

Input: $x, y \in \{0,1\}^k$, $n_k = \max(\lceil \log(k-1) \rceil, 1)$
Output: $z = x + y \mod 2^k$

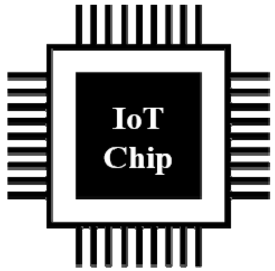
1: $P \leftarrow x \oplus y$
2: $G \leftarrow x \wedge y$
3: **for** $i := 1$ **to** $n_k - 1$ **do**
4: $\quad G \leftarrow (P \wedge (G \ll 2^{i-1})) \oplus G$
5: $\quad P \leftarrow P \wedge (P \ll 2^{i-1})$
6: **end for**
7: $G \leftarrow (P \wedge (G \ll 2^{n-1})) \oplus G$
8: **return** $x \oplus y \oplus (2G)$

[FSE 2015] CGTV method : Limitation

※ Notation

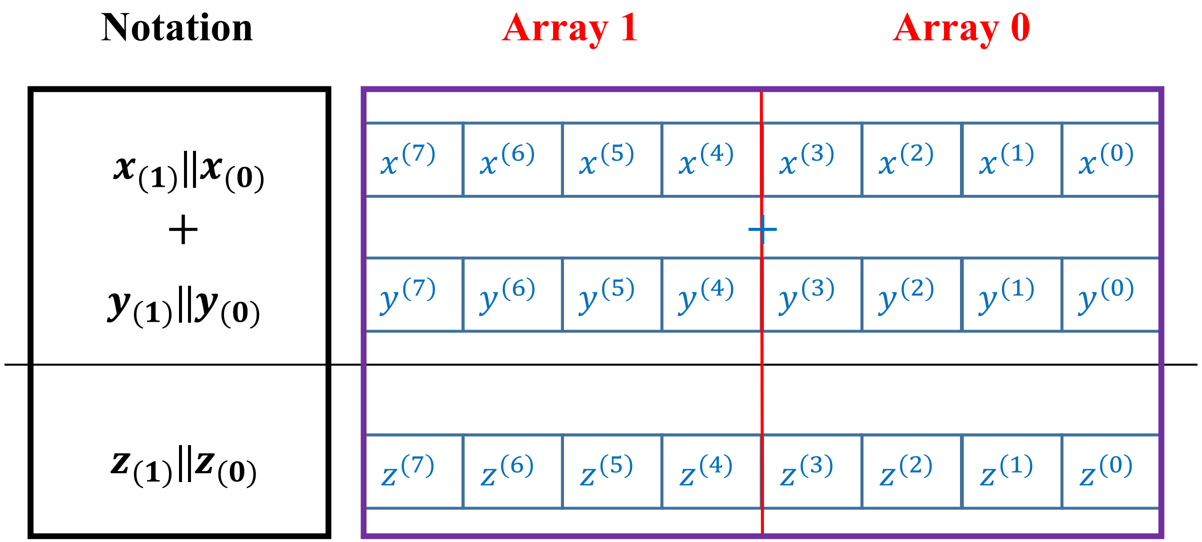
The size of register bit : l bit

The size of arithmetic operation bit : k bit



The constrained device

- [FSE 2015] CGTV method is based on Kogge-Stone Adder
  - $k = 8,\ l = 4$



8-bit Generic Variant Kogge-Stone Adder

Algorithm 1 Kogge-Stone Adder

Input: $x, y \in \{0,1\}^k$, $n_k = \max(\lceil \log(k-1) \rceil, 1)$
Output: $z = x + y \mod 2^k$

1: $P \leftarrow x \oplus y$
2: $G \leftarrow x \wedge y$
3: **for** $i := 1$ **to** $n_k - 1$ **do**
4: $\quad G \leftarrow (P \wedge (G \ll 2^{i-1})) \oplus G$
5: $\quad P \leftarrow P \wedge (P \ll 2^{i-1})$
6: **end for**
7: $G \leftarrow (P \wedge (G \ll 2^{n-1})) \oplus G$
8: **return** $x \oplus y \oplus (2G)$

Using array concept

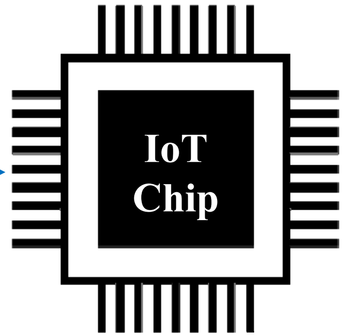
Algorithm 4 Generic Variant for Kogge-Stone Adder

Input: $x = (x_{(m-1)} \| \cdots \| x_{(0)})$, $y = (y_{(m-1)} \| \cdots \| y_{(0)})$
$\quad n = \max(\lceil \log(k-1) \rceil, 1)$
Output: $z = (z_{(m-1)} \| \cdots \| z_{(0)}) = x + y \mod 2^k$

1: $(p_{(m-1)} \| \cdots \| p_{(0)}) \leftarrow (x_{(m-1)} \| \cdots \| x_{(0)}) \oplus (y_{(m-1)} \| \cdots \| y_{(0)})$
2: $(g_{(m-1)} \| \cdots \| g_{(0)}) \leftarrow (x_{(m-1)} \| \cdots \| x_{(0)}) \wedge (y_{(m-1)} \| \cdots \| y_{(0)})$
3: **for** $i := 1$ **to** $n - 1$ **do**
4: $\quad (h_{(m-1)} \| \cdots \| h_{(0)}) \leftarrow \mathsf{Shift}[g, 2^{i-1}]$
5: $\quad (h_{(m-1)} \| \cdots \| h_{(0)}) \leftarrow (p_{(m-1)} \| \cdots \| p_{(0)}) \wedge (h_{(m-1)} \| \cdots \| h_{(0)})$
6: $\quad (g_{(m-1)} \| \cdots \| g_{(0)}) \leftarrow (h_{(m-1)} \| \cdots \| h_{(0)}) \oplus (g_{(m-1)} \| \cdots \| g_{(0)})$
7: $\quad (h_{(m-1)} \| \cdots \| h_{(0)}) \leftarrow \mathsf{Shift}[p, 2^{i-1}]$
8: $\quad (p_{(m-1)} \| \cdots \| p_{(0)}) \leftarrow (p_{(m-1)} \| \cdots \| p_{(0)}) \wedge (h_{(m-1)} \| \cdots \| h_{(0)})$
9: **end for**
10: $(h_{(m-1)} \| \cdots \| h_{(0)}) \leftarrow \mathsf{Shift}[g, 2^{n-1}]$
11: $(h_{(m-1)} \| \cdots \| h_{(0)}) \leftarrow (p_{(m-1)} \| \cdots \| p_{(0)}) \wedge (h_{(m-1)} \| \cdots \| h_{(0)})$
12: $(g_{(m-1)} \| \cdots \| g_{(0)}) \leftarrow (h_{(m-1)} \| \cdots \| h_{(0)}) \oplus (g_{(m-1)} \| \cdots \| g_{(0)})$
13: $(h_{(m-1)} \| \cdots \| h_{(0)}) \leftarrow \mathsf{Shift}[p, 2^{n-1}]$
14: **return** $(x_{(m-1)} \oplus y_{(m-1)} \oplus h_{(m-1)} \| \cdots \| x_{(0)} \oplus y_{(0)} \oplus h_{(0)})$

Contents

Our Contributions



The constrained device

Existing Scheme

Generic Kogge-Stone
Arithmetic to Boolean Conversion

- Using array concept
- High cost consumes because of secure operations
  - ✓ The size of operation unit : the size of arithmetic operation bit

Our proposal

Enhanced Kogge-Stone
Arithmetic to Boolean Conversion

- Low cost consumes when using secure operations
  - ✓ The size of operation unit : the size of register bit
- Easily control the carry bit

Our Contributions – Generic Kogge-Stone AtoB Conversion

- **[FSE 2015] CGTV method**
  - **Basic operation : Secure Shift, Secure And, Secure Xor**

Algorithm 5 Kogge-Stone Arithmetic to Boolean Conversion

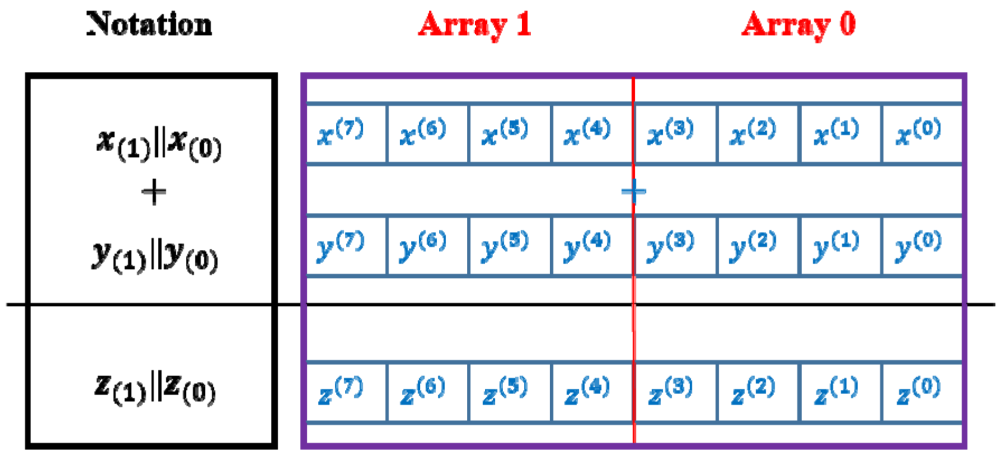
Input: $A, r \in \{0,1\}^k$ and $n_k = \max(\lceil \log(k-1) \rceil, 1)$
Output: x' such that $x' \oplus r = A + r \mod 2^k$

Let $s \leftarrow \{0,1\}^k$, $t \leftarrow \{0,1\}^k$, $u \leftarrow \{0,1\}^k$
$P' \leftarrow A \oplus s$
$P' \leftarrow P' \oplus s$
$G' \leftarrow s \oplus ((A \oplus t) \wedge r)$
$G' \leftarrow G' \oplus (t \wedge r)$
for $i := 1$ **to** $n-1$ **do**
    $H \leftarrow \mathrm{SecShift}(G', s, t, 2^{i-1})$
    $U \leftarrow \mathrm{SecAnd}(P', H, s, t, u)$
    $G' \leftarrow \mathrm{SecXor}(G', U, u)$
    $H \leftarrow \mathrm{SecShift}(P', s, t, 2^{i-1})$
    $P' \leftarrow \mathrm{SecAnd}(P', H, s, t, u)$
    $P' \leftarrow P' \oplus s$
    $P' \leftarrow P' \oplus u$
end for
$H \leftarrow \mathrm{SecShift}(G', s, t, 2^{n-1})$
$U \leftarrow \mathrm{SecAnd}(P', H, s, t, u)$
$G' \leftarrow \mathrm{SecXor}(G', U, u)$
$x' \leftarrow A \oplus 2G'$
$x' \leftarrow x' \oplus 2s$
return x'

Our Contributions – Underlying Concept

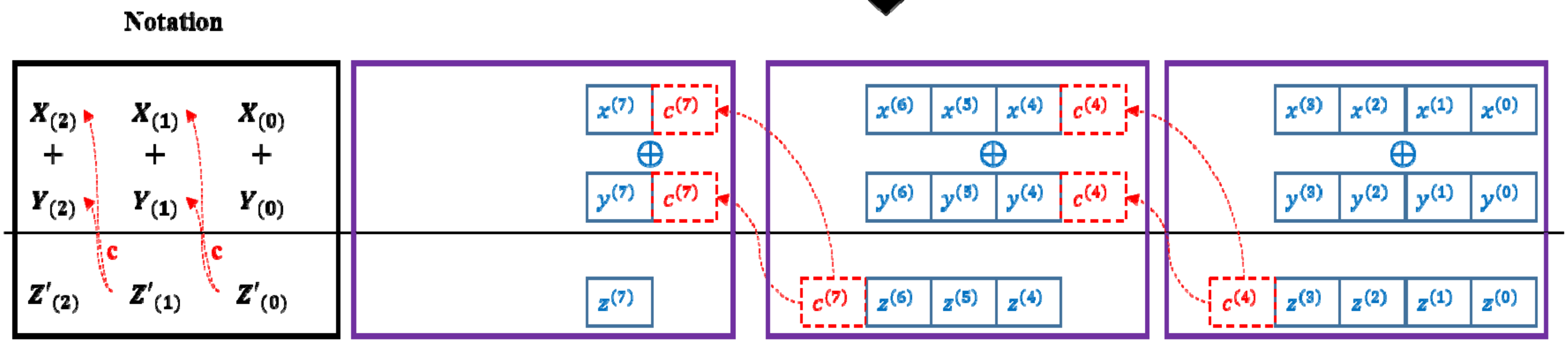
The size of arithmetic operation bit
$k = 8$

The size of register bit
$l = 4$



8-bit Generic Variant Kogge-Stone Adder

Enhanced Variant Algorithm



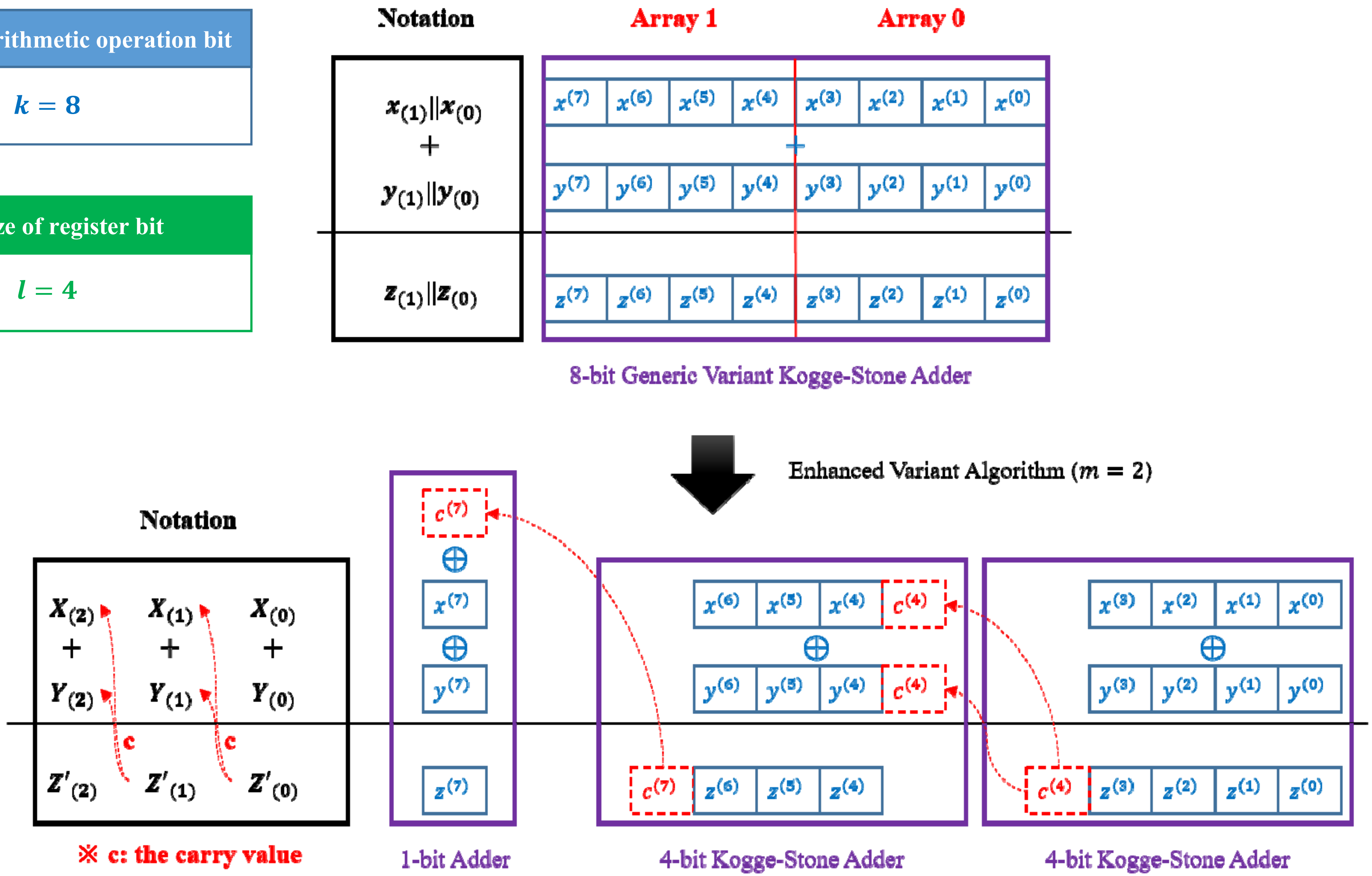
※ c: the carry value

4-bit Kogge-Stone Adder | 4-bit Kogge-Stone Adder | 4-bit Kogge-Stone Adder

Our Contributions – Underlying Concept [$m = 2$]

The size of arithmetic operation bit
$k = 8$

The size of register bit
$l = 4$



Our Contributions – Pseudo Code for Enhanced AtoB Conversion

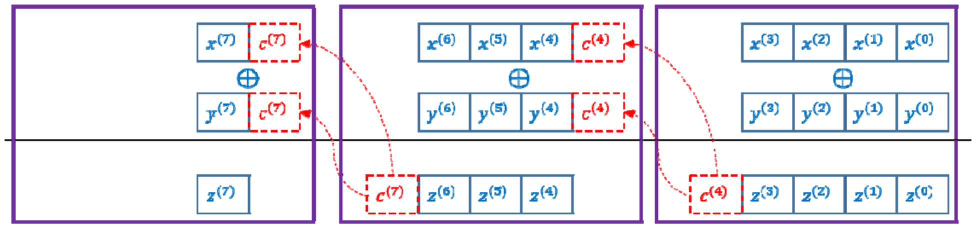
Algorithm 3 Enhanced Variant for AtoB Masking

Input: $a = \left(a_{(m-1)}||\cdots||a_{(0)}\right), r = \left(r_{(m-1)}||\cdots||r_{(0)}\right) \in \{0,1\}^k$
$n_l = \max(\lceil \log(l-1) \rceil, 1)$
Output: $x = \left(x_{(m-1)}||\cdots||x_{(0)}\right)$ such that $x \oplus r = a + r \mod 2^k$

1: $s \leftarrow \{0,1\}^l,\ t \leftarrow \{0,1\}^l,\ u \leftarrow \{0,1\}^l,\ \delta \leftarrow \{0,1\}^l$
2: $a_{(0)} \leftarrow \left(a^{(l-1)}||\cdots||a^{(0)}\right),\ r_{(0)} \leftarrow \left(r^{(l-1)}||\cdots||r^{(0)}\right),\ C \leftarrow \delta$
3: **for** $i := 1$ **to** m **do**
4: $\quad a_{(i)} \leftarrow \left(a^{(i(l-1)+l-1)}||\cdots||a^{(i(l-1)+1)}||0\right)$
5: $\quad r_{(i)} \leftarrow \left(r^{(i(l-1)+l-1)}||\cdots||r^{(i(l-1)+1)}||0\right)$
6: **end for**
7: **for** $j := 0$ **to** m **do**
8: $\quad P \leftarrow a_{(j)} \oplus s$
9: $\quad P \leftarrow P \oplus r_{(j)}$
10: $\quad G \leftarrow s \oplus \left((a_{(j)} \oplus t) \wedge r_{(j)}\right) \oplus C$
11: $\quad G \leftarrow G \oplus \left(t \wedge r_{(j)}\right) \oplus \delta$
12: $\quad$ **for** $i := 1$ **to** $n_l - 1$ **do**
13: $\quad\quad H \leftarrow \mathsf{SecShift}_l[G, s, t, 2^{i-1}]$
14: $\quad\quad W \leftarrow \mathsf{SecAnd}_l[P, H, s, t, u]$
15: $\quad\quad G \leftarrow \mathsf{SecXor}_l[G, W, u]$
16: $\quad\quad H \leftarrow \mathsf{SecShift}_l[P, s, t, 2^{i-1}]$
17: $\quad\quad P \leftarrow \mathsf{SecAnd}_l[P, H, s, t, u]$
18: $\quad\quad P \leftarrow P \oplus s$
19: $\quad\quad P \leftarrow P \oplus u$
20: $\quad$ **end for**
21: $\quad H \leftarrow \mathsf{SecShift}_l[G, s, t, 2^{n-1}]$
22: $\quad W \leftarrow \mathsf{SecAnd}_l[P, H, s, t, u]$
23: $\quad G \leftarrow \mathsf{SecXor}_l[G, W, u]$
24: $\quad X'_{(j)} \leftarrow a_{(j)} \oplus (2G)$
25: $\quad X'_{(j)} \leftarrow X'_{(j)} \oplus (2s)$
26: $\quad$ **if** $j \neq 0$ **then** $X'_{(j)} \leftarrow X'_{(j)} \gg 1$
27: $\quad$ **end if**
28: $\quad C \leftarrow [\{G \gg (l-1)\} \oplus \delta] \oplus \{s \gg (l-1)\}$
29: **end for**
30: $\left(x_{(m-1)}||\cdots||x_{(0)}\right) \leftarrow \left(X'_{(m)}||\cdots||X'_{(0)}\right) \mod 2^k$
31: **return** $x = \left(x_{(m-1)}||\cdots||x_{(0)}\right)$

The number of blocks

4-bit Kogge-Stone AtoB conversion (Step.8 ~ Step. 27)



✓ The size of operation unit : the size of register bit (4 bit)

Securely control the carry bit

Our Contributions – Pseudo Code for Enhanced AtoB Conversion

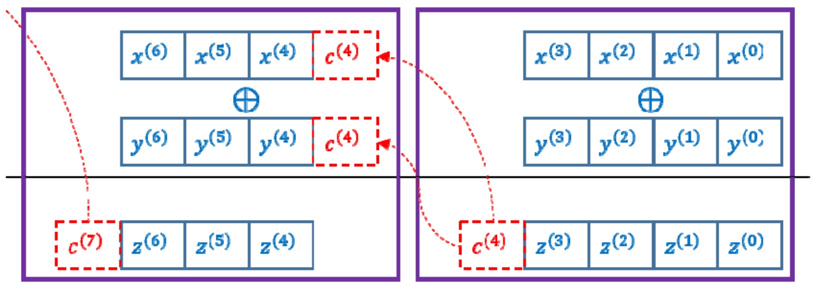
Algorithm 4 Enhanced Variant for AtoB Masking ($m = 2$)

Input: $a = \left(a_{(m-1)} || \cdots || a_{(0)}\right), r = \left(r_{(m-1)} || \cdots || r_{(0)}\right) \in \{0,1\}^k$
$n_l = \max(\lceil \log(l-1) \rceil, 1)$
Output: $x = \left(x_{(m-1)} || \cdots || x_{(0)}\right)$ such that $x \oplus r = a + r \mod 2^k$

1: $s \leftarrow \{0,1\}^l$, $t \leftarrow \{0,1\}^l$, $u \leftarrow \{0,1\}^l$, $\delta \leftarrow \{0,1\}^l$
2: $a_{(0)} \leftarrow \left(a^{(l-1)} || \cdots || a^{(0)}\right)$, $r_{(0)} \leftarrow \left(r^{(l-1)} || \cdots || r^{(0)}\right)$, $C \leftarrow \delta$
3: **for** $i := 1$ **to** m **do**
4: $\quad a_{(i)} \leftarrow \left(a^{(i(l-1)+l-1)} || \cdots || a^{(i(l-1)+1)} || 0\right)$
5: $\quad r_{(i)} \leftarrow \left(r^{(i(l-1)+l-1)} || \cdots || r^{(i(l-1)+1)} || 0\right)$
6: **end for**
7: **for** $j := 0$ **to** $m-1$ **do**
8: $\quad P \leftarrow a_{(j)} \oplus s$
9: $\quad P \leftarrow P \oplus r_{(j)}$
10: $\quad G \leftarrow s \oplus \left(\left(a_{(j)} \oplus t\right) \wedge r_{(j)}\right) \oplus C$
11: $\quad G \leftarrow G \oplus \left(t \wedge r_{(j)}\right) \oplus \delta$
12: $\quad$ **for** $i := 1$ **to** $n_l - 1$ **do**
13: $\quad\quad H \leftarrow \mathsf{SecShift}_l[G, s, t, 2^{i-1}]$
14: $\quad\quad W \leftarrow \mathsf{SecAnd}_l[P, H, s, t, u]$
15: $\quad\quad G \leftarrow \mathsf{SecXor}_l[G, W, u]$
16: $\quad\quad H \leftarrow \mathsf{SecShift}_l[P, s, t, 2^{i-1}]$
17: $\quad\quad P \leftarrow \mathsf{SecAnd}_l[P, H, s, t, u]$
18: $\quad\quad P \leftarrow P \oplus s$
19: $\quad\quad P \leftarrow P \oplus u$
20: $\quad$ **end for**
21: $\quad H \leftarrow \mathsf{SecShift}_l[G, s, t, 2^{n-1}]$
22: $\quad W \leftarrow \mathsf{SecAnd}_l[P, H, s, t, u]$
23: $\quad G \leftarrow \mathsf{SecXor}_l[G, W, u]$
24: $\quad X'_{(j)} \leftarrow a_{(j)} \oplus (2G)$
25: $\quad X'_{(j)} \leftarrow X'_{(j)} \oplus (2s)$
26: $\quad$ **if** $j \neq 0$ **then**
27: $\quad\quad X'_{(j)} \leftarrow X'_{(j)} \gg 1$
28: $\quad$ **end if**
29: $\quad C \leftarrow [\{G \gg (l-1)\} \oplus \delta] \oplus \{s \gg (l-1)\}$
30: **end for**
31: $x_{(m)} \leftarrow \left(a_{(m)} \oplus C\right) \oplus \delta$
32: $\left(x_{(m-1)} || \cdots || x_{(0)}\right) \leftarrow \left(X'_{(m)} || \cdots || X'_{(0)}\right) \mod 2^k$
33: **return** $x = \left(x_{(m-1)} || \cdots || x_{(0)}\right)$

The reduced the number of blocks to $m-1$

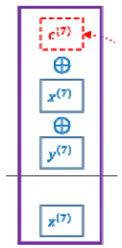
4-bit Kogge-Stone AtoB conversion (Step.8 ~ Step. 30)



4-bit Kogge-Stone Adder 4-bit Kogge-Stone Adder

✓ The size of operation unit : the size of register bit (4 bit)

Most Significant Single Bit



1-bit Adder

Contents

Analysis and Implementations : Comparison (1)

※ Notation

k: the size of arithmetic operation bit

l: the size of the register bit

m: the number of blocks

n_α: $\max(\lceil \log(\alpha - 1) \rceil, 1)$ ($\alpha = k$ or l)

Algorithm	Rand	Computational Complexity		
		Xor	And	Shift
[FSE 2015]	$3k$	$20mn_k - m - 4n_k$	$8mn_k - 2m$	$8mn_k - 4n_k$
Enhanced Kogge-Stone AtoB Conversion	$4l$	$16mn_l + 5m + 16n_l - 1$	$8mn_l - m + 8n_l - 4$	$4mn_l + 7m + 4n_l - 5$
Enhanced Kogge-Stone AtoB Conversion ($m = 2$)	$4l$	$16mn_l + 3m + 20$	$8mn_l - 2m + 7$	$4mn_l + 3m + 9$

■ Analysis and Implementations : Comparison (1)

※ Notation

k: the size of arithmetic operation bit

l: the size of the register bit

m: the number of blocks

n_α: $\max(\lceil \log(\alpha - 1) \rceil, 1)$ $(\alpha = k$ or $l)$

Algorithm	Rand	Computational Complexity		
		Xor	And	Shift
[FSE 2015]	$3k$	$20mn_k - m - 4n_k$	$8mn_k - 2m$	$8mn_k - 4n_k$
Enhanced Kogge-Stone AtoB Conversion	$4l$	$16mn_l + 5m + 16n_l - 1$	$8mn_l - m + 8n_l - 4$	$4mn_l + 7m + 4n_l - 5$
Enhanced Kogge-Stone AtoB Conversion $(m = 2)$	$4l$	$16mn_l + 3m + 20$	$8mn_l - 2m + 7$	$4mn_l + 3m + 9$

■ Analysis and Implementations : in the simulated AVR, MSP, ARM boards

Algorithm	*l*	*k*	Clock Cycle	Penalty Factor
[FSE 2015]	8	64	2,864	1.00
Enhanced Kogge-Stone AtoB Conversion	8	64	1,217	0.42

※ **Simulation Program : AVR Studio 6.2**

[FSE 2015]	16	64	2,705	1.00
Enhanced Kogge-Stone AtoB Conversion	16	64	765	0.28

※ **Simulation Program : IAR Embedded Workbench Evaluation**

[FSE 2015]	32	64	1,196	1.00
Enhanced Kogge-Stone AtoB Conversion ($m = 2$)	32	64	384	0.32

※ **Simulation Program : ARM Developer Suite v1.2**

■ Analysis and Implementations : Application to First-Order Masked SPECK

Algorithm	*l*	*k*	Clock Cycle	Penalty Factor
Non-masked SPECK	8	64	24,360	1.00
Masked SPECK with [FSE 2015]	8	64	177,303	7.27
Masked SPECK with Enhanced Kogge-Stone AtoB Conversion	8	64	112,951	4.64
Non-masked SPECK	16	64	21,446	1.00
Masked SPECK with [FSE 2015]	16	64	143,642	6.70
Masked SPECK with Enhanced Kogge-Stone AtoB Conversion	16	64	81,562	3.80
Non-masked SPECK	32	64	10,279	1.00
Masked SPECK with [FSE 2015]	32	64	71,006	6.91
Masked SPECK with Enhanced Kogge-Stone AtoB Conversion ($m = 2$)	32	64	44,936	4.37

Contents

Conclusion

- **Our solution applies to directly low-resource device**
  - **Suitable to IoT device**

- **Implementation performance increases approximately 58~72% over the original algorithm results**
  - **When applied to SPECK, 36~43% improvements**

- **Extension to higher-order AtoB masking scheme and arithmetic operation without conversion**

Q&A : mathwys87@kookmin.ac.kr