Integrative Acceleration of First-Order Boolean Masking for Embedded IoT Devices

Yuichi Komano[1], Hideo Shimizu[1], and Hideyuki Miyake[1]

Toshiba Corporation, Kawasaki, Japan
{yuichi1.komano,hideo.shimizu,hideyuki.miyake}@toshiba.co.jp

Abstract. Physical attacks, especially side-channel attacks, are threats to IoT devices which are located everywhere in the field. For these devices, the authentic functionality is important so that the IoT system becomes correct, and securing this functionality against side-channel attacks is one of our emerging issues. Toward that, Coron et al. gave an efficient arithmetic-to-Boolean mask conversion algorithm which enables us to protect cryptographic algorithms including arithmetic operations, such as hash functions, from the attacks. Recently, Biryukov et al. improved it by locally optimizing subroutines of the conversion algorithm. In this paper, we revisit the algorithm. Unlike Biryukov et al., we improve the Coron et al.'s algorithm with integrative optimizations over the subroutines. The gains against these algorithms are about 22.6% and 7.0% in the general setting. We also apply our algorithm to HMAC-SHA-1 and have an experiment to show that the implementation on a test vehicle smartcard leaks no sensitive information with the ISO/IEC17825 test.

Keywords: side-channel attack, mask conversion, IoT, embedded device

1 Introduction

Internet of Things (IoT) has been widely spread to make our lives smart and comfortable. In the IoT system, devices are located in the field and communicate each other to collect their sensing data and to control actuators.

Securing the devices in the IoT system is one of emerging issues. Especially, side-channel attacks including the power analyses [10, 11, 4] should be serious threats to the devices. This is because, in the IoT system, attackers easily get hold of devices from the field and physically analyze them. Moreover, most of cost-constrained devices lack the tamper protection mechanism and they can be vulnerable to such attacks.

Among the security functionalities of the IoT devices, the authenticity is strongly required more than others, such as the confidentiality and the anonymity, to keep the system correct. In this paper, we therefore focus on the countermeasure against the side-channel attack which realizes the secure authentication.

1.1 First-Order Boolean and Arithmetic Maskings

The hash-based message authentication code (HMAC, [2, 18]) is widely used for the authentication. Unlike the block cipher, like AES [17], each of the secure hash algorithms (SHA, [19]) includes arithmetic additions besides Boolean operations. Against the HMAC with SHA, side-channel vulnerabilities have been reported [13, 15, 1].

In order to protect the HMAC against the side-channel attack, both of the Boolean and arithmetic operations should be randomized with different types of masks. Hence, algorithms, converting a Boolean mask to an arithmetic one and vice versa, are required.

Goubin [6] proposed promising conversion algorithms for both directions. Since then, improvements and extensions, such as ones for higher-order masking, have been reported. However, to the best of our knowledge, his Boolean-to-arithmetic conversion is the best algorithm with fewer operations, up to now.

Recently, Coron et al. [5] proposed an arithmetic-to-Boolean conversion by changing the basis, an arithmetic addition, of the algorithm. Unlike the previous conversion based on the ripple-carry adder, they revisited the Kogge-Stone carry look-ahead adder [12] to construct a new conversion. With this approach, the amount of computation is dramatically decreased from $\mathcal{O}(k)$ to $\mathcal{O}(\log k)$ where k is the addition bit size. They also gave a masked addition algorithm which computes, with inputs randomized with Boolean masks, a sum with one of the Boolean masks.

In CARDIS 2017, Biryukov et al. [3] improved the Coron et al.'s masked addition. They searched optimal subroutines called in the Coron et al.'s masked addition in formal manner and applied them to construct an improved masked addition algorithm. They also applied their masked addition algorithm to the lightweight block ciphers, SIMON, SPECK, and RECTANGLE.

In addition to the above two, Schneider et al. [20] discussed an efficient hardware implementation of the conversion. Won and Han [21] modified the Kogge-Stone carry look-ahead adder with a divide and conquer approach.

1.2 Our Contributions

In this paper, we propose improved algorithms of the Coron et al.'s conversion and masked addition algorithms. Biryukov et al. took a bottom-up approach to individually optimize subroutines; however, our approach is integrative. We decrease the number of mask operations beyond the subroutines. Our tricks are to unify the subroutines[1] and to break the symmetry of subroutine calls in iterations.

We succeeded to decrease the coefficient (for $\log_2 k$ in Table 1) in the computational complexity by seven and one against the Coron et al.'s conversion and the Biryukov et al.'s one, respectively. Against the Coron et al.'s algorithm, Biryukov et al.'s one requires fewer numbers of AND and XOR operations

[1] Recently, Jungk et al. [9] also proposed more efficient algorithms independently.

whereas it additionally executes the OR (orn) operations. On the other hand, our algorithm focuses to decrease the numbers of XOR operations; and eventually, ours requires fewer operations than both algorithms in total (as shown in Table 2). The gains against the previous conversions are 22.6% and 7.0% if the size of addition unit is supposed to be $k = 32$.

We then applied our conversion algorithm to protect the authentication with HMAC-SHA-1. The gains, on the theoretical numbers of required operations, are still 17.1% and 4.9% against the HMAC-SHA-1 implementations with previous conversion algorithms (as shown in Table 3). We also developed a software implementation for an IC test vehicle smartcard [7] from Information-technology Promotion Agency (IPA), Japan. In this software, we use the xorshift [14] as a mask generation function. From the power consumption traces measured with the smartcard, the testing method of ISO/IEC 17825 [8] finds no vulnerability which confirms the security of our conversion.

1.3 Organization

The remaining of this paper is organized as follows. Section 2 reviews the previous works. In Section 3, we explain our strategy and propose our arithmetic-to-Boolean conversion and masked addition algorithms. We then check the first-order security of our algorithm with the IPA test vehicle smartcard in Section 4. Section 5 gives discussions on the detail and the extension of our algorithms. Finally, Section 6 concludes this paper.

2 Related Works

In this section, we review previous works related to the maskings.

2.1 Goubin's Mask Conversion Algorithm [6]

We recall the Boolean-to-arithmetic conversion algorithm by [6]. From $x' = x \oplus r$ and r, this algorithm efficiently computes $A = x - r$ as in Algorithm 1. We use this algorithm in our experiments later. In [6], Goubin also proposed a reverse conversion, the arithmetic-to-Boolean conversion, but we omit its detail in this paper.

2.2 Coron et al.'s Algorithms [5]

Coron et al. [5] proposed an efficient arithmetic-to-Boolean conversion applicable to the first-order masking. They took a new approach to construct their conversion by securing the Kogge-Stone carry look-ahead adder [12], although the Goubin's conversion was based on the ripple-carry adder. They also proposed a Kogge-Stone masked addition which, with inputs with Boolean masks, computes a sum with one of the Boolean masks for inputs.

Algorithm 1 Goubin's Boolean-to-arithmetic Conversion Algorithm

Input: $x', r \in \{0,1\}^k$ such that $x' = x \oplus r$ for secret $x \in \{0,1\}^k$
Output: $A \in \mathbb{F}_{2^k}$ and r such that $A = x - r$
1: $\gamma \leftarrow \{0,1\}^k$
2: $t \leftarrow x' \oplus \gamma$
3: $t \leftarrow t - \gamma$
4: $t \leftarrow t \oplus x'$
5: $\gamma = \gamma \oplus r$
6: $a \leftarrow x' \oplus \gamma$
7: $a \leftarrow a - \gamma$
8: $a \leftarrow a \oplus t$
9: **return** a, r

In [5], they also reported the implementations of HMAC-SHA-1 with first-order masking with the Kogge-Stone arithmetic-to-Boolean conversion and the Kogge-Stone masked addition. From their result, the implementation with their masked addition is less effective, requiring about 2.28 times clock cycles, compared to one with their conversion. Hence, this section only review the Kogge-Stone arithmetic-to-Boolean mask conversion as depicted in Algorithm 2.

Algorithm 2 calls subroutines [5] labeled as SecShift, SecAnd, and SecXor which securely execute operations of Shift, AND, and XOR, respectively, by using masks.

2.3 Biryukov et al.'s Masked Addition Algorithm [3]

Biryukov et al. [3] took a comprehensive approach to search optimal algorithms of subroutines which securely execute operations of AND and OR. They then applied these subroutines to construct improved masked addition and subtraction algorithms.

Algorithm 3 shows the improved masked addition algorithm from [3]. This algorithm calls the improved SecAnd labeled as SecAnd2, and two other subroutines labeled as SecShift2 and SecXor2.

3 New Algorithms

We first explain our strategy to improve the arithmetic-to-Boolean conversion algorithm and the masked addition. We then give our algorithms and compare their efficiencies with ones of the previous algorithms.

3.1 Strategy

Our approach is to improve the Coron et al.'s algorithm. We have two ideas to enhance this algorithm.

The first one is to decrease the numbers of required masks from three to two, by replacing the third mask (t in Algorithm 2) with an XOR of other two masks

Algorithm 2 Kogge-Stone Arithmetic-to-Boolean Conversion [5]

Input: $A, r \in \{0,1\}^k$ and $n = \max(\lceil \log_2(k-1) \rceil, 1)$ such that $A = x - r \in \mathbb{F}_{2^k}$
Output: x' such that $x' \oplus r = A + r \mod 2^k$
1: Let $s \leftarrow \{0,1\}^k, t \leftarrow \{0,1\}^k, u \leftarrow \{0,1\}^k$
2: $P' \leftarrow A \oplus s$
3: $P' \leftarrow P' \oplus r$
4: $G' \leftarrow s \oplus ((A \oplus t) \wedge r)$
5: $G' \leftarrow G' \oplus (t \wedge r)$
6: **for** $i := 1$ to $n-1$ **do**
7: $\quad H \leftarrow \mathsf{SecShift}(G', s, t, 2^{i-1})$
8: $\quad U \leftarrow \mathsf{SecAnd}(P', H, s, t, u)$
9: $\quad G' \leftarrow \mathsf{SecXor}(G', U, u)$
10: $\quad H \leftarrow \mathsf{SecShift}(P', s, t, 2^{i-1})$
11: $\quad P' \leftarrow \mathsf{SecAnd}(P', H, s, t, u)$
12: $\quad P' \leftarrow P' \oplus s$
13: $\quad P' \leftarrow P' \oplus u$
14: **end for**
15: $H \leftarrow \mathsf{SecShift}(G', s, t, 2^{n-1})$
16: $U \leftarrow \mathsf{SecAnd}(P', H, s, t, u)$
17: $G' \leftarrow \mathsf{SecXor}(G', U, u)$
18: $x' \leftarrow A \oplus 2G'$
19: $x' \leftarrow x' \oplus 2s$
20: **return** x'

(as in the second line in Algorithm 5). This replacement degrades the level of higher-order security by one; however, masking an internal variable with one of two independent masks and their XOR value is enough to ensure the first-order security. This replacement decreases not only the number of masks itself but also that of XOR operations (for Steps 12 and 13 in Algorithm 2, etc.).

The second one is to decrease the number of re-masking operations within subroutines. In their algorithm, SecShift is followed by SecAnd. In each of SecShift and SecAnd, XOR operations are executed in order for the output to be masked with a certain mask. Our second idea is to remove the operations for re-masking by integrating these two subroutines (named SecShiftAnd in Algorithm 4).

In addition to above two, we also change the initialization steps (Steps 3 to 5 in Algorithm 2) to decrease the number of XOR operations (with input A).

Although our ideas seem to naturally lead an improved algorithm, unfortunately, it is incorrect. This is because, by reducing the number of independent masks, we have to use the mask in different order from the original algorithm so that masks are uncanceled. As seen in the next subsection, we prepare two sequences of operations (Steps 8 to 11 and 13 to 15, *etc.*) to keep the internal variables being masked throughout the conversion.

Algorithm 3 Biryukov et al.'s Masked Addition Algorithm [3]

Input: $x_1, x_2, y_1, y_2 \in \{0,1\}^k$ and $n = \max(\lceil \log_2(k-1) \rceil, 1)$ such that $x = x_1 \oplus x_2$ and $y = y_1 \oplus y_2$
Output: z_1, z_2 such that $z = z_1 \oplus z_2 = (x + y) \mod 2^k$
1: $p_1, p_2 \leftarrow \mathsf{SecXor2}(x_1, x_2, y_1, y_2)$
2: $g_1, g_2 \leftarrow \mathsf{SecAnd2}(x_1, x_2, y_1, y_2)$
3: $g_1, g_2 \leftarrow ((g_1 \oplus x_2) \oplus g_2, x_2)$
4: **for** $i := 1$ to $n-1$ **do**
5: $\quad h_1, h_2 \leftarrow \mathsf{SecShift2}(g_1, g_2, 2^{i-1})$
6: $\quad u_1, u_2 \leftarrow \mathsf{SecAnd2}(p_1, p_2, h_1, h_2)$
7: $\quad g_1, g_2 \leftarrow \mathsf{SecXor2}(g_1, g_2, u_1, u_2)$
8: $\quad h_1, h_2 \leftarrow \mathsf{SecShift2}(p_1, p, 2^{i-1})$
9: $\quad h_1, h_2 \leftarrow ((h_1 \oplus x_2) \oplus h_2, x_2)$
10: $\quad p_1, p_2 \leftarrow \mathsf{SecAnd2}(p_1, p_2, h_1, h_2)$
11: $\quad p_1, p_2 \leftarrow ((p_1 \oplus y_2) \oplus p_2, y_2)$
12: **end for**
13: $h_1, h_2 \leftarrow \mathsf{SecShift2}(g_1, g, 2^{n-1})$
14: $u_1, u_2 \leftarrow \mathsf{SecAnd2}(p_1, p_2, h_1, h_2)$
15: $g_1, g_2 \leftarrow \mathsf{SecXor2}(g_1, g_2, u_1, u_2)$
16: $z_1, z_2 \leftarrow \mathsf{SecXor2}(y_1, y_2, x_1, x_2)$
17: $z_1, z_2 \leftarrow (z_1 \oplus 2g_1, z_2 \oplus 2g_2)$
18: **return** z_1, z_2

3.2 Algorithms

As mentioned, we introduce a combined subroutine SecShiftAnd to accelerate the conversion. Algorithm 4 gives its procedure, which is naturally derived by combining SecShift and SecAnd of [5] without the XOR operations in SecShift.

Algorithm 4 SecShiftAnd

Input: $x'_1, s_1, j, x'_2, s_2, u$ such that $x'_i, s_i, u \in \{0,1\}^k$ and $j \in \mathbb{Z}$ where $x'_i = x_i \oplus s_i$
Output: z' such that $z' = ((x_1 << j) \wedge x_2) \oplus u$
1: $y \leftarrow x'_1 << j$
2: $s' \leftarrow s_1 << j$
3: $z' \leftarrow u \oplus (x'_2 \wedge y)$
4: $z' \leftarrow z' \oplus (x'_2 \wedge s')$
5: $z' \leftarrow z' \oplus (s_2 \wedge y)$
6: $z' \leftarrow z' \oplus (s_2 \wedge s')$
7: **return** z'

We then give our conversion in Algorithm 5. As mentioned in the previous subsection, we use two sequences of operations to be selectively used by the conditions of i (loop counter) and n. Note that these conditions are public and the branches leak no sensitive information of the internal variables.

Algorithm 5 Our Arithmetic-to-Boolean Conversion

Input: $A, r \in \{0,1\}^k$ and $n = \max(\lceil \log_2(k-1) \rceil, 1)$ such that $A = x - r \in \mathbb{F}_{2^k}$
Output: x' such that $x' \oplus r = A + r \mod 2^k$

$s, u \leftarrow \{0,1\}^k$
$t \leftarrow s \oplus u$
$P' \leftarrow A \oplus s$
$G' \leftarrow t \oplus (P' \wedge r)$
$G' \leftarrow G' \oplus (s \wedge r)$
$P' \leftarrow P' \oplus r$
for $i := 1$ to $n-1$ **do**
    if i is odd **then**
        $U \leftarrow \mathsf{SecShiftAnd}(G', t, 2^{i-1}, P', s, u)$
        $G' \leftarrow G' \oplus U$
        $P' \leftarrow \mathsf{SecShiftAnd}(P', s, 2^{i-1}, P', s, t)$
    else
        $U \leftarrow \mathsf{SecShiftAnd}(G', s, 2^{i-1}, P', t, u)$
        $G' \leftarrow G' \oplus U$
        $P' \leftarrow \mathsf{SecShiftAnd}(P', t, 2^{i-1}, P', t, s)$
    end if
end for
if n is odd **then**
    $U \leftarrow \mathsf{SecShiftAnd}(G', t, 2^{n-1}, P', s, u)$
else
    $U \leftarrow \mathsf{SecShiftAnd}(G', s, 2^{n-1}, P', t, u)$
end if
$G' \leftarrow G' \oplus U$
$x' \leftarrow A \oplus 2G'$
if n is odd **then**
    $x' \leftarrow x' \oplus 2s$
else
    $x' \leftarrow x' \oplus 2t$
end if
return x'

3.3 Comparison with Previous Algorithms

Let us compare our algorithm with the previous ones. Table 1 summarizes the numbers of operations required in each algorithm. In [3], Biryukov et al. gave, not an arithmetic-to-Boolean algorithm, but the masked addition and subtraction algorithms. From their masked addition algorithm, an arithmetic-to-Boolean conversion algorithm are naturally derived, which we call the Biryukov et al.'s conversion algorithm. We give its detail as Algorithm 7 in Appendix A.

This table shows that our conversion and addition algorithms require fewer cycles compared to previous ones. For example, the gains of our conversion algorithm against the Coron et al.'s and Biryukov et at.'s ones, for $k = 32$, are 31 ($\approx 22.6\%$) and 8 ($\approx 7.0\%$), respectively.

Algorithm 6 Our Masked Addition Algorithm

Input: $x', y', r, s \in \{0,1\}^k$ and $n = \max(\lceil \log_2(k-1) \rceil, 1)$ such that $x' = x \oplus r$ and $y' = y \oplus s$
Output: z' such that $z \oplus r = x + y \mod 2^k$

$u \leftarrow \{0,1\}^k$
$t \leftarrow s \oplus u$
$z' \leftarrow x' \oplus y'$
$P' \leftarrow z' \oplus r$
$z' \leftarrow z' \oplus s'$
$G' \leftarrow \mathsf{SecAnd}(x', y', s, r, t)$
for $i := 1$ to $n-1$ **do**
    if i is even **then**
        $U \leftarrow \mathsf{SecShiftAnd}(G', t, 2^{i-1}, P', s, u)$
        $G' \leftarrow G' \oplus U$
        $P' \leftarrow \mathsf{SecShiftAnd}(P', s, 2^{i-1}, P', s, u)$
    else
        $U \leftarrow \mathsf{SecShiftAnd}(G', s, 2^{i-1}, P', t, u)$
        $G' \leftarrow G' \oplus U$
        $P' \leftarrow \mathsf{SecShiftAnd}(P', t, 2^{i-1}, P', t, s)$
    end if
end for
if n is even **then**
    $U \leftarrow \mathsf{SecShiftAnd}(G', t, 2^{n-1}, P', s, u)$
else
    $U \leftarrow \mathsf{SecShiftAnd}(G', s, 2^{n-1}, P', t, u)$
end if
$G' \leftarrow G' \oplus U$
$z' \leftarrow z' \oplus 2G'$
if n is even **then**
    $z' \leftarrow z' \oplus 2s$
else
    $z' \leftarrow z' \oplus 2t$
end if
return z'

Let us check the detail of the conversion algorithm. Table 2 summarizes the number of operations required in each algorithm for $k = 32$. Compared to the Coron et al.'s conversion algorithm, although the Biryukov et al.'s one additionally requires 20 orn operations, it decreases the numbers for and and eor operations. In total, the gain is 23.

Our conversion algorithm, on the other hand, decreases the number of operations (only) for eor by 31 from the Coron et al.'s one. The gain is enough large to compensate the overhead on and operation against the Biryukov et al.'s one.

Algorithm	rand	$k=8$	$k=16$	$k=32$	$k=64$	k
Coron et al.'s conversion	3	81	109	137	165	$28\log_2 k - 3$
Biryukov et al.'s conversion	2	70	92	114	136	$22\log_2 k + 4$
Our conversion	2	64	85	106	127	$21\log_2 k + 1$
Coron et al.'s addition	2	88	116	144	172	$28\log_2 k + 4$
Biryukov et al.'s addition	0	70	92	114	136	$22\log_2 k + 4$
Our addition	1	69	90	111	132	$21\log_2 k + 6$

Table 1. Number of operations in each algorithm

Algorithm	and	orn	sft	eor	total
Coron et al.'s conversion	38	0	20	79	137
Biryukov et al.'s conversion	20	20	20	54	114
Our conversion	38	0	20	48	106

Table 2. Number of operations in each algorithm for $k = 32$ (in detail)

4 Experiments

We apply our conversion algorithm to give a first-order secure implementation of HMAC-SHA-1. In considering the Coron et al.'s result, we implement it, *not with the masked addition*, but with Goubin's Boolean-to-arithmetic conversion algorithm and ours. Namely, internal values of (HMAC-)SHA-1 is basically randomized with a first order Boolean mask. In each round operation, we use the rapid Goubin's algorithm to convert inputs with arithmetic masks suitable for additions, and our algorithm *once* to convert the summation back to the data with a Boolean mask.

4.1 Equipments

We developed a software for HMAC-SHA-1 [2, 18, 19] with MDK-lite for Windows, version 5.24.1. The C code was compiled with armcc v5.06 update 5 (build 528) using the O3 optimization to generate an assembly code. We then checked the assembly code not to remove the masking and reverted the code to retrieve the masking back in the assemble level. Finally, the assembly code was compiled with armasm v5.06 update 5 (build 528) to generate a binary code.

We then downloaded the binary code into an IC test vehicle smartcard [7] from Information-technology Promotion Agency (IPA), Japan. This smartcard includes the ARM7 based SC100 with 28MHz system clock, the 512KB flash memory and the 18KB RAM. The interface follows the ISO7816-3 with T=0.

We measured power consumption traces from the smartcard with the SASEBO-W board [16] and the digital oscilloscope LECROY WavePro715Zi. We controlled the SASEBO-W board, with external 2.5V power supply and 3.57MHz

frequency, from a Windows based laptop PC to run the smartcard. We acquired power consumption traces using the oscilloscope with 1G Samples/s.

4.2 Implementation of HMAC-SHA-1

Prototype with Python: Before implementing with C, we first implemented HMAC-SHA-1 with Python from scratch. Table 3 summarizes the number of randomnesses and operations required for HMAC-SHA-1 with each algorithm. For the latter, we count the operations of add, sub, and, or, eor, orr (only for the Biryukov et al.'s conversion), shift, and rot which are supposed to be executed in one clock cycle with ARM processor.

Implementation	♯rand	♯ops	ratio
Without countermeasure	0	4,004	1
With Coron et al.'s conversion	313	63,358	15.82
With Biryukov et al.'s conversion	72	55,173	13.78
With our conversion	72	52,493	13.11

Table 3. Numbers of randomnesses and operations required for HMAC-SHA-1 with Python

From this table, the implementations of HMAC-SHA-1 with a first-order masking require more than tenfold operations compared to the one without any countermeasure. Among those with a first-order masking, our conversion leads the fast implementation. The gains against those with the Coron et al.'s and Biryukov et al.'s algorithms are 10,865 ($\approx$ 17.1%) and 2,680 ($\approx$ 4.9%), respectively.

As for the randomness, implementations with Biryukov et al.'s conversion and ours require 72 masks: 5 masks for the initial five words in each two hashes (subtotal 10), 16 ones for the sixteen words to each three blocks (subtotal 48), 11 ones for the remaining block (the first five words out of sixteen ones, which is an output of the inner hash, are already masked), one for the Boolean-to-arithmetic conversion, and two for the arithmetic-to-Boolean conversion.

C implementation for IPA Test Vehicle Smartcard: We then implemented HMAC-SHA-1, in C with assembly modification, with/without a countermeasure. In the implementations with countermeasures, the masks are generated by xorshift [14] with a random seed. Table 4 summarizes the numbers of cycles required for HMAC-SHA-1 in the test vehicle smartcard.

In this table, "opt0" means implementations where they are masked throughout the 80 rounds, but two results of SHA-1 compressions for the fixed keys are pre-computed. "opt1" and "opt2" mean optimized implementations where internal values in 40 and 60 middle rounds of SHA-1 are unmasked, respectively. The

Implementation	♯cycles	ratio
Without countermeasure	12,391	1
With Coron et al.'s conversion (opt0)	68,711	5.55
With Biryukov et al.'s conversion (opt0)	66,344	5.35
With our conversion (opt0)	63,546	5.13
With Coron et al.'s conversion (opt1)	41,914	3.38
With Biryukov et al.'s conversion (opt1)	40,913	3.30
With our conversion (opt1)	39,471	3.19
With Coron et al.'s conversion (opt2)	29,150	2.35
With Biryukov et al.'s conversion (opt2)	28,629	2.31
With our conversion (opt2)	27,862	2.25

Table 4. Number of cycles required for HMAC-SHA-1 in IPA test vehicle smartcard

implementations with the "opt0" countermeasure require more than five times cycles compared to one without a countermeasure. If two hashes for the fixed keys are not precomputed, they should be tenfold as in Table 3. Compared to those with Coron et al.'s conversion, our gains are 5,165 ($\approx 7.5\%$, "opt0"), 2,443 ($\approx 5.8\%$, "opt1"), and 1,288 ($\approx 4.4\%$, "opt2"), respectively. Compared to those with Biryukov et al.'s conversion, our gains are 2,798 ($\approx 4.2\%$, "opt0"), 1,442 ($\approx 3.5\%$, "opt1"), and 767 ($\approx 2.7\%$, "opt2"), respectively.

Table 5 summarized the code sizes of C implementations. From the table, our algorithm leads the smallest code.

Implementation	size (byte)
With Coron et al.'s conversion (opt2)	2,780
With Biryukov et al.'s conversion (opt2)	2,700
With our conversion (opt2)	2,640

Table 5. Code sizes for HMAC-SHA-1 in IPA test vehicle smartcard

4.3 Test for the First-Order Side-Channel Leakage

We implemented a software of HMAC-SHA-1 with our conversion (opt2) on the test vehicle smartcard and acquired 100,000 power consumption traces, as specified for the security level 4 in ISO/IEC 17825 [8], with a fixed key and random inputs.

HMAC-SHA-1, with a key *key* and a message *msg*, computes $\mathsf{sha1}((key \oplus opad)||\mathsf{sha1}((key \oplus ipad)||msg))$, where *ipad* and *opad* are the constants $\mathsf{0x3636\cdots36}$ and $\mathsf{0x5c5c\cdots5c}$, respectively. As explained, in each sha1 computations, the compression of first block with the fixed key is precomputed. Our software computes

the compression of the second block with *msg* in the inner hash, using the compression of the first block as an initial vector. We regard the output of first round in the second compression with *msg* as an attack target.

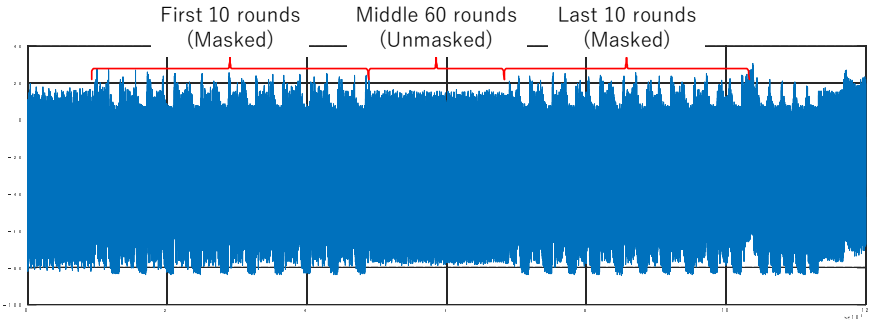

Fig. 1. Average traces for $\mathsf{sha1}(msg)$ with IV= $\mathsf{sha1}(key \oplus ipad)$

Figure 1 depicts the average traces of 1,000 first power consumption traces, where its horizontal and vertical axes represent time and power consumption (in voltage), respectively. The iterations of rounds are observed as specified in this figure. From each trace, we extract a subtrace for the first round by removing a random delay in the processing time for synchronization.

We divided the 100,000 traces into two groups by the first byte of the target, whether its Hamming weight is more than 16 or less than 16. We then applied the test from ISO/IEC 17825 [8]. The graph in Figure 2 shows the result, where its horizontal and vertical axes represent time and T-value, respectively. In this figure, there is no point which exceeds the threshold 4.5 (less than ± 0.5) and we concludes that our conversion protect HMAC-SHA-1 from the first-order attack.

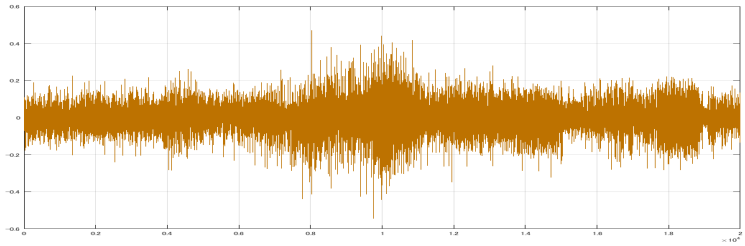

Fig. 2. ISO/IEC 17825 tests for our target

5 Discussions

As we explained in Section 3.3, our algorithm requires fewer operations than the previous algorithms to convert the arithmetic mask to the Boolean one. Moreover, as shown in Section 4.3, our algorithm gives a secure implementation against the first-order side-channel attack.

5.1 Branches

Unlike the previous algorithms, ours has branches of operations based on i and n as in Algorithm 5. Note that, since i and n do not depend on the sensitive data, these branches leak no information about secret inputs. However, it increases the execution time as it is. As for the branches conditioned by $n = \max(\lceil \log_2(k-1) \rceil, 1)$, if an architecture, especially, the size of addition k, is determined in advance, it is sufficient to implement a corresponding one of the operation sequences without a branch on n. As for the branch conditioned by i, on the other hand, we can remove it if the implementation of **for** loop is unrolled. This leads a trade-off between the time and the code size. In our previous experiment, we implemented the unrolled[2] architecture in Algorithm 5.

5.2 Conversion of Higher-Order Masking

In this paper, we gave an arithmetic-to-Boolean conversion algorithm for a first-order masking. Following the discussion of [5, Section 6], our algorithm is extensible to a conversion of higher-order masking, as well as the Coron et al.'s algorithm.

6 Conclusion

In this paper, we proposed another improved conversion algorithm from the Coron et al.'s one, by reducing operations over subroutines. Our experiments, with the IPA test vehicle smartcard, showed that our conversion correctly worked as a countermeasure against the first-order attack. Discussions on sophisticated attacks such as (non-)profiled attacks and their countermeasures are our future works.

Acknowledgement

In this paper, we use the test vehicle smartcard from IPA. We would like to thank anonymous reviewers for their fruitful comments on the previous version of this manuscript.

[2] Note that the implementation of SHA-1 is in the rolled architecture.

References

1. Sonia Belaïd, Luk Bettale, Emmanuelle Dottax, Laurie Genelle, and Franck Rondepierre. Differential power analysis of HMAC SHA-2 in the hamming weight model. In Pierangela Samarati, editor, *SECRYPT 2013 - Proceedings of the 10th International Conference on Security and Cryptography*, pages 230–241. SciTePress, 2013.
2. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1996.
3. Alex Biryukov, Daniel Dinu, Yann Le Corre, and Aleksei Udovenko. Optimal first-order Boolean masking for embedded IoT devices. In Thomas Eisenbarth and Yannick Teglia, editors, *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017*, volume 10728 of *Lecture Notes in Computer Science*, pages 22–41. Springer, 2018.
4. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
5. Jean-Sébastien Coron, Johann Großschädl, Mehdi Tibouchi, and Praveen Kumar Vadnala. Conversion from arithmetic to Boolean masking with logarithmic complexity. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015*, volume 9054 of *Lecture Notes in Computer Science*, pages 130–149. Springer, 2015.
6. Louis Goubin. A sound method for switching between Boolean and arithmetic masking. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 3–15. Springer, 2001.
7. Toru Hashimoto and Boutheina Chetali. High level CC certification in Japan. In *The 2013 International Common Criteria Conference, ICCC 2013*, 2013. `https://www.commoncriteriaportal.org/iccc/ICCC_arc/presentations/T2_D1_4_30pm_Hashimoto_High_Level_CC_Certs.pdf`.
8. ISO/IEC. *ISO/IEC 17825. Information technology – Security techniques – Testing methods for the mitigation of non-invasive attack classes against cryptographic modules*. ISO/IEC, 2016.
9. Bernhard Jungk, Richard Petri, and Marc Stöttinger. Efficient side-channel protections of ARX ciphers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 1(3):627–653, 2018.
10. Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
11. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

12. Peter M. Kogge and Harold S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans. Computers*, 22(8):786–793, 1973.
13. Kerstin Lemke, Kai Schramm, and Christof Paar. DPA on n-bit sized Boolean and arithmetic operations and its application to IDEA, RC6, and the HMAC-construction. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 205–219. Springer, 2004.
14. George Marsaglia. Xorshift RNGs. *Journal of Statistical Software*, 8:1–6, 2003.
15. Robert P. McEvoy, Michael Tunstall, Colin C. Murphy, and William P. Marnane. Differential power analysis of HMAC based on SHA-2, and countermeasures. In Sehun Kim, Moti Yung, and Hyung-Woo Lee, editors, *Information Security Applications, 8th International Workshop, WISA 2007*, volume 4867, pages 317–332. Springer, 2007.
16. National Institute of Advanced Industrial Science and Technology. Side-channel Attack Standard Evaluation Board (SASEBO), SASEBO-W. 2012. `http://satoh.cs.uec.ac.jp/SASEBO/en/board/sasebo-w.html`.
17. National Institute of Standards and Technology (NIST). Federal Information Processing Standards Publication (FIPS) 197, Advanced Encryption Standard (AES). 2001. `https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf`.
18. National Institute of Standards and Technology (NIST). Federal Information Processing Standards Publication (FIPS) 198-1, The Keyed-Hash Message Authentication Code (HMAC). 2008. `https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.198-1.pdf`.
19. National Institute of Standards and Technology (NIST). Federal Information Processing Standards Publication (FIPS) 180-4, Secure Hash Standard (SHS). 2015. `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf`.
20. Tobias Schneider, Amir Moradi, and Tim Güneysu. Arithmetic addition over boolean masking - towards first- and second-order resistance in hardware. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*, volume 9092 of *Lecture Notes in Computer Science*, pages 559–578. Springer, 2015.
21. Yoo-Seung Won and Dong-Guk Han. Efficient conversion method from arithmetic to boolean masking in constrained devices. In Sylvain Guilley, editor, *Constructive Side-Channel Analysis and Secure Design - 8th International Workshop, COSADE 2017, Paris, France, April 13-14, 2017, Revised Selected Papers*, volume 10348 of *Lecture Notes in Computer Science*, pages 120–137. Springer, 2017.

A Arithmetic-to-Boolean Conversion based on Biryukov et al.'s Addition

Based on the Biryukov et al.'s masked addition of Algorithm 3, an arithmetic-to-Boolean conversion is naturally derived. Algorithm 7 shows the conversion. It requires two random masks as ours does.

Algorithm 7 Arithmetic-to-Boolean Conversion based on Biryukov et al.'s Addition

Input: $A, r \in \{0,1\}^k$ and $n = \max(\lceil \log_2(k-1) \rceil, 1)$ such that $A = x - r \in \mathbb{F}_{2^k}$
Output: x' such that $x' \oplus r = A + r \mod 2^k$

$x_2, y_2 \leftarrow \{0,1\}^k$
$x_1 \leftarrow A \oplus x_2$
$y_1 \leftarrow r \oplus y_2$
$p_1, p_2 \leftarrow \mathsf{SecXor2}(x_1, x_2, y_1, y_2)$
$g_1, g_2 \leftarrow \mathsf{SecAnd2}(x_1, x_2, y_1, y_2)$
$g_1, g_2 \leftarrow ((g_1 \oplus x_2) \oplus g_2, x_2)$
for $i := 1$ to $n-1$ **do**
    $h_1, h_2 \leftarrow \mathsf{SecShift2}(g_1, g_2, 2^{i-1})$
    $u_1, u_2 \leftarrow \mathsf{SecAnd2}(p_1, p_2, h_1, h_2)$
    $g_1, g_2 \leftarrow \mathsf{SecXor2}(g_1, g_2, u_1, u_2)$
    $h_1, h_2 \leftarrow \mathsf{SecShift2}(p_1, p, 2^{i-1})$
    $h_1, h_2 \leftarrow ((h_1 \oplus x_2) \oplus h_2, x_2)$
    $p_1, p_2 \leftarrow \mathsf{SecAnd2}(p_1, p_2, h_1, h_2)$
    $p_1, p_2 \leftarrow ((p_1 \oplus y_2) \oplus p_2, y_2)$
end for
$h_1, h_2 \leftarrow \mathsf{SecShift2}(g_1, g, 2^{n-1})$
$u_1, u_2 \leftarrow \mathsf{SecAnd2}(p_1, p_2, h_1, h_2)$
$g_1, g_2 \leftarrow \mathsf{SecXor2}(g_1, g_2, u_1, u_2)$
$x' \leftarrow A \oplus 2g_1 \oplus 2g_2$
return x'