Asymmetric Leakage from Multiplier and Collision-Based Single-Shot Side-Channel Attack

doi: 10.1587/transfun.E99.A.1323

PAPER Special Section on Design Methodologies for System on a Chip

Asymmetric Leakage from Multiplier and Collision-Based Single-Shot Side-Channel Attack**

Takeshi SUGAWARA[†a)], **Daisuke SUZUKI**[†], ***Members***, ***and*** **Minoru SAEKI**[††*], ***Nonmember***

SUMMARY The single-shot collision attack on RSA proposed by Hanley et al. is studied focusing on the difference between two operands of multiplier. It is shown that how leakage from integer multiplier and long-integer multiplication algorithm can be asymmetric between two operands. The asymmetric leakage is verified with experiments on FPGA and micro-controller platforms. Moreover, we show an experimental result in which success and failure of the attack is determined by the order of operands. Therefore, designing operand order can be a cost-effective countermeasure. Meanwhile we also show a case in which a particular countermeasure becomes ineffective when the asymmetric leakage is considered. In addition to the above main contribution, an extension of the attack by Hanley et al. using the signal-processing technique of Big Mac Attack is presented.

key words: *RSA, side-channel attack, collision attack, Montgomery multiplication*

1. Introduction

Side-channel attack (SCA) is one of the main research subjects in cryptography implementation. The attack exploits side-channel information, such as power dissipation and electromagnetic radiation, caused as a side-effect of cryptographic operation. With the help of side-channel information, attacker may successfully analyze cryptography [2]. The first attacks known as simple power analysis (SPA) and differential power analysis (DPA) were presented by Kocher et al. in 1999 [3]. Since then, new attack and countermeasure have been studied so far.

Kocher et al. presented the first SCA against RSA [3]. The basic concept of the attack is to directly decode a secret exponent by distinguishing multiplication from squaring in a common modular exponentiation algorithm. That is a single-shot attack. In other words, a trace of single RSA execution is sufficient to attack the cryptography. The single-shot attack is feasible only when the difference between multiplication and squaring are easily observable e.g., there is a data-dependent conditional branch. In order to counteract the attack, branch-less algorithms are developed. The well-known multiply-always method in Alg. 1 falls in this category.

Even after data-dependent branch is removed, there is residual side-channel leakage correlated to the operands of the multiplication and squaring [4]. The residual leakage is small [5], but further research has been conducted in order to exploit it. Using multiple traces is one direction of research, however, there are efficient countermeasures against multiple-shot attacks [6]. Therefore, improving single-shot attack has become an important challenge. As a result of improvements in measurement and signal processing, successful single-shot attacks on FPGA implementations were recently reported [7]–[9]. That means the residual leakage should be suppressed for secure implementation.

Consequently, suppressing the residual leakage has become a new challenge. Hanley et al. suggested to use multiplier-level countermeasure [10]–[12] and that seems to be a promising approach. For leakage suppression, deep understanding of leakage from multiplier is needed. Since multiplier is a core component in RSA implementation, it has been extensively studied from arithmetic-circuit to architectural levels. In contrast, there are quite few conventional studies on the way how multiplier makes side-channel leakage [10], [13], [14].

In this paper, leakage from multiplier is studied focusing on difference between two operands. It is shown that how leakage from integer multiplier and long-integer multiplication algorithm can be asymmetric between operands. The asymmetric leakage is verified with experiments on FPGA and micro-controller platforms. Moreover, the experimental results show that the order of operands significantly affect the success rate of a single-shot attack. Its consequence is two folds: (i) designing operand order can be a cost-effective countermeasure, while (ii) some countermeasures become ineffective when the asymmetric leakage is considered.

A1 It is shown that how leakage from integer multiplier and long-integer multiplication algorithm can be asymmetric between two operands.
A2 The asymmetric leakage is verified in FPGA and micro-controller platforms. In the experiment, it is also shown that the order of operands significantly affect the success rate of a single-shot attack.
A3 It is shown that designing the order of operands can be a cost-effective countermeasure.
A4 It is shown that a specific countermeasure becomes

Manuscript received September 11, 2015.
Manuscript revised January 6, 2016.
[†]The authors are with Mitsubishi Electric Corporation, Kamakura-shi, 247-8501 Japan.
[††]The author is with Information-technology Promotion Agency, Tokyo, 113-6591 Japan.
[*]The research was conducted while the author was with Mitsubishi Electric Corporation.
[**]The paper is based on a preliminary version appeared at the sixth international workshop on constructive side-channel analysis and secure design (COSADE 2015) [1].
a) E-mail: Sugawara.Takeshi@bp.MitsubishiElectric.co.jp
DOI: 10.1587/transfun.E99.A.1323

ineffective when the asymmetric leakage is considered.

In addition to the above main results, there are two additional contributions.

B1 The single-shot attack by Hanley et al. [9] is extended using the technique of Big Mac Attack [13].
B2 The first experimental result of successfully analyzing an FPGA implementation of RSA with the multiply-always method using single-shot internal-collision attack is presented.

The paper is organized as follows. Its correspondence to the contributions **A1**-**A4** and **B1**-**B2** is also described. In Sect. 2, the related studies are reviewed followed by the proposed extension of the attack by Hanley et al. (**B1**). In Sect. 3, the way how leakage from integer multiplier and long-integer multiplication algorithm can be asymmetric is discussed (**A1**). Then, the asymmetric leakage is verified with experiments in Sect. 4 (**A1**). In the section, the attack in Sect. 2 is applied to an FPGA implementation with various operand orders (**B2**). The experimental results are discussed in Sect. 5. It is shown that the asymmetry can be used as a cost-effective countermeasure (**A3**) meanwhile it can cause a problem in a certain countermeasure (**A4**). It is also shown that the asymmetric leakage is measurable in a microcontroller and thus the results in this paper are also relevant to software implementation of cryptography (**A2**). Sect. 6 is a concluding remark.

2. Single-Shot Collision Attack

Conventional attacks are briefly reviewed. Then, the two most relevant attacks namely (i) the multiple-shot attack by Witteman et al. [15] and (ii) the single-shot attack by Hanley et al. [9] are described in detail. Finally, the proposed extension of the attack by Hanley et al. is described.

2.1 Conventional Single-Shot Attacks

Simple Power Analysis (SPA) [3] As described in the introduction, Kocher et al. proposed the first single-shot attack on RSA. The attack exploits binary method used for modular exponentiation. In binary method, there is a conditional branch between multiplication and squaring depending a secret bit. The idea of the attack is to directly decode a secret exponent by distinguishing multiplication from squaring by inspecting a side-channel trace.
Big Mac Attack (BMA) [13] Walter proposed BMA to attack another modular exponentiation algorithm called the window method [13]. In the attack, two distinct parts of a trace are compared to find collision. Information of presence (or equivalently absence) of collision can be used to attack cryptography. Along with the attack, a sophisticated signal-processing technique is introduced in order to efficiently find the collision. Firstly, a segment to be compared is split into multiple sub-segments. Then the sub-segments are averaged together and an averaged segment is obtained. Finally, two averaged segments are compared to detect collision. Since signal-to-noise ratio (SNR) of the averaged segments are higher because of the averaging, collision is efficiently detected. Feasibility of the attack is proved with simulation [13]. However, no practical result has been reported as described by Clavier et al. [16].

Algorithm 1 Multiply-Always Method with Left-To-Right Scanning

Input: Message M, Modulo N, Secret exponent $d = (d_{t-1}, \cdots, d_0)_2$
Output: Ciphertext M^d
1: $R_0 \leftarrow 1$
2: **for** $j = t-1$ downto 0 **do**
3: $\quad \overline{d_j} \leftarrow 1 - d_j$
4: $\quad R_0 \leftarrow R_0^2 \mod N$
5: $\quad R_{\overline{d_j}} \leftarrow R_0 \times M \mod N$
6: **end for**
7: Return R_0

Horizontal Correlation Power Analysis (HCPA) [10] HCPA proposed by Clavier et al. is a successor of BMA [10]. In the attack, a single trace is split into many sub-traces in the same manner as BMA. Then, a multiple-shot attack is mounted to the virtual multiple traces. There are experimental results successfully attacking software implementations [10].
Clustering-based attack [7] Heyszl et al. proposed an attack using the k-means clustering algorithm [7]. In the attack, traces are classified into two clusters that supposedly correspond to the value of a secret bit. In addition, high-quality electromagnetic traces are measured and used for the analysis. The attack is then improved by Perin et al. [8]. As a result of the improvements both in distinguisher and measurement, successful attack on FPGA platform is reported [7], [8]. Notably, Perin et al. succeeded in attacking an FPGA implementation with a multiplier-level countermeasure called the leak resilient arithmetic [12] by exploiting the remaining first-order leakage.

2.2 Multiple-Shot Internal-Collision Attack by Witteman et al. [15]

Witteman et al. proposed a new multiple-shot attack which exploits collision between consecutive operations i.e., internal collision [15]. The attack on the multiply-always method (Alg. 1) is described.

The consecutive multiplication and squaring in Alg. 1 are considered. For clarity they are rewritten as

$$R'_{\overline{d_j}} \leftarrow R_0 \times M \mod N, \quad (1)$$

$$R''_0 \leftarrow R'^2_0 \mod N. \quad (2)$$

If $\overline{d_j} = 1$, the memory R_0 is not updated in Eq. (1) and thus $R_0 = R'_0$. Therefore, the consecutive multiplication and squaring have the same input i.e., there is collision. Alternatively when $\overline{d_j} = 0$, $R_0 \neq R'_0$ and there is no collision. As a result, a secret bit is revealed every single time presence/absence of collision is observed.

The collision is detected using correlation-coefficient matrix. Suppose L different messages are encrypted with the same exponent. Traces for multiplication and squaring for the i-th message are denoted by m_t^i and s_t^i, respectively. Note that the subscript t is time index. The correlation coefficient matrix $C_{x,y}$ is calculated as follows:

$$C_{x,y} = \mathcal{F}_j[m_x^j, s_y^j]. \tag{3}$$

Here, $\mathcal{F}_j$ is the correlation-coefficient operator defined by

$$\mathcal{F}_j[m^j, s^j] := \frac{1}{L}\sum_{j=0}^{L-1} \frac{(m^j - \mathcal{E}_j[m^j]) \cdot (s^j - \mathcal{E}_j[s^j])}{\sqrt{\mathcal{V}_j[m^j] \cdot \mathcal{V}_j[s^j]}}, \tag{4}$$

$$\mathcal{E}_j[m^j] := \frac{1}{L}\sum_{j=0}^{L-1} m^j \tag{5}$$

$$\mathcal{V}_j[m^j] := \mathcal{E}_j[(m^j)^2] - \mathcal{E}_j[m^j]^2. \tag{6}$$

If there is a collision, $C_{x,y}$ contains a non-zero value. Therefore, the collision can be found by inspecting the matrix.

2.3 Single-Shot Internal-Collision Attacks by Hanley et al. [9]

Hanley et al. proposed a single-shot attack against various addition-chain algorithms. The attack uses internal collision similarly to the one by Witteman et al. but it uses only a single trace. That enables the new attack to defeat the blinding countermeasure [6]. In the following description, the attack for the multiply-always method is described.

The attack by Witteman et al. cannot be used for the single-shot setting because Eq. (4) is useless for $L = 1$[†]. Instead, Hanley et al. proposed to compare the traces m_x^0 and s_x^0 in time domain. Two different ways are proposed for measuring the similarity: the Euclidean distance and the time-domain correlation coefficient given by $\mathcal{F}_x[m_x^0, s_x^0]$.

Hanley et al. applied the attack to a software implementation and successfully recovered 99% of the exponent bits. They also applied the attack to an FPGA implementation, however, the attempt was unsuccessful for the multiply-always method. The result can be reasoned by lower SNR in side-channel information from FPGA. SNR has a big impact on the success rate of single-shot attack.

The attack uses multiple points of interest. That is the advantage of the attack over the clustering-based attacks [8]. Therefore, the multiply-always method can be defeated even if there is no first-order leakage [8]. In addition, the attack is advantageous to HCPA on the point that is feasible with unknown message. In other words, the attack by Hanley et al. defeats the message-blinding countermeasure [6].

2.4 Proposed Extension of the Attack by Hanley et al.

An extension of the attack by Hanley et al. is described. The idea is to use the technique from BMA as a preprocessing thereby improve SNR [††].

Multiplication and squaring found in modular exponentiation algorithm is realized by long-integer multiplication. Hereafter, long-integer multiplication $A \times B$ is considered. A and B are composed of s words and denoted by $A = \{a_{s-1}, \cdots, a_0\}$ and $B = \{b_{s-1}, \cdots, b_0\}$ where a_j and b_i are words. The long-integer multiplication comprises partial-product generation namely $a_j \times b_i$. The leakage of $a_j \times b_i$ is denoted by $l(j, i)$.

The trace $l(j, i)$ is pre-processed as shown in Fig. 1. $l(j, i)$ is compressed into two s-dimensional vectors $l_a(j)$ and $l_b(i)$ defined by

$$l_a(j) = \frac{1}{s}\sum_{i=0}^{s-1} l(j, i), \tag{7}$$

$$l_b(i) = \frac{1}{s}\sum_{j=0}^{s-1} l(j, i). \tag{8}$$

$l_a(j)$ and $l_b(i)$ are called the compressed vectors. By the compression, the effect of one operand is removed thereby improving SNR of another. The compressed vectors $l_a(j)$ and $l_b(i)$ correlate to a_j and b_i, respectively. This is the same approach as BMA.

Finally, the compressed vectors from multiplication and squaring traces are compared in the same manner as the original attack by Hanley et al. The measured traces of multiplication and squaring are denoted by $l(j, i)$ and $l'(j, i)$, respectively. The corresponding compressed vectors are denoted by $l_a(j)$, $l'_a(j)$, $l_b(i)$, and $l'_b(i)$. The compressed vectors are compared with correlation coefficient as follows:

$$\mathcal{F}_j[l_a(j), l'_a(j)] \in [-1, 1], \tag{9}$$

$$\mathcal{F}_i[l_b(i), l'_b(i)] \in [-1, 1]. \tag{10}$$

Either Eqs. (9) or (10) is used depending on which operand has collision.

In order to conduct the attack, the attacker should know the positions of $l(j, i)$ in a raw trace. That is the same prerequisite as BMA and HCPA. Even if the prior knowledge is unavailable, the attacker can possibly reverse-engineer the points of $l(j, i)$ by analyzing the correlation matrix in Eq. (3). That is because patterns found in the matrix reflect the underlying long-integer multiplication algorithm. That is explained in Sect. 4.2 with experimental results. Note that in order to get a meaningful correlation matrix, the exponent blinding should be disabled. There are some realistic situations in which the condition is satisfied. Firstly, the attacker with an open sample can possibly profile the device while disabling the countermeasure. Secondly, the same co-processor for modular exponentiation may be used for another purpose without the exponent blinding. One such example is signature verification in which no secret is involved.

[†]Note that Clavier et al. proposed another single-shot extension [16] which aim at distinguishing multiplication from squaring.

[††]The method can be thought as a missing variant with "regular algorithm + unknown message" in the categorization by Bauer et al. [11].

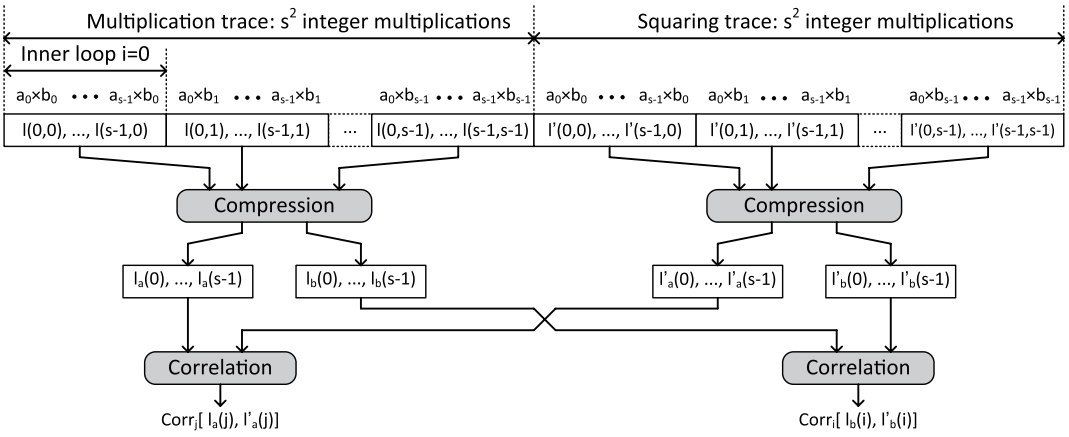


Fig. 1 The distinguisher based on Big Mac Attack.

3. Asymmetric Leakage

Difference between two operands of multiplier is discussed in (i) integer multiplier and (ii) long-integer multiplication (LIM) levels.

3.1 Asymmetry at Integer Multiplier Level

In the paper of BMA, Walter showed that two operands of a simple integer multiplier are symmetric in terms of side-channel leakage [13]. However, sophisticated multipliers can be asymmetric as described below.

The Booth recoding is a common technique for partial product generation † [17]. The technique enables to reduce the total number of partial products thereby improving performance. Figure 2 shows a circuit for generating one partial product using the radix-4 Booth recoding. The word length is denoted by w. Firstly, the multiplicand A is expanded to $\{\overline{2A}, \overline{A}, 0, A, 2A\}$. The expansion is efficiently implemented using arithmetic shift and NOT gates. Then, one out of five candidates is selected at the 5:1 selector. The selector output is the partial product. The selector is controlled by a 3-bit chunk of the multiplier namely $\{x_{i+1}, x_i, x_{i-1}\}$.

The circuit in Fig. 2 has asymmetry in terms of operands. Leakage from a 32-bit integer multiplier with the radix-4 Booth recoding is investigated with simulation. The multiplier is synthesized and post-synthesis logic simulation is conducted. During the simulation, the number of signal-transition events i.e., toggles is counted and recorded.

Two testvectors namely A and B are examined. Firstly, 32-bit integers c and h_i are generated for $0 \le i < 10,000$. In the testvector A, $c \times h_i$ is calculated. In this case, the circuit is driven with fixed multiplier and variable multiplicand. In the testvector B, $h_i \times c$ is calculated wherein the circuit is driven with variable multiplier and fixed multiplicand.

Histograms of the measured toggle counts are shown

†Note that Walter and Samyde noticed that leakage from the Booth recoding does not obey by the Hamming-weight model [14]. However, the difference between operands was not mentioned.

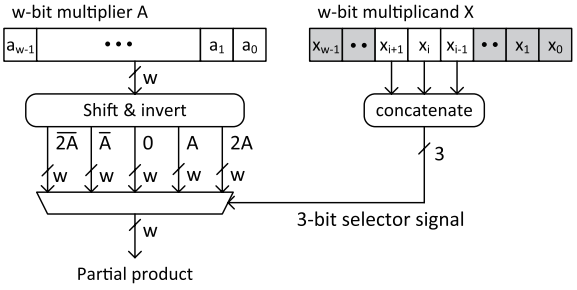


Fig. 2 Partial-product generator for the radix-4 Booth recoding.

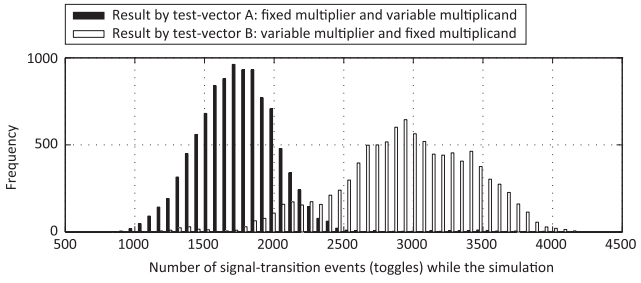


Fig. 3 Toggle counts of a 32-bit multiplier.

in Fig. 3. The black and white bars correspond to the result by two sets of test-vectors. The result shows that histograms distribute differently depending on the test-vector. As shown in the histograms, more toggles are observed when the multiplicand is fixed. In other words, the number of signal-transition events is more susceptible to the multiplier port compared the multiplicand port. The result is explained by an empirical fact that a selector signal has stronger effect on toggle counts. In Fig. 2, the 3-bit selector signal $\{x_{i+1}, x_i, x_{i-1}\}$ drives the w-bit data path. Therefore, changes in the 3-bit signal are amplified to w bits.

3.2 Asymmetry at Long-Integer Multiplication Level

Difference of operands at the LIM level is discussed. The Montgomery multiplication [18] with the coarsely integrated operand scanning (CIOS) [19] shown in Alg. 2 is

Algorithm 2 Coarsely Integrated Operand Scanning [19]

Input: Word $A = \{a_{s-1}, \cdots, a_0\}$ and $B = \{b_{s-1}, \cdots, b_0\}$
Output: Product t_i for $i \in [0, s-1]$

1: **for** $i = 0$ to $s-1$ **do**
2: $C \leftarrow 0$
3: **for** $j = 0$ to $s-1$ **do**
4: $(C, S) \leftarrow t_j + a_j \times b_i + C$
5: $t_j \leftarrow S$
6: **end for**
7: $(C, S) \leftarrow t_s + C$
8: $t_s \leftarrow S$
9: $t_{s+1} \leftarrow C$
10: $C \leftarrow 0$
11: $m \leftarrow t_0 \times n'_0 \mod W$
12: **for** $j = 0$ to $s-1$ **do**
13: $(C, S) \leftarrow t_j + m \times n_j + C$
14: $t_j \leftarrow S$
15: **end for**
16: $(C, S) \leftarrow t_s + C$
17: $t_s \leftarrow S$
18: $t_{s+1} \leftarrow t_{s+1} + C$
19: **for** $j = 0$ to s **do**
20: $t_j \leftarrow t_{j+1}$
21: **end for**
22: **end for**
23: Return t_j

considered. Inputs are expressed as $A = \{a_{s-1}, \cdots, a_0\}$ and $B = \{b_{s-1}, \cdots, b_0\}$ where a_j and b_i are words. The core operation is partial-product generation $a_j \times b_i$ at the line 4 of Alg. 2. Fig. 4 shows a common circuit architecture for LIM using a w-bit multiply-and-accumulate (MAC) unit [20]. The MAC unit operates $(x, y, z) \mapsto z + x \times y$ where z is usually an accumulator. The words a_j and b_i are read from the memory and fed to the MAC unit via w-bit temporal registers labelled `regA` and `regB`.

Suppose `regA` and `regB` store the long integers A and B, respectively. Figure 4 also shows an operation sequence describing the contents of the registers. As shown in the table, `regB` is updated less frequently because b_i is scanned at the outer loop in Alg. 2. For s-word long-integer multiplication, `regA` and `regB` are updated s^2 and s times, respectively.

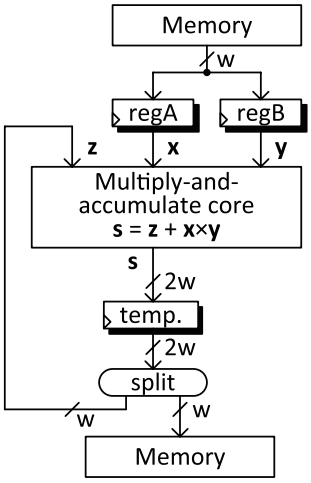
In CMOS, a strong dynamic current is caused when input is changed [2]. As a result, the operand scanned at the inner loop (i.e., A) has stronger leakage. This LIM-level asymmetry is verified through experiments in Sect. 4.

4. Experiments

The attack described in Sect. 2.4 is applied to an FPGA implementation of the Montgomery multiplication. The experiment is conducted for all the possible operand orders in order to investigate how asymmetric leakage affect success rate of the attack.

4.1 Setup

A circuit implementing the 1024-bit Montgomery multiplication is examined. The circuit uses the MAC-based architecture in Fig. 4. The MAC unit has a 64-bit integer multiplier and thus the number of words $s = 16 = 1024/64$. The words are scanned with the CIOS method in Alg. 2. The MAC unit has a special 1-bit input with which operands to the integer multiplier is swapped. The signal is used in order to evaluate the asymmetry at the integer-multiplier level.



An operation sequence when s=3

Seq. no.	regA	regB
0	a_0	b_0
1	a_1	b_0
2	a_2	b_0
3	a_0	b_1
4	a_1	b_1
5	a_2	b_1
6	a_0	b_2
7	a_1	b_2
8	a_2	b_2

Fig. 4 Common circuit architecture for long-integer multiplication using an multiply-and-accumulate unit.

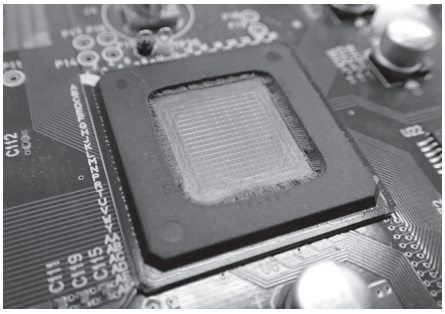


Fig. 5 A de-packaged FPGA chip for measurement.

The design of the 1024-bit Montgomery multiplication is implemented on Virtex-II Pro FPGA on SASEBO [21]. For preparation, the chip is de-packaged as shown in Fig. 5. Then, traces are measured by putting a magnetic-field probe on the die surface [22]. The probe is 0.1 mm in diameter. Traces are captured using an oscilloscope with the bandwidth of 12.5 GHz and the sampling rate of 25.0 GSa/s.

Test-vectors are designed to emulate RSA with the multiply-always method (see Alg. 1). Firstly, 1024-bit random numbers u_k, v_k, and w_k are generated for $k \in [0, 999]$. For each triplet (u_k, v_k, w_k), the Montgomery multiplication denoted by $\mathcal{M}(\cdot, \cdot)$ is called in five ways: $\mathcal{M}(u_k, v_k)$, $\mathcal{M}(u_k, v_k)$ with its integer operand swapped, $\mathcal{M}(u_k, u_k)$, $\mathcal{M}(v_k, v_k)$, and $\mathcal{M}(w_k, w_k)$. The five Montgomery multiplications are called with identifiers namely (UV), $(\overline{UV})$, (UU), (VV), and (WW). (UV) and $(\overline{UV})$ are different in the order of operands to the 64-bit integer multiplier. That is realized by the afore-mentioned special feature of the MAC unit. Table 1 also summarizes the correspondence between

Table 1 Test-vectors of the Montgomery multiplication.

			Correspondence between (u_k, v_k, w_k) and multiplier operand			
			LIM level		integer-multiplier level	
Identifier	Operation	Trace	Inner loop (a_j in Alg. 2)	Outer loop (b_i in Alg. 2)	multiplier	multiplicand
(UV)	$\mathcal{M}(u_k, v_k)$	$T_{uv}(k,t)$	u_k	v_k	v_k	u_k
$(\overline{UV})$	$\mathcal{M}(u_k, v_k)$	$T_{\overline{uv}}(k,t)$	u_k	v_k	u_k	v_k
(UU)	$\mathcal{M}(u_k, u_k)$	$T_{uu}(k,t)$	u_k	u_k	u_k	u_k
(VV)	$\mathcal{M}(v_k, v_k)$	$T_{vv}(k,t)$	v_k	v_k	v_k	v_k
(WW)	$\mathcal{M}(w_k, w_k)$	$T_{ww}(k,t)$	w_k	w_k	w_k	w_k

Table 2 Examined pairs of traces.

	Traces		Colliding operand	
Identifier	Multiplication	Squaring	LIM-level	integer-multiplier-level
(UV, UU)	$T_{uv}(k,t)$	$T_{uu}(k,t)$	Inner loop (a_j in Alg. 2)	multiplicand
(UV, VV)	$T_{uv}(k,t)$	$T_{vv}(k,t)$	Outer loop (b_i in Alg. 2)	multiplier
$(\overline{UV}, UU)$	$T_{\overline{uv}}(k,t)$	$T_{uu}(k,t)$	Inner loop (a_j in Alg. 2)	multiplier
$(\overline{UV}, VV)$	$T_{\overline{uv}}(k,t)$	$T_{vv}(k,t)$	Outer loop (b_i in Alg. 2)	multiplicand
(UV, WW)	$T_{uv}(k,t)$	$T_{ww}(k,t)$	—	—
$(\overline{UV}, WW)$	$T_{\overline{uv}}(k,t)$	$T_{ww}(k,t)$	—	—

(u_k, v_k, w_k) and the operands in the integer-multiplier and LIM levels. The measured side-channel traces for the k-th triplet are denoted by $T_{uv}(k,t)$, $T_{\overline{uv}}(k,t)$, $T_{uu}(k,t)$, $T_{vv}(k,t)$, and $T_{ww}(k,t)$ where t is time index.

The traces are examined in pair. Six pairs namely

$$\{(UV), (\overline{UV})\} \times \{(UU), (VV), (WW)\}.$$

are evaluated as summarized in Table 2. The pair corresponds to consecutive multiplication and squaring in the multiply-always method: (UV), $(\overline{UV})$ correspond to multiplication while (UU), (VV), and (WW) are squaring. The pairs are referred by identifiers namely (UV, UU), (UV, VV), $(\overline{UV}, UU)$, $(\overline{UV}, VV)$, (UV, WW), and $(\overline{UV}, WW)$.

Table 2 also summarizes colliding operand both at the integer-multiplier and LIM levels. In (UV, UU) and $(\overline{UV}, UU)$, the operand scanned at the inner loop of Alg. 2 has collision at the LIM-level. On the other hand, the operand scanned at the outer loop has collision in (UV, VV) and $(\overline{UV}, VV)$. At the integer-multiplier level, multiplicand collides in (UV, UU) and $(\overline{UV}, VV)$. Meanwhile multiplier collides in (UV, VV) and $(\overline{UV}, UU)$. There is no collision in (UV, WW) and $(\overline{UV}, WW)$.

The pairs are compared under the multiple- and single-shot attacks in the following sections.

4.2 Application of the Multiple-Shot Attack by Witteman et al.

As a preliminary experiment, the pairs of the traces are analyzed using the attack by Witteman et al. For the pairs (UV, UU), (UV, VV), $(\overline{UV}, UU)$, and $(\overline{UV}, VV)$, correlation matrices are obtained as follows:

$$C_{x,y}^{(UV,UU)} = \mathcal{F}_k[T_{uv}(k,x), T_{uu}(k,y)], \tag{11}$$

$$C_{x,y}^{(UV,VV)} = \mathcal{F}_k[T_{uv}(k,x), T_{vv}(k,y)], \tag{12}$$

$$C_{x,y}^{(\overline{UV},UU)} = \mathcal{F}_k[T_{\overline{uv}}(k,x), T_{uu}(k,y)], \tag{13}$$

$$C_{x,y}^{(\overline{UV},VV)} = \mathcal{F}_k[T_{\overline{uv}}(k,x), T_{vv}(k,y)], \tag{14}$$

where $\mathcal{F}_k[\cdot,\cdot]$ is the correlation-coefficient operator in Eq. (4).

The matrices are shown as bitmap images in Fig. 6. The bitmap images show different patterns depending on the colliding operands at the LIM level. There are repeated slash lines on Fig. 6-(i) and -(iii) in which there are collisions at the inner loop. On the other hand, collision at the outer loop makes rectangle patterns as shown in Fig. 6-(ii).

The bitmap images also show the difference caused by the asymmetry at the integer-multiplier level. The multiplier (cf. the multiplicand) shows higher correlation as expected in Sect. 3.1. The slash lines are brighter in Fig 6-(iii) compared to the ones in Fig. 6-(i). Similarly, the rectangle patterns are more distinct in Fig. 6-(ii). Quantitative comparison is conducted in the next section.

In Sect. 2.4, the way how to find points of interest namely $l(j,i)$ is discussed. The discussion is revisited considering the experimental result. In Fig. 6, a bright pixel corresponds to integer multiplications with collision. Therefore, their time indices can be used as the points of interest. Figure 7 shows how the slash lines in Fig. 6 is sampled as points of interest (for the number of words $s = 4$).

4.3 Application of the Proposed Single-Shot Attack

The proposed attack described in Sect. 2.4 is applied to the traces. That is applied to all the six pairs in Table 2. Considering colliding operands, Eq. (9) is used for (UV, UU) and $(\overline{UV}, UU)$ while Eq. (10) is used for (UV, VV) and $(\overline{UV}, VV)$.

Correlation coefficients are obtained by applying the attack to the pairs. Since there are $1,000$ traces, $1,000$ correlation coefficients are obtained for each pair. They are

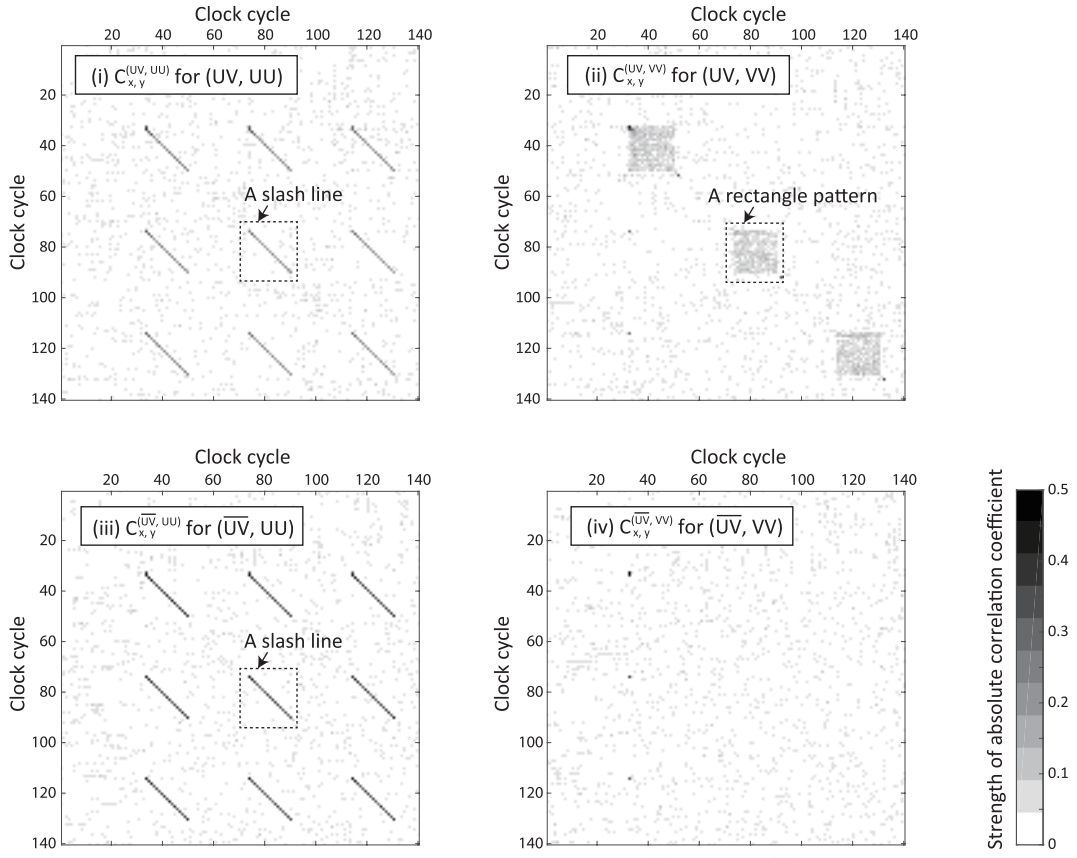


Fig. 6 Correlation-coefficient matrices $C^{(UV,UU)}_{x,y}$, $C^{(UV,VV)}_{x,y}$, $C^{(\overline{UV},UU)}_{x,y}$, and $C^{(\overline{UV},VV)}_{x,y}$ shown as bitmap images. (i)-(iv) correspond to different operand orders.

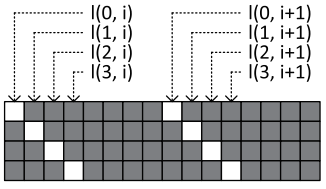


Fig. 7 Determining points of interest from correlation matrix.

shown as histograms in Fig. 8. Figure 8-(i) to -(iv) correspond to the pairs (UV, UU), (UV, VV), $(\overline{UV}, UU)$ and $(\overline{UV}, VV)$, respectively. The no-collision pairs (UV, WW) and $(\overline{UV}, WW)$ are also shown in each sub-figure for comparison. In each sub-figure, black and white bars represent correlation coefficients in cases with and without collision. The figure shows that correlation coefficients distribute separably in some pairs.

Distinguishability between distributions with and without collision is evaluated. The black and white bars in Fig. 8 are composed of $1,000$ samples each. A mix of them comprising $2,000$ samples is considered. The performance is evaluated by classifying the mix without using prior knowledge.

In this experiment, the following simple classification is used. Firstly, the $2,000$ samples are sorted. Then, $1,000$ samples with highest correlation coefficients are decided to have collision. The remaining $1,000$ samples are decided to have no collision. The above classification is made considering the following attack on RSA. Suppose the RSA uses t-bit exponent. By applying the proposed attack, t correlation coefficients are obtained. The exponent is expected to have $t/2$-bit zeros and $t/2$-bit ones. Therefore, the attacker divides the correlation coefficients into upper and lower halves.

As a result of the thresholding, the success rates are 98.3%, 93.0%, 99.5%, and 52.7% in Fig. 8-(i) to -(iv), respectively. More than 99% is achieved with $(\overline{UV}, UU)$. That is the first successful single-shot internal-collision attack of the multiply-always method on FPGA. In contrast, the attack is unsuccessful in $(\overline{UV}, VV)$. The result clearly shows that operand order has a significant impact on the success rate of the attack.

5. Discussion

The experimental results show that the operand order has significant impact on side-channel leakage. In this section, consequences of the results are discussed from three different perspectives. Firstly from circuit designer's perspective, leakage can be efficiently reduced by appropriately designing the order of operands. Secondly from attacker's perspective, the asymmetry can be exploited as a new type of infor-

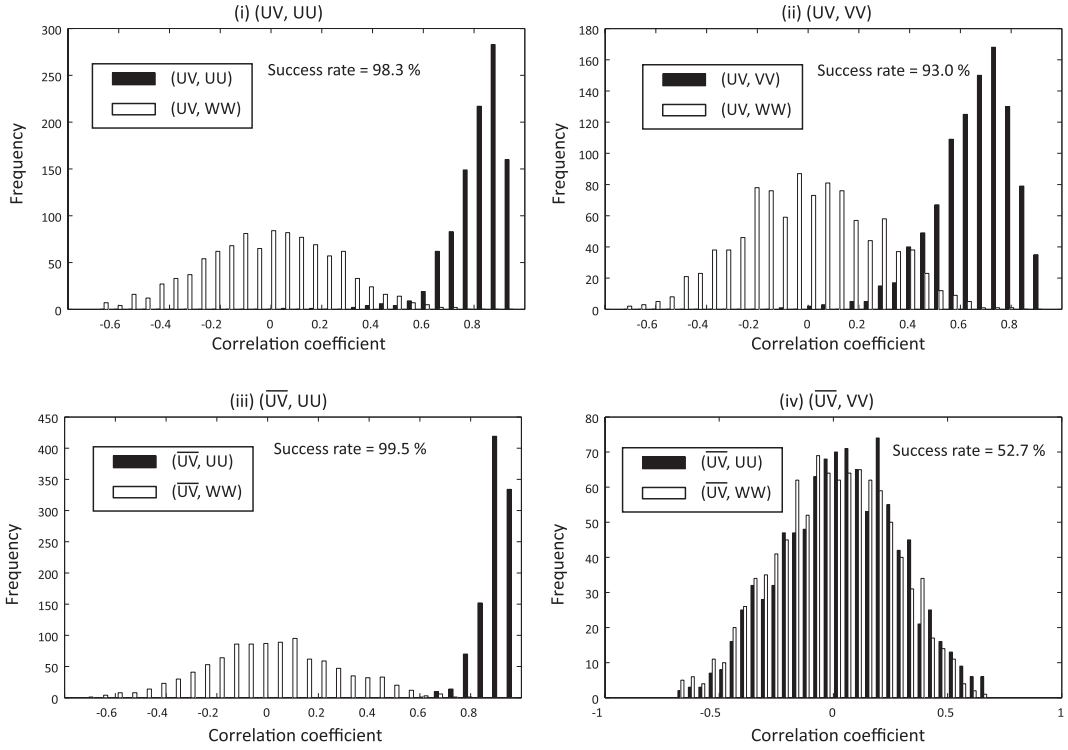


Fig. 8 Histograms of correlation coefficients for the single-shot attack. Sub-figures (i)-(iv) correspond to different operand orders.

mation leakage to attack a countermeasure. Finally from software engineer's perspective, it is shown that the asymmetry is measurable from microcontroller and thus it is also relevant to software implementation of cryptography.

5.1 Leakage Reduction by Designing Operand Order

The experimental results show that operand order can divide success and failure of the attack. That means that leakage can be reduced by appropriately designing the order of operands. In the previous experiment, The pair $(\overline{UV}, VV)$ is the best option in terms of leakage suppression (see Fig. 8-(iv)). The operand order can be changed at almost no cost. In addition to the cost effectiveness, the proposed method can easily be combined with other conventional countermeasures (e.g., the randomized operand scanning [10], [11]).

Although a specific design using the Montgomery multiplication with CIOS is discussed in this paper, the same idea can be easily extended to many other methods. That is because the causes of asymmetry, the partial-product generation and operand scanning, are common in long-integer multiplication. On the other hand, an exceptional case is worth noting. The Finely Integrated Operand Scanning (FIOS) [19] falls in such a case. In FIOS, the register containing the variable scanned at the outer cannot be kept while the outer loop. That is because another word-wise multiplication, needed for the Montgomery reduction, should be interleaved. As a result, the leakage from the operand scanned at outer loop is not necessarily smaller.

It is also worth mentioning that toggle simulation can be used to determine a good operand order. As shown in Sect. 3.1, the asymmetry at the integer-multiplier level can be simulated. The LIM-level asymmetry is not simulated in the paper, however, the frequency of register update can be covered by the toggle simulation.

5.2 Attack on Montgomery Powering Ladder

In contrast to the previous result, the asymmetric leakage can make some countermeasures ineffective. Algorithm 3 shows the Montgomery powering ladder (MPL) [23]. We focus on collisions between inputs[†].

In MPL, consecutive operations always collides as follow:

$$R_{\bar{a}} \leftarrow R_0 \times R_1 \mod N, \tag{15}$$

$$R_a \leftarrow R_a \times R_a \mod N. \tag{16}$$

Therefore, the presence of collision does not leak k_j. However, the colliding operand depends on k_j. There is collision at the first and second operand when $k_j = 0$ and $k_j = 1$, respectively. Therefore, attacker can recover a secret bit by distinguishing the collisions in different operands.

Interestingly, the attack is no longer effective when Eq. (15) and Eq. (16) are replaced with the following statements found in [9].

[†]Hanley et al. considered more general cases considering a collision between input and output. However, the input-output collision was very weak in our setup.

Algorithm 3 Montgomery Powering Ladder [23]

Input: Message M, scalar $k = (k_{t-1}, \cdots, k_0)_2$
Output: Ciphertext M^d

1: $R_0 \leftarrow 1; R_1 \leftarrow M$
2: **for** $j = t - 1$ downto 0 **do**
3: $\quad a \leftarrow k_j; \overline{a} \leftarrow 1 - a$
4: $\quad R_{\overline{a}} \leftarrow R_0 \times R_1 \mod N$
5: $\quad R_a \leftarrow R_a \times R_a \mod N$
6: **end for**
7: Return R_0

```
00. nop
01. nop
02. muls r0, r1
03. nop
04. nop
```

Fig. 9 Assembly code loaded on ARM Cortex-M0 core.

Table 3 Test vector to `muls` instruction.

Trace name	Operands to `muls`
$T_{\alpha\beta}(k,t)$	$\alpha_k \times \beta_k$
$T_{\alpha\alpha}(k,t)$	$\alpha_k \times \alpha_k$
$T_{\beta\beta}(k,t)$	$\beta_k \times \beta_k$
$T_{\gamma\gamma}(k,t)$	$\gamma_k \times \gamma_k$

$$R_{\overline{a}} \leftarrow R_a \times R_{\overline{a}} \mod N, \tag{17}$$
$$R_a \leftarrow R_a \times R_a \mod N. \tag{18}$$

Now, collision unconditionally occurs at the first operand. Therefore, collision becomes independent of k_j. This is another example showing the importance of designing operand order.

5.3 Asymmetric Leakage in Software Implementation

So far, asymmetric leakage is discussed for hardware implementation. It would be natural to ask if it is relevant to software implementation. Since the Booth recoding is commonly used in commercial logic synthesizers [24], integer multiplier on processor likely has asymmetric leakage. That is investigated with experiment.

NXP LPC1114 [25] with ARM Cortex-M0 [26] core is chosen as a target chip. That chip is selected because ARM Cortex-M series is frequently used as an evaluation platform for software implementation of cryptography [27].

Figure 9 shows a small program used for the experiment. The registers `r0` and `r1` are initialized with input values to the multiplier before entering to the program. Then, the multiplication instruction `muls` is called with `r0` and `r1`. The target instruction is surrounded by `nop` instructions in order to reduce perturbation by neighboring instructions processed simultaneously in a pipeline.

While executing the program in Fig. 9, side-channel trace is measured. The LPC1114 chip is de-packaged in the same manner as the FPGA shown in Fig. 5. Then, the chip is measured by putting the magnetic-field probe on the die surface.

Input to the instruction is designed similarly to Sect. 4.3. 16-bit random integers α_k, β_k, and γ_k are generated. For each triplet $(\alpha_k, \beta_k, \gamma_k)$, four multiplications are executed as summarized in Table 3. The four cases are: $\alpha_k \times \beta_k$, $\alpha_k \times \alpha_k$, $\beta_k \times \beta_k$, and $\gamma_k \times \gamma_k$. The measured traces for the k-th triplet are denoted by $T_{\alpha\beta}(k,t)$, $T_{\alpha\alpha}(k,t)$, $T_{\beta\beta}(k,t)$, and $T_{\gamma\gamma}(k,t)$ wherein t is time index. Note that α_k, β_k, and γ_k are limited to 16 bits in order to avoid overflow in the 32-bit multiplier.

The traces are analyzed using the approach by Witteman et al. Correlation-coefficient matrices are evaluated for the following three combinations:

$$C_{x,y}^{(\alpha\beta,\alpha\alpha)} = \mathcal{F}_k[T_{\alpha\beta}(k,x), T_{\alpha\alpha}(k,y)], \tag{19}$$
$$C_{x,y}^{(\alpha\beta,\beta\beta)} = \mathcal{F}_k[T_{\alpha\beta}(k,x), T_{\beta\beta}(k,y)], \tag{20}$$
$$C_{x,y}^{(\alpha\beta,\gamma\gamma)} = \mathcal{F}_k[T_{\alpha\beta}(k,x), T_{\gamma\gamma}(k,y)]. \tag{21}$$

The notation follows Eq. (4). $C_{x,y}^{(\alpha\beta,\alpha\alpha)}$ and $C_{x,y}^{(\alpha\beta,\beta\beta)}$ have collision in different operand. Meanwhile $C_{x,y}^{(\alpha\beta,\gamma\gamma)}$ has no collision.

Bitmap images of matrices $C_{x,y}^{(\alpha\beta,\alpha\alpha)}$, $C_{x,y}^{(\alpha\beta,\beta\beta)}$, and $C_{x,y}^{(\alpha\beta,\gamma\gamma)}$ are shown in Fig. 10 (1), (2), and (3), respectively. The figures involve a single clock cycle in which `muls` is executed.

The time when the correlation coefficients are maximized is indicated by dashed lines. For clarity, cross-section of the matrices at the dash lines are shown in Fig. 10 (1'), (2'), and (3'). There are spikes in $C_{x,y}^{(\alpha\beta,\alpha\alpha)}$ and $C_{x,y}^{(\alpha\beta,\beta\beta)}$ as expected. There is difference in the strengths of the correlation coefficients. The peak of correlation coefficient in $C_{x,y}^{(\alpha\beta,\alpha\alpha)}$ is around 0.2 while that is about 0.8 in $C_{x,y}^{(\alpha\beta,\beta\beta)}$. The result clearly shows that the second operand (argument) has stronger correlation compared to the first operand. The result shows that asymmetric leakage is observable in microcontroller.

6. Conclusion

Side-channel leakage from multiplier is asymmetric in terms of its two operands. The asymmetry can be reasoned and predicted when arithmetic-circuit and micro-architecture levels are considered. Therefore, designing operand order can be a cost-effective countermeasure. On the other hand, some countermeasure can be defeated if the leakages from first and second operands are distinguishable.

There are a lot of interesting problems yet to be solved. Notably, the attack using input-to-output collision is an interesting challenge. Another important open problem is on incomplete exponent recovery. The successful rate more than 99% is clearly dangerous. The ideal goal is 50.0%, however, it could possibly be relaxed.

Acknowledgement

The authors would like to thank the anonymous reviewers at COSADE 2015 for their valuable comments. The study was

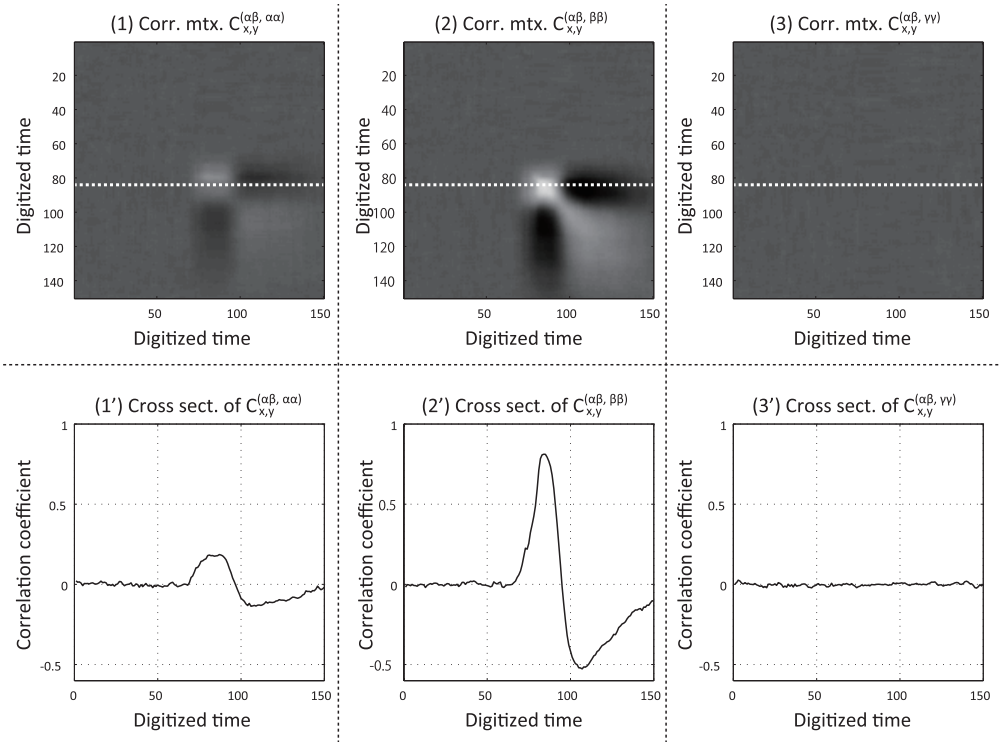


Fig. 10 Correlation-coefficient matrices $C_{x,y}^{(\alpha\beta,\alpha\alpha)}$, $C_{x,y}^{(\alpha\beta,\beta\beta)}$, and $C_{x,y}^{(\alpha\beta,\gamma\gamma)}$ in sub-figures (1)–(3). Their cross sections at dashed lines are in (1')–(3').

conducted as a part of the CREST Dependable VLSI Systems Project funded by the Japan Science and Technology Agency.

References

[1] T. Sugawara, D. Suzuki, and M. Saeki, "Two operands of multipliers in side-channel attack," Constructive Side-Channel Analysis and Secure Design, Lecture Notes in Computer Science, vol.9064, pp.64–78, 2015.

[2] S. Mangard, E. Oswald, and T. Popp, Power Analysis Attacks: Revealing the Secrets of Smart Cards, Springer-Verlag, 2007.

[3] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," Advances in Cryptology, CRYPTO'99, Lecture Notes in Computer Science, vol.1666, pp.388–397, 1999.

[4] F. Amiel, B. Feix, M. Tunstall, C. Whelan, and W.P. Marnane, "Distinguishing multiplications from squaring operations," Selected Areas in Cryptography, Lecture Notes in Computer Science, vol.5381, pp.346–360, 2009.

[5] N. Homma, A. Miyamoto, T. Aoki, A. Satoh, and A. Shamir, "Collision-based power analysis of modular exponentiation using chosen-message pairs," CHES 2008, LNCS, vol.5154, pp.100–112, 2008.

[6] J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," Cryptographic Hardware and Embedded Systems, Lecture Notes in Computer Science, vol.1717, pp.292–302, 1999.

[7] J. Heyszl, A. Ibing, S. Mangard, F. De Santis, and G. Sigl, "Clustering algorithms for non-profiled single-execution attacks on exponentiations," Smart Card Research and Advanced Applications, Lecture Notes in Computer Science, pp.79–93, 2014.

[8] G. Perin, L. Imbert, L. Torres, and P. Maurine, "Attacking randomized exponentiations using unsupervised learning," Constructive Side-Channel Analysis and Secure Design, Lecture Notes in Computer Science, vol.8622, pp.144–160, 2014.

[9] N. Hanley, H. Kim, and M. Tunstall, "Exploiting collisions in addition chain-based exponentiation algorithms using a single trace," Cryptography ePrint Archive: Report 2012/485, http://eprint.iacr.org/2012/485

[10] C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil, "Horizontal correlation analysis on exponentiation," Information and Communications Security, Lecture Notes in Computer Science, vol.6476, pp.46–61, 2010.

[11] A. Bauer, E. Jaulmes, E. Prouff, and J. Wild, "Horizontal and vertical side-channel attacks against secure rsa implementations," Topics in Cryptology, CT-RSA 2013, Lecture Notes in Computer Science, vol.7779, pp.1–17, 2013.

[12] J.-C. Bajard, L. Imbert, P.-Y. Liardet, and Y. Teglia, "Leak resistant arithmetic," Cryptographic Hardware and Embedded Systems, CHES 2004, Lecture Notes in Computer Science, vol.3156, pp.62–75, 2004.

[13] C.D. Walter, "Sliding windows succumbs to big mac attack," Cryptographic Hardware and Embedded Systems, CHES 2001, Lecture Notes in Computer Science, vol.2162, pp.286–299, 2001.

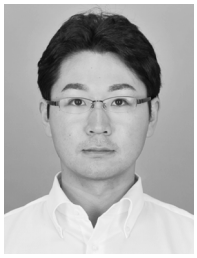
[14] C.D. Walter and D. Samyde, "Data dependent power use in multipliers," 17th IEEE Symposium on Computer Arithmetic (ARITH'05), pp.4–12, 2005.

[15] M.F. Witteman, J.G.J. van Woudenberg, and F. Menarini, "Defeating RSA multiply-always and message blinding countermeasures," Topics in Cryptology, CT-RSA 2011, Lecture Notes in Computer Science, vol.6558, pp.77–88, 2011.

[16] C. Clavier, B. Feix, G. Gagnerot, C. Giraud, M. Roussellet, and V. Verneuil, "ROSETTA for single trace analysis: Recovery of secret exponent by triangular trace analysis," Progress in Cryptology, INDOCRYPT 2012, Lecture Notes in Computer Science, vol.7668, pp.140–155, 2012.

[17] I. Koren, Computer Arithmetic Algorithms 2nd Ed., A K Peters/CRC Press, 2001.

[18] P.L. Montgomery, "Modular multiplication without trial division," Math. Comput., vol.44, no.170, pp.519–519, 1985.
[19] C.K. Koç, T. Acar, and B.S. Kaliski, Jr., "Analyzing and comparing Montgomery multiplication algorithms," IEEE Micro, vol.16, no.3, pp.26–33, 1996.
[20] A. Miyamoto, N. Homma, T. Aoki, and A. Satoh, "Systematic design of RSA processors based on high-radix montgomery multipliers," IEEE Trans. VLSI Syst., vol.19, no.7, pp.1136–1146, 2011.
[21] AIST, "Side-channel attack standard evaluation board," http://www.risec.aist.go.jp/project/sasebo/
[22] T. Sugawara, D. Suzuki, M. Saeki, M. Shiozaki, and T. Fujino, "On measurable side-channel leaks inside ASIC design primitives," Cryptographic Hardware and Embedded Systems, CHES 2013, Lecture Notes in Computer Science, vol.8086, pp.159–178, 2013.
[23] M. Joye and S.-M. Yen, "The Montgomery powering ladder," Cryptographic Hardware and Embedded Systems, CHES 2002, Lecture Notes in Computer Science, vol.2523, pp.291–302, 2003.
[24] Synopsys DesignWare Technical Bulletin Article, "2007.03 designware library datapath and building block IP — DesignWare library introduces 19 new building block IPs in the 2007.03 release," http://www.synopsys.com/dw/dwtb.php?a=dwbb_0701
[25] NXP Semiconductors, "LPC1110/11/12/13/14/15 Product Data Sheet," http://www.nxp.com/documents/data_sheet/LPC111X.pdf
[26] ARM, "Cortex-M0 Technical Reference Manual Revision r0p0," http://infocenter.arm.com/help/topic/com.arm.doc.ddi0432c/DDI0432C_cortex_m0_r0p0_trm.pdf
[27] R. de Clercq, L. Uhsadel, A. Van Herrewege, and I. Verbauwhede, "Ultra low-power implementation of ECC on the ARM cortex-M0+," Proc. The 51st Annual Design Automation Conference on Design Automation Conference — DAC'14, pp.1–6, 2014.



Takeshi Sugawara received the B.E., M.Is., and Ph.D. degrees from Tohoku University, Japan, in 2006, 2008 and 2011, respectively. He is currently a researcher at Mitsubishi Electric Corporation since 2011 and is involved in the research and development on information security. His research interests involve high-performance implementation and side-channel security of cryptographic hardware.

Daisuke Suzuki received the B.E. and M.E. degrees from Tokyo University of Science, Japan, in 1999 and 2001. In 2001, he joined Mitsubishi Electric Corporation. He received the Ph.D. degree from Yokohama National University, Japan, in 2011. His research interests include implementation of cryptosystems. He was awarded the SCIS 2005 paper prize.

Minoru Saeki received the B.E. degree from the University of Tokyo in 1988. In 1988, he joined Mitsubishi Electric Corporation. He is currently with Information-technology Promotion Agency, Japan since 2015. His research interests include computer architecture, anti-tamper design, and security architecture.