CHOSEN-MESSAGE SPA ATTACKS AGAINST FPGA-BASED RSA HARDWARE IMPLEMENTATIONS

Atsushi Miyamoto, Naofumi Homma and Takafumi Aoki

Graduate School of Information Sciences,
Tohoku University, Japan
{miyamoto, homma}@aoki.ecei.tohoku.ac.jp

Akashi Satoh

Research Center for
Information Security,
AIST, Japan
akashi.satoh@aist.go.jp

ABSTRACT

This paper presents SPA (Simple Power Analysis) attacks against public-key cryptosystems implemented on an FPGA platform. The SPA attack investigates a power waveform generated by a cryptographic module, and reveals a secret key in the module. We focus on chosen-message SPA attacks, which enhances the differences of operating waveforms between multiplication and squaring correlated to the secret key by using the input of particular messages. In particular, Yen showed a unique SPA attack against RSA cryptosystem, but no verification experiment using actual software or hardware was performed. In this paper, we implemented four-types of RSA processors on an FPGA platform in combination with two variants of the Montgomery multiplication algorithm and two different types of multipliers for SPA attacks experiments. Then we demonstrated effectiveness of various chosen-message attacks as well as Yen's method, and investigated the characteristics of the attacks depending on the hardware architectures.

1. INTRODUCTION

Physical attacks on cryptographic modules using side channel information are attracting extensive attention. In order to reveal the secret parameters the power dissipation, electromagnetic radiation, or operating times are measured and correlated to internal operations. Simple Power Analysis (SPA) and Differential Power Analysis (DPA) proposed by Kocher et al. [1] are known as basic and powerful side-channel attacks, and many papers have been published about them. SPA and DPA attacks against RSA were first investigated by Kocher and Messerges [2], respectively.

The original idea of SPA introduced by Kocher is to reveal differences between multiplication and squaring operations performed during modular exponentiation according to the bit pattern of the secret key. Such differences, however, are not always observable for some implementations. In order to make secret information leak via the operating waveforms, chosen message attacks for RSA that use specific data depending on the target cryptographic module were proposed [3, 4, 5, 6]. The timing attack against RSA with CRT [3] measures the operating times caused by extra reductions in the Montgomery multiplication [7] that depend on the input data. The SPA with adaptively chosen messages [4] can be applied to an RSA implementation using CRT based on Garner's algorithm, where an extra modular reduction at the end of a CRT is repeatedly searched for changes due to the input message. The DPA using the Hamming weight of an intermediate value [5] was also applied to RSA with CRT. These attacks focused on the RSA implementations using specific algorithms, and thus information about the implementations is critical to reveal the secret keys. The first two methods can be defeated by inserting dummy reductions, and the DPA of [5] cannot be used with implementations using the Montgomery algorithm.

In [6], Yen proposed an SPA attack using chosen messages to break public-key cryptosystems based on modular exponentiation, including those using Montgomery multiplication and/or CRT algorithms. He also described a capability of breaking the most popular SPA countermeasures using dummy multiplication [8] by using the particular input data of -1. However, no experiment on actual software or hardware implementation was demonstrated.

In order to bear out effectiveness of the chosen-message SPA attack, we designed four types of experimental RSA processors on an FPGA platform combining two typical algorithms (CIOS and FIOS) [9] with two kinds of multipliers (an embedded multiplier in the FPGA and our custom design). Then we analyzed the characteristic of the attack depending on the hardware architectures and input message pattern by monitoring a number of power waveforms.

2. MODULAR EXPONENTIATION ALGORITHM

Modular exponentiation is one of the most important arithmetic operations for public-key cryptography such as the RSA scheme, the ElGamal encryption scheme, and for the Diffie-Hellman key agreement. The RSA cryptosystem em-

ALGORITHM 1
MODULAR EXPONENTIATION (L-TO-R BINARY METHOD)

Input:	$X, N,$ $E = (e_{k-1}, ..., e_1, e_0)_2$
Output:	$Z = X^E$ mod N

```
1 :  $Z := 1$;
2 :  for $i = k - 1$ downto 0
3 :     $Z := Z * Z$ mod $N$;          – squaring
4 :     if ($e_i = 1$) then
5 :        $Z := Z * X$ mod $N$;       – multiplication
6 :     end if
7 :  end for
```

ALGORITHM 2
HIGH-RADIX MONTGOMERY MULTIPLICATION (*MontMult*)

Input:	$X = (x_{m-1}, ..., x_1, x_0)_{2^r}$, $Y = (y_{m-1}, ..., y_1, y_0)_{2^r}$, $N = (n_{m-1}, ..., n_1, n_0)_{2^r}$, $W = -N^{-1}$ mod 2^r
Output:	$Z = XY2^{-r \cdot m}$ mod N

```
1 :  $Z := 0$;
2 :  for $i = 0$ to $m - 1$
3 :     $C := 0$;
4 :     $t_i := (z_0 + x_i y_0)W$ mod $2^r$;
5 :     for $j = 0$ to $m - 1$
6 :        $Q := z_j + x_i y_j + t_i n_j + C$;
7 :        if ($j \neq 0$) then $z_{j-1} := Q$ mod $2^r$;
8 :        $C := Q/2^r$;
9 :     end for
10:     $z_{m-1} := C$;
11:  end for
12:  if ($Z > N$) then $Z := Z - N$;
```

ploys modular exponentiation for encryption and decryption as follows:

$$C = P^E \bmod N, \tag{1}$$

$$P = C^D \bmod N, \tag{2}$$

where P is the plaintext, C is the ciphertext, E and N are the public keys, and D is the secret key. P, C, N, and D are at least 1,024 bits in length for security reasons.

The binary method which is most commonly used for the modular exponentiation, does multiplication and squaring sequentially according to the bit pattern of the exponent E or D. There are two variations of the algorithm. The left-to-right binary method starts at the exponent's MSB and works downward. The right-to-left binary method, on the other hand, starts at the exponent's LSB and works upward. **ALGORITHM 1** shows a left-to-right binary method for scanning the bits of the exponent from MSB to LSB, where k indicates the bit length of the secret keys. This algorithm always performs a squaring at Line 3 regardless of the scanned bit value, but the multiply operation at Line 5 is only executed if the scanned bit is 1.

One popular method to speed up the exponentiation is to use Montgomery's modular multiplication algorithm (*MontMult*) [7]. For integers X and Y , where $0 \leq X, Y < N < 2^k = R$, define *MontMult* to be $XYR^{-1} \bmod N$. **ALGORITHM 2** shows a high-radix Montgomery multiplication

ALGORITHM 3
MODULAR EXPONENTIATION WITH *MontMult*

Input:	X, N $E = (e_{k-1}, ..., e_1, e_0)_2$,
Output:	$Z = X^E$ mod N

```
1 :  $W := -N^{-1}$ mod $R$;
2 :  $Y := XR$ mod $N$;
3 :  $Z := R$ mod $N$;
4 :  for $i = k - 1$ downto 0
5 :     $Z :=$ MontMult $(Z, Z, N, W)$;        – squaring
6 :     if ($e_i = 1$) then
7 :        $Z :=$ MontMult $(Z, Y, N, W)$;     – multiplication
8 :     end if
9 :  end for
10:  $Z :=$ MontMult $(Z, 1, N, W)$;
```

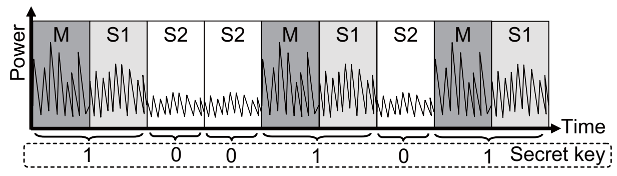


Fig. 1. Chosen-message SPA against RSA.

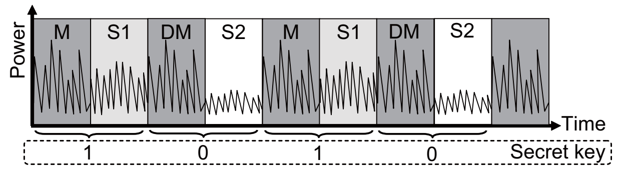


Fig. 2. Chosen-message SPA against RSA with dummy multiplication.

algorithm [9] for the fast and compact implementation [10]. The k-bit operands are divided into m r-bit blocks and processed in a nested loop. **ALGORITHM 3** shows a modular exponentiation algorithm combining **ALGORITHM 1** and **ALGORITHM 2**.

3. SPA USING SPECIFIC INPUT DATA

The basic idea of Yen's SPA [6] is to enhance the differences between the multiplication and squaring operations to be observed by using a chosen message that is -1. Then we can maintain the multiplication and squaring results for the modular exponentiation as constants. For example, let $-1 \bmod N (= N - 1)$ be the input X in **ALGORITHM 1**. Then the outputs of the multiplication and squaring operations during exponentiation are $-1 \bmod N$ and $1 \bmod N$, respectively. For brevity, we write $-1 \bmod N$ and $1 \bmod N$ as -1 and 1 in the following.

According to the above example, the multiplication and squaring operations during exponentiation can be classified into three types: (M) multiplication after squaring, (S1) squaring after multiplication, and (S2) squaring after squaring. These operations M, S1 and S2 in **ALGORITHM 1** are

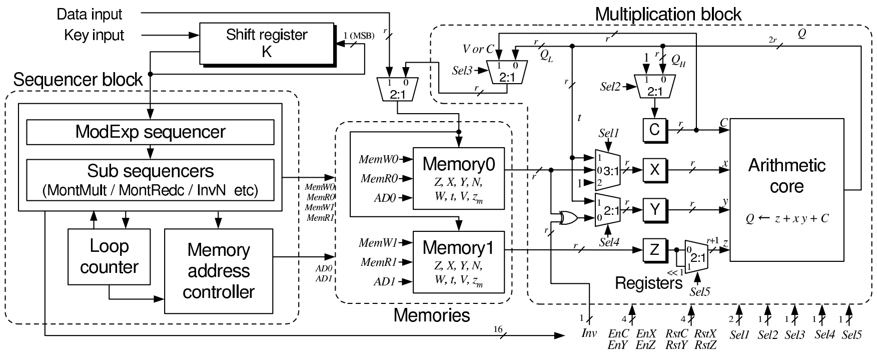


Fig. 3. RSA processor architecture.

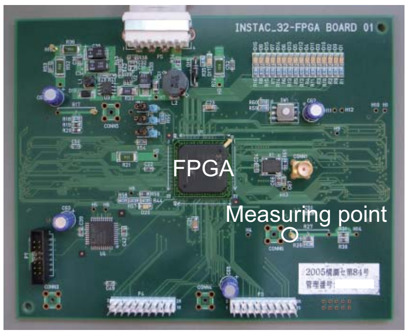


Fig. 4. Evaluation board (INSTAC-32).

given as

$$Z = 1 * (-1) \bmod N = -1 \bmod N, \quad (3)$$
$$Z = (-1) * (-1) \bmod N = 1 \bmod N, \quad (4)$$
$$Z = 1 * 1 \bmod N = 1 \bmod N, \quad (5)$$

respectively. When the exponentiation employs the Montgomery multiplication algorithm in **ALGORITHM 3**, the three operations are given as

$$\begin{aligned} Z &= R * (-R) * R^{-1} \bmod N \\ &= -R \bmod N, \quad (6) \\ Z &= (-R) * (-R) * R^{-1} \bmod N \\ &= R \bmod N, \quad (7) \\ Z &= R * R * R^{-1} \bmod N \\ &= R \bmod N, \quad (8) \end{aligned}$$

respectively. Note that the multiplication and squaring operations in the right-to-left binary method are also classified by the -1 value input.

Fig. 1 illustrates an image of the above SPA against an RSA using the left-to-right binary method for the secret key exponent "100101." As described above, the squaring S1 follows the multiplication M, and the squaring S2 follows S2 or S1. In other words, M is never followed by S2. Thus, the bit pattern of the secret exponent can be obtained if one of the three operations is distinct from the others.

By confining the operation sequences as well as the data values Yen's method can also defeat the well-known SPA countermeasure "square-and-multiply-always" algorithms [8]. The countermeasure inserts dummy multiplications for the zero bits of the exponent so as to perform squaring and multiplication for each bit. As shown in Fig. 2, the dummy multiplications should be inserted before the S2 states to hide the key bit pattern. The dummy multiplication outputs the usual value of -1, but it is discarded, and the value of 1 is used in the following squaring that is the S2 operation, Therefore, a dummy multiplication DM followed by the S2 can easily be distinguished from the true multiplication M followed by the S1.

Table 1. Synthesis report

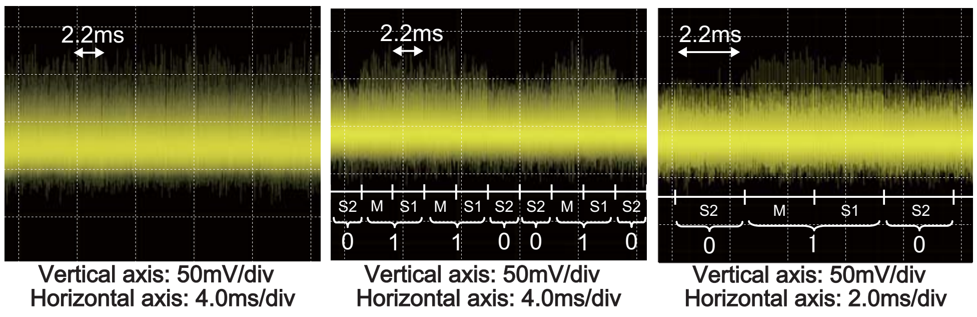
	Type-I		Type-II	
	CIOS	FIOS	CIOS	FIOS
# of slices	3,028	3,098	4,079	4,111
# of FFs	3,215	3,252	3,369	3,333
# of LUTs	4,891	5,031	6,873	6,899
# of MULTs	4	4	0	0
Delay (ns)	15.57	15.83	34.69	43.33

4. EXPERIMENTS

4.1. RSA processor setup

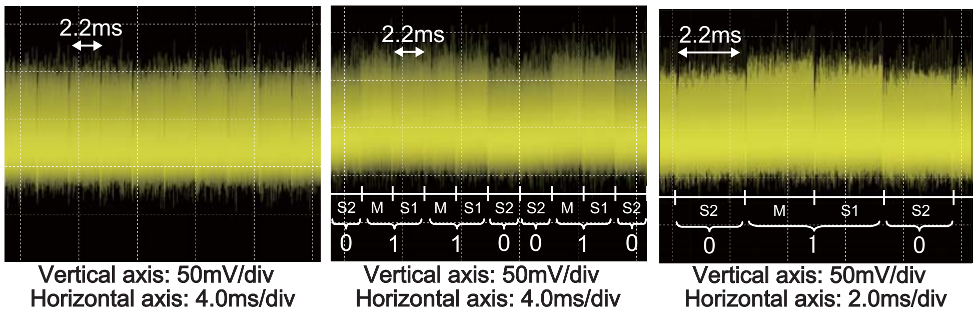
We designed RSA processors using **ALGORITHM 3** with the high-radix Montgomery multiplication based on **ALGORITHM 2** to demonstrate the effectiveness of chosen-message SPA. One multiplier was used in our processor, and thus the arithmetic operation $Q := z_j + x_i y_j + t_i n_j + C$ in **ALGORITHM 2** is divided into two steps: a multiplication step ($Q := z_j + x_i y_j + C$) and a reduction step ($Q := z_j + t_i n_j + C$). Reference [9] classified the high-radix Montgomery multiplication algorithms using one multiplier into five algorithms, and the Coarsely Integrated Operand Scanning (CIOS) and the Finely Integrated Operand Scanning (FIOS) methods are typical approaches. The CIOS algorithm executes the multiplication and reduction steps separately by modifying **ALGORITHM 2** using two j-loops, and the FIOS algorithm alternates between the steps in each word for j. The characteristics of the power waveforms are strongly dependent on the sequence of multiplication steps and reduction steps, and hence, we used the CIOS and FIOS methods for this experiment.

Fig. 3 shows a block diagram of our RSA processor using the CIOS algorithm, where the data bus width is r bits to fit the word size of the multiplier. The "Multiplication block" repeats the multiply-additions in accord with the bit pattern output from the shift register K to perform modular exponentiation. The r-bit Arithmetic core in the Multiplication block performs multiply-additions ($Q := z_j + x_i y_j + C$ or $Q := z_j + t_i n_j + C$) In this experiment, the operand and word sizes are $k = 1,024$ and $r = 32$.



(a) (b) (c)

Fig. 5. Power waveforms of CIOS Type-I processor: (a) random value input, (b) -1 value input, (c) magnified view of (b).



(a) (b) (c)

Fig. 6. Power waveforms of FIOS Type-I processor: (a) random value input, (b) -1 value input, (c) magnified view of (b).

The $1,024$-bit RSA processors with CIOS and FIOS methods were implemented on a Xilinx FPGA Virtex-II (xc2v1000). Power consumption of the arithmetic core is greatly influenced by the multiplier architecture, and so we used two types of 32-bit multipliers for each method: an embedded multiplier in the FPGA (Type-I) and a custom array multiplier (Type-II). Table 1 shows the synthesis report of the four RSA processors using the Xilinx ISE 7.1.

4.2. SPAs against RSA processors

Fig. 4 shows the experimental FPGA board INSTAC-32, and the measurement point where a resistor (5Ohm) is inserted between the FPGA ground pin and the ground plane of the board. The power traces were monitored with an oscilloscope (Agilent MSO 6104A) as voltage drops caused by the resistor. The RSA operations were performed at a 2-MHz operating frequency, and the sampling rate of the oscilloscope was 80 MSa/s (million samples per second). The modulus N was arbitrary in this experiment.

Figs. 5 (a) and (b) show the power traces generated by the CIOS Type-I processor for input data with random values and -1, respectively. Fig. 5 (c) is a magnified view of Fig. 5 (b). In Fig. 5 (a), we can not see any relationship between the waveform patterns and the operations. In contrast, differences for the operations are clearly visible in Fig. 5 (b). The multiplication M generated the 2.2-ms waveforms with the highest voltage peak, and the waveforms for the squarings S1 and S2 were at the middle and the lowest ranges, respectively. As described in Section 3, we only need to distinguish any one of the operations from the others to reveal the secret key bits. Also, Fig. 6 shows the power traces of the FIOS Type-I processor corresponding to Fig. 5. In Fig. 6 (b), we can see the differences for the multiplication and squaring operations more clearly than in the CIOS processor. The main reason is that the FIOS algorithm always changes two operands for the multiplier and thus has larger differences in transistor switching. We confirmed here that Yen's method can enhance SPA against RSA circuits with two typical Montgomery multiplication algorithms.

Figs. 7 and 8 show the power traces of the Type-II processors corresponding to Figs. 5 and 6, respectively. The relative power consumption of the arithmetic core was very high, almost double compared to the Type-I processor. The result shows that we can still expose the differences of the waveform patterns using the chosen-message approach (Figs. 7 (b) and 8 (b)). This suggests that the chosen-message approach would be applied to other platforms (e.g. ASICs), in which the power consumed by the arithmetic core would dominate the power waveforms.

Finally, Fig. 9 shows the power trace for the FIOS Type-I processor with the most popular SPA countermeasure, which is the square-and-multiply-always method [8]. The dummy

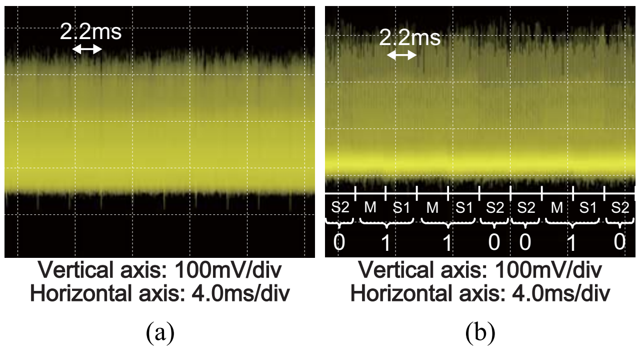


Fig. 7. Power waveforms of CIOS Type-II processor: (a) random value input, and (b) -1 value input.

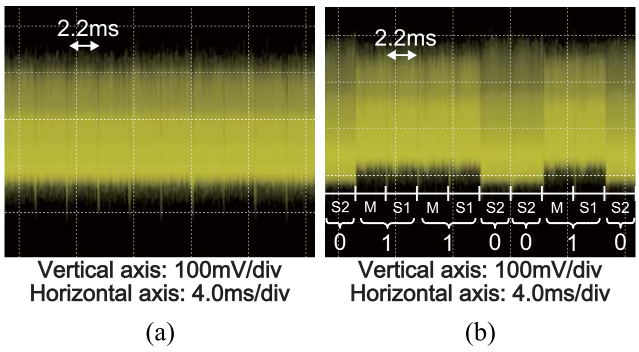


Fig. 8. Power waveforms of FIOS Type-II processor: (a) random value input, and (b) -1 value input.

multiplications DM are inserted to alternate between multiplication and squaring without regard to the secret key bits. However, we can distinguish between the true multiplications M and the dummy multiplications DM by checking if the following squaring is S1 (which means a true multiplication M) or S2 (which means a dummy multiplication DM).

As we examined using the four types of RSA implementations on the FPGA board, the chosen-message method resulted in large differences in the operating waveforms depending on the secret key bits, which are easily visible to the eye, and no special equipment is required. Not only the straightforward implementations, but even the countermeasure using dummy multiplications was defeated.

4.3. Discussion on other fixed-value input SPAs

In the following, we discuss the possibilities of other chosen-message input SPAs against RSA hardware implementations.

When an RSA processor is implemented by using the Montgomery multiplication algorithm like **ALGORITHM 2**, the input value of $R^{-1} = 2^{-k}$ also produces differences in the power waveforms as large as the input value of -1 discussed above. This is because the input value of R^{-1} is converted into the Montgomery domain with Equation (9) at Line 2 in **ALGORITHM 3**, prior to the Montgomery

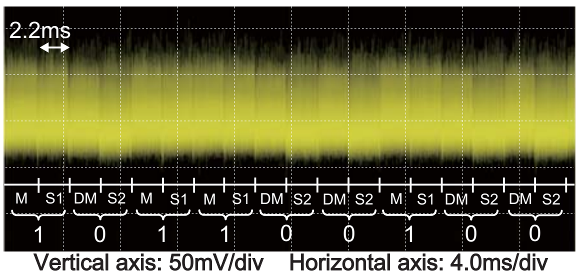


Fig. 9. Power waveform of FIOS Type-I processor using dummy multiplication countermeasure with -1 value input.

multiplication.

$$Y = R^{-1}R = 1 \bmod N \tag{9}$$

Therefore, $Y = 1$ is always multiplied in **ALGORITHM 2**, and the output of the modular multiplication is identical to the other input (not one). More precisely, $y_j = 0$ for $j = 1 \sim m-1$ at Line 6 in **ALGORITHM 2** ($Q := z_j + x_i y_j + t_i n_j + C$). The power consumed by the multiplier for the modular multiplication should be much lower than that of the modular squaring whose input is not one.

Fig. 10 shows the power traces generated by the CIOS and FIOS processors with the Type-I/II multipliers for the input of $R^{-1} = 2^{-1024}$. The lower peaks for the M in these pictures indicate that less power is consumed. According to the result from the Xilinx XPower simulator, the power consumed by the CIOS arithmetic core at Line 6 in **ALGORITHM 2** for the R^{-1} input was only one seventh in comparison with random input. In addition, the data of the three registers (X, Y, and C) in the CIOS core is almost unchanged during the multiplication. That also reduces the power consumption. As a result, we can find the secret key bits for all of the processors. We also examined the all-one value in the Montgomery domain for the four processors and obtained the secret key bits for the same reason, because the data in the registers and the inputs to the multiplier do not change.

The above observation suggests that there are many potential inputs that produce large differences in the power waveforms. For example, there are a lot of almost all-zero or all-one values in the Montgomery domain. Fig. 11 shows the power traces obtained from the four processors whose $1,024$-bit input contained an 800-bit stream of zeros with 224 random bits. We could determine the secret key bits for all of the processors. These results showed that there are at least 2^{224} variations of the input data that will expose the secret key bits on the $1,024$-bit processors.

5. CONCLUSIONS

This paper demonstrated and analyzed effectiveness of the chose-message SPA attacks for RSA implementations on an

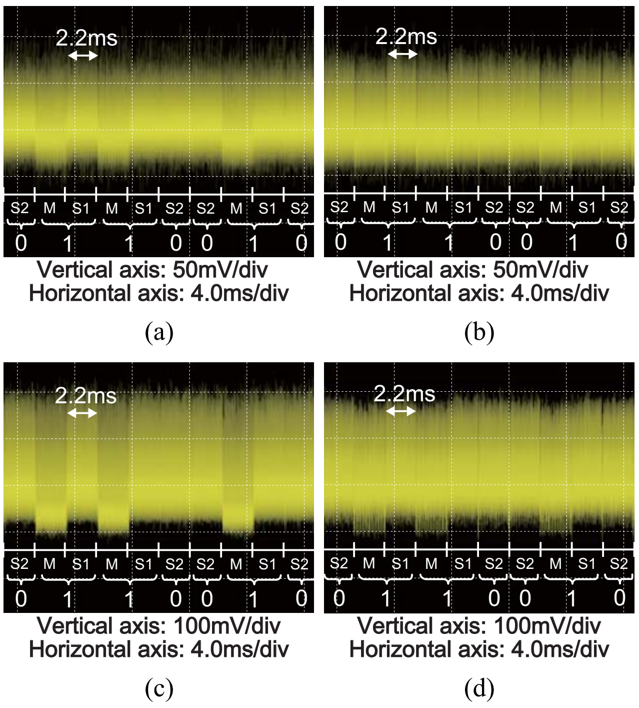


Fig. 10. Power waveforms of the four processors with R^{-1} value input: (a) CIOS Type-I, (b) FIOS Type-I, (c) CIOS Type-II, and (d) FIOS Type-II.

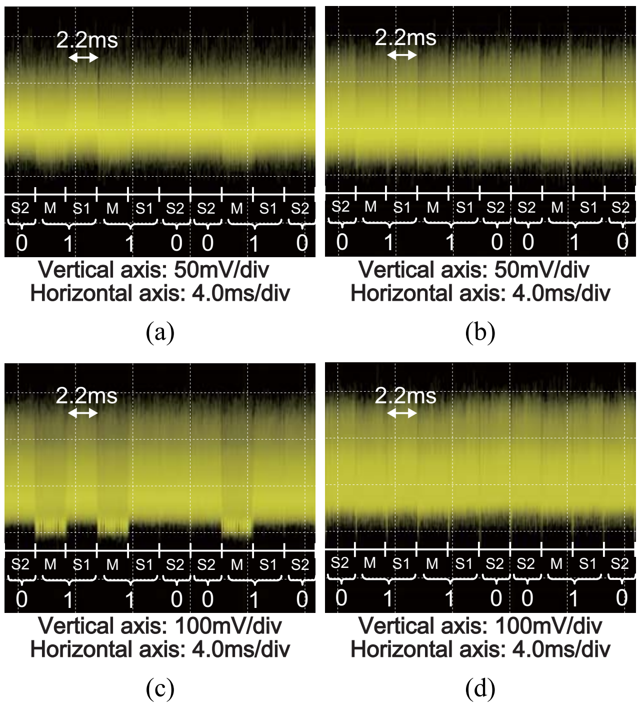


Fig. 11. Power waveforms of the four processors with almost-zero value input in the Montgomery domain: (a) CIOS Type-I, (b) FIOS Type-I, (c) CIOS Type-II, and (d) FIOS Type-II.

FPGA platform. We implemented four types of RSA processors (two algorithms with two multipliers) on the Xilinx FPGA. The experimental results clearly showed that the differences between multiplication and squaring operations for all of the implementations were easily visible to the eye, while the power traces using random inputs did not expose the secret information. In addition, the chosen-message SPA can also defeat the most popular SPA countermeasures using square-and-multiply-always algorithms.

We mainly analyzed the waveform characteristics for the input of -1 value, but there are many other values that cause large difference on the waveform patterns corresponding to the secret key, such as R^{-1} and values with a long zero bit stream, as discussed above. The effects of such values depend on the algorithm, architecture, and platform of the cryptographic modules, and thus we will continue to investigate the relationships between the data patterns and the characteristics of the implementations.

6. REFERENCES

[1] P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” *CRYPTO 1999, Lecture Notes in Computer Science*, vol. 1666, pp. 388 – 397, aug 1999.

[2] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, “Power analysis attacks of modular exponentiation in smartcards,” *CHES 1999, Lecture Notes in Computer Science*, vol. 1717, pp. 144–157, aug 1999.

[3] W. Schindler, “A timing attack against rsa with the chinese remainder theorem,” *CHES 2000, Lecture Notes in Computer Science*, vol. 1965, pp. 109 – 124, aug 2000.

[4] R. Novak, “SPA-based adaptive chosen-ciphertext attack on RSA implementation,” *PKC 2002, Lecture Notes in Computer Science*, vol. 2274, pp. 252–262, feb 2002.

[5] B. D. Boer, K. Lemke, and G. Wicke, “A DPA attack against the modular reduction within a crt implementation of RSA,” *CHES 2002, Lecture Notes in Computer Science*, vol. 2523, pp. 228–243, aug 2002.

[6] S. M. Yen, W. C. Lien, S. J. Moon, and J. C. Ha, “Power analysis by exploiting chosen message and internal collisions - vulnerability of checking mechanism for RSA-decryption.” *Mycrypt 2005, Lecture Notes in Computer Science*, vol. 3715, pp. 183–195, sep 2005.

[7] P. L. Montgomery, “Modular multiplication without trial division,” *Math. Comp.*, vol. 44, no. 170, pp. 519–521, 1985.

[8] J. S. Coron, “Resistance against differential power analysis for elliptic curve cryptosystems,” *CHES 1999, Lecture Notes in Computer Science*, vol. 1717, pp. 192–302, aug 1999.

[9] T. Koc, C. K.and Acar and B. S. Kaliski, “Analyzing and comparing montgomery multiplication algorithms,” *IEEE Micro*, vol. 16, no. 3, pp. 26–33, jun 1996.

[10] A. Satoh and K. Takano, “A scalable dual-field elliptic curve cryptographic processor,” *IEEE Trans. Comput.*, vol. 52, no. 4, pp. 449–460, apr 2003.