SPA against an FPGA-Based RSA Implementation with a High-Radix Montgomery Multiplier

Atsushi Miyamoto*, Naofumi Homma*, Takafumi Aoki* and Akashi Satoh†
* Graduate School of Information Sciences, Tohoku University
6-6-05, Aramaki Aza Aoba, Aoba-ku, Sendai-shi 980-8579, Japan
Phone: +81-22-795-7169, Fax: +81-22-263-9308,
E-mail: miyamoto@aoki.ecei.tohoku.ac.jp
† IBM Research, Tokyo Research Laboratory, IBM Japan, Ltd.
1623-14 Shimo-tsuruma, Yamato-shi, Kanagawa, 242-8502, Japan

***Abstract*— Simple Power Analysis (SPA) was applied to an RSA processor with a high-radix Montgomery multiplier on an FPGA platform, and the different characteristics of power waveforms caused by two types of multiplier (built-in and custom) were investigated in detail. We also applied an active attack where input data was set to a specific pattern to control the modular multiplication. The power dissipation for the multiplication was greatly reduced in comparison with modular squaring, resulting in success in revealing all of the secret key bits.**

I. Introduction

Physical attacks on cryptographic modules using side-channel information are attracting extensive attention. In order to reveal secret parameters, the power dissipation, electromagnetic radiation, or operating times as correlated to internal operations are measured. Simple Power Analysis (SPA) and Differential Power Analysis (DPA) [1], [2] are known as basic and powerful side-channel attacks, and many papers have been published. However, there are not many reports about vulnerabilities of cryptographic hardware against such attacks based on experimental results using actual implementations. Therefore, we used an FPGA platform to develop a cryptographic processor for the de facto standard public key cipher RSA and evaluated its vulnerability against SPA. The processor uses a high-radix Montgomery multiplication algorithm [3], [4] without division to accelerate the modular multiplication and squaring. The SPA attack on RSA is to detect the sequence of multiplication and squaring operations repeatedly performed using to a secret key (an exponent). Therefore, the structure of the multiplier has strong influence on the accuracy of SPA. For this work, we implemented two types of RSA processors using an embedded multiplier block in an FPGA and a custom made multiplier, and the characteristics of the multipliers for SPA are compared.

II. RSA processor

A. Modular exponentiation algorithm

The RSA cryptosystem employs modular exponentiation for encryption and decryption as follow:

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

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

ALGORITHM 1
Modular Exponentiation (*MSB First*)

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

```
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
```

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.

Binary methods are commonly used for the modular exponentiations, which perform multiplication and squaring sequentially according to the bit pattern of the exponent E or D. **ALGORITHM 1** (*MSB First*) shows a left-to-right binary method for scanning the bits of the exponent from MSB to LSB. This algorithm always performs a squaring at Line 3 independently of the scanned bit value, but the multiply operation at Line 5 is only executed if the scanned bit is 1.

ALGORITHM 2 (*MontMult*) shows the high-radix Montgomery multiplication we implemented, and **ALGORITHM 3** (*ModExp*) is the left-to-right binary method using the Montgomery multiplication, where *MontRedc* and *InvN* indicate the preprocesses to calculate $X2^k \bmod N$ and $-N^{-1} \bmod 2^k$, respectively.

B. Processor architecture

Fig. 1 shows a block diagram of our RSA processor where the data bus width is r bits to fit the word size of the multiplier. The Multiplication block repeats the multiply-additions in response to the bit pattern output from the shift register Key to perform *ModExp*. The r-bit Arithmetic core in the Multiplication block performs a multiply-addition operation (e.g., $Q = z + xy + C$) using operands stored in the registers X, Y, C and Z. Our design uses single port memories, so the multiply-addition takes two cycles to read the operands

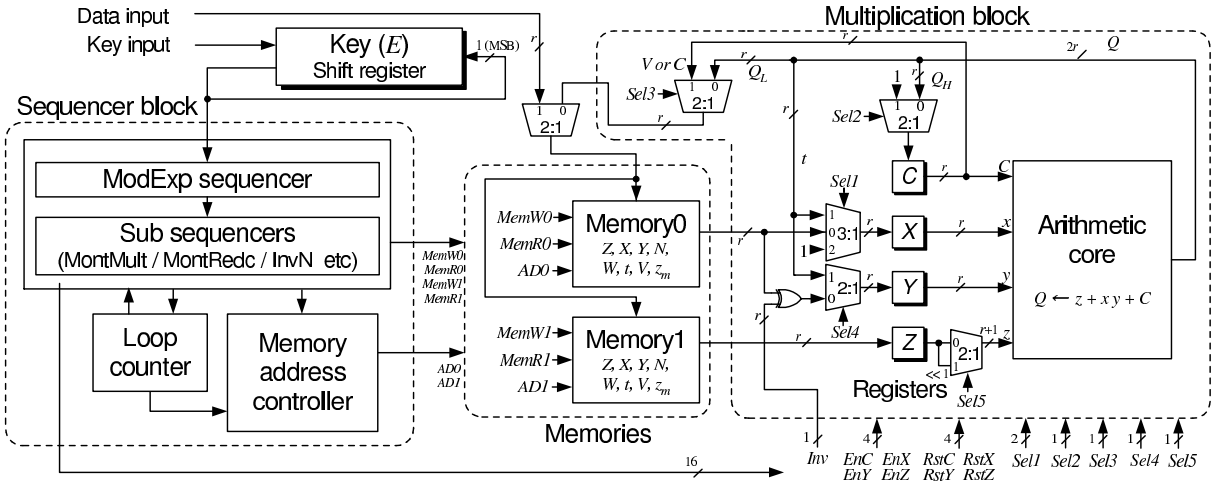

Fig. 1. RSA processor architecture.

ALGORITHM 2
MONTGOMERY MULTIPLICATION (*MontMult*)

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

1 : $Z := 0$; $V := 0$;
2 : **for** $i = 0$ **to** $m - 1$
3 : $C := 0$;
4 : $t_i := (z_0 + x_i y_0) \bmod 2^r$;
5 : $t_i := t_i W \bmod 2^r$;
6 : **for** $j = 0$ **to** $m - 1$
7 : $Q := z_j + x_i y_j + C$;
8 : $z_j := Q \bmod 2^r$; $C := Q/2^r$;
9 : **end for**
10: $z_m := C$;
11: $C := 0$;
12: **for** $j = 0$ **to** $m - 1$
13: $Q := z_j + n_j t_i + C$;
14: **if** $(j \neq 0)$ **then** $z_{j-1} := Q \bmod 2^r$;
15: $C := Q/2^r$;
16: **end for**
17: $Q := z_m + V + C$;
18: $z_{m-1} := Q \bmod 2^r$; $V := Q/2^r$;
19: **end for**
20: $C := 1$;
21: **for** $j = 0$ **to** $m - 1$
22: $Q := z_j + !n_j + C$;
23: $z_j := Q \bmod 2^r$; $C := Q/2^r$;
24: **end for**
25: **if** $(C == 1 \;||\; V == 1)$ **then return**
26: $C := 0$;
27: **for** $j = 0$ **to** $m - 1$
28: $Q := z_j + n_j + C$;
29: $z_j := Q \bmod 2^r$; $C := Q/2^r$;
30: **end for**

from the memories Memory0 and Memory1, and to write back the calculated result. When the operand and word sizes are $k = 1,024$ and $r = 32$, the Montgomery multiplication takes 4,386 cycles, and the total number of cycles for the modular exponentiation is around 7 million.

ALGORITHM 3
MODULAR EXPONENTIATION (*ModExp*)

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

1 : W := *InvN* (N);
2 : Y := *MontRedc* (X, N);
3 : $Z := 2^k \bmod N$;
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* $(Z, 1, N, W)$;

III. SPA AGAINST RSA PROCESSORS

A. Experimental condition

SPA for the RSA cryptosystem tries to obtain the secret key by distinguishing between the power waveforms of multiplication and squaring. Fig. 2 is an image of the SPA for the RSA cryptosystem using the left-to-right binary method, where the key bit pattern is "1001".

In order to capture this kind of power trace, we implemented the 1,024-bit RSA processors on a Xilinx FPGA Virtex-II with two types of 32-bit multipliers. One is an embedded multiplier block in the FPGA (Type-I) and the other is a custom array multiplier with Booth encoder (Type-II). Table I shows the synthesis report of RSA processors using the Xilinx ISE 7.1. Fig. 3 shows the experimental FPGA board INSTAC-32 [5], and the measurement point where a register is inserted between the FPGA ground pin and the ground plane of the board. The power traces were monitored by an oscilloscope 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 400 MSa/s (million samples per second). Table II summarizes the experimental condition.

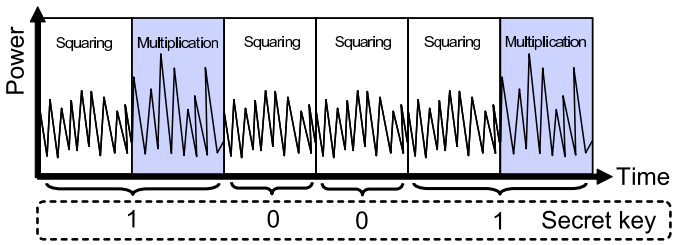

Fig. 2. SPA attack against RSA cryptosystem.

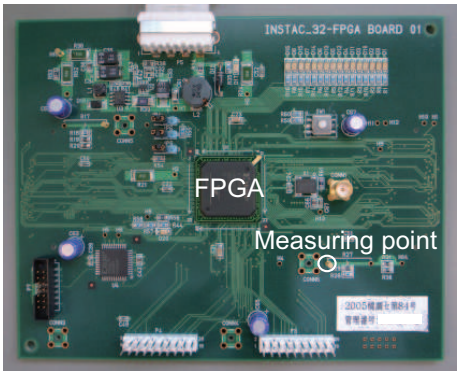

Fig. 3. Experimental FPGA board.

TABLE I

SYNTHESIS REPORT

	Type-I	Type-II
Number of slices	2,596 / 5,120 (50%)	3,592 / 5,120 (70%)
Number of FFs	3,197 / 1,0240 (31%)	3,336 / 1,0240 (32%)
Number of LUTs	4,152 / 1,0240 (40%)	6,024 / 1,0240 (58%)
Number of multipliers	4 / 40 (10%)	0 / 40 (0%)
Delay	15.63ns (63.97MHz)	37.52ns (26.65MHz)

TABLE II

EXPERIMENTAL CONDITION

Experimental FPGA board (INSTAC-32)	
FPGA	Xilinx Virtex-II xc2v1000
Crystal oscillator	2-MHz
Resistance value	5 Ohm
Experimental equipment	
Oscilloscope	Agilent DSO6104A Sampling rate: 400 MSa/s
DC-voltage power supply	Output voltage: 3.3 V

B. Experimental results

Fig. 4 shows the power trace obtained from the Type-I processor, where the horizontal and vertical axes indicate time and voltage, respectively. Two types of waveform patterns of 2.2-ms period appear in the figure, and the periods (b), (d), and (g) show higher voltage peaks in comparison with the other 4 periods. The two types of waveforms are caused by modular multiplication and squaring, and two multiplications are never executed in series by the binary method of **ALGORITHM 1** and **3**. Therefore, it is easy to determine the secret key bits for this part as "1101".

By magnifying the periods (c) and (d) of Fig. 4, a time of

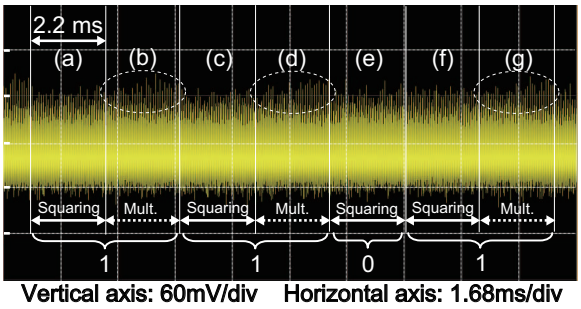

Fig. 4. Power trace of Type-I processor at 1.68ms/div.

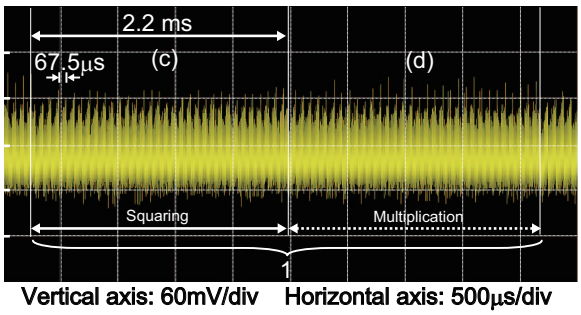

Fig. 5. Power trace of Type-I processor at 500μs/div.

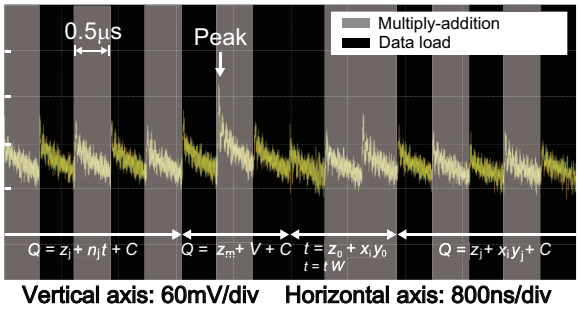

Fig. 6. Power trace of Type-I processor at 800ns/div.

67.5-μs cycle is observed between the highest peaks as shown in Fig. 5, and Fig. 6 is the more highly magnified view around the peak. The 2-MHz clock is fed to the RSA processor, and thus the cycle time is 0.5-μs. Multiply-addition and memory read access are performed in the cycles colored in gray and black, respectively.

There is no significant difference in the power waveforms of the gray and black cycles. This suggests that the arithmetic core is not the main reason for the differing characteristic of the waveforms for modular multiplication and squaring. At the highest peak in Fig. 6, the processor was executing Lines 17-18 of **ALGORITHM 2** and loading the y_0 data at Line 4. These operations are common for modular multiplication and squaring, but accessing memory is different. When we swapped the memory access patterns of the two operations, the waveform patterns were also swapped. This clearly shows that the power waveform of the Type-I RSA processor is dominated by the memory access pattern.

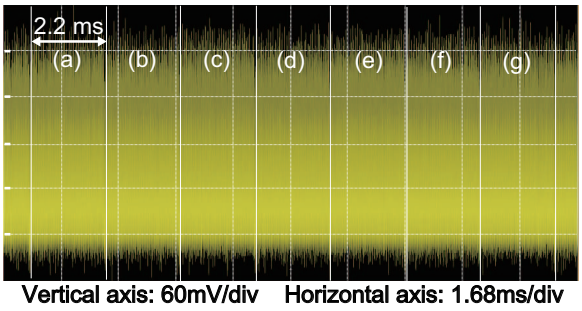

Fig. 7. Power trace of Type-II processor at 1.68ms/div.

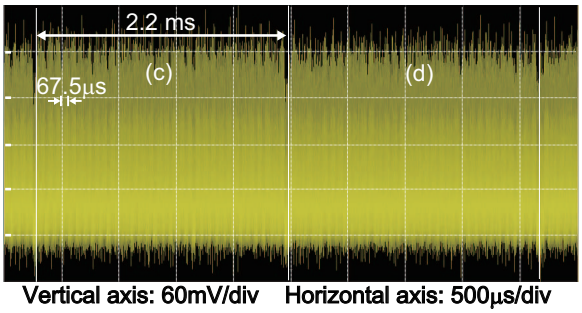

Fig. 8. Power trace of Type-II processor at 500μs/div.

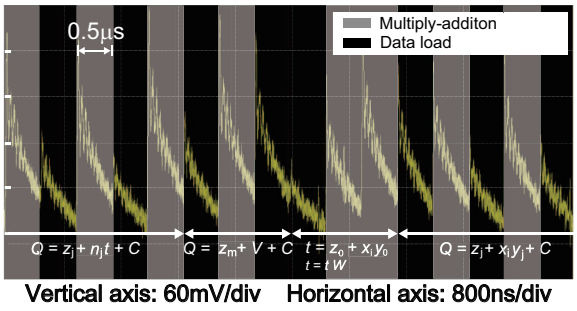

Fig. 9. Power trace of Type-II processor at 800ns/div.

Figs. 7-9 show the power traces of the Type-II processor corresponding to Figs. 4-6. The power consumption for the multiply-addition in the Type-II processor is very high, and the power used by memory access was hidden. Therefore, the multiplication and the squaring operation could not be distinguished in this case.

However, the high power characteristic of the custom multiplier could be used for another active attack. As shown in Fig. 10, the power consumption of the multiplication was significantly reduced by setting the operand Y to the specific data of all ones. This is not difficult because the operand Y is calculated from the plaintext input X and the public parameter N in **ALGORITHM 3**. Simulation results also showed that the power consumed by the multiplier at Lines 6-9 in **ALGORITHM 2** (i.e., $Q = z_j + x_i y_j + C$) was only one seventh for random Y data. This is because the inputs of the arithmetic core and the three registers do not change during the j-loop operation of Lines 6-9 when Y="111...1".

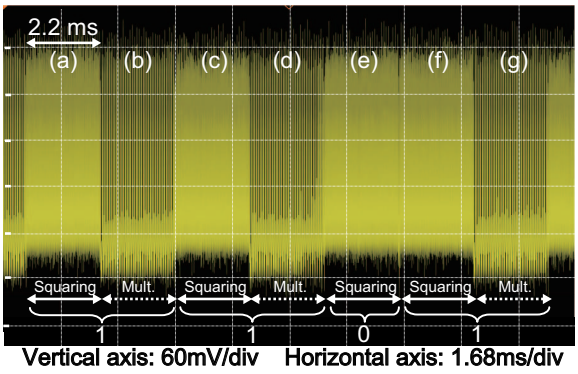

Fig. 10. Power trace of Type-II processor for specific input.

Not just for the Type-II processor, but this input data control technique can also be applied to the Type-I processor, and its high effectiveness is confirmed by both simulation and experiment with FPGA implementations.

IV. Conclusion

We implemented RSA processors with a high-radix Montgomery multiplier on an FPGA, and investigated the vulnerability to SPA. When the power consumption of the arithmetic core using the built-in multiplier was low, it was easy to distinguish between the power waveforms for modular multiplication and the squaring operations even for random inputs. By investigating the power traces in detail, we found that the different memory access patterns of the two operations highly affected the peak levels of power dissipation. When we used a custom multiplier, it consumed much more power, and thus the peak differences caused by the specific memory access patterns could not be observed. However, we introduced a technique to control the bit pattern of the operand in the modular multiplication, and succeeded in greatly reducing the power dissipation in comparison with modular squaring. As a result, we could determine the bit pattern of the secret key used to control the multiplication and squaring when no countermeasure was used in our FPGA implementations. We are going to implement our RSA processor on an ASIC chip, and will report the SPA experiments using that chip in the future.

References

[1] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *CRYPTO '99: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, pp. 388–397, Springer-Verlag, 1999.

[2] P. Kocher, R. Lee, G. McGraw, and A. Raghunathan, "Security as a new dimension in embedded system design," in *DAC '04: Proceedings of the 41st annual conference on Design automation*, pp. 753–760, ACM Press, 2004.

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

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

[5] T. Matsumoto, S. Kawamura, K. Fujisaki, N. Torii, S. Ishida, Y. Tsunoo, M. Saeki, and A. Yamagishi, "Tamper-resistance standardization research committee report," *The 2006 Symposium on Cryptography and Information Security*, pp. 1–6, 2006.