PAPER Special Section on Information and Communication System Security

Collision-Based Power Attack for RSA with Small Public Exponent

Kouichi ITOH[†,††a)], **Dai YAMAMOTO**[†b)], **Jun YAJIMA**[†c)], ***and*** **Wakaha OGATA**[††d)], ***Members***

SUMMARY This paper proposes a new side channel attack to RSA cryptography. Our target is an implementation with a combination of countermeasures. These are an SPA countermeasure by m-ary method and a DPA countermeasure by randomizing exponent techniques. Here, randomizing exponent techniques shows two DPA countermeasures to randomize the secret exponent d. One is an exponent randomizing technique using $d'_i = d + r_i\phi(N)$ to calculate $c^{d'_i} \pmod N$, and another is a technique using $d_{i,1} = \lfloor d/r_i \rfloor$ and $d_{i,2} = (d \pmod{r_i})$ to calculate $(c^{d_{i,1}})^{r_i} \times c^{d_{i,2}} \pmod N$. Using the combination of countermeasures, it was supposed that the implementation is secure against power attack. However, we firstly show the result to successfully attack the implementation of the combination of these countermeasures. We performed the experiment of this search on a PC, and complete d has been successfully revealed less than 10 hours for both attacks.

key words: *power attack, collision attack, SPA, DPA, RSA, window method, countermeasure*

1. Introduction

Power attack can reveal the secret key stored in a cryptographic device by using the information obtained from power consumption of the device. Currently, this attack is widely known as a big threat on smart cards and other cryptographic devices. Power attack is classified into two types, simple power attack (SPA) and differential power attack (DPA). For securing the cryptographic device against power attack, the device must resist to SPA and DPA.

The main concern for the countermeasure against power attack is minimizing the extra cost for resisting to these attacks. For securing RSA device, the largest interest is minimizing the overhead of the processing time and many speed-efficient countermeasures are proposed.

1.1 History of Countermeasures against Power Attack on RSA

For attacking RSA by SPA, the attacker tries to distinguish square and multiply operations performed in the device by observing the power trace. This attack can reveal the secret exponent d if binary method is implemented, because square and multiply operation is directly related with the bit value of d: square and multiply operation is performed if a bit value of d is 1, and only the square operation is performed if it is 0. For resisting to SPA, some countermeasures are known, which perform square and multiply operation in a constant sequence regardless to d. One countermeasure of this type is a square-and-multiply method [1]. This method is based on the binary method, but dummy multiply operation is performed when a bit value of d is 0. Joye's method [13] is based on the right-to-left binary method, and also performs square and multiply operations in a constant sequence without dummy operation. Another countermeasure is m-ary (window method), which also perform square (S) and multiply (M) operation in a constant sequence as $SSSSMSSSSMSSSM\ldots$.

For resisting to DPA, data randomizing is a common countermeasure technique to hide the correlation between the secret exponent d and the power trace. Most popular DPA countermeasure is to randomize a exponent by using the multiplication (RMUL) [1]. In the RMUL, the randomized exponent $d'_i = d + r_i\phi(N)$ is used instead of d, where r_i is 20-bit random number and $\phi(N)$ is Euler function of N.

Besides this method, the DPA countermeasures by dividing the exponent with the random number are also proposed (RDIV) [7], [2]. In these methods, random number r_i is generated and the quotient $d_{i,1} = \lfloor d/r_{i,1} \rfloor$ and the remainder $d_{i,2} = d \pmod{r_i}$ is calculated for the secret exponent d. Then $c^d \pmod N$ is calculated by $(c^{d_{i,1}})^r \times c^{d_{i,2}} \pmod N$. In [7], r_i is recommended to be $20 \sim 30$-bit random number to reduce the overhead of the processing time.

The bit length of r_i on RDIV is set to be long as $(\log_2 d)/2$ in [2]. If such a long bit random number is used, the overhead of the processing time can be huge, but this problem is solved by the Shamir's trick mentioned in [3]. With this technique, the overhead of the processing time is very small.

RMUL and RDIV are the technique to randomize the exponent. Other than the exponent, the modulus or the message can be also randomized. For randomizing the modulus, $N'_i = N \times r_i$ is used for random number r_i to decrypt c by $(c^d \pmod{N'_i}) \pmod N$ (RMOD) [19]. If r_i is $20 \sim 30$-bit, the overhead of the processing time is small.

For randomizing the message, c is decrypted by $c = ((R_i)^e \times c)^d \times R_i^{-1} \pmod N$ for random number R_i (RMES) [5]. This technique involves to compute a pair of the random number and its inverse, but it can be easily computed by $(R_i)^e = (\gamma^e)^{r_i} \pmod N$ and $R_i^{-1} = (\gamma^{-1})^{r_i} \pmod N$

Manuscript received August 4, 2008.
Manuscript revised December 29, 2008.

[†]The authors are with the FUJITSU LABORATORIES Ltd., Kawasaki-shi, 211–8588 Japan.
[††]The author is with the Tokyo Institute of Technology, Tokyo, 152–8550 Japan.
a) E-mail: kito@labs.fujitsu.com
b) E-mail: ydai@labs.fujitsu.com
c) E-mail: jyajima@labs.fujitsu.com
d) E-mail: wakaha@mot.titech.ac.jp

DOI: 10.1587/transinf.E92.D.897

when pre-computed constant values $\gamma^e \pmod N$ and $\gamma^{-1} \pmod N$ are provided. If r_i is 20 ∼ 30-bit, the overhead of the processing time is also small.

A DPA countermeasure based on the window method is proposed in [12]. This technique randomizes a table data in m-ary method by 20 ∼ 30-bit random number. This countermeasure can also resist to SPA because this technique is combined with m-ary method.

Another DPA countermeasure based on the window method is proposed in [20]. This countermeasure uses the technique to randomize addition chain of window method by repeating to divide the exponent d with small random numbers.

BRIP [14], [16] and RIP [10], [11] are DPA countermeasures to randomize the initial data of the modular exponent operation. BRIP provides SPA and DPA countermeasure, while RIP provides DPA countermeasure. RIP can resist to SPA by combining it with SPA countermeasure, but the variation of the combination is limited because it assumes binary method. For using RIP, square-and-multiply method is recommended for resisting to SPA.

1.2 An Analysis of DPA Countermeasures by Randomizing the Exponent

Among countermeasures described in Sect. 1.1, RMUL, RDIV, RMOD and RMES randomize the intermediate data for resisting to DPA, but these countermeasures do not resist to SPA. In other words, the strength of RMUL, RDIV, RMOD and RMES against SPA depends on the implementation of modular exponentiation, namely the choice of the binary method or the window method.

Among these countermeasures, RMUL and RDIV are the best choice to minimize the overhead of the processing time, because entire processing time is $O(n^2(n + s))$ in RMUL and RDIV, while it is $O(n(n + s)^2)$ in RMOD and $O(n^2(n + 2s))$ in RMES, where n is bit length of N and s is bit length of the random number r_i. So we discuss the best choice of modular exponentiation method to be combined with RMUL or RDIV.

Binary method is out of the choice, because (randomized) exponent is directly revealed when the attacker distinguished the sequence of square and multiply operation.

A next choice is the square-and-multiply method [1]. In this method, the sequence of square and multiply operation is repeated in a constant sequence regardless to the exponent, so this method seems to be good choice. But it is analyzed by the power attack proposed by Yen et al. [21] for the left-to-right square-and-multiply method. If this attack is used, the dummy multiply operation can be detected by the attacker, which allows the attacker to reveal the exponent, even when the exponent is randomized by RMUL or RDIV. In this attack, ciphertext c is chosen as $c = -1 \pmod N$ by the attacker. By this choice, the data pattern of the square and multiply operation is limited just only to 3 patterns: 1×1, $(-1) \times 1$ and $1 \times (-1)$. Dummy multiply operation in the square-and-multiply method can be detected if these 3 pattern are distinguished. Another choice is SPA countermeasure based on right-to-left binary method, which are right-to-left square-and-multiply method [1] and Joye's method [13]. By using these methods, SPA can be prevented, but overhead of the processing time is large because 2 square/multiply operation are required per 1-bit of the exponent.

Now, we can see the window-based modular exponentiation method is the best choice be combined with RMUL and RDIV for securing against SPA with minimizing the overhead of the processing time.

But if the sliding window method is combined with RMUL, it is proposed to be attacked by Fouque et al. when the public exponent e is small [6]. In general, the sliding window method is known to leak the partial bits of the exponent if the square and multiply operations are distinguished. In the Fouque et al.'s attack, the attacker collects non-consecutive partial bit(s) of the randomized exponent with distinguishing the square and multiply operation in sliding window. Then the attacker reveals the secret exponent d by using special search technique which uses these non-consecutive partial bit(s) as the hint to reveal the secret exponent d [6].

So, we see square and multiply operation must be repeated in a constant sequence when using the window-based modular exponentiation method. For satisfying this requirement in RSA, m-ary method is suitable. By using m-ary method, square and multiply operations are always constantly repeated as $SSSSMSSSSM, \ldots$. Since this sequence is not related to the exponent, no information is leaked by distinguishing the square and multiply operation. And m-ary method is also secure against the attack to detect dummy multiply operation, because no dummy operation is performed. Hence, the best choice of the modular multiplication method combined with RMUL and RDIV is m-ary method.

1.3 Overview of Our Proposal

We propose a new attack on the SPA and DPA-resistant implementation providing the RMUL or RDIV with m-ary method, if the public exponent e is small as $e = 3$ or 65537, and the random number is small as 20-bit. We note our attack can not break RMOD and RMES. Our result means RMUL and RDIV are faster than RMOD and RMES, but not secure against our attack. Our attack to RDIV can be applied to Fouque et al.'s attack [6], which did not include the attack to RDIV.

In m-ary method, the sequence of square and multiply operation is repeated in a constant pattern regardless to the exponent, which looks to leak no information about the exponent. So how can we get the information of the exponent? As we have already discussed, no information is leaked by distinguishing square and multiply operation because it is repeated in a constant pattern like $SSSSMSSSSM \ldots$. For obtaining the information related with the bit value of the exponent, we applied the Yen el at.'s attack [21] to m-ary

method. By choosing $c = -1 \pmod N$ as the ciphertext, we can limit the data pattern of the multiply operation to 2 patterns, 1×1 and $1\times(-1)$. These 2 pattern occur in accordance with the bit value of every m-th bit of the exponent (i.e. least bit of each m-bit window). With our collision-based power attack technique, these two multiply operations can be distinguished from just a single power trace. Therefore, our proposal is very effective and efficient.

Next, we also propose recovery technique of the secret exponent d from these non-consecutive partial bit values. Our technique is an improvement of the special search technique in [6]. With our technique, the secret exponent d can be recovered from the partial bit values with running time $O(2^s e)$, where s is bit length of the random number r_i and e is the public exponent.

Therefore, RMUL or RDIV countermeasure with m-ary method can be totally attacked by using our proposed techniques.

As Fouque et al.'s attack [6], our attack assumes the case when the public exponent e is small as $e = 3, 65537$ and the bit length of the random number is 20-bit. These parameters are mostly used in the real application and are propriety assumption. Success of the our attack depends on the running time $O(2^s e)$, not on the value of e or r_i. When running time is $O(2^s e) = O(2^{36})$, we confirmed 2048-bit d was recovered within 10 hours by the experiment ran on a PC. So our attack will work for other parameters with similar running time.

For examining our attack really works in real world, we performed two experiments. First is an experiment of our collision-based power attack to collect partial bits of the randomized exponent. Second is an experiment to reveal the secret exponent d by using the collected information. In the first experiment, we tried to distinguish 1×1 and $1 \times (-1)$ from a single power trace, and succeeded to distinguish it. In the second experiment, we tried to find d with our search technique by running a search program on a PC. When running time is $O(2^{36})$, we succeeded to reveal the secret exponent d within practically short time (less than 10 hours on Core2Quad-3.6 GHz PC) and high probability of success (around 0.9), by collecting the partial bit value of 14 randomized exponents when setting $m = 4$ on m-ary.

1.4 Organization of the Paper

Rest of this paper is organized as follows. In Sect. 2, we define the notations. In Sect. 3, we describe Fouque et al.'s attack on RMUL combined with sliding window. In Sect. 4, we describe the overview of our attack on RMUL and RDIV combined with m-ary method. In Sect. 5, we describe the idea of our collision-based power attack and show its experimental result. In Sect. 6, we describe our technique to reveal d by using the information collected by the collision-based power attack, and show its experiment result in Sect. 7. In Sect. 8, we analyze the impact of our attack on the other window-based countermeasures. In Sect. 9, we discuss how to prevent our attack when using the RMUL and RDIV, and describe conclusion in Sect. 10.

2. Preliminaries

In this section, we define some notations. In the RSA key setting, we use following notations:

- e: Public exponent.
- d: Secret exponent.
- N: A public modulus satisfies $N = pq$ for two large primes p and q, where $n = log_2 N$.
- $\phi(N)$: order of modulus N, $\phi(N) = (p-1)(q-1)$.
- c: A ciphertext, where decryption operation without CRT is $c^d \pmod N$.
- k: An small integer satisfies $ed = 1 + k\phi(N)$ for $0 < k < e$.

In RMUL countermeasure, we use following notations.

- r_i: Random numbers ($1 \le i \le v$).
- v: Number of r_i.
- s: Bit length of r_i with setting $s = 20$.
- d'_i: Randomized exponent $d'_i = d + r_i\phi(N)$.

In RDIV countermeasure, we use following notations.

- r_i: Random numbers ($1 \le i \le v$).
- v: Number of r_i.
- s: Bit length of r_i with setting $s = 20$.
- $d'_{i,1}$: First randomized exponent $d'_{i,1} = \lfloor d/r_i \rfloor$.
- $d'_{i,2}$: Second randomized exponent $d'_{i,2} = d \pmod{r_i}$.

In our attack, we use following notations.

- d_H: A value given by $d_H = (1 + kN)/e$,
- $[x]_{a,b}$: Partial bit value of x from a-th to b-th bit ($a \ge b \ge 0$).
  E.g., Let $x = 111010$, $[x]_{4,1} = 1101$.
- $Leak(d')_{a,b}$: Partial bit value of d' from a-th to b-th bit leaked by power attack ($a \ge b \ge 0$).
  E.g., let $d' = 11101011$ and assume 0,2,3,5-th bit of d' is leaked by power attack, $Leak(d')_{4,0} = *10*1$ where $*$ represents the unknown bit.
- $[x]_{a,b} \doteq Leak[y]_{a,b}$: Partial bit value of x and y are all the same from a-th to b-th bit except unknown bit of y.
  E.g., let $x = 10111101$ and $Leak[y]_{7,0} = *01*10*1$. $[x]_{7,4} \doteq Leak[y]_{7,4}$ is true, and $[x]_{7,3} \doteq Leak[y]_{7,3}$ is also true, but $[x]_{7,2} \doteq Leak[y]_{7,2}$ is false.

3. Fouque et al.'s Attack on RMUL with Sliding Window Method [6]

Fouque et al. proposed an SPA-based attack on RMUL implementation combined with sliding window method [6]. Entire steps of the attack is shown in Fig. 1. This attack consists of four steps.

In step 1, the attacker calculates upper half of d. In generic RSA key setting, upper half bits of d is difficult to calculate, but if e is very small as 3 or 65537, it is easy to calculate. In such a small e, upper half bits of d is same as

that of $d_H = (1+kN)/e$, because d satisfies $d = (1+k\phi(N))/e$ and upper half bits (e.g. leftmost $\log_2 d + 30$ bits) of $\phi(N)$ is equal to that of N with high probability. To calculate d_H, k is the only unknown parameter. But this is not problem because k is very small value that satisfies $0 < k < e$. Especially, k is known to satisfy $k = 2$ when $e = 3$ [6]. When $e = 65537$, the attacker can perform brute-force search for every possible value $0 < k < e$. If assumed value of k is not correct, it is detected in the matching algorithms of step 3 described in later. Hence, we can assume that value of k is correct in step 4.

In step 2, the attacker collects the power trace to reveal partial bit value of the randomized exponent $d'_i (1 \le i \le v)$. In this step, the attacker distinguishes the square and multiply operation performed in the sliding window method to collect non-consecutive bits of d'_i as shown in Fig. 2. If multiply operation was observed, the attacker can know the corresponding bit of d'_i is 1. If continuous square operations were observed more than m-times, the attacker knows the corresponding bit of d'_i is 0, where m is the bit length of the window.

In step 3, the attacker tries to find all random numbers $r_i, (1 \le i \le v)$ from partial bit information of d'_i obtained in step 2. Since each r_i is 20-bit value, all of r_i can be revealed by brute-force search. In this search, the attacker assumes a 20-bit value of r_i for some i. Then he calculates $d_H + r_i N$ to simulate upper half bits of $d'_i = d + r_i\phi(N)$. Since upper half bits of d_H and N are the same as that of d and $\phi(N)$ respectively, upper half bits of $d_H + r_i N$ are equal to d'_i with high probability. So matching algorithm can be used by comparing the partial bit value of d'_i observed in step 2 and the corresponding bit value of $d_H + r_i N$ to choose correct r_i for some i.

If the number of the observed bits of d'_i in step 2 is longer than 20-bit, the attacker can choose a unique r_i. Furthermore, if the observed bits of d'_i is longer than 37-bit (Maximum value of $20 + \log_2 k$), the attacker detect the assumed value of k is correct or not.

r_i and k are uniquely chosen with high probability for typical parameter of RSA and sliding window method. If we assume a case d is 1024-bit and $m = 4$, at least 1/4 of upper half of d'_i (e.g. 480 bits) would be leaked, which means 120-bit of d'_i is leaked. This bit length is enough to uniquely determine unique r_i and k. Hence, if no matched r_i is found for some k and $i = 1$, the assumed value of k is turned to be wrong, and the attacker returns to step 1 without proceeding the search of r_i for $i \ge 2$. Most time-consuming process through entire steps is the loop between step 1 to step 3 until correct k is found, and the running time is estimated as $O(2^s e)$. After the correct k and r_1 is found, all r_i are obtained by repeating the search of r_i for every $i = 2, \ldots, v$.

In step 4, the attacker finds lower-half bits of d. This is done by repeating 8-bit brute-force search, which runs from lowest 8-bit to higher 8-bits. In this search, d is revealed by comparing the partial bit value of d'_i obtained in step 2 and simulated value of $d + r_i\phi(N)$ for all $1 \le i \le v$. For speeding up this search, the attacker utilizes the relationship $d = (1 + k\phi(N))/e$ to eliminate d. Hence, the attacker performs the brute-force search for only the value of $\phi(N)$. At first, the attacker assumes lowest 8-bit value of as $\phi(N)$ =????????, and if matched 8-bit value is found (e.g. 11011001), he performs brute-force search for next 8-bit higher bits as $\phi(N)$ =????????11011001 and repeats similar search process. If the search is finished for lowest-half bits of $\phi(N)$, the attacker knows all bits of $\phi(N)$ because upper half bits are equal to N with high probability. Thus the attack is completed.

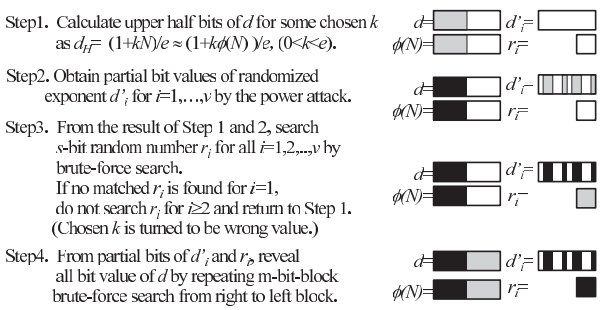

Fig. 1 Entire steps of Fouque et al.'s attack [6].

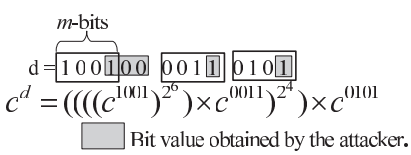

Fig. 2 Bit leakage of previous attack [6] on m-bit sliding-window method.

4. An Overview of Our Attack

In this section, we propose an attack on RMUL and RDIV implementation combined with m-ary method shown in Table 1. Entire steps of the attack is shown in Fig. 3.

As previous attack [6], our attack also consists of four steps. And the objective of each step is also the same as previous attack with the same running time $O(2^s e)$, but the method differs in step 2, 3 and 4, which are summarized as

Table 1 m-ary method.

INPUT: $c, N, d = (d_{\gamma-1}, \ldots, d_1, d_0)_{2^m}$
OUTPUT: $X = c^d \pmod N$
1: $w[0] := 1$;
2: for $(i = 1; i < 2^m; i++)$
3: $w[i] := w[i-1] \times c \pmod N$;
4: next i;
5: $X := w[d_{\gamma-1}]$;
6: for $(i = \gamma - 2; i \ge 0; i--)$
7: for $(j = 0; j < m; j++)$
8: $Z := Z \times Z \pmod N$
9: next j;
10: $X = X \times w[d_i] \pmod N$;
11:next j;
12:return X;

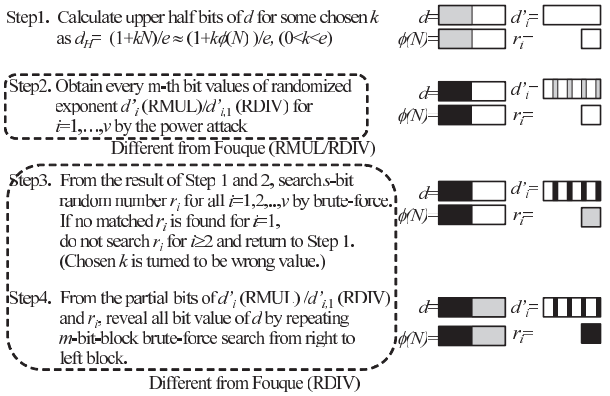

Fig. 3 Entire steps of our attack.

below.

- Step 2: In the previous attack, the attacker obtains partial bit values of the randomized exponent by distinguishing square and multiply operation. But in our attack, the attacker obtains partial bit values of the randomized exponent by distinguishing multiply operation performs 1×1 or $1 \times (-1)$ for a chosen message $c = -1 \pmod N$.
- Step 3, 4: When attacking RMUL, method of this step is almost the same as that of previous attack. When attacking RDIV, method of this step is different from previous attack.

Largest difference is step 2. It comes from the assumption of the attack. The previous attack assumes the target device implements sliding-window method, which allows to leak partial information of the randomized exponent by distinguishing square and multiply operations. But our attack assumes the target device implements m-ary method, which does not leak any information of the randomized exponent by distinguishing square and multiply operations. So we adopt another approach to obtain partial bit value of the randomized exponent.

Our approach is to observe the power trace by choosing the ciphertext c as $c = -1 \pmod N$. When $c = -1 \pmod N$ is chosen, data pattern of the square and multiply operation is limited only to 3 patterns, 1×1, $1 \times (-1)$ or $(-1) \times (-1)$. This idea is based on the Yen et al.'s attack to binary-based method [21], which enables to directly reveal all bits of the randomized exponent. By applying this idea into m-ary method, we found partial bit values are leaked for every m-th bit. (See Fig. 4.)

By using this leaked partial bit values, the attacker tries to obtain the rest of information to reveal d in step 3 and 4, which are the almost same as previous method when attacking RMUL, or modified method when attacking RDIV.

Our attack in step 2 and its experimental result is explained in Sect. 5. Our attack in step 3 and 4 are explained in Sect. 6.

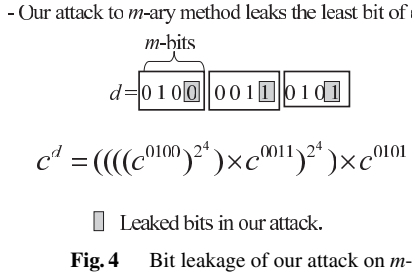

Fig. 4 Bit leakage of our attack on m-ary method.

5. Our Power Attack to Obtain the Partial Bit Value of Randomized Exponent on m-ary Method

In this section, we describe our method to obtain the partial bit value of the randomized exponent using power attack. Since our target is RMUL and RDIV countermeasures, the attacker must observe the bit information of the randomized exponent just from a single power trace because random number is varied in every time. This observation is not easy because noise always appear in the real measurement of the power trace.

For this goal, we focused on the power attack technique using a chosen message. This technique assumes the attacker can input any value of c as the ciphertext of RSA decryption, which allows to obtain more information than the attack inputting random ciphertext. For attacking RSA, many results with this technique are proposed [4], [8], [9], [21]. We extended the Yen's technique [21] to m-ary method.

5.1 Previous Power Attack Using the Chosen Message

In this section, we describe previous power attack techniques using a chosen message on RSA. First attack of this type was doubling attack proposed by Fouque et al. [4]. In this attack, the attacker observes a pair of power traces when $c = X \pmod N$ and $c = X^2 \pmod N$ for some chosen integer X. The attacker guesses some bit of the exponent, and if the guess is correct, same intermediate value appear for these two different inputs. This is so-called "collision", and the attacker can detect it by comparing the power trace of these two inputs.

Another type of the chosen message attack is proposed by Yen et al. [21]. This attack is based on the following observation: if $c = -1 \pmod N$ is input, the data pattern of the square or the multiply operation is limited to only 3 patterns, 1×1, $1 \times (-1)$ or $(-1) \times (-1)$, which make it easy to distinguish the power trace. Their attack is applicable on both binary method and square-and-multiply always method [1]. It can directly reveal all bits of the exponent from the observed power trace.

By using this idea on binary method, square and multiply operations are easily distinguished because $1 \times (-1)$ corresponds to multiply operation, while 1×1 and $(-1) \times (-1)$ correspond to square operation. This attack is also applicable on the square-and-multiply-always method [1], because

the attacker can distinguish the dummy multiply operation by distinguishing square operation is 1×1 or $(-1) \times (-1)$[†]. This attack is extended to the attack using a message pair $c = X \pmod N$ and $c = -X \pmod N$, which is generalized as using the pair (X, Y) satisfying $X^\alpha = Y^\alpha \pmod N$ [21]. They did not show the experimental result to distinguish the power trace of these attacks, but it was shown by Homma et al. [8].

Yen et al.'s message pair attack was generalized by Homma et al. [9]. They proposed to use a message pair (X, Y) satisfying $X^\alpha = Y^\beta \pmod N$ and showed the validness of the attack by the experiment result.

5.2 Basic Idea of Our Chosen Message Attack

For attacking RMUL or RDIV, we assumed that a power attack using chosen message pair is not available, because data pattern of the square/multiply operation is randomly varied between the first and the second message. So we focused on the chosen message attack using $c = -1 \pmod N$ [21]. However, it does not make any sense to distinguish the square/multiply operation or dummy multiply operation on m-ary method. So what type of the information is obtained?

We focused on the window calculated in the process of m-ary method. In Table 1, $w[i]$ represents the window which satisfies $w[i] = c^i \pmod N$ for $0 \le i \le 2^m - 1$. If $c = -1 \pmod N$ is chosen, we can see that $w[i] = 1$ when i is even and $w[i] = -1$ when i is odd. So the multiply operation shown in line 10 of Table 1 are divided into 2 cases: when d_i is even, 1×1 is performed, and when d_i is odd, $1 \times (-1)$ is performed. Hence, if we can distinguish the multiply operation is 1×1 or $1 \times (-1)$ from a single power trace, every m-th bit of the exponent of m-ary will be leaked. By using this idea to RMUL or RDIV, partial bit value of the randomized exponent will be leaked.

"Randomized exponent" is equal to $d'_i = d + r_i\phi(N)$ in RMUL, and $d'_{i,1} = \lfloor d/r_i \rfloor$ in RDIV. We note that this attack has exception to fail get partial information of the randomized exponent in RDIV. If RDIV is implemented as to calculate $c^{r_i} \pmod N$ and $(c^{r_i})^{d'_{i,1}} \pmod N$ in this order, partial information of the randomized exponent can not be obtained when r_i is even because the multiplication is always 1×1. To avoid this case, the attacker must ignore the power trace when r_i is even. If r_i is even, it can be detected by distinguishing the final multiply operation of $c^{r_i} \pmod N$ is 1×1.

5.3 How to Distinguish the Multiplication

If we can distinguish 1×1 and $1 \times (-1)$ just by observing the waveform of a single power trace, there is no problem in step 2 of our attack in Fig. 3. But it will be difficult because we assume to distinguish it from a single power trace including the noise content.

So we introduce collision-based power attack technique to distinguish it easier. Our collision-based power is illustrated in Fig. 5. Let $Power_i(t)$ be the power trace of d'_i in RMUL or $d'_{i,1}$ in RDIV, and Δt be the processing time of square or multiply operation, a differential power trace

$$Power_i(t) - Power_i(t - \Delta t) \tag{1}$$

is made by the attacker in our collision-based power attack. If the multiplication is 1×1, it causes a collision to squaring operation of 1×1 performed just before the multiply operation, the differential power trace of this part will be plain. In contrast, if the multiplication is $1 \times (-1)$, the differential power trace of this part will be peak, whose length will be $3\Delta t$ as illustrated in Fig. 5.

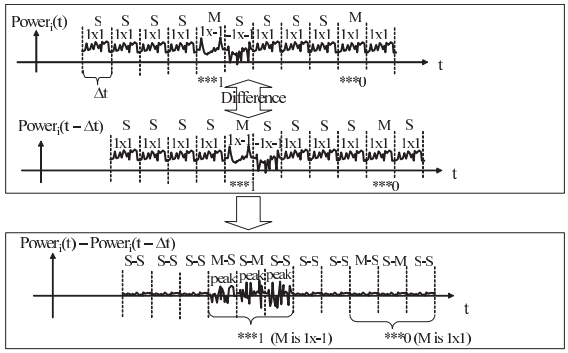

Fig. 5 An overview of our collision-based power attack.

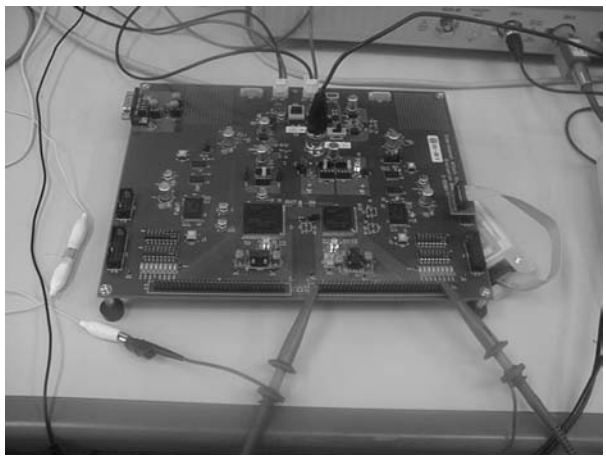

Fig. 6 The platform board used in our power attack experiment, SASEBO (Sidechannel attack standard evaluation BOard) [18].

5.4 Experimental Result of Our Collision-Based Power Attack

In this section, we show the experiment result of our collision-based power attack. As a platform device, we used SASEBO (Sidechannel Attack Standard Evaluation BOard) board [18] shown in Fig. 6, which is a FPGA-based standard platform for evaluating the strength of side-channel analy-

[†]This attack can be extended to attack on the binary-method version of BRIP [14], [16].

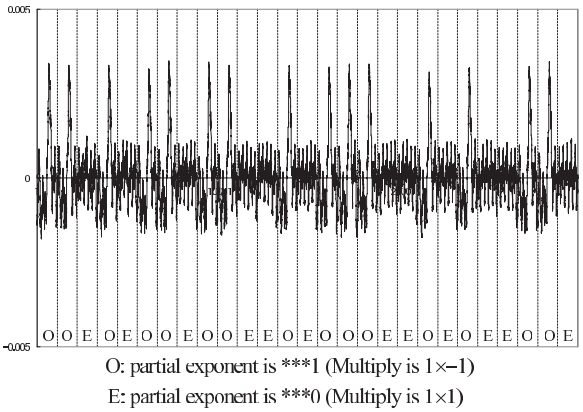

Fig. 7 The differential power trace of Fig. 5 in our experiment.

sis. We implemented 128-bit† modular exponentiation algorithm for the randomized exponent with m-ary method for $m = 4$ on FPGA.

Through this experiment, we set 24-MHz as the clock of the FPGA and 125 Ms/sec as the sampling ratio of the oscilloscope. Our experiment result of the differential power trace by (1) is shown in Fig. 7. When making each differential power trace, we tuned it with simple moving average to reduce the noise of the single power trace, and the differential trace is made from a single power trace for a randomized exponent d'.

We can see both trace reveals us d' =???? $*\,*\,*\,1\,*\,*\,*\,1\,*\,*\,*\,0\,*\,*\,*\,1\,*\,*\,*\,0\,*\,*\,*\,1\,*\,*\,*\,1\,*\,*\,*\,0\,*\,*\,*\,1\,*\,*\,*\,1\,*\,*\,*\,0\,*\,*\,*\,0\,*\,*\,*\,1\,*\,*\,*\,0\,*\,*\,*\,1\,*\,*\,*\,1\,*\,*\,*\,1\,*\,*\,*\,0\,*\,*\,*\,0\,*\,*\,*\,1\,*\,*\,*\,0\,*\,*\,*\,1\,*\,*\,*\,0\,*\,*\,*\,0\,*\,*\,*\,1\,*\,*\,*\,1\,*\,*\,*\,0\ldots$,†† which perfectly matches the value of d' used in the real experiment as d' = 0x71f8d6bf6b3c07a73d60187aadba6...

From this result, we can see the power trace of Fig. 7 clearly shows us the partial bit information of d'.

6. Our Attack to Reveal d

In this section we describe our attack to reveal d, which are step 3 and step 4. In step 3, all random numbers r_i are revealed from the partial bit value of the randomized exponent obtained by the power attack in step 2. In step 4, lower-half bits of d is revealed from r_i and partial bit value of the randomized exponent obtained in step 2. When step 4 is finished, the attacker can obtain all bits of d, because upper half bits of d is already revealed in step 1.

6.1 Our Attack to Reveal r_i

Our attack to reveal r_i on RMUL is shown in Fig. 8, and that in RDIV is shown in Fig. 9. Both of these attacks are based on the simple idea: choose 20-bit r_i by brute-force attack, that matches the partial bit value of the randomized exponent obtained by power attack. This attack is possible because upper-half bit value of d'_i is simulated as $d_H + r_i N$ in RMUL, and $d'_{i,1}$ is simulated as $\lfloor d/r_i \rfloor$ in RDIV for the candidate of r_i.

```
Input: n, N, e, k, h=(n+50)/2, Leak(d'_i) for all 1≤ i ≤ v.
Output: 20-bit random numbers r_i for all 1 ≤ i ≤v.
 d_H = ⌈ (1+Nk)/e ⌉;
 for (i=1; i≤v ; i++) { /* Search i-th random number r_i */
    for (z=1; z<2^20; z++) { /* z is a candidate of r_i */
       y = d_H + z×N; /* simulate random exponent */
       if ([y]_((20+n), h) = Leak(d'_i)_((20+n), h) )) {
             r_i = z;  /* if y is matched, set r_i to z */
             break; /* proceed to search r_(i+1) */
       }
    }
 }
 return r_i;
```

Fig. 8 Our attack algorithm to reveal r_i on RMUL.

```
Input: N, k, e, n, h=(n+10)/2, Leak(d'_(i,1)) for all 1≤ i ≤ v.
Output: 20-bit random numbers r_i for all 1 ≤ i ≤v.
 d_H = ⌊ (1+kN)/e ⌋;
 for (i=1; i≤v ; i++) { /* Search i-th random number r_i */
    for (z=1; z<2^20; z++) { /* z is a candidate of r_i */
       y = ⌊d_H / z⌋; /* simulate random exponent */
       if ([y]_((n-20), h) = Leak(d'_(i,1))_((n-20), h) )) {
             r_i = z;  /* if y is matched, set r_i to z */
             break; /* proceed to search r_(i+1) */
       }
    }
 }
 return r_i;
```

Fig. 9 Our attack algorithm to reveal r_i on RDIV.

Figure 8 is the same as Fouque et al.'s attack [6]. Figure 9 is based on the same idea of Fouque et al.'s attack, but modified for RDIV countermeasure.

Both attack fails to reveal any random number r_i when input value of k was wrong. So, these attack can be used to check the value of k is correct. ($v = 1$ is enough to check k.) This checking is unnecessary when $e = 3$, but necessary when $e = 65537$ because k is unknown value. Since k is small as $0 < k < e$, brute-force search allows the attacker to find correct k. But number of the candidates of the search is large as $2^{16} \times 2^{20} = 2^{36}$, which is the largest bottle-neck of our all attack steps on revealing d.

6.2 Our Attack to Reveal Lower-Half Bits of d

Our attack to reveal lower-half bits of d on RMUL is shown in Fig. 10, and that on RDIV is shown in Fig. 12. Figure 10 is the almost same as the Fouque et al.'s attack [6], and Fig. 12 is modified from Fouque et al.'s attack [6].

6.2.1 Revealing Lower-Half Bits of d on RMUL

In the attack of Fig. 10 on RMUL, right-to-left brute-force

†We implemented shorter (128) bit length of RSA due to the resource constraint of the FPGA device. But we assumed our experiment can emulate the 1024-bit or 2048-bit environment, because we suppose our collision-based power attack is not affected by the bit length of RSA.

††Note that leftmost 4-bit of d' cannot be detected from this differential power trace, but the attacker doesn't have to know these bits because it is unnecessary for step 3 and 4 of our attack in Sect. 4.

```
Input: N, n, e, k, h=⌈(n+50)/(2w)⌉, Leak(d'_i) and r_i for all 1≤ i ≤ v.
Output: Search result of φ(N)
α := −k^{-1} (mod e); /* φ(N) satisfies φ(N) =eλ + α for some integer λ */
X := 0;
for (j=0; j≤h ; j++) {
 for (z=0; z<2^{m+1}; z++) {
   y := (z × 2^{jm} + X) e + α;
   u_i := (1+ky)/e + r_i × y; /* simulate random exponent */
   if ([u_i]_{jm+m, jm} ≐ Leak(d'_i)_{jm+m, m} is true for all 1≤ i ≤ v)) {
     X := (z mod 2^m) × 2^{jm} + X);
     /* if all u_i are matched, set lower m-bit of z to X */
     break; /* proceed to search next m-bit */
   }
 }
}
Y := ([N]_{n,hw} || [Xe+ α]_{hw-1,0} );
/* combine higher and lower bit of φ(N) */
return Y;
```

Fig. 10 Our attack algorithm to reveal $\phi(N)$ on RMUL.

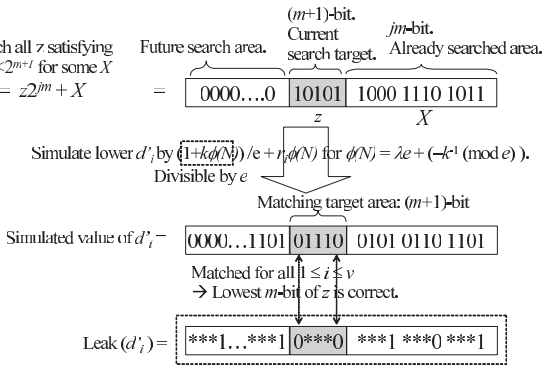

Fig. 11 Matching algorithm of $\phi(N)$ in Fig. 10.

search is performed for $\phi(N)$ like [6]. This is because d'_i is simulated by only $\phi(N)$ as $d'_i = d + r_i\phi(N) = (1 + k\phi(N))/e + r_i\phi(N)$ for known r_i. In our search, candidate value of $\phi(N)$ is generated as $\phi(N) = e\lambda + (-k^{-1} \pmod e)$, for integer λ, which is slightly modified technique from [6]. This technique is convenient because $1 + k\phi(N)$ is always divisible by e, which makes the right-to-left brute-force search easier because lowest bit of d'_i is perfectly simulated with the same length of the lowest bit of λ.

In this search, λ is divided into m-bit blocks, and brute-force search is performed for every m-bit block running from the lowest block to the higher block. (not to the highest block, because upper-half bits of $\phi(N)$ is known). That is, m-bit brute-force search is done for the lowest m-bit of λ from 2^m candidate values as $\lambda = 0000, 0001 \ldots, 1111$. And if matched m-bit is found as 1101, next upper m-bit is searched from 2^m candidate values as $\lambda = 00001101, 00011101, \ldots 11111101$, and the similar process is repeated until all bits are obtained.

Here our concern is how to find the matched value from candidate values. It is illustrated in Fig. 11. Matching is done by comparing the simulated value of d'_i and $Leak(d'_i)$, a partial bit value of d'_i obtained by power attack in step 3, for all $1 \le i \le v$. Candidate values of $\phi(N)$ is generated by setting $\lambda = z2^{jm} + X$ for $z = 0, 1, \ldots, 2^{m+1} - 1$, where X represents the lower bits of λ already found in the search. We call z 'search target', which represents the number of candidate values to determine m-bit in our brute-force search.

In our search on RMUL, search target has $m + 1$-bit length to match simulated value of d'_i to match $(mj + m)$-th and mj-th bit of $Leak(d'_i)$ as illustrated in Fig. 11. After matching candidate is found, lower m-bit of z is selected as next higher m-bit of λ by updating X as $X := (z \pmod{2^m}) \times 2^{jm} + X$.

Though the leaked bit $Leak(d'_i)$ is non-consecutive, all bits of $\phi(N)$ are determined by this search. This is because d'_i is generated by multiplying $\phi(N)$ with random number r_i, and the random number makes it possible to leak the various bit value information of $\phi(N)$. If d'_i is not generated by random number as $d'_i = 3\phi(N)$, all bits of $\phi(N)$ can not be leaked from $Leak(d'_i)$, but it reveals us just only partial non-consecutive bits of $\phi(N)$.

We performed an experiment on a PC to examine this attack really reveals all bits of $\phi(N)$. We succeeded to reveal $\phi(N)$ with high probability as 0.9, if $m = 4$ and $v = 14$. See Sect. 7 for details.

6.2.2 Revealing Lower-Half Bits of d on RDIV

In the attack of Fig. 12 on RDIV, left-to-right brute-force search is performed for d. The direction of the search on RDIV is inverse to that on RMUL. This is because of the difference between the multiplication and division. That is, the answer is determined from lowest to highest bit in multiplication, while the answer is determined from highest to lowest bit in division.

As similar in the search on RMUL, d is divided into m-bit blocks, and the brute-force search is performed for every m-bit block running from the center block to the lower block. This search does not start from the highest block, because upper-half bits of d is known. Upper bit is initialized with that of $\lfloor d_H = (1 + kN)/e \rfloor$ as shown $X := [d_H]_{n,hm}$ in Fig. 12.

Our method to match every m-bit block is illustrated in Fig. 13. In this method, matching is done by comparing the simulated value of $d'_{i,1}$ and $Leak(d'_i)$, a partial bit value of d'_i obtained by power attack in step 3, for all $1 \le i \le v$.

This idea is similar to our matching algorithm on RMUL, but there is a difference in selecting matched m-bit value from candidates. In our matching on RDIV, the selected candidate value might be wrong, because $d'_{i,1}$ is simulated by dividing the candidate value by r_i. Since the candidate value is determined from upper bit to lower bit, the simulated value of $d'_{i,1}$ is approximated value of $d'_{i,1}$, but contains error with some probability.

For example, let us assume the case d is $d = 0x9876ff34\ldots$ for $m = 4$. In this case, left-to-right search goes well until the intermediate search result is $d = 0x987?????\ldots$, but the problem is next 4-bit search. We know the answer is 6, but the search algorithm might regard both 6 and 7 are matched candidate value, because '$6ff$' in $d = 0x9876ff34\ldots$ is close to '700', and both of them might match the search algorithm because of the error in simulat-

```
Input: n, d_H=(1+Nk)/e, k, g, h=⌈(n+10)/(2w)⌉, Leak(d'_{i,1}) and r_i for all 1≤ i ≤ v.
Output: Search result of X, satisfies |X − d| < max (2^20, 2^mg)
X:= [d_H]_{n, hm}; /* Set upper-half bit of d to d_H */
for (j:=0; j≤ h − g ; j++) τ_j:=0; X_{C,j} ← ∅; /* Initialize candidate value set */
for (i:=1; i≤v; i++) s_i = ⌈log_2(2^hm/r_i)⌉; /* Initialize matching bit of each d'_{i,1} */
for (j:=h − g, a=0; j>0 ; j−−, a++) {
  F = 0; /* 1:search was success for some z, 0: search failed for all z. */
  for (z:=0; z<2^mg; z++) {
    y := X + z × 2^jm;   /* candidate value of d */
    u_i := ⌊ y/r_i ⌋;      /* simulate random exponent */
    if ([u_i]_{s_i−am−1, s_i−am−m} ≐ Leak(d'_{i,1})_{s_i−am−1, s_i−am−m}   is true for all 1≤ i ≤ v)) {
      if [z]_{mg−1, mg−m} ∉ X_{C,j} {τ_j := τ_j+1;} /* count elements of X_{C,j} */
      X_{C,j} ← X_{C,j} ∪ [z]_{mg−1, mg−m}; /* append top m-bit of z to X_{C,j} */
      F=1; /* set flag */
    }
    if (F=1) {T:=(τ_j-th element of X_{C,j}); X := X+ T × 2^{jm+mg−m}; }
    else { /* If no matched value is found, roll-back the search */
      X_{C,j} ← X_{C,j} − (τ_j-th element of X_{C,j}); /* remove bad element of X_{C,j} */
      τ_j := τ_j − 1; /* reduce element counter of X_{C,j} */
      /* roll-back the counter value, but if j > h−g, abort this algorithm */
      do {j:= j+1; a := a − 1; if (j>h−g) return ERROR;  } while {τ_j=0};
      /* roll-back the value of X_{C,j} */
      T:=(τ_j-th element of X_{C,j}); X:= [X]_{n, jm+m} + T × 2^jm;
    }
  }
}
return X;
```

Fig. 12 Our attack algorithm to reveal d on RDIV.

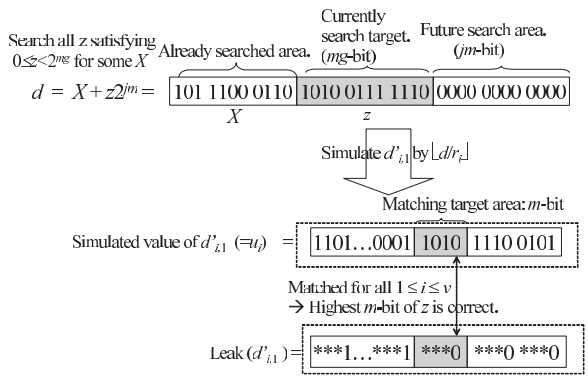

Fig. 13 Matching algorithm of d in Fig. 12.

ing $d'_{i,1}$.

We can't prevent this error completely unless (almost of) all bits of d are known, but we can reduce the probability of error. To reduce the probability of the error, we recommend to enlarge the bit length of the search target. This idea is included in Fig. 12, in which $d'_{i,1}$ is simulated by "$u_i = \lfloor y/r_i \rfloor$". We note y is the candidate value of d given as $y = X + z \times 2^{jm}$ by z and X, where z is the search target of mg-bit value and X is the upper bit value of d already found in the search. In Fig. 10, bit length of the search target z is $m + 1$, but it is enlarged to mg by the parameter g to reduce error. As parameter g is larger, the probability of error is reduced, but the running time becomes larger. We recommend to set g as to mg is close or equal to 20.

Unlike our search on RMUL, plurality of the matched candidate can be found in our search on RDIV, no matter how g is large. Hence, our search in RDIV picks up plurality of matched candidate values. It is noted as '$X_{C,j}$' in Fig. 12, which is a set of m-bit matched candidate values. For example, if both of 0x6ff and 0x700 are matched for $m = 4$, $X_{C,j}$ becomes $X_{C,j} = \{6, 7\}$. The operation to the set $X_{C,j}$ is noted by '$\leftarrow$' (e.g. $X_{C,j} \leftarrow X_{C,j} \cup [z]_{mg-1,mg-m}$;), to distinguish it from the operation to other variables. Number of the set of $X_{C,j}$ is stored in τ_j of Fig. 12.

Due to the error in simulating $d'_{i,1}$, this search algorithm does 'roll-back' when no matched candidate is found. The 'F' flag in Fig. 12 holds the result whether matched candidate value is found or not. If some candidate is found, F is 1. But if no candidate is found, F is 0. And the 'roll-back' routine is performed if F is 0 after all search for $0 \le z < 2^{mg}$ is finished. By the 'roll-back' routine, brute-force search of d temporary goes back to upper m bit. For example, if the intermediate search result of d is 'd = 0x986?...' or 'd = 0x987?...', and no candidate value is matched in searching '?' of '0x987?', it is eliminated from the candidate of value of d, then the search goes back to find '?' of '0x986?' to continue the left-to-right search.

After the search of Fig. 12 is finished, the output value X is approximated value of d which satisfies $|X - d| < MAX(2^{20}, 2^{mg})$, where $MAX(a, b)$ represents the larger value of a and b. The rest bits of d can be revealed by the similar search by comparing the simulated value of $d'_{i,2} = d \pmod{r_i}$ and $Leak(d'_{i,2})$. If few candidates still remains after this search is finished, final value of d is easily determined by confirming the decrypting operation succeeds for some pair of the ciphertext and the plaintext, where the pair of the ciphertext and the plaintext is easily obtained by only public key (e, N).

We performed an experiment on a PC to examine this attack really reveals all bits of d. We succeeded to reveal d with high probability 0.91, if $m = 4$ and $v = 14$, which is similar result on RMUL. See Sect. 7 for details.

7. Experimental Result to Reveal d

In this section, we report the experiment result to reveal d for validating our attack. As described in Sect. 4, our attack consists of four steps. The attack method of step 2 is power attack, and the attack method of other steps is the programming code runs on a PC. This section reports the experimental result of step1, 3 and 4 ran on a PC. Experimental result of the step 2 is reported in Sect. 5.

In the experiment, we performed our attack on RMUL and RDIV. In both attack, we set RSA the key parameter as $e = 3, 65537$ and $n = 1024, 2048$. Parameter of m-ary is fixed to $m = 4$, bit length of the random number s is set to $s = 20$, the number of the random exponent attacked in step 2 is fixed to $v = 14$, and the bit length of the search target of our attack on RDIV is set to $mg = 20 (g = 5)$.

We implemented our attack with the Shoup's NTL library [17], and ran it on a Fedora 7 PC with Core2-Quad 3.6 GHz CPU.

In the experiment, we measured the running time and the success ratio of our attack in the experiment. All of results are summarized in Table 2.

In all cases, our attack succeeded with high probability (from 0.8 to 0.95) even when v is not so large ($v = 14$). From

Table 2 Experimental Results to Reveal d using our attack.
($v = 14, m = 4$, NTL library [17] on Fedora 7 with Core2Quad-3.6 GHz CPU)

Countermeasure	n	Running time		Success ratio		Bit length of search target (= mg)
		$e = 3$	$e = 65537$	$e = 3$	$e = 65537$	
RMUL	1024	3 seconds	3.5 hour	18/20	17/20	N/A
	2048	3 seconds	4 hour	18/20	19/20	N/A
RDIV	1024	16 minutes	4.6 hour	19/20	19/20	20
	2048	51 minutes	7 hour	16/20	19/20	20

the result of the attack on RDIV, we can see that $g = 5$ gives enough high probability to reveal correct d.

If we compare the running time of $e = 3$ and $e = 65537$, the case $e = 65537$ is much longer than the case $e = 3$. This difference come from the running time is $O(2^s e)$, that is, brute-force attack of k is required in the attack $e = 65537$, while it is not required in $e = 3$. The step to find correct k takes the longest time in these parameter settings, but we can see the running time practically short because all attacks finished within 10 hours.

From these results, we can see the validness of our attacks, which can reveal d within short time (at most few days even when using middle-range PC) and high probability (around 0.9).

8. Impact of Our Attack on the Other Window-Based Countermeasure

In this section, we discuss the impact of our attack on other window-based countermeasures for RSA.

8.1 Window Method Version of BRIP [14], [16]

In this countermeasure, the exponent is not randomized. By using our attack in step 2, d can be obtained for every m-th bit, but information of other bit position of d is not leaked. Hence it is difficult to reveal d from these information.

8.2 Ciet-Joye Countermeasure [2]

This is variant of RDIV, where the randomized exponent $\lfloor d/r_i \rfloor$ and $d \pmod{r_i}$ are calculated for random number r_i whose bit length is large as $(log_2 d)/2$. If such a large bit length is used, it is difficult to reveal random number r_i by brute-force search as step 3. So this countermeasure is difficult to attack.

8.3 MIST Countermeasure [20]

In this countermeasure, random addition chain is constructed by repeating to divide the exponent d by small random numbers. Similar as Ciet-Joye's countermeasure [2], such a long sequences of random number looks like to be difficult to attack by brute-force attack, but further discussion be needed to prove its security.

8.4 Randomized Window Method [12]

In this method, all window data are randomized as $w[j] = c^{j2^{20}+r_i}$ for $0 \le j < 2^m$ and 20-bit random number r_i. Such a short random number might be a good target by brute-force search in step 3, but this search is impossible because the attacker can not get any partial information in step 2. In this countermeasure, the table data is calculated as $w[j] = c^{j2^{20}+r_i}$ and all table data hold the same data when inputting $c = -1 \pmod{N}$. This cannot satisfy the assumption of the attack in step 2. So this countermeasure is difficult to attack.

9. Countermeasures against Our Attack when Using RMUL or RDIV

This section discusses how to resist our attack by using RMUL or RDIV countermeasure with small public exponent.

The simplest method is to avoid non-CRT decryption. If CRT decryption is used, our attack does not work because the attacker can't calculate d_H to obtain upper-half of d in step 1, d_H will be $d_H = \lfloor (1 + kp)/e \rfloor$ or $d_H = \lfloor (1 + kq)/e \rfloor$ in the CRT decryption, but this value can not be calculated without p or q. Without the information obtained in step 1, the attacker can not proceed to next steps.

If non-CRT decryption must be used, most effective protection is a message filtering shown in the presentation of [8]. That is, if $c = -1 \pmod{N}$ is input, the device returns the RSA decryption as $c^{d \pmod 2} \pmod{N}$. If the device is used for RSA only (not used as DH or DSA engine), the last bit of d is always 1. So the decryption result is calculated as c. Otherwise, combining with RMOD or RMES is also good countermeasure, but the overhead of the processing time will be larger.

Extending the bit length of random number is also effective, because it is obtained by brute-force attack in step 3. So if its bit length is large enough to be unbreakable by brute-force search, the attacker can not obtain d. When using this method, we recommend to use 80-bit or longer bit of random number. Problem of this method is the overhead of the processing time will be larger.

In case of using RDIV, our attack is avoided by adding two limitations. The first is to force the random number to be even, and the second is to calculate modular exponentiation $(c^{r_i}) \pmod{N}$ and $(c^{r_i})^{\lfloor d/r_i \rfloor} \pmod{N}$ in this order. These limitations prevent the attack in step 2, because the partial information of $\lfloor d/r_i \rfloor$ is not obtained since $c^{r_i} \pmod{N}$ is always 1.

10. Conclusion

In this paper, we have proposed a new attack RSA imple-

mentation using RMUL and RDIV countermeasure combined with m-ary method, which was supposed to be secure combination. To attack the implementation with this combination, we have proposed a collision-based power attack using only a chosen message to leak every m-th bit of the exponent on m-ary implementation. This attack is very effective even when the exponent is randomized for every execution of the modular exponentiation because of using only a single power trace. We have attempted the experiment using only a single power trace, and revealed every m-th bit of the exponent. We have also proposed technique to reveal d from these partial bits of the randomized exponent, and showed our attack can really reveal d by the experiment ran on a PC within 10 hours.

Acknowledgment

We express many thanks to Adi Shamir and anonymous referees for their valuable advices on their helpful technical and editorial comments.

SASEBO (Side-channel Attack Standard Evaluation BOard) is the Crypto-FPGA board specialized for side-channel attack experiments developed by AIST and Tohoku University in a research project funded by METI (Ministry of Economy, Trade and Industry, Japan).

References

[1] J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," Cryptographic Hardware and Embedded Systems, CHES 1999, LNCS 1717, pp.292–302, Springer-Verlag, Aug. 1999.

[2] M. Ciet and M. Joye, "(Virtually) Free randomization techniques for elliptic curve cryptography," International Conference on Information and Communications Security, ICICS 2003, LNCS 2836, pp.348–359, Springer-Verlag, Oct. 2003.

[3] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," IEEE Trans. Inf. Theory, vol.31, no.4, pp.469–472. July 1985.

[4] P.-A. Fouque and F. Valette, "The doubling attack—Why upwards is better than downwards," Cryptographic Hardware and Embedded Systems, CHES 2003, LNCS 2779, pp.269–280, Springer-Verlag, Sept. 2003.

[5] P.-A. Fouque, G. Martinet, and G. Poupard, "Attacking unbalanced RSA-CRT using SPA," Cryptographic Hardware and Embedded Systems, CHES 2003, LNCS 2779, pp.243–253, Springer-Verlag, Sept. 2003.

[6] P.-A. Fouque, S.K. Jacques, G. Martinet, F. Muller, and F. Valette, "Power attack on small RSA public exponent," Cryptographic Hardware and Embedded Systems, CHES 2006, LNCS 4249, pp.339–353, Springer-Verlag, Oct. 2006.

[7] D. Hermann and V. Harald, "Portable data carrier provided with access protection by dividing up codes," European Patent no.EP1262037, Dec. 2002.

[8] N. Homma, A. Miyamoto, T. Aoki, and A. Satoh, "SPA using a steady value input against RSA hardware implementation," Proc. Symposium on Cryptography and Information Security, SCIS 2007, Jan. 2007 (in Japanese).

[9] N. Homma, A. Miyamoto, T. Aoki, A. Satoh, and A. Shamir, "Collision-based power analysis of modular exponentiation using chosen-message pairs," Cryptographic Hardware and Embedded Systems, CHES 2008, LNCS 5154, pp.15–29, Springer-Verlag, Aug. 2008.

[10] K. Itoh, T. Izu, and M. Takenaka, "Efficient countermeasures against power analysis for elliptic curve cryptosystems," Smart Card Research and Advanced Applications, CARDIS 2004, pp.99–114, ISBN 1-4020-8146-4, Aug. 2004.

[11] K. Itoh, T. Izu, and M. Takenaka, "Improving the randomized initial point countermeasure against DPA," Applied Cryptography and Network Security, ACNS 2006, LNCS 3989, pp.459–469, Springer-Verlag, June 2006.

[12] K. Itoh, J. Yajima, M. Takenaka, and N. Torii, "DPA countermeasures by improving the window method," Cryptographic Hardware and Embedded Systems, CHES 2002, LNCS 2523, pp.303–317, Springer-Verlag, Aug. 2002.

[13] M. Joye, "Highly regular right-to-left algorithm for scalar multiplication," Cryptographic Hardware and Embedded Systems, CHES 2007, LNCS 4727, pp.135–147, Springer-Verlag, Sept. 2007.

[14] C.K. Kim, J.C. Ha., S.-H. Kim, S. Kim, S.-M. Yen, and S.J. Moon, "A secure and practical CRT-based RSA to resist side channel attacks," International Conference on Computational Science and Its Applications, ICCSA 2004, LNCS 3043, pp.150–166, Springer-Verlag, May 2004.

[15] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," International Cryptology Conference, CRYPTO 1999, LNCS 1666, pp.388–397, Springer-Verlag, Aug. 1999.

[16] H. Mamiya, A. Miyaji, and H. Morimoto, "Efficient countermeasures against RPA, DPA, and SPA," Cryptographic Hardware and Embedded Systems, CHES 2004, LNCS 3156, pp.343–356, Springer-Verlag, Aug. 2004.

[17] V. Shoup, Number Theory Library (NTL), available on http://www.shoup.net/ntl/

[18] Side-channel Attack Standard Evaluation Board (SASEBO), http://www.rcis.aist.go.jp/special/SASEBO/

[19] A. Shamir, "Method and apparatus for protecting public key schemes from timing and fault attacks," U.S.Patent Number 5,991,415, Nov. 1999; also presented at the rump session of EUROCRYPT'97.

[20] C.D. Walter, "MIST: An efficient, randomized exponentiation algorithm for resisting power analysis," Cryptographers' Track at RSA Conference, CT-RSA 2002, LNCS 2271, pp.53–66, Springer-Verlag, Feb. 2002.

[21] S.-M. Yen, W.-C. Lien, S. Moon, and J.C. Ha, "Power analysis by exploiting chosen message and internal collisions - vulnerability of checking mechanism for RSA-decryption," First International Conference on Cryptology in Malaysia, Mycrypt 2005, LNCS 3715, pp.183–195, Springer-Verlag, Sept. 2005.

Kouichi Itoh received his B.E. degree from Tokyo Institute of Technology in 1995, and M.E. degree from Japan Advanced Institute of Science and Technology in 1997, respectively. In 1997, he joined FUJITSU LABORATORIES LTD., where he has been involved in research and development of implementation of public-key and common-key cryptosystems and side channel attacks. He was awarded SCIS (Symposium on Cryptography and Information Security) paper prize in 2002, and CSS (Computer Security Symposium) paper prize in 2002.

Dai Yamamoto received the B.E. degree in information systems engineering, and M.E. degree in information networking from Osaka University, Osaka, Japan, in 2005 and 2007, respectively. He has been engaged in research and development on high-efficiency hardware architecture and cryptography at FUJITSU LABORATORIES LTD. since 2007.

Jun Yajima received his B.E. and M.E. degrees in information and system engineering from Chuo University in 1997 and 1999, respectively. Since 1999, he has been a researcher at FUJITSU LABORATORIES LTD. His current research interest includes information security and cryptography. He was awarded the SCIS2007 paper prize.

Wakaha Ogata received B.S., M.E. and D.E. degrees in electrical and electronic engineering in 1989, 1991 and 1994, respectively, from Tokyo Institute of Technology. From 1995 to 2000, she was an Assistant Professor at Himeji Institute of Technology. Since 2000, she has been an Associate Professor at Tokyo Institute of Technology. Her current interests are information security and cryptography.