New Strategy of Efficient SPA-resistant Exponentiations

Wu-Chuan Yang

Abstract

In this paper, we propose a new strategy of SPA-resistant exponentiation for RSA cryptosystems. In the previous resistant strategy, the attackers can only detect one computation type. Based on asynchronous strategy, we modify the computation so that the attacker still detect the square-only and the square-and-multiply computations in evaluating exponentiation. The goal of the modification is that the probabilities of digits in these two computations are the same; therefore the attackers do not get any information.

Keywords: RSA cryptosystems, simple power analysis (SPA), asynchronous strategy, folding exponentiations

1. Introduction

Modular exponentiation is the famous computation in cryptosystems [1-8]. Therefore, it is very important to securely and efficiently compute exponentiation. Unlike traditional cryptanalysis, side channel attacks obtain information from the physical implementation of a cryptosystem. The most investigated side channel attacks include simple power analysis (SPA), differential power analysis (DPA), timing attack (TA), and fault attack (FA) [9, 10, 11]. It is difficult to resist all possible side channel attacks, because these attacks have many approaches [12, 13, 14]. In this paper, we focus on the SPA-resistant exponentiations, and propose a new strategy to do secure and efficient SPA-resistant exponentiation.

The famous algorithms for SPA-resistant exponentiations include dummy computations [15], Montgomery powering ladder [16], and side channel atomicity [17]. Suppose **S** denotes the computation cost of squaring, and **M** denotes the cost of multiplication. Dummy computation is a straightforward method in which we insert a dummy computation in the classical square-and-multiply algorithm to balance the power bias with the computation cost increasing from $n\mathbf{S}+0.5n\mathbf{M}$ to $n\mathbf{S}+n\mathbf{M}$. The if-else decision is removed from the Montgomery powering ladder algorithm to avoid the fault attack in addition [18]. However, the computation cost also requires $n\mathbf{S}+n\mathbf{M}$. The main idea of side channel atomicity is to separate the computation in all possible digits into several equivalent parts to avoid SPA with a $1.5n\mathbf{M}$ computation cost.

In this paper, we proposed new strategy of SPA-resistant exponentiations. The attacker can detect the square-only and square-and-multiply computations; if the probabilities of all possible digits are all equivalent in all computations, no secret information is leaked. This technology can be achieved by modifying *asynchronous strategy* [20,21] and *folding exponentiations* [22,23,24,25].

2. The Previous Strategy of SPA-Resistant Exponentiations

The classical square-and-multiply algorithm is shown in *Algorithm* 1. The 0 and 1 can be detected by identifying the square-only (**SO**) computations and the square-and-multiply (**SM**) computations. The attacker gets d_i=0 for detecting **SO** and d_i=1 for detecting **SM** as shown in Figure 1.

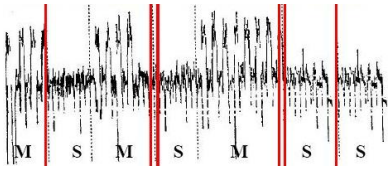

Figure 1: Possible SPA detection in *Algorithm* 1

For example, the attacker finds d=57(111001_2) as shown in Table 1. The digit probabilities of the computations are in Table 2, where shows that *Algorithm* 1 is vulnerable to SPA.

*Corresponding Author: Wu-Chuan Yang
(E-mail: wcyang@isu.edu.tw)
[1]Department of Information Engineering, I-Shou University,No.1, Sec. 1, Syuecheng Rd., Dashu District, Kaohsiung City 84001,Taiwan

Algorithm 1 Classical square-and-multiply algorithm

I/P: $x,d=(d_n,d_{n-1},\cdots,d_0)_2$
O/P: $z=x^d$
1. z=1;
2. **for** (i from $n-1$ to 0) {
3. $z=z^2$;
4. **if** ($d_i\neq 0$) $z=z\cdot x$;
5. }

Table 1: A SPA example in *Algorithm* 1

	initial	1	1	1	0	0	1
y	1	1	x^2	x^6	x^{14}	x^{28}	x^{56}
		x	x^3	x^7			x^{57}
detect		**SM**	**SM**	**SM**	**SO**	**SO**	**SM**
d_i		1	1	1	0	0	1

Table: 2 Probabilities of digits in *Algorithm* 1

Detection	**SO**	**SM**
0	100%	0
1	0	100%

A simple SPA-resistant exponentiation inserts a dummy multiplication in the if-else decision. A good dummy-multiplication version is shown in *Algorithm* 2 in which the if-else decision is eliminated [15, 16]. Because of only one computation (**SM**) in the for-loop, the attacker does not get d_i by SPA. The computation cost of *Algorithm* 2 is n**S**+n**M**. An example of computing $y = x^{57}$ ($57=111001_2$) is illustrated in Table 3.

Algorithm 2 Dummy-multiplication exponentiation

I/P: $x, d = (d_n, d_{n-1}, \cdots, d_0)_2$

O/P: $z = x^d$

1. $z_0 = 1, z_1 = 1$;
2. **for** (i from $n-1$ to 0) {
3. $z_1 = z_1^2$;
4. $z_{d_i} = z_{d_i} \cdot x$;
5. }
6. $z = z_1$;

Table 3: different detections in *Algorithm* 2

	initial	1	1	1	0	0	1
z_0	1				x	x^2	
z_1	1	1	x^2	x^6	x^{14}	x^{28}	x^{56}
		x	x^3	x^7			x^{57}
detect		**SM**	**SM**	**SM**	**SM**	**SM**	**SM**

In *Algorithm* 1, if we detect the computation is **SO**, the relative bits are zeros; and if the detected computation is **SM**, the relative bits are ones as shown in Figure 2. In *Algorithm* 2, if d_i=0 we do a dummy multiplication in line 4; therefore the detected computations are all **SM**, and the relative bits are 50% ones and 50% zeros as shown in Figure 3. Although *Algorithm* 2 is SPA-resistant, its performance is reduced.

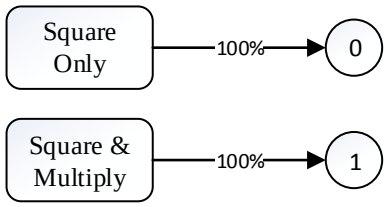

Figure 2: Computation and its relative bits in *Algorithm* 1

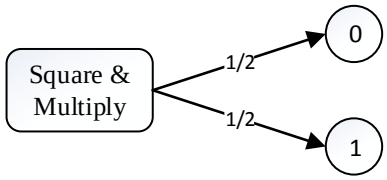

Figure 3: Computation and its relative bits in *Algorithm* 2

Many SPA-resistant algorithms of exponentiation, include the above dummy computation, and the Montgomery powering ladder [16] and the side channel atomicity [17] are integrating all possible computations into one with equal computation cost. Many efficient algorithms to evaluate exponentiation cannot be used. In the next section, we propose a totally new SPA-resistant strategy: many computations (in this paper, it is two) can be detected, so the probabilities of the digits in all possible computations are all the same to provide the SPA-resistant effect.

3. The Proposed Strategy

The proposed strategy is based on asynchronous strategy and folding exponentiation. The folding exponentiation uses the concept of multi-exponentiation [26], so we can fold the exponent by half, which then evaluates the result as shown in *Algorithm* 3. Note that all possible values of $u^{a_i}v^{b_i}$, including $u, v,$ and uv, should be precomputed and stored. The probability of $(a_i, b_i) \neq (0,0)$ is about 0.75; thus the average computation is $(0.5n + 0.5n)\boldsymbol{S} + \frac{0.75n}{2}\boldsymbol{M} = n\boldsymbol{S} + 0.375n\boldsymbol{M}$. Table 4 shows an example of *Algorithm* 3. In the example, we must precompute and store x^8 and x^9; $57=(111001)_2$ can be separated to 3 pairs (1,0), (1,0), (1,1).

Algorithm 3 --- Basic folding exponentiation

I/P: $x, d = (d_n, d_{n-1}, \cdots, d_0)_2$
O/P: $z = x^d$

1. $z = 1$;
2. $y = x^{\frac{n}{2}}$;
3. Set $\begin{cases} a = \left(d_{\frac{n}{2}-1}, d_{\frac{n}{2}-2}, \cdots, d_1, d_0\right)_2 \\ b = \left(d_{n-1}, d_{n-2}, \cdots, d_{\frac{n}{2}+1}, d_{\frac{n}{2}}\right)_2 \end{cases}$
4. **for** $(i$ from $\frac{n}{2} - 1$ to $0)$ {
5. $z = z^2$;
6. **if** $(a_i, b_i) \neq (0,0)$ { $z = z \cdot (x^{a_i} y^{b_i})$;}
7. **}**

Table 4: An example of Algorithm 3

	Precomputation					(1,0)	(1,0)	(1,1)
	x (=u)	x^2	x^4	x^8 (=v)	x^9 (=uv)			
y						1	x^{16}	x^{48}
						x^8	x^{24}	x^{57}
detect		**S**	**S**	**S**	**M**	**SM**	**SM**	**SM**

In Algorithm 3, if we detect the computation is **SO**, the relative bits are zero-pairs, 00; and if the detected computation is **SM**, the relative bits are nonzero-pairs, 01, 11, and 10, as shown in Figure 4. It is advantage to resist SPA. However the attacker can get some information of the secret exponent d by detecting the computation.

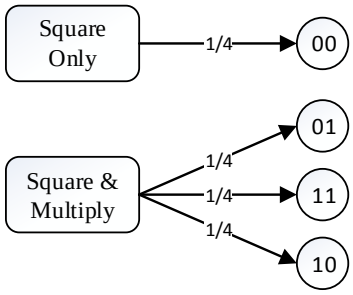

Figure 4: Computation and its relative bits in *Algorithm* 3

In 2007, Yang, Guan, and Laih proposed an efficient strategy for multi-computations, e.g. multi-exponentiation and multi-scalar multiplication. They also proposed a series of algorithms, which improve the performance of multi-computation [20]. The concept of an asynchronous strategy is to compute partial results by choosing nearby digits which are both zeros and both non-zeros. This can be described by shifting digits in a and b. For example, let $a = 010101011$ and $b = 101010101$, the joint Hamming distance (the number of nonzero columns) $\omega(a, b)$=9.

$$\begin{cases} a = 0\,1\,0\,1\,0\,1\,0\,1\,1 \\ b = 1\,0\,1\,0\,1\,0\,1\,0\,1 \end{cases} \rightarrow \omega(a, b) = 9$$

We can match more zeros by left-shifting a one-bit to reduce the join Hamming weight to 5 as follows.

$$\begin{cases} a = 0\,1\,0\,1\,0\,1\,0\,1 \quad 1 \\ b = \quad 1\,0\,1\,0\,1\,0\,1\,0\,1 \end{cases} \rightarrow \omega(a, b) = 5$$

In the above observation, the computational cost can be reduced by shifting a one-bit to the left. In order to get the correct computation, we must modify the corresponding computations to obtain correct result: the first is the rules to determine the position of the blocks to be shifted, and the second is the relative computations for shifted blocks.

We denote the computation with the same weight digit-pair (a_i, b_i), the normal state S_o, the computation with different weight digit-pair (a_i, b_j) with $i \neq j$, the shift state S_x. A simple computing sequence of the asynchronous strategy is shown in Figure 5.

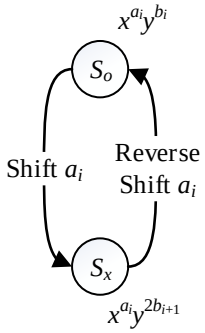

Figure 5: A simple example of the asynchronous strategy

For example, the computing sequence of the following pair (a, b)

$$\begin{cases} a = 0\,1\,0\,1\,0\,1\,0\,1 \quad 1\,(171) \\ b = \quad 1\,0\,1\,0\,1\,0\,1\,0\,1\,(341) \end{cases}.$$

The computing sequence of $x^a y^b$ is

$$1 \rightarrow xy^2 \rightarrow x^2y^4 \rightarrow x^5y^{10} \rightarrow x^{10}y^{20} \rightarrow x^{21}y^{42} \rightarrow x^{42}y^{84} \rightarrow x^{85}y^{170} \rightarrow x^{171}y^{341}$$

In order to improve performance, the rule of state transformation should be simple, and the computation can reduce the number of multiplication as possible. Therefore, a good corresponding computation of the state transform should be modified as shown in Table 5.

Table 5: Corresponding computation of *Algorithm* 4

State	Computation	Condition
$S_o \to S_o$	$z = z^2 \cdot (x^{a_i} y^{b_i})$	$a_i = b_i$
$S_o \to S_x$	$z = z^2 \cdot (x^{a_i})$	$a_i \neq b_i$
$S_x \to S_x$	$z = z^2 \cdot (x^{a_i} y^{2b_{i+1}})$	$\sim(a_i = b_i \neq b_{i+1})$
$S_x \to S_o$	$z = (z \cdot y^{b_{i+1}})^2 \cdot (x^{a_i} y^{b_i})$	$a_i = b_i \neq b_{i+1}$

Note that we must pre-process the relative exponent bit, a_i, when the exponent a is shifted. The relative exponent bits, a_i, b_i, b_{i+1} must be processed when the reverse-shift is done. As illustrated above, the basic algorithm of asynchronous strategy is illustrated in *Algorithm* 4. For simplicity the instruction $z = z^2 \cdot t^{e_i}$ (note that e_i is 0 or 1) means

if $(e_i \neq 0)$ { $z = z^2 \cdot t$; }
else { $z = z^2$; }

Algorithm 4 Basic algorithm of the asynchronous strategy

I/P: $x, y, a = (a_n, a_{n-1}, \cdots, a_0)_2, a = (b_n, b_{n-1}, \cdots, b_0)_2$
O/P: $z = x^a y^b$

1. $y = 1, s = S_o$;
2. $u = xy, v = xy^2$;
3. **for** (i from $n-1$ to 0) {
4. **if** $(s = S_o)$ {
5. **if** $(a_i = b_i)$ { $z = z^2 \cdot u^{a_i}$; }
6. **else** { $s = S_x, z = z^2 \cdot x^{a_i}$; }
7. }
8. **else** {
9. **if** $(a_i = b_i$ **and** $b_i \neq b_{i+1})$ { $s = S_o, z = (z \cdot y^{b_{i+1}})^2 \cdot u^{a_i}$; }
10. **else** { $z = z^2 \cdot v^{a_i}$; }
11. }
12. }
13. **if** $(s = S_x)$ { $z = z \cdot y^{b_0}$; }

Detail of the asynchronous strategy is described in [20, 21]. Due to the property of asynchronous weight digits in computing sequence, the asynchronous strategy has also advantage to resist SPA. In evaluating single exponentiation, we can combine the strategy and fold exponentiation, so we can do exponentiation by *Algorithm* 5.

Algorithm 5 The proposed strategy

I/P: $x = (x_n, x_{n-1}, \cdots, x_0)_2, d = (d_n, d_{n-1}, \cdots, d_0)_2$
O/P: $z = x^d$

1. $z = 1, s = S_o$;
2. $y = x^{\frac{n}{2}}, u = x^{\frac{n}{2}+1}, v = x^{n+1}$;
3. $\begin{cases} a = \left(a_{\frac{n}{2}-1}, a_{\frac{n}{2}-2}, \cdots, a_0\right) = \left(d_{\frac{n}{2}-1}, d_{\frac{n}{2}-2}, \cdots, d_0\right)_2 \\ b = \left(b_{\frac{n}{2}-1}, b_{\frac{n}{2}-2}, \cdots, b_0\right) = \left(d_{n-1}, d_{n-2}, \cdots, d_{\frac{n}{2}}\right)_2 \end{cases}$
4. **for** (i from $\frac{n}{2}-1$ to 0) {
5. **if** $(s = S_o)$ {
6. **if** $(a_i = b_i)$ { $z = z^2 \cdot u^{a_i}$ };
7. **else** { $s = S_x,\ z = z^2 \cdot x^{a_i}$};
8. }
9. **else** {
10. **if** $(a_i = b_i$ **and** $b_i \neq b_{i+1})$ { $s = S_o, z = (z \cdot y^{b_{i+1}})^2 \cdot u^{a_i}$; }
11. **else** { $z = z^2 \cdot z^{a_i}$; }
12. }
13. }
14. **if** $(s = S_x)$ { $z = z \cdot y^{b_0}$; }

The corresponding computation of *Algorithm* 5 is similar to *Algorithm* 4 as shown in Table 6.

Table 6: Corresponding computation of *Algorithm* 5

State	Computation	Condition
$S_o \to S_o$	$z = z^2 \cdot x^{\left(\frac{n}{2}+1\right)d_i}$	$d_i = d_{\frac{n}{2}+i}$
$S_o \to S_x$	$z = z^2 \cdot x^{a_i}$	$d_i \neq d_{\frac{n}{2}+i}$
$S_x \to S_x$	$z = z^2 \cdot x^{(n+1)d_i}$	$\sim(d_i = d_{\frac{n}{2}+i} \neq d_{\frac{n}{2}+i+1})$
$S_x \to S_o$	$z = \left(z \cdot x^{\frac{n}{2}b_{i+1}}\right)^2 \cdot x^{\left(\frac{n}{2}+1\right)d_i}$	$d_i = d_{\frac{n}{2}+i} \neq d_{\frac{n}{2}+i+1}$

4. Performance and Security Analysis

In *Algorithm* 3, the main computations are in line 5, 6. If the attacker detects the computation is **SO**, he can get $(a_i, b_i) = (0,0)$, that is $\left(d_i, d_{i+\frac{n}{2}}\right) = (0,0)$. If the attacker detects the computation is **SM**, $(a_i, b_i) \neq (0,0)$ is known, that is $\left(d_i, d_{i+\frac{n}{2}}\right) = (0,1)$ *or* $(1,0)$ or $(1,1)$. The attacker can get the information: $P_{d_i=0} = \frac{1}{3}$, $P_{d_i=1} = \frac{2}{3}$, $P_{d_{i+\frac{n}{2}}=0} = \frac{1}{3}$, and $P_{d_{i+\frac{n}{2}}=1} = \frac{2}{3}$, but he cannot make sure of the value of d_i and $d_{i+\frac{n}{2}}$. However, *Algorithm* 3 is weak to SPA, so about $\frac{1}{4}n$ bits will be detected, and some extra information will be leaked.

The analysis of *Algorithm* 5 is more complex; the main computations are in line 6, 7, 10, and 11 as shown in Table 7. We must find the probability of each state and computation in Table 7. In performance analysis, Lemma 1 shows the probability of each state.

Table 7: Analysis of *Algorithm* 5

P.S.	N.S.	$b_{i+1}a_ib_i$	computation	Probability
S_o	S_o	x 0 0	**SO**	$\frac{1}{3}\cdot\frac{1}{4}=\frac{1}{12}$
		x 1 1	**SM**	$\frac{1}{3}\cdot\frac{1}{4}=\frac{1}{12}$
	S_x	x 0 1	**SO**	$\frac{1}{3}\cdot\frac{1}{4}=\frac{1}{12}$
		x 1 0	**SM**	$\frac{1}{3}\cdot\frac{1}{4}=\frac{1}{12}$
S_x	S_o	0 1 1	**SM**	$\frac{2}{3}\cdot\frac{1}{8}=\frac{1}{12}$
		1 0 0	**SM**	$\frac{2}{3}\cdot\frac{1}{8}=\frac{1}{12}$
	S_x	0 0 0	**SO**	$\frac{2}{3}\cdot\frac{1}{8}=\frac{1}{12}$
		0 0 1	**SO**	$\frac{2}{3}\cdot\frac{1}{8}=\frac{1}{12}$
		0 1 0	**SM**	$\frac{2}{3}\cdot\frac{1}{8}=\frac{1}{12}$
		1 0 1	**SM**	$\frac{2}{3}\cdot\frac{1}{8}=\frac{1}{12}$
		1 1 0	**SM**	$\frac{2}{3}\cdot\frac{1}{8}=\frac{1}{12}$
		1 1 1	**SM**	$\frac{2}{3}\cdot\frac{1}{8}=\frac{1}{12}$

Lemma 1: In *Algorithm* 4, $P_{s=S_o}=\frac{1}{3}$ and $P_{s=S_x}=\frac{2}{3}$.

Proof: Suppose $P_{s=S_o}=p$, and $P_{s=S_x}=1-p$.

We know $P_{x_i=0}=P_{x_i=1}=\frac{1}{2}$.

The following case will cause the next state to S_o:

(1) PS=S_o, and $(x_i,y_i)=(00)$ or (11).

(2) PS=S_x, and $(y_{i+1}x_iy_i)=(100)\ or\ (011)$.

Thus $p=\frac{1+1}{4}p+\frac{1+1}{8}(1-p)\rightarrow p=\frac{1}{3}$.

That is $P_{S_o}=\frac{1}{3}$ and $P_{S_x}=\frac{2}{3}$.

Lemma 1 is useful to analyze the security and performance of *Algorithm* 5. Because of uniform distribution of the bits, we can derive the average number of multiplications, $0.333n$ in Theorem 1.

Theorem 1: The average number of multiplications in *Algorithm* 6 is $\frac{1}{3}n\approx 0.333n$.

Proof: According to Lemma 1, we can get the average number of multiplication by

$$\sum_{\text{Computation=SM}} P_{s=State}\times P_{x_iy_i}$$
$$=\frac{n}{2}\left(\frac{1}{3}\cdot\frac{1}{4}\cdot(1+1)+\frac{2}{3}\cdot\frac{1}{8}\cdot(1+1+1+1+2)\right)=$$
$$\frac{1}{3}n\approx 0.333n$$

In security analysis, Theorem 2 shows when **SO** or **SM** is detected, the probabilities of $(a_i,b_i)=(0,0)$ or $(a_i,b_i)\neq(0,0)$ are all $\frac{1}{2}$. That is the attacker cannot find any information of (a_i,b_i). This is the ideal result that the performance of *Algorithm* 5 is better than *Algorithm* 3; and each probabilities of all pair are all equal in each computation.

Theorem 2: In *Algorithm* 4, the digit probabilities of the detection is

$P_{(a_i,b_i)=(0,0)|\text{detection=SO}}=\frac{1}{2}$, $P_{(a_i,b_i)\neq(0,0)|\text{detection=SO}}=\frac{1}{2}$,
$P_{(a_i,b_i)=(0,0)|\text{detection=SM}}=\frac{1}{8}$, $P_{(a_i,b_i)\neq(0,0)|\text{detection=SM}}=\frac{7}{8}$.

Proof: According to Table 7, we can easily get the result.

If the detected computation of *Algorithm* 5 is **SO**, the relative bits is 00 and 01, that is we can derive $d_{\frac{n}{2}+i}$ is 0. If the detected computation of *Algorithm* 5 is **SM**, the relative bits is 00, 01, 11 and 10, that is we cannot derive the content of the relative bits.

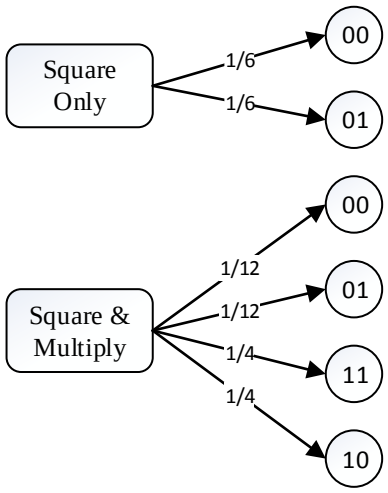

Figure 6: Computation and its relative bits in *Algorithm* 5

Because original asynchronous strategy focuses on performance improvement, the bit probability of SM detected is not equivalent, that is some information will be leaked by directly using *Algorithm* 5. In order to get secure computation by equivalent bit probability of each detected computation, we must modify the rules of the state decision and the modified computation sequence in asynchronous strategy.

5. Conclusions

In this paper, we mainly proposed the concept of balancing digit probability. Even the different computations are detected; if the probabilities of the bits in the detected computations are equivalent, no information is leaked. However we do not propose an ideal algorithm for the proposed strategy. Based on folding exponentiation and asynchronous strategy, the computation cost is $n\boldsymbol{S} + 0.333n\boldsymbol{M}$ and the bit detected rate is 12.5% in the proposed algorithm (*Algorithm* 5). In comparison with the classical square-and-multiply exponentiation (*Algorithm* 1), the cost is $n\boldsymbol{S} + \frac{1}{2}n\boldsymbol{M}$ and the bit detected is 100%; for the dummy-multiplication exponentiation algorithm (*Algorithm* 2), the cost is $n\boldsymbol{S} + n\boldsymbol{M}$ and the bit detected is 0%; and directly using folding exponentiation (*Algorithm* 3), the cost is $n\boldsymbol{S} + \frac{3}{8}n\boldsymbol{M}$ and the bit detected is 25%. All the computation cost and bit detected rate are shown in Table 8. *Algorithm* 4 is used to evaluate multi-exponentiation, so it does not be considered in Table 8.

Table 8: Comparison of single exponentiation algorithms

Algorithm	Computation Cost ($\boldsymbol{x^d}$)	Bit detected rate
Algorithm 1	$n\boldsymbol{S} + 0.500n\boldsymbol{M}$	100%
Algorithm 2	$n\boldsymbol{S} + 1.000n\boldsymbol{M}$	0%
Algorithm 3	$n\boldsymbol{S} + 0.375n\boldsymbol{M}$	25%
Algorithm 5	$n\boldsymbol{S} + 0.333n\boldsymbol{M}$	12.5%

In order to get better performance and security simultaneously, we use the strategy that compute the asynchronous weight of exponent in this paper. Based on the strategy, we proposed an efficient algorithm, whose average computation cost is $n\boldsymbol{S} + 0.333n\boldsymbol{M}$ **,** and 12.5% of bits will be detected in the simple power analysis. In addition, some extra information will be leaked in detected computation square-and-multiply. That is the proposed algorithm is only a compromise prototype, so it can be improved by adjusting the transformation rule and computing sequence in asynchronous strategy. Based on the strategy, more secure and efficient SPA-resistant algorithms can be implemented in future.

References

[1]. R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Communication of ACM, vol. 21, pp. 120 126, 1978.

[2]. T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," IEEE Transactions on Information Theory, vol. 31, no. 4, pp. 469-472, Jul. 1985.

[3]. FIPS186-2, "Digital signature standard (DSS)," http://csrc.nist.gov/ publications/fips/, 2001.

[4]. D. E. Knuth, The Art of Computer Programming, vol 2. Seminumerical algorithms, Addi-son-Wesley, 1969, 2nd edition 1982, 3rd edition 1998.

[5]. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, Handbook of Applied Cryptography, CRC Press, 1997.

[6]. Ç. K. Koç, "High-speed RSA implementations," RSA Laboratories, Technique Notes TR201, Available from URL: http://www.rsasecurity.com/ rsalabs/, pp. 9-32, Nov. 1994.

[7]. D. M. Gordon, "A survey of fast exponentiation methods," Journal of Algorithms, vol. 27, pp. 129-146, 1998.

[8]. B. Möller, "Algorithms for multi-exponentiations," 8th Annual Workshop on Selected Areas in Cryptography -SAC 2001, LNCS 2259, Springer-Verlag, pp. 165-180, 2001.

[9]. P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," Advances in Cryptology - CRYPTO'96, LNCS 1109, Springer- Verlag, pp. 104-113. 1996.

[10]. D. Boneh, R. A. DeMillo, and R.J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," Advances in Cryptology - EUROCRYPT '97, LNCS 1233, Springer-Verlag, pp. 37-51, 1997.

[11]. P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," Advances in Cryptology - CRYPTO '99, LNCS 1666, Springer- Verlag, pp. 388-397, 1999.

[12]. S. M. Yen and M. Joye, "Checking Before Output May Not be Enough against Fault-Based Cryptanalysis," IEEE Trans. on Computers, vol. 49, no. 9, pp.967-970, Sept. 2000.

[13]. S. M. Yen, S. J. Kim, S. G. Lim and S. J. Moon, "A countermeasure against one physical cryptanalysis may benefit another attack," Information Security and Cryptology - ICISC '01, LNCS 2288, Springer-Verlag, pp. 414-427, 2002.

[14]. C. Giraud, "An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis," IEEE Trans. Computers, vol. 55, no. 9, pp. 1116-1120, Sep. 2006.

[15]. J. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," Cryptographic Hardware and Embedded Systems - CHES'99, LNCS 1717, Springer-Verlag, pp. 292-302, 1999.

[16]. M. Joye and S. M. Yen, "The Montgomery Powering Ladder," Cryptographic Hardware and Embedded Systems - CHES'2002, LNCS 2523, Springer-Verlag, pp. 291-302, 2003.
[17]. B. Chevallier-Mames, Mathieu Ciet, and Marc Joye, "Low-Cost Solutions for Preventing Simple Side-Channel Analysis：Side-Channel Atomicity," IEEE Trans. on Computers, vol. 53, no. 6, pp. 760-768, June 2004.
[18]. S. M. Yen, S. J. Kim, S. G. Lim and S. J. Moon, "RSA Speedup with Chinese Remainder Theorem Immune against Hardwarw Fault Cryptanalysis," IEEE Trans. on Computers, vol. 52, no. 4, pp. 461-472, Apr. 2003.
[19]. W. C. Yang, P. Y. Hsieh, and C. S. Laih, "Efficient Squaring of Large Integers," IEICE Transactions on Fundamentals, vol. E87-A, no. 5, pp. 1189-1192, May. 2004.
[20]. W. C. Yang, D. J. Guan, and C. S. Laih, "Fast Multi-Computations with Asynchronous Strategy," IEEE Trans. on Computers, vol. 56, no. 2, pp. 234-242, Feb. 2007.
[21]. W. C. Yang, "The Study of Multi-computation in Public Key Cryptography," Ph. D. dissertation of National Cheng Kung University, Jan. 2005.
[22]. D. C. Lou and C. C. Chang, "Fast exponentiation method obtained by folding the exponent in half," Electronics Letters, vol. 32, Issue 11, pp. 984-985, May 1996.
[23]. E. F. Brickelland, D. M. Gordon, K. S. McCurley, and D. Wilson, "Fast exponentiation with precomputation," Advances in Cryptology-Eurocrypt'92, LNCS 658, Springer-Verlag, pp. 200-207, 1992.
[24]. C. H. Lim and P. J. Lee, "More flexible exponentiation with precomputation," Advances in Cryptology-Crypto'94, LNCS 839, Springer-Verlag, pp. 95-107, 1994.
[25]. W. C. Yang, "New Strategy of Efficient SPA-resistant Exponentiations," The Fifth International Conference on Information Assurance and Security (IAS 2009), pp.348-351, 2009.08.
[26]. S. M. Yen, C. S. Laih, and A. K. Lenstra, "Multi-exponentiation," IEE Proceedings, Computers and Digital Techniques, vol. 141, no. 6, pp. 325-326, 1994.

W u-Chuan Yang received his BS, MS, and Ph.D. degree all in Electrical Engineering from National Cheng Kung University, Tainan, Taiwan, in 1988, 1991, and 2006, respectively. From 1991 to 2005, he was with the Department of Electronic Engineering at Nan-Jeon Junior College of Technique and Commerce. In 2003, he obtained 2003 Annual Best Paper Award of JISE(Journal of Information Science and Engineering). From 2005 to 2007, he was with the Department of Technology Management at Aletheia University. Since 2007, he is an associate professor in the Department of Information Engineering at I-Shou University, located in Kaohsiung, Taiwan. His research interests include cryptography, algorithm, information security and software engineering.