Two Exponentiation Algorithms Resistant to Cross-correlation Power Analysis and to Other Known Attacks

Yaacov Belenky, Zeev Geyzel, Michael Kara-Ivanov, and Avraham Entelis

NDS Technologies Israel Ltd., Shlomo haLevi Str. 5, Jerusalem, Israel

Abstract. In order to prevent the SPA (Simple Power Analysis) attack against modular exponentiation algorithms, a multiply-always implementation is generally used. Witteman et al. introduced in [14] a new cross-correlation power analysis attack against the multiply-always implementation. We suggest two new algorithms, resistant to this attack and also to other known attacks.
The first algorithm is an alternative approach to exponentiation algorithms used in cryptography, which usually receive as an input some representation (e.g. binary) of the exponent. In our approach both the exponent and the result are functions (not necessarily easily invertible) of the exponentiation algorithm input. We show that this approach can have a good performance and that it is also resistant to several known attacks, especially to the cross-correlation power analysis and also to the attack described in [9]. It is particularly relevant for cryptographic schemes in which the private exponent can be chosen arbitrarily.
Another exponentiation algorithm that we present here may be preferable for use with RSA in certain settings. It is resistant to the cross-correlation power analysis attack, C safe error attack, and other attacks; although it involves squaring operations.

Keywords: Exponentiation algorithms, addition chains, cross-correlation attacks, safe-error attacks, power analysis attacks

1 Introduction

Fast algorithms to calculate x^d, where d is a very big (e.g. 2048 bits long) integer, have wide application and have been an object of study for a long time. In cryptographic applications d is usually secret, and the resistance of an algorithm against various attacks becomes another, not less important than performance, factor in the choice of an algorithm.

Optimal performance of exponentiation algorithms has been intensively studied under the name of "addition chains" (see [11] p. 465-485 for an overview). The main problem to be solved is: given an integer d, find a way to calculate x^d that involves minimal possible number of multiplications. A generalization of this problem is: given a finite set of integers $d_1, d_2, \ldots, d_n$, find a way to calculate all of $x^{d_1}, x^{d_2}, \ldots, x^{d_n}$ that involves the minimal possible number of multiplications. The latter generalized problem has been proven to be NP-complete [5]. For the first problem, no general polynomial time algorithm that finds an optimal sequence of multiplications is known.

Things are obviously much easier if the goal is to build an efficient addition chain without having to worry about what the final result will be. Fortunately, in most cryptographic algorithms there are no special requirements for the private exponent, except for high enough entropy. On the other hand, the protection from side-channel attacks (like SPA or DPA) is very important in the cryptographic applications.

The multiple always countermeasure (see [4]) is an important protection from both SPA and some forms of DPA. Witteman et al. [14] used the observation that different stages of multiple-always algorithm can be identified by cross-correlation power analysis. They noted that the power consumptions of operations with one common operand, is more correlated than the power consumptions of the same operations with different operands.

In this article we analyze two algorithms resistant to this kind of attacks. In the first algorithm we use the above mentioned approach, where both the exponent and the result are functions (not necessarily easily invertible) of the exponentiation algorithm input.

The RSA algorithm is a remarkable exception, since the private exponent cannot be chosen arbitrarily. In particular, if the public exponent is a fixed number, e.g . $2^{16}+1$ (which is a common practice), then the private exponent is a function of the prime factors of the modulus, and therefore cannot be produced as an output of another function that we do not know how to efficiently invert. For these cases, we suggest a modification of a different exponentiation algorithm. It is based on the classical right-to-left exponentiation algorithm and has the same performance (2 multiplications per bit of the exponent) as algorithms described in [10], but is additionally secure against Bellcore attacks [1]. It should be noted, however, that if the performance of private key operations is more critical than the performance of public key operations, then it is still possible to use the first approach that gives significantly better performance.

The rest of the paper is organized as follows. In Section 2 we introduce addition chains, and we also present our squaring-free algorithm. In Section 3 we present our new algorithm intended for RSA. Section 4 contains our conclusions.

2 Efficient Squaring-Free Addition Chain Algorithm

Firstly, we define the addition chains:

Definition 1. *A sequence of natural numbers $x_0, x_1, \dots x_n$ is called an* ***addition chain*** *if $\forall i \in [0, n] : (x_i = 0 \vee x_i = 1 \vee \exists j, k : (j \le k < i \wedge x_i = x_j + x_k))$.*

This is a slight modification of the definition given in [11]. See there also how the addition chain theory is used in the fast exponentiation problem.

We suggest an algorithm, which is a kind of the addition chain exponentiation scheme. It uses only two registers for the storage of intermediate results. Moreover, in order to defend against cross-correlation attacks (like [14]), we want to avoid squaring operations. Thus, we call our algorithm **SFTS** (squaring-free two-slot). In fact, SFTS is also known as euclidean addition chain algorithm. Let U be an arbitrary set, and $\circ : U^2 \to U$ be an associative binary operation on this set. We use multiplicative notation, and write a^n instead of $\overbrace{a \circ a \circ \dots \circ a}^{n \text{ times}}$. All the results are equally applicable to the elliptic curves cryptography, where additional notation is traditionally used. We also assume that there exists a neutral element 1 such that $\forall a : 1 \circ a = a \circ 1 = a$. This assumption holds for operations used in cryptography, such as multiplication in finite fields and addition of points on an elliptic curve over a finite field. However this assumption is not essential, and all the algorithms discussed below can be easily modified if it does not hold. The algorithm can be written in the following form:

SFTS Exponentiation Algorithm

Inputs:

$x \in U$;

$n \in \mathbb{Z}$, $n \geq 2$;

$h(j) = 0, 1$ for $j \in \mathbb{Z}$, $0 \leq j \leq n-2$.

```
1. $r_0 \leftarrow r_1 \leftarrow x$
2. for $j = 0$ to $n-2$ do
3.     $r_{h(j)} \leftarrow r_0 \circ r_1$
4. end for
5. return $r_0 \circ r_1$
```

In other words, in a SFTS exponentiation algorithm are only two registers r_0 and r_1, both initialized with x, and at each iteration $r_0 \circ r_1$ is calculated and stored to $r_{h(i)}$.

Definition 2. *If $b = \langle b_0, b_1, \ldots, b_{r-1} \rangle$ is a sequence of positive integers, then its **derived index sequence** $H(b)$ is the bit sequence $H(b) = 1^{b_0} 0^{b_1} 1^{b_2} \ldots$. Sequence b is called the **generating sequence** of SFTS addition chain.*

Let $A(b) = \langle A_0(b), A_1(b), \ldots, A_{n-1}(b) \rangle$ be the sequence of results of the execution loop in the SFTS algorithm. We will denote the return value $A_{n-1}(b)$ also as $R(b)$. It is obvious, that flipping of all h bits (permutation between r_0 and r_1) does not affect $A(b)$ and $R(b)$, so without losing of any generality we can assume that $h_0 = 1$.

The following two propositions give equivalent alternative expressions for $A(b)$ and $R(b)$.

Proposition 1. *If $b = \langle b_0, b_1, \ldots, b_{r-1} \rangle$ is a generating sequence, and for any $i \in [0, r-1]$ the continued fraction W_i is defined as*

$$W_i = b_i + \cfrac{1}{b_{i-1} + \cfrac{1}{b_{i-2} + \ldots + \cfrac{1}{b_0 + 1}}} \tag{1}$$

then for any $i \in [0, r-1]$ $\quad A_{\sum_{j=0}^{i} b_j}(b)$ is the numerator of W_i, represented as an irreducible fraction, and $R(b)$ is the sum of the numerator and denominator of W_{r-1} represented as an irreducible fraction.

Proposition 2. *If $b = \langle b_0, b_1, \ldots, b_{r-1} \rangle$ is a generating sequence, then for any $i \in [0, r-1]$*

$$A_{\sum_{j=0}^{i} b_j}(b) = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} b_{i-1} & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} b_{i-2} & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} b_0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{2}$$

and

$$R(b) = \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} b_{r-1} & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} b_{r-2} & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} b_0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{3}$$

Given an n-bit number d, it is possible to choose an arbitrary $c \in [\frac{d}{2}, d]$ relatively prime with d, to represent $\frac{d}{c}$ as a continued fraction

$$\frac{d}{c} = 1 + \cfrac{1}{b_{r-1} + \cfrac{1}{b_{r-2} + \ldots + \cfrac{1}{b_0 + 1}}} \tag{4}$$

and to use the SFTS Exponentiation Algorithm with x and b as inputs in order to calculate x^d. This was suggested in [13] and [3] in the context of elliptic curve cryptography. However for an arbitrary choice of c the resulting algorithm may be inefficient. If c is chosen randomly, it is shown in [12] that the average number of multiplications is $6\pi^{-2}(\ln n)^2 + O(\log n(\log\log n)^2)$. For $n = 160$ the main term is ≈ 7700. Experiments give an average of about 2500 multiplications total, or ≈ 15.6 (sic!) multiplications per bit. In order to improve the performance, in [13] it is suggested to search for c around d/ϕ, where ϕ is the golden ratio, so that the generating sequence starts from many 1's. The following table ([13]) gives average and worst number of choices of c necessary in order to achieve different performance values for 160-bit integers.

multiplications per bit	2	1.875	1.75	1.6875	1.625
on average	29	121	2,353	46.454	7,795,840
worst case	521	3,454	44,254	1,554,011	79,402,210

Another method (GRASC) and its modification (GRAC) to shorten the chain have been suggested in [7] and [8]. These methods achieve better performance than in [13] and [3], but require significantly more memory (up to 26 registers), and are more complicated in hardware implementation.

However, if any value of d of predefined bit length is acceptable, then it is possible to use sequences b that require just two registers, provide significantly better performance and do not require precalculations. We suggest using generating sequences b in which all elements are 1's and 2's. We have proven that out of all generating sequences b with a given value of $m = \sum_{i=0}^{r-1} b_i$ and $b_0 = 1$, the sequence $b = \langle 1, 2, 1, 2, 1, \ldots, 2, 1, 2, 1\rangle$ or $b = \langle 1, 2, 2, 1, 2, 1, \ldots, 2, 1, 2, 1\rangle$ or $b = \langle 1, 2, 2, 2, 1, 2, 1, \ldots, 2, 1, 2, 1\rangle$ depending on m mod 3, has the minimal possible value $R(b)$. This value asymptotically grows as $(\sqrt{3} + 2)^{\frac{n}{3}} \approx 1.5511^n$, which means worst performance ≈ 1.5790 multiplications per bit. The average performance for a randomly chosen bit string $H(b)$ is ≈ 1.54 regardless of the bit length, which is only slightly slower than classic right-to-left and left-to-right algorithms. Another motivation of this limitation $b_i \leq 2$ is a security consideration. As we mentioned above, the power analysis enables to identify two different multiplications with the same first or second operand [14]. Changing of the operand order can protect against this attack. Obviously, it is possible only when the same multiplicand is not used more than twice. See below an improved SFTS exponentiation algorithm.

The question remains whether there is enough entropy in the numbers $R(b)$ where b is a generating sequence that contains only 1's and 2's. Experimental results show that the number of possible numbers $R(b)$ are asymptotically greater than $0.42R(b)$.

It has been already mentioned in [3] that the scheme is resistant to SPA, even in elliptic curve cryptography where multiplication and squaring (called addition and doubling) typically have different power consumption signature, because no squaring operations, except for the very first multiplication of the base by itself, are involved; there are also no conditional operators. For the same reason - lack of squaring operations - the scheme is resistant to the cross-correlation attacks.

Regarding safe error attacks, there are two different kinds of them. C safe error attacks are applicable only when there are fake operations that do not affect the final result ([16]), and are clearly irrelevant here. However, M safe error attacks ([15]) must be dealt with.

M safe errors attacks can be illustrated on the following modular multiplication algorithm.

Modular Multiplication Algorithm

Inputs:
$X = (X_{n-1}X_{n-2}\dots X_0)_{2^T}, Y, N \in \mathbb{N}$

```
1. $R \leftarrow 0$
2. for $i = n-1$ downto $0$ do
3.     $R \leftarrow (R \cdot 2^T + X_i \cdot Y) \mod N$
4. end for
5. return $R$
```

Let us suppose that an assignment $X \leftarrow X \cdot Y \mod N$ is one step of a modular exponentiation, using the above modular multiplication algorithm. The modification of a block X_i after it has been used does not affect the result of the modular multiplication (because block X_i is used only once). Thus this modification does not affect also the result of the entire modular exponentiation (because the variable X gets overwritten after each multiplication). Therefore, if the attacker knows how to induce a random fault in the most significant half of operand X during the second part of the modular multiplication, and how to check whether the final result of the modular exponentiation is correct, then it is possible to find out whether the result of calculation $X \cdot Y$ is stored in X (and the fault has had no effect), or in another location (and the corrupted value in X affects the final result).

In the implementation of the SFTS Exponentiation Algorithm suggested in [3] the order of the operands is fixed, e.g. $r_0 \circ r_1$. In this case faults in r_0 have a chance to be safe only at the iterations for which $h(i) = 0$ (the result is stored to r_0). Therefore the M safe error attack reveals the secret. Fortunately, changing the order of operands defends against the M safe error attack. Namely, if the first operand (the one scanned in the outer loop) is always different from the destination, then no safe errors are left. The improved version of the algorithm is shown below.

Improved SFTS Exponentiation Algorithm

Inputs:

$x \in U$;

$n \in \mathbb{Z}$, $n \geq 2$;

$h(j) = 0, 1$ for $j \in \mathbb{Z}$, $0 \leq j \leq n-2$.

```
1. $r_0 \leftarrow r_1 \leftarrow x$
2. for $j = 0$ to $n-1$ do
3.     $r_{h(j)} \leftarrow r_{1-h(j)} \circ r_{h(j)}$
4. end for
5. return $r_0 \circ r_1$
```

This version is resistant not only to cross-correlation attacks, but also to M safe error attacks.

3 Exponentiation Algorithm for RSA

In the case of RSA, the private exponent cannot be chosen arbitrarily (e.g., if the public exponent must be a small constant), the algorithm described in the previous section that calculates the private exponent as $R(b)$ for some generating sequence b cannot be used. In [10] three highly regular algorithms with performance 2 multiplications per bit were suggested; one of them (Add-only scalar multiplication algorithm) does not involve squaring operations, and is therefore resistant to cross-correlation attack. We suggest a modification of another algorithm [2] with similar parameters, but with following differences:

- It is resistant not only to SPA, cross-correlation and fault attacks, but also to Bellcore attacks [1]
- It does involve squaring operations

First of all, we notice that when the cross-correlation attack finds squaring operations, it may help to find the private exponent in two ways:

- The sequence of squaring and multiplication operations reveals the secret
- The values being squared reveal the secret

For the left-to-right exponentiation algorithm, both the values and the sequence bear information about the secret. On the other hand, for the right-to-left exponentiation algorithm, the values being squared do not depend on the private exponent, but the order of operations does. Adding fake multiplication after squaring that has been proposed against SPA , since then neither values nor order depend on the private exponent. However, fake multiplications enable C safe error attacks.

We intend to defend the right-to-left exponentiation algorithm with added fake multiplications against C safe error attacks, and on the same occasion against another class of attacks known as Bellcore attacks [1]. Bellcore attacks assume that the attacker can cause a fault in one (unknown) bit of an intermediate result, and receive the final result of the exponentiation. These assumptions about attacker's capabilities are much stronger than for safe error attacks in two aspects:

- The attacker must be able to receive the final result (while for safe error attacks it is enough to know whether the result is correct)
- The fault must be limited to one or few bits (compared to an arbitrary fault in safe error attacks)

Therefore safe error attacks are more dangerous than Bellcore attacks. However, there is a defense that is good against both of them. In the suggested algorithm the final part of right-to-left multiplication is modified in such a way that the final result will be calculated in two ways (see [2]). We suggest a check of the two versions of the final result. Any single fault in an intermediate result causes them to be different; if this happens, the algorithm fails and returns no result, therefore providing resistance against both C safe error attacks and Bellcore attacks. This algorithm is highly regular in the same sense as all algorithms suggested in [10].

Improved Right-to-Left Exponentiation Algorithm

Inputs:

$x \in U$

$d = (d_{n-1}, \ldots, d_0)_2 \in \mathbb{N}$

```
 1. $r_2 \leftarrow r_0 \leftarrow x$
 2. $r_1 \leftarrow 1$
 3. for $j = 0$ to $n - 2$ do
 4.    $r_{d(i)} \leftarrow r_{d(i)} \circ r_2$
 5.    $r_2 \leftarrow r_2 \circ r_2$
 6. end for
 7. $r_0 = r_0 \circ r_1$
 8. $r_0 = r_0 \circ r_1$
 9. $r_1 = r_1 \circ r_2$
10. if $r_0 \neq r_1$
11.    return error
12. end if
13. return $r_0$
```

4 Conclusion

In this paper we presented two exponentiation algorithms resistant to the cross-correlation power analysis attack and other attacks. The doubling-free addition scheme has been previously mentioned in [13] and [3] in the context of elliptic curve cryptography. We suggest them for use with virtually all exponentiation-based cryptographic schemes, except for RSA in certain settings, with the following improvements:

- Using generating sequences that significantly improves performance and requires no precalculations

– Change of the order of operands that provides a defense against M safe error attacks.

The new exponentiation algorithm for RSA extends the family of highly regular algorithms from [10], having the same performance and additional protection against Bellcore attacks.

References

1. Boneh, D., DeMillo, R.A., Lipton, R.J. On the importance of eliminating errors in cryptographic computations. In: W. Fumy (Ed.): Advances in Cryptology - EUROCRYPT '97, LNCS 1233, pp. 37-51. Springer, Heidelberg (1997)
2. Boscher, A., Naciri, R., and Prouff, E., CRT RSA Algorithm Protected against Fault Attacks. In Sauveron, D., Markantonakis, K., Bilas, A., and Quisquater, J.-J., editors, *WISTP 2007*, volume 4462 of *LNCS*, pages 229243. Springer, 2007.
3. Byrne, A., Meloni, N., Crowe, F., Marnane, W.P., Tisserand, A., Popovici, E.M.: SPA resistant elliptic curve cryptosystem using addition chains. In: International Conference on Information Technology-ITNG07, pp.995-1000 (2007)
4. J.S. Coron, Resistance against differential power analysis for elliptic curve cryptosystems In Cryptographic Hardware and Embedded Systems CHES99, LNCS 1717 , pp 292-302, Springer Verlag (1999)
5. Downey, P., Leong, B., Sethi, Computing sequences with addition chains, SIAM Journal of Computing 10 (1981), 638-646
6. Fouque, A.P., Valette, F.: The doubling attack — why upwards is better than downwards. In: D.Walter, C., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 269280. Springer, Heidelberg (2003)
7. Goundar, R.R., Shiota, K., Toyonaga, M.: New strategy for doubling-free short addition-subtraction chain. In: International Journal of Applied Mathematics, vol. 2, no. 3 (2007)
8. Goundar, R.R., Shiota, K., Toyonaga, M.: SPA resistant scalar multiplication using golden ratio addition chain method. In: IAENG International Journal of Applied Mathematics, 38:2, IJAM_38_2_02 (Revised online publication: 21 June 2008)
9. Homma, N., Miyamoto, A., Aoki, T., Satoh, A., Shamir, A.: Collision-based power analysis of modular exponentiation using chosen-message pairs. In: E. Oswald and P. Rohatgi (Eds.): CHES 2008, LNCS 5154, pp. 15-29. Springer, Heidelberg (2008)
10. Joye, M.: Highly regular right-to-left algorithms for scalar multiplication. In: Pallier, P., Verbauwhede, I., (Eds.) CHES 2007, LNCS 4727, pp. 135-147. Springer, Heidelberg (2007)
11. Knuth, D.E. The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 3rd edition, §4.6.3. Addison-Wesley, San Francisco (1998)
12. Knuth, D., Yao, A: Analysis of subtractive algorithm for greatest common divisors. In: Proc. Nat. Acad. Sci. USA, 1975 December, Volume 72(12), pp. 4720-4722
13. Meloni, N.: Fast and secure elliptic curve scalar multiplication over prime fields using special addition chains.Cryptology ePrint Archive, Report 2006/216, 2006,`http://eprint.iacr.org/2006/216.pdf`
14. Witteman, M.F. , van Woudenberg, J.G.J., Menarini, F., Defeating RSA multiply-always and message blinding countermeasures. In Proceedings of the 11th international conference on Topics in cryptology: CT-RSA 2011 (CT-RSA'11), Aggelos Kiayias (Ed.). Springer-Verlag, Berlin, Heidelberg, 77-88. (2011)
15. Yen, S.-M., Joye, M.: Checking before output may not be enough against fault-based cryptoanalysis. In: IEEE Trans. on Computers, 49(9):967-970, September 2000
16. Yen, S.-M., Kim, S.-J., Lim, S.-G., Moon, S.-J. A countermeasure against one physical cryptanalysis may benefit another attack. In: K. Kim (Ed.): Information Security and Cryptology – ICICS 2001, LNCS 2288, pp. 414-427. Springer, Heidelberg (2002)
17. Yen, S.-M., Lien, W.-C., Moon, S.-J., Ha, J.-C.: Power analysis by exploiting chosen message and internal collisions - vulnerability of checking mechanism for RSA decryption. In: Dawson, E., Vaudenay, S. (eds.) Mycrypt 2005. LNCS, vol. 3715, pp. 183195. Springer, Heidelberg (2005)