HOW TO AVOID THE N-1 ATTACK WITHOUT COSTLY IMPLEMENTATIONS

David Tinoco Varela

Computational Science Graduate Program, Facultad de Estudios Superiores Cuautitlán, Universidad Nacional Autónoma de México, Edo. de Mex. 54740, México
dativa19@comunidad.unam.mx

ABSTRACT

Simple Power Analysis (SPA) attacks are widely used against several cryptosystems, principally against cryptosystems based on modular exponentiation. Many types of SPA have been reported in the literature, Yen et al. introduced the N-1 attack, which uses chosen input messages to obtain relevant information from the attacked cryptosystem. Their attack was implemented on the square-and-multiply always and on the BRIP algorithm, both algorithms in left-to-right form. There are possible countermeasures against this attack, but all of them are costly and time consuming. In this paper, a computationally efficient and effective method to avoid the N-1 attack is investigated.

KEYWORDS

Simple Power Analysis, N-1 Attack, Modular Exponentiation, Embedded Devices.

1. INTRODUCTION

Since *Kocher* [1] described the first *Side Channel Attack* (SCA), many physical attacks against cryptosystems implemented on embedded devices have been developed. Because SCAs represent a serious threat, it is necessary to avoid them to obtain reliable cryptographic modules.

When a device executes operations of encryption or decryption, there are several measurable physical signals, such as operating times [1], power consumption [2], and electromagnetic radiation [3], that are correlated with the secret parameters of the cryptosystem. Basically, the SCA obtains useful information from a device, such as a s*mart card*, by measuring these physical signals. The information can be used to reveal the secret key used to protect the processed data within the appliance, and thus, the security of the cryptosystem can be compromised.

Simple Power Analysis (SPA) and *Differential Power Analysis* (DPA) are well-known attacks, both described by Kocher in [2]. The secret values from a cryptosystem can be obtained using SPA by measuring one or few power traces from the device that is executing the algorithm. It is possible to determine the value of the system's secret key from these measurements. To obtain the secret key using DPA, many power consumption traces of the same algorithm with different input messages are collected, and a statistical analysis of the traces is performed.

Primarily, an SCA is used to attack the *modular exponentiation*, which is the core operation of many cryptosystems, such as RSA. Because the modular exponentiation is mainly calculated in binary form, attacks use the measured signals to determine the binary representation of the exponent; usually, the exponent is the secret key of the cryptosystem.

Square-and-multiply is a classic algorithm that is used to calculate the modular exponentiation. This algorithm can be used in *right-to-left* or *left-to-right* form. This algorithm uses few

DOI : 10.5121/ijnsa.2012.4407

registers, and it is the simplest and the fastest of all known algorithms. However, this algorithm is vulnerable to SPA because it calculates a modular multiplication and a square if the exponent bit being processed is 1 and only a square if the exponent bit being processed is 0. Thus, an attacker can measure the power consumption or the operating times of the algorithm's execution to determine the binary string of the cryptosystem's secret key.

Many modular exponentiation algorithms have been designed to prevent SPA, e.g., [4], [5], [6], [7], [8], [9], [10], and [11]. *Coron* [12] provided the first algorithm created specifically to defeat SPA using the *square-and-multiply always* (SaMA) algorithm, which works in a regular form. That is, the algorithm will always calculate a multiplication followed by a square, independent of the value of the exponent bit being processed (0 or 1).

There are many SPA techniques [13], [14]. For example, *chosen-message* is a technique used to obtain relevant information from a device that is executing a cryptographic algorithm. Chosen-message uses an input message that has a specific behaviour during the execution of the algorithm, and that behaviour can be observed by the attacker who can obtain the secret parameters of the cryptosystem.

Fouque and *Valette* proposed and implemented the *Doubling Attack* (DA) [13], a chosen-message attack, against the left-to-right SaMA. In that attack, two related messages, M and M^2, are chosen, and the behaviours of both messages in the attacked algorithm are used to obtain the secret key. The r*elative Doubling Attack* (RDA), an idea similar to DA, was proposed by Yen et al. in [14]. These authors used RDA to attack the *Montgomery powering ladder* algorithm [5].

One of the simplest chosen-message attacks is the $N-1$ attack proposed by Yen et al. in [15]. Though they explained the theoretical form of their attack against the left-to-right SaMA and BRIP algorithms [16], *Miyamoto* et al. demonstrated the effectiveness of an $N-1$ attack in practice [17].

Intermediate even exponents were proposed in [19] to protect the Montgomery powering ladder algorithm against an attack that is based in the Jacobi symbol.

2. PRELIMINARIES

2.1. Modular exponentiation

The classical modular exponentiation algorithm (algorithm 1) uses the binary string of the exponent to calculate the correct result of $m^d = m^{\sum_{i=0}^{n-1} 2^i d_i}$. In each of the iterations, this algorithm executes a square operation and a conditional modular multiplication. Because of the conditional multiplication, an attacker can determine the binary string of the exponent by observing the power consumption of the device. The algorithm consumes more power when the bit of the exponent is 1 than when the bit is 0.

Coron designed the SaMA algorithm (algorithm 2) to avoid the SPA. This algorithm is regular, i.e., it does not have conditional multiplications and executes the same number of operations in each of the iterations. However, algorithm 2 uses dummy operations, which can be vulnerable to *Safe Error Attacks* [18].

Mamiya et al. proposed the BRIP algorithm, algorithm 3. This algorithm is regular, but it does not use dummy operations. Randomization of the input message protects the algorithm against

DPA. The authors claimed that their algorithm is secure against SPA, but Yen et al. demonstrated that algorithm 3 is vulnerable to an $N-1$ attack, which is a type of SPA.

Algorithm 1 Square-and-multiply left-to-right

Input $m \in G$, $d = (d_{n-1} \ldots d_0)_2$
Output $s = m^d \in G$
$R[0] \leftarrow 1$
for $n-1$ to 0 **do**
    $R[0] \leftarrow R[0]^2 \bmod N$
    if $d_i = 1$ **then**
        $R[0] \leftarrow R[0] \cdot m \bmod N$
    end if
end for
Return $R[0]$

Algorithm 2 Square-and-multiply always, left-to right

Input $m \in G$, $d = (d_{n-1} \ldots d_0)_2$
Output $s = m^d \in G$
$R \leftarrow 1$
for $n-1$ to 0 **do**
    $R[0] \leftarrow R^2 \bmod N$
    $R[1] \leftarrow R[0] \cdot m \bmod N$
    $R \leftarrow R[d_i] \bmod N$
end for
Return R

Algorithm 3 BRIP

Input $m \in G$, $d = (d_{n-1} \ldots d_0)_2$
Output $s = m^d \in G$
$R[0] \leftarrow r$
$R[1] \leftarrow r^{-1}$
$R[2] \leftarrow m \cdot r^{-1}$
for $n-1$ to 0 **do**
    $R[0] \leftarrow R[0]^2 \bmod N$
        if $d_i = 0$ **then**
            $R[0] = R[0] \cdot R[1] \bmod N$
        else
            $R[0] = R[0] \cdot R[2] \bmod N$
        end if
end for
Return $R[0] \cdot R[1] \bmod N$

2.2. *N*-1 Attack

In 2005, Yen et al. proposed the $N-1$ attack in [15]. They implemented it against modular exponentiation algorithms 2 and 3. This attack assumes that the power traces obtained from a device that is performing computations can be measured and compared to determine when two values or operations are equal, even if the adversary does not know what values are being processed. For example, if A and B are values, where $B = A$, the collisions of A^2 and B^2 can be detected, even if A and B are unknown.

This attack is based on two observations:

1. - If a cryptosystem works with modulo N, $(N-1)^2 \equiv 1 \bmod N$. This observation can be extended to obtain $(N-1)^j \equiv 1 \bmod N$, for any even integer j.

2. - If a cryptosystem works with modulo N, $(N-1)^k \equiv (N-1) \bmod N$, for any odd integer k.

If a chosen value equal to $N-1$ is the input message, the value at the end of each of the iterations of the attacked algorithm can take only one of two values: 1 if the bit being processed is 0 or $N-1$ if the bit being processed is 1. Therefore, an attacker can observe only two patterns in the power traces and thus can determine the binary string of the exponent.

In 2008, Miyamoto et al. [17] demonstrated and analyzed the effectiveness of the $N-1$ attack in practice. They obtained power traces in which it is possible to identify when an exponent bit is 0 and when it is 1.

In [19], it was claimed that any m^d could be calculated using only even intermediate exponents in a modular exponentiation algorithm. This idea was used to protect the Montgomery powering ladder algorithm against an attack based on the Jacobi symbol. In section 3, this idea is used to avoid the $N-1$ attack.

3. PROPOSED ALGORITHMS

In this section, the algorithms previously described and attacked with the $N-1$ attack are modified to protect them against the mentioned threat.

3.1. Modified algorithms

If an attacker wants to break a cryptosystem and knows the value of the modulus N of the cryptosystem, he can choose an input message that is equal to $N-1$. The attacker sends the chosen message to a device that runs an algorithm to encrypt and decrypt secret information. The attacker collects one or few power traces from the device and searches for two patterns in the power traces. Each pattern corresponds to a specific value. In this case, the value is 1 if $d_i = 0$ and $N-1$ if $d_i = 1$.

When the attacker has found the patterns, he can determine two possible exponents, d and $\bar{d}$, where $\bar{d}$ is the binary inverse of d, i.e., if $\mathrm{d} = 101101$, then $\bar{d} = 010010$. Using d and $\bar{d}$, a trial-and-error approach can be used to find the correct exponent and to break down the security of the cryptosystem.

This scheme is illustrated in table 1, which shows the behaviour of algorithm 3 when it is subjected to an $N-1$ attack. In this example, it is assumed that $m = N-1$ and d =89=1011001. Algorithm 3 was chosen to demonstrate that two patterns could be recognized, even with a blinded message.

Table 1. Algorithm 3 executed with an input message equal to $N-1$.

i	d_i	Intermediate steps	Result (Pattern)
6	1	$R[0] = r^2$ $R[0] = m \cdot r$	$(N-1) \cdot r$
5	0	$R[0] = m^2 \cdot r^2$ $R[0] = m^2 \cdot r$	$(1) \cdot r$
4	1	$R[0] = m^4 \cdot r^2$ $R[0] = m^5 \cdot r$	$(N-1) \cdot r$
3	1	$R[0] = m^{10} \cdot r^2$ $R[0] = m^{11} \cdot r$	$(N-1) \cdot r$
2	0	$R[0] = m^{22} \cdot r^2$ $R[0] = m^{22} \cdot r$	$(1) \cdot r$
1	0	$R[0] = m^{44} \cdot r^2$ $R[0] = m^{44} \cdot r$	$(1) \cdot r$
0	1	$R[0] = m^{88} \cdot r^2$ $R[0] = m^{89} \cdot r$	$(N-1) \cdot r$

As shown in table 1, the presence of odd and even exponents in the intermediate steps of the algorithm lead to two patterns in the power trace, and thus, the secret key of the cryptosystem can be discovered.

This attack is powerful and can be easily implemented. However, to obtain the secret key, two patterns must be present in the power trace. This attack can be avoided if there is only one pattern in the power trace of each iteration, regardless of the processed bit. This idea is the premise of the proposed algorithms. To implement this idea, recall that it is possible to work with only even intermediate exponents in an algorithm [19]. Therefore, it is possible to obtain only one visible pattern in the measured power trace. Based on this statement, algorithms 2 and 3 have been modified to obtain algorithms 4 and 5, respectively.

In algorithms 4 and 5, the first step is to change the input value m to m^2, this square value will affect all of the calculations in the algorithms. In each of the iterations, only even intermediate exponents will be used in calculations. Therefore, an attacker will not be able to obtain any information because all of the executed iterations in the algorithm will have the same pattern, independent of the processed bit. This behavior is illustrated in table 2.

Algorithm 4 Modified square-and-multiply always, left-to-right

Input $m \in G$, $d = (d_{n-1} \ldots d_0)_2$
Output $s = m^d \in G$
$R \leftarrow 1$
$M = m \cdot m$
for $n-1$ to 1 **do**
    $R[0] \leftarrow R^2 \bmod N$
    $R[1] \leftarrow R[0] \cdot M \bmod N$
    $R \leftarrow R[d_i] \bmod N$
end for
if $d_0 = 0$ **then**
    Return R
else
    Return $R \cdot m$
end if

Algorithm 5 Modified BRIP

Input $m \in G$, $d = (d_{n-1} \ldots d_0)_2$
Output $s = m^d \in G$
$R[0] \leftarrow r$
$R[1] \leftarrow r^{-1}$
$R[2] \leftarrow m \cdot m \cdot r^{-1}$
for $n-1$ to 1 **do**
    $R[0] \leftarrow R[0]^2 \bmod N$
    if $d_i = 0$ **then**
        $R[0] = R[0] \cdot R[1] \bmod N$
    else
        $R[0] = R[0] \cdot R[2] \bmod N$
    end if
end for
if $d_0 = 1$ **then**
    $R[0] = R[0] \cdot R[1] \cdot m \bmod N$
else
    $R[0] = R[0] \cdot R[1] \bmod N$
end if
Return $R[0]$

3.2. Principle of the proposed algorithms

The explanation presented in this subsection is an extended version of that given in [19], but here; equation (13) is presented for first time. This equation is a mathematical representation of algorithms that work with intermediate even exponents.

Table 2. Algorithm 5 executed with an input message equal to $N-1$ and an exponent d =89=1011001.

i	d_i	Intermediate steps	Result (Pattern)
6	1	$R[0]=r^2$ $R[0]=m^2\cdot r$	$(1)\cdot r$
5	0	$R[0]=m^4\cdot r^2$ $R[0]=m^4\cdot r$	$(1)\cdot r$
4	1	$R[0]=m^8\cdot r^2$ $R[0]=m^{10}\cdot r$	$(1)\cdot r$
3	1	$R[0]=m^{20}\cdot r^2$ $R[0]=m^{22}\cdot r$	$(1)\cdot r$
2	0	$R[0]=m^{44}\cdot r^2$ $R[0]=m^{44}\cdot r$	$(1)\cdot r$
1	0	$R[0]=m^{88}\cdot r^2$ $R[0]=m^{88}\cdot r$	$(1)\cdot r$
0	1	$R[0]=R[0]\cdot R[1]\cdot m=m^{88}\cdot r\cdot r^{-1}\cdot m=m^{89}$	

The attacked algorithms (2 and 3) are in the left-to-right form. According to Moreno and Hasan [11], this execution form is based in the following property; given the result of m^a, it is possible to obtain $m^{a'}$, where a' is calculated by adding the bit b to a. Note that $a'=2a+b$, and thus

$$m^{a'}=m^{2a+b}=(m^a)^2\cdot m^b. \quad (1)$$

If equation (1) is used to calculate m^d in a modular exponentiation algorithm (algorithms 2 and 3), where $d=\sum_{i=0}^{n-1} d_i\cdot 2^i$, it is possible to obtain the following representation of it

$$(\cdots((m^{d_{n-1}})^2\cdot m^{d_{n-2}})^2\cdots m^{d_1})^2\cdot m^{d_0}, \quad (2)$$

where n is the bit length of the exponent being processed.

If the modified algorithms (algorithms 4 and 5) were executed from iteration $n-1$ to 0, the following representation could be obtained

$$(\cdots((m^{2d_{n-1}})^2\cdot m^{2d_{n-2}})^2\cdots m^{2d_1})^2\cdot m^{2d_0}. \quad (3)$$

To understand why the proposed algorithms are executed from $n-1$ to 1 and why an *if statement* is used in their last lines, we must consider the behaviours of equations (2) and (3) when they are calculated using a modular exponentiation algorithm. The behaviours of both equations will be represented using the assumption that they are executed from bit d_{n-1} to d_0. In both representations, $k = n-1-i$, and each step represents an iteration. The behaviour of equation (2) at each of the iterations of an algorithm is given by equations (4) - (7):

$$\text{Step } 1 \qquad (\cdots((m^{2^1(d_{n-1})+2^0(d_{n-2})})^2 \cdot m^{(d_{n-3})})^2 \cdots m^{(d_1)})^2 \cdot m^{(d_0)} \tag{4}$$

$$\vdots \qquad\qquad \vdots$$

$$\text{Step } i \qquad (\cdots((m^{2^i(d_{n-1})+2^{i-1}(d_{n-2})+\cdots+2^0(d_k)})^2 \cdot m^{(d_{k-1})})^2 \cdots m^{(d_1)})^2 \cdot m^{(d_0)} \tag{5}$$

$$\vdots \qquad\qquad \vdots$$

$$\text{Step } n-2 \qquad (m^{2^{n-2}(d_{n-1})+2^{n-3}(d_{n-2})+\cdots+2^0(d_1)})^2 \cdot m^{(d_0)} \tag{6}$$

$$\text{Step } n-1 \qquad m^{2^{n-1}(d_{n-1})+2^{n-2}(d_{n-2})+\cdots+2^1(d_1)} \cdot m^{(d_0)} \tag{7}$$

The behaviour of equation (3) is given by equations (8) - (11):

$$\text{Step } 1 \qquad (\cdots((m^{2^{1+1}(d_{n-1})+2^{0+1}(d_{n-2})})^2 \cdot m^{2(d_{n-3})})^2 \cdots m^{2(d_1)})^2 \cdot m^{2(d_0)} \tag{8}$$

$$\vdots \qquad\qquad \vdots$$

$$\text{Step } i \qquad (\cdots((m^{2^{i+1}(d_{n-1})+2^{i-1+1}(d_{n-2})+\cdots+2^{0+1}(d_k)})^2 \cdot m^{2(d_{k-1})})^2 \cdots m^{2(d_1)})^2 \cdot m^{2(d_0)} \tag{9}$$

$$\vdots \qquad\qquad \vdots$$

$$\text{Step } n-2 \qquad (\underline{m^{2^{n-2+1}(d_{n-1})+2^{n-3+1}(d_{n-2})+\cdots+2^{0+1}(d_1)}})^2 \cdot m^{2(d_0)} \tag{10}$$

$$\text{Step } n-1 \qquad m^{2^{n-1+1}(d_{n-1})+2^{n-2+1}(d_{n-2})+\cdots+2^{1+1}(d_1)} \cdot m^{2(d_0)} \tag{11}$$

It is possible to see that equation (7) is the correct result of calculating m^d. Note that the underlined part of equation (10), ($\underline{m^{2^{n-1}(d_{n-1})+2^{n-2}(d_{n-2})+\cdots+2^1(d_1)}}$), is similar to equation (7).

To obtain equality between equations (7) and (10), the last square and the multiplication by $m^{2(d_0)}$ in equation (10) are deleted. The following expression is obtained:

$$m^{2^{n-1}(d_{n-1})+2^{n-2}(d_{n-2})+\cdots+2^1(d_1)} \tag{12}$$

Now, equation (12) must be multiplied by $m^{(d_0)}$, and thus, the correct value of m^d is obtained.

Equation (3) is the representation of classic exponentiation algorithms. Here, the equation (13) is presented, which is the representation of algorithms that work with intermediate even exponents:

$$(\cdots((m^{2d_{n-1}})^2 \cdot m^{2d_{n-2}})^2 \cdots m^{2d_2})^2 \cdot m^{2d_1} \cdot m^{d_0} \tag{13}$$

Because the proposed algorithms run from $n-1$ to 1, thereby eliminating the last square and the last multiplication by $m^{2(d_0)}$, and use the *if statement*, which is the multiplication by $m^{(d_0)}$ where $d_0 \in \{0,1\}$, the correct value of m^d is computed.

3.3. Possible countermeasures

There are possible countermeasures against the $N-1$ attack. However, the countermeasures have characteristics that make them impractical. Some possible countermeasures and their disadvantages are discussed below:

1. *Blind the message.* In this countermeasure, the message, m, is blinded using a random value, r. This idea can be seen as a correct form to protect the modular exponentiation. However, it can be observed that algorithm 3 blinds the message and it remains vulnerable to the $N-1$ attack [15]. The best technique for message blinding under this scheme is that proposed by *Fumaroli* and *Vigilant* [20]. However, their technique requires an additional register to save r^{-1} and n additional square operations, where n is the number of bits of the exponent.

2. *Blind the modulus*, mentioned in [6, section 3.2]. In this countermeasure, the modulus, N, is blinded using a random value, r. This idea seems like a good proposal, however, there are two disadvantages in its implementation:

   2.1. The bit length of the modulus is increased, for what the operations will be calculated with bigger integers; thus, the runtime of the algorithm will be longer, and the power consumption of the algorithm will be higher.

   2.2. There are standards used in the manufacturing of crypto devices, such as smart cards. If the bit length of the modulus is increased, the modulus will not be coupled with the standards.

3. *Block the special input message,* $N-1$. This might be the most logical reaction against this type of attack. However, according to Yen et al. [15], it is possible to choose messages with a modified $N-1$ value. For example, the attacker can choose two input values, m_i and $m_j = (N-1) \cdot m_i$, and collect the two related power consumption traces of computing $m_i{}^d \bmod N$ and $m_j{}^d \bmod N$. The adversary can detect the collisions when $m_i{}^k \equiv m_j{}^k \bmod N$ and deduce the value of d. To avoid this modified attack, the relationship between two input messages can be checked to determine if $m_j \equiv m_i \cdot (N-1) \bmod N$ for every pair i and j. According to Yen et al. this is extremely difficult and infeasible to implement in practice because the input messages, m_i and m_j, might not be consecutive messages, and it is not possible to store all of the previous messages required to perform the detection of that relationship.

In table 3, it is possible to see the behaviour of two executions of the algorithm 2 when two input messages equal to m_1 and $m_2 = (N-1) \cdot m_1$ are used in the algorithm. In this example, it was assumed that d =89=1011001.

Table 3 shows that the visible patterns in both executions of algorithm 2 are equal when the bit being processed is 0 and are different when the bit being processed is 1. This characteristic can be used by an attacker to recognize the binary string of the secret key, d.

Table 3. Comparison of executions of algorithm 2 using input message m_1 and input message $m_2 = m_1 \cdot (N-1)$.

i	d_i	**Intermediate steps when the input is equal to m_1.**	**Intermediate steps when the input is equal to m_2.**	**Comparison of the pattern of the register R.**
6	1	$R[0] = 1$ $R[1] = m_1$ $R = m_1$	$R[0] = 1$ $R[1] = m_1 \cdot (N-1)$ $R = m_1 \cdot (N-1)$	Different
5	0	$R[0] = {m_1}^2$ $R[1] = {m_1}^3$ $R = {m_1}^2$	$R[0] = {m_1}^2$ $R[1] = {m_1}^3 \cdot (N-1)$ $R = {m_1}^2$	Equal
4	1	$R[0] = {m_1}^4$ $R[1] = {m_1}^5$ $R = {m_1}^5$	$R[0] = {m_1}^4$ $R[1] = {m_1}^5 \cdot (N-1)$ $R = {m_1}^5 \cdot (N-1)$	Different
3	1	$R[0] = {m_1}^{10}$ $R[1] = {m_1}^{11}$ $R = {m_1}^{11}$	$R[0] = {m_1}^{10}$ $R[1] = {m_1}^{11} \cdot (N-1)$ $R = {m_1}^{11} \cdot (N-1)$	Different
2	0	$R[0] = {m_1}^{22}$ $R[1] = {m_1}^{23}$ $R = {m_1}^{22}$	$R[0] = {m_1}^{22}$ $R[1] = {m_1}^{23} \cdot (N-1)$ $R = {m_1}^{22}$	Equal
1	0	$R[0] = {m_1}^{44}$ $R[1] = {m_1}^{45}$ $R = {m_1}^{44}$	$R[0] = {m_1}^{44}$ $R[1] = {m_1}^{45} \cdot (N-1)$ $R = {m_1}^{44}$	Equal
0	1	$R[0] = {m_1}^{88}$ $R[1] = {m_1}^{89}$ $R = {m_1}^{89}$	$R[0] = {m_1}^{88}$ $R[1] = {m_1}^{89} \cdot (N-1)$ $R = {m_1}^{89} \cdot (N-1)$	Different

3.4. Advantages of and commentaries on our modified algorithms

The modified algorithms have almost the same runtime as the original versions of them. They only require an additional *if statement*. Algorithm 4 requires one more register than algorithm 2 to save $M = m \cdot m$, but algorithm 5 does not require any extra register.

The proposed algorithms do not require that the message be blinded to avoid an $N-1$ attack[1] because in each intermediate step, the resulting value will always be equal to 1 (if the input message is equal to $N-1$) regardless of the value of the exponent's bit being processed by the algorithm. The attacker cannot determine the secret key because there is only one observable pattern in the power trace.

The bit length of the modulus, or any other value, does not need to be increased in the proposed algorithms. Therefore, the algorithms are in accordance with the standards used to manufacture crypto devices.

Additionally, this strategy is useful to avoid attacks that use a modified $N-1$ value because the congruence $m_i{}^k \cdot 1 \equiv m_j{}^k \bmod N$ will always be true in each intermediate step of the algorithm, disregarding the value of the exponent bit being processed. Therefore, it is not necessary to check the relationship between input messages, which is very costly and impractical. This protection can be observed in table 4, where it is possible to see the behaviour of two executions of the algorithm 4 when input messages, m_1 and $m_2 = (N-1) \cdot m_1$, are used in the algorithm. In this example, it was assumed that d =89=1011001.

In table 4, it is possible to see that the patterns of the two executions of the algorithm are equal. Therefore, an attacker cannot distinguish between a bit that is 0 and a bit that is 1. It is important to note that line 4 in algorithm 4 eliminates the value $N-1$ of the message m_2:

$$M = m_2 \cdot m_2 = (m_1 \cdot (N-1)) \cdot (m_1 \cdot (N-1)) = (m_1 \cdot (N-1))^2 = m_1{}^2 \cdot (N-1)^2 = m_1{}^2 \cdot 1 = m_1{}^2$$

Owing to the characteristics described in this section, it is possible to say that this method is better than existing methods.

As was said before, this method can be easily implemented in practice. Therefore, it can be implemented in protocols where embedded devices and cryptosystems based in modular exponentiation are used; an example of a possible application is given in [21].

4. CONCLUSIONS

In this paper, we have presented two algorithms based in the intermediate even exponents that are secure against the $N-1$ attack.

The proposed algorithms do not require additional runtime because they perform the same number of operations as the original versions (algorithms 2 and 3). Additionally, they do not increase the bit length of their parameters, i.e., they do not require a larger modulus to be secure against this type of attack. Therefore, they are in accordance with the manufacturing standards of embedded devices, such as smart cards.

Through this paper, has been shown that the proposed algorithms provide security against attacks in which the input message is equal to $N-1$ and in which the input message is a modification of the $N-1$ value, like $N-1 \cdot m$, without costly steps and without impractical solutions.

[1] We are referring to the characteristics of the method against an $N-1$ attack, but it is necessary to blind the message to avoid other types of attacks on the algorithms.

This protection against $N-1$ attacks is easy to implement. Because it does not require additional implementation costs, it can be used in practice. This strategy does not sacrifice runtime and offers better security than existing methods.

ACKNOWLEDGEMENTS

We would like to thank the referee for carefully reading this paper and for his constructive suggestions. This paper was supported, in part, by project CONS-18 of FES-C UNAM.

Table 4. Comparison of two executions of algorithm 4, one using an input message m_1 and the other using an input message $m_2 = m_1 \cdot (N-1)$.

i	d_i	**Intermediate steps when the input is equal to m_1.**	**Intermediate steps when the input is equal to m_2.**	**Comparison of the pattern of the register *R*.**
6	1	$R[0] = 1$ $R[1] = m_1^2$ $R = m_1^2$	$R[0] = 1$ $R[1] = m_1^2$ $R = m_1^2$	Equal
5	0	$R[0] = m_1^4$ $R[1] = m_1^6$ $R = m_1^4$	$R[0] = m_1^4$ $R[1] = m_1^6$ $R = m_1^4$	Equal
4	1	$R[0] = m_1^8$ $R[1] = m_1^{10}$ $R = m_1^{10}$	$R[0] = m_1^8$ $R[1] = m_1^{10}$ $R = m_1^{10}$	Equal
3	1	$R[0] = m_1^{20}$ $R[1] = m_1^{22}$ $R = m_1^{22}$	$R[0] = m_1^{20}$ $R[1] = m_1^{22}$ $R = m_1^{22}$	Equal
2	0	$R[0] = m_1^{44}$ $R[1] = m_1^{46}$ $R = m_1^{44}$	$R[0] = m_1^{44}$ $R[1] = m_1^{46}$ $R = m_1^{44}$	Equal
1	0	$R[0] = m_1^{88}$ $R[1] = m_1^{90}$ $R = m_1^{88}$	$R[0] = m_1^{88}$ $R[1] = m_1^{90}$ $R = m_1^{88}$	Equal
0	1	$R = R \cdot m_1 = m_1^{88} \cdot m_1$ $R = m_1^{89}$	$R = R \cdot m_1 \cdot (N-1)$ $R = m_1^{88} \cdot m_1 \cdot (N-1)$ $R = m_1^{89} \cdot (N-1)$	Different

REFERENCES

[1] P. Kocher, (1996), "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems." *In Koblitz, N., ed.: Advances in Cryptology-CRYPTO 96. Volume 1109 of Lecture in Notes in Computer Science*, pp. 104-113.

[2] P.C. Kocher, J. Jaffe, and B. Jun, (1999), "Differential Power Analysis." *In Wiener, M., Ed.: Advances in Cryptology-CRYPTO '99. Volume 1666 of Lecture Notes in Computer Science*, pp. 388-397.

[3] J.J. Quisquater and D. Samyde, (2001), "Electromagnetic analysis (ema): Measures and counter-measures for smart cards". *Smart Card Programming and Securit,* pp. 200-210.

[4] M. Joye, (2002), "Recovering lost efficiency of exponentiation algorithms on smart cards". IET *Electronics Letters*, Vol. 38, No. 19, pp. 1095-1097.

[5] M. Joye and S.M. Yen. (2002), "The montgomery powering ladder." *In Cryptographic Hardware and Embedded Systems-CHES 2002, 2523 of Lecture Notes in Computer Science*, pp. 291-302.

[6] C. Giraud, (2006), "An rsa implementation resistant to fault attacks and to simple power analysis". *IEEE Transactions on computers*, Vol. 55, No. 9, pp. 1116-1120.

[7] J.C. Ha, J.H. Park, S.J. Moon and S.M. Yen, (2007), "Provably secure countermeasure resistant to several types of power attack for ECC". Springer *Information Security Applications*, pp. 333-344.

[8] D.Z. Sun, J.P. Huai, J.Z. Sun and Z.F. Cao, (2007), "An efficient modular exponentiation algorithm against simple power analysis attacks", IEEE *Transactions on Consumer Electronics*, Vol. 53, No. 4, pp. 1718-1723.

[9] M. Joye, (2007), "Highly regular right-to-left algorithms for scalar multiplication." *Cryptographic Hardware and Embedded Systems-CHES 2007, 4727 of Lecture in Notes in Computer Science*, pp. 135–147.

[10] J.C. Ha, C.H. Jun, J.H. Park, S.J. Moon, and C.K. Kim, (2008), "A new crt-rsa scheme resistant to power analysis and fault attacks." *Third 2008 International Conference on Convergence and Hybrid Information Technology*, pp. 351-356.

[11] C. Moreno and M.A. Hasan, (2011), "SPA-resistant binary exponentiation with optimal execution time", Springer *Journal of Cryptographic Engineering*, pp. 1-13.

[12] J.S. Coron, (1999), "Resistance against differential power analysis for elliptic curve cryptosystems." *In Ko, Paar, C., Eds.: Cryptographic Hardware and Embedded Systems-CHES 2002. Vol. 1717 of Lecture Notes in Computer Science*, pp. 292-302.

[13] P.A. Fouque and F. Valette, (2003), "The doubling attack–why upwards is better than downwards." *In Cryptographic Hardware and Embedded Systems-CHES 2003, LNCS 2779*, pp. 269-280.

[14] S.M. Yen, L.C. Ko, S.J. Moon, and J.C. Ha, (2005), "Relative doubling attack against montgomery ladder." *In Information Security and Cryptology-ICISC 2005, 3935 of Lecture Notes in Computer Science*, pp. 117-128.

[15] S.M. Yen, W.C. Lien, S.J. Moon, and J.C. Ha, (2005), "Power analysis by exploiting chosen message and internal collisions-vulnerability of checking mechanism for rsa-decryption." *Progress in Cryptology–Mycrypt 2005, 3715 of Lecture Notes in Computer Science*, pp. 183-195.

[16] H. Mamiya, A. Miyaji, and H. Morimoto, (2004), "Efficient countermeasures against rpa, dpa, and spa." *Cryptographic Hardware and Embedded Systems-CHES 2004, 3156 of Lecture Notes in Computer Science*, pp. 343-356.

[17] A. Miyamoto, N. Homma, T. Aoki and A. Satoh, (2008), "Enhanced power analysis attack using chosen message against RSA hardware implementations." IEEE *International Symposium on Circuits and Systems,* pp. 3282-3285.

[18] S.M. Yen, S. Kim, S. Lim, and S. Moon, (2001), "A countermeasure against one physical cryptanalysis may benefit another attack". *Information Security and Cryptology-ICISC 2001, 2288 of Lecture Notes in Computer Science*, pp.414-427.

[19] D. Tinoco Varela, (2012), "Blinded Montgomery Powering Ladder Protected Against the Jacobi Symbol Attack", *International Journal of Security* (IJS), Vol. 6, No. 3, pp. 15-27.

[20] G. Fumaroli and D. Vigilant, (2006), "Blinded fault resistant exponentiation." *Fault Diagnosis and Tolerance in Cryptography, 4236 of Lecture Notes in Computer Science,* pp. 62-70.

[21] Hayam K. Al-Anie, Mohammad A. Alia and Adnan A. Hnaif, (2011), "E-Voting protocol based on public-key cryptography." *International Journal of Network Security & Its Applications* (IJNSA), Vol.3, No.4, pp. 87-98.