Blinded Montgomery Powering Ladder Protected Against the Jacobi Symbol Attack

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

Abstract

Recently, many physical attack types (e.g., timing attacks, power consumption attacks, fault attacks) have been developed against cryptosystems, specifically against the modular exponentiation that is the core operation of many security systems. Indeed, there is a real need to eliminate the vulnerabilities of cryptosystems, such as RSA or the Elliptic Curve Cryptosystem, that make them susceptible to such attacks. In 2006, Boreale described a new type of physical attack based in the Jacobi symbol concept, and later, Schmidt used the same concept as Boreale to break the security of the blinded Montgomery powering ladder. In this paper, a countermeasure against Schmidt's attack is presented to make the blinded Montgomery powering ladder resistant to the Jacobi symbol attack.

Keywords: Modular Exponentiation, Cryptography, Jacobi Symbol, Montgomery Ladder, Fault Attacks.

1. INTRODUCTION

Kocher [1] was the first to point out the existence of physical attacks called *Side Channel Attacks (SCA)*. He observed that when a cryptographic algorithm is implemented in an embedded device, an attacker can obtain the binary string of the secret key by simply observing the power traces or the timing consumption of the device in an electronic test instrument, such as an oscilloscope. SCAs are, first of all, used to attack modular exponentiation (*Add and double* is the analogous function in the Elliptic Curve Cryptosystem, ECC), which is the core operation in cryptosystems such as RSA.

SCAs opened the door to a new type of physical attacks, one of which was the *Fault Attack* (FA) proposed by Bonhe, DeMillo and Lipton [2]. FAs are more aggressive than SCAs because FAs physically disturb the execution of the device that is running the cryptographic algorithm.

To prevent SCAs and FAs, many modular exponentiation algorithms have been created, but Coron [3] provided the first algorithm specifically designed to defeat SCAs when he proposed the s*quare-and-multiply always algorithm*. However, this algorithm was attacked by the denominated *Safe Error Attack* (SEA) [4].

The *Montgomery powering ladder* [5] was a new idea proposed by Joye and Yen to protect cryptosystems against SCAs and FAs. This algorithm works in a regular form: that is, regardless of the value of the bit being processed (0 or 1), the algorithm will always calculate a multiplication followed by a squaring. The Montgomery ladder was widely accepted and attracted the attention of many researchers. Giraud [6] modified the Montgomery ladder to protect it against FAs; he proposed a *Coherence Test* based on a characteristic of the algorithm: the registers in all the iterations have the form $R[0]=m^x$, $R[1]=m^{x+1}$. As a result, if the coherence test $R[0]\cdot m=R[1]$ is true, then return $R[0]$; if not, return "error".

The Montgomery ladder was attacked by the *Relative Doubling Attack* (RDA) [7], a modification of the *Doubling Attack* (DA) [8], but Fumaroli and Vigilant [9] added a random value to the Montgomery ladder to blind the modular exponentiation. The algorithm proposed by Fumaroli and Vigilant was secure against SCAs, DA, RDA, and in a partial form against FA.

A new type of attack was presented by Boreale in 2006 [10]. This attack uses a combination of FA and SCA, and using the Jacobi symbol (JS) it is possible to obtain the binary string of the secret key d. He used his model against the *square-and-multiply right-to-left* algorithm and proved that his attack is effective even in the presence of message blinding. On the other hand, Schmidt and Medwed [11] used the Jacobi symbol concept to create an attack that breaks the security of the Montgomery powering ladder in its blinded form.

There are more modular exponentiation algorithms ([12], [13], [14], [15], [16], [17]) trying to defeat all the physical attacks ([18], [19], [20], [4], [21]) that threaten the security of the cryptosystems, but here, we focus our attention only on the blinded form of the Montgomery ladder algorithm and on the goal of avoiding Jacobi symbol attacks.

2. PRELIMINARIES

2.1 Jacobi Symbol

The first necessary concept is the *quadratic residue*: for a given prime p, a is a quadratic residue if $\gcd(a,p)=1$ and $a=y^2 \bmod p$ for some y. If $\gcd(a,p)=1$ but a is not a quadratic residue mod p, a is called a quadratic non-residue mod p.

$\left(\frac{a}{p}\right)$ is called the *Legendre symbol* of $a \bmod p$, and we can see that

$$\left(\frac{a}{p}\right)=\begin{cases} 1 & \text{If } a \text{ is a quadratic residue mod } p \\ -1 & \text{If } a \text{ is a quadratic non-residue mod } p \\ 0 & \text{If there is a common factor} \end{cases}$$

Now, we have that $\left(\frac{a}{n}\right)=\left(\frac{a}{p_1}\right)\cdots\left(\frac{a}{p_k}\right)$ is the Jacobi symbol, where n is odd, $n=p_1\cdots p_k$, and the p_i are prime factors of n. The Jacobi symbol is a generalization of the Legendre symbol.

2.1 Fault Attacks

Bone, DeMillo and Lipton showed that it is possible to disturb an embedded device while it is executing a cryptographic algorithm [2] and that with the erroneous output value, an attacker can obtain secret information that can break the security of the cryptosystem. A disturbance can be induced, principally, by variation in supply voltage, and it may cause the device to misinterpret data or even skip a complete instruction.

2.2 Montgomery Powering Ladder and its Blinded Form

Many modular exponentiation algorithms have been developed. Joye and Yen proposed a new kind of algorithm to calculate the modular exponentiation, called the *Montgomery powering ladder* [5]. Their model was based on a different idea from those algorithms designed before it. The principal concept was that

$$L_j=\sum_{i=j}^{t-1} d_i 2^{i-j} \text{ and } H_j=L_j+1$$

Some characteristics of the Montgomery ladder introduced in [5] are as follows:

- The algorithm is highly regular; that is, there is always a multiplication followed by a squaring, regardless of the processed bit.
- $R[1]/R[0]=m$ is invariant throughout the execution of the algorithm.
- The two multiplications performed at each iteration share a common operand, for which the *Common-multiplicand multiplication* [22] can be used.
- The two multiplications performed are independent at each iteration, and therefore, they can be calculated in parallel form.

The Montgomery ladder was improved by Fumaroli and Vigilant (FV scheme), who added a random element r to protect the algorithm; they used one more register than the simple Montgomery ladder to save the inverse value of the random element r (Algorithm 1).

Algorithm 1 FV scheme

```
1: Input $m \in G$ , $d=(d_{n-1} \dots d_0)_2$
2: Output $s=m^d \in G$
3: $R[0] \leftarrow r$ ; $R[1] \leftarrow m \cdot r$
4: $R[2]=r^{-1}$
5: for $n-1$ to $0$ do
6:        $R[\overline{d_i}] \leftarrow R[\overline{d_i}] \cdot R[d_i] \bmod N$
7:        $R[d_i] \leftarrow R[d_i] \cdot R[d_i] \bmod N$
8:        $R[2]=R[2] \cdot R[2] \bmod N$
9: end for
10: Return $R[0] \cdot R[2] \bmod N$
```

2.3 Attacks Based on the Jacobi Symbol

In 2006, Boreale proposed a new kind of attack against the modular exponentiation, implemented over the *binary square-and-multiply right-to-left algorithm* (Algorithm 2) [10]. He put a fault z in $R[1]$ when a squaring is executed in the iteration $i-1$ of the *for* loop. Then, depending on the value of (S/N), where S is the attacked output value, it can be possible to determine the value of the bit d_i. This scheme works by assuming that $(m/N)=1$, where m is the input value, and its behavior is similar to the *Safe error* : if the value of the bit in the i-th iteration is equal to 0, the fault does not affect the calculation of the JS of $(R[0]_i/N)=1$, but if $d_i=1$, z affects the register $R[0]_i$ which can provoke a JS value of $(R[0]_i/N)=-1$,and thus a JS value of $(S/N)=-1$. Here, two options are given: if (S/N) is always equal to 1, then $d_i=0$, but if $(S/N)=-1$, d_i is equal to 1. Thus, an enemy can deduce the secret key of the cryptosystem.

Table 1 shows the behavior of algorithm 2 under the attack described by Boreale. In the example, it was assumed that $(m/N)=1$, $(z/N)=-1$, and $d=25=11001$.

In 2010, Schmidt [11] proposed an attack that consisted of giving a message m with $(m/N)=-1$ to the FV scheme and skipping the operation $R[d_i]=R[d_i]^2$. Then, observing the resulting value could identify the values of d_i and d_{i+1}. If $(S/N)=-1$, then $d_i=d_{i+1}$. The procedure of this attack is shown as algorithm 3.

An example of the attack described in the algorithm 3 against the FV scheme is observed in table 2. In this example, it was supposed that $(m/N)=-1$ and $d=19=10011$.

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

```
1: Input $m \in G$ , $d = (d_{n-1} \ldots d_0)_2$
2: Output $s = m^d \in G$
3: $R[0] \leftarrow 1;\ R[1] \leftarrow m$
4: for $0$ to $n-1$ do
5:        if $d_i = 1$ then
6:            $R[0] \leftarrow R[0] \cdot R[1] \bmod N$
7:        end if
8:         $R[1] \leftarrow R[1]^2 \bmod N$
9: end for
10: Return $R[0]$
```

i	d_i	Intermediate products	Jacobi symbol
0	1	$R[0] = m$ $R[1] = (m)^2 = m^2$	$(R[0]/N) = (1) =$ 1 $(R[1]/N) = (1) =$ 1
1	0	$R[0] = m$ $R[1] = (m^2)^2 = m^4$	$(R[0]/N) = (1) =$ 1 $(R[1]/N) = (1) =$ 1
2	0	$R[0] = m$ $R[1] = (m^4)^2 = m^8 = z$	$(R[0]/N) = (1) =$ 1 $(R[1]/N) = (-1) =$ -1
3	1	$R[0] = m \cdot z$ $R[1] = (z)^2 = z^2$	$(R[0]/N) = (1)(-1) =$ -1 $(R[1]/N) = (1) =$ 1
4	1	$R[0] = m \cdot z^3$ $R[1] = (z^2)^2 = z^4$	$(R[0]/N) = (1)(-1) =$ -1 $(R[1]/N) = (1) =$ 1

TABLE 1: Algorithm 2 performed with a JS attack, FA in $i-1$ and $d_i = 1$.

Algorithm 3 Attack proposed in [11]

```
1: Ensure Exponent $d = (d_{n-1} \ldots d_0)_2$ is used by the device.
2: Set $d_{n-1} = 1$
6: for $n-2$ to $0$ do
5:        Chose $m \in Z_N$ with $\left(\frac{m}{N}\right) = -1$
6:        Calculate $S$ with the $i$-th squaring skipped
6:        if $\left(\frac{S}{N}\right) = -1$ then
7:            $d_i = d_{i+1}$
8:        else
9:            $d_i = 1 \oplus d_{i+1}$
10:       end if
11: end for
12: Return $d$
```

i	d_i	Intermediate products	Jacobi symbol
4	1	$R[0] = m \cdot r^2$ $R[1] = m^2 \cdot r^2$	$(R[0]/N) = (-1)(1) = -1$ $(R[1]/N) = (1)(1) = 1$
3	0	$R[0] = (m \cdot r^2)^2 = m^2 \cdot r^4 = FA$ $R[1] = m^3 \cdot r^4$	$(R[0]/N) = (1)(1) = 1$ $(R[1]/N) = (-1)(1) = -1$
2	0	$R[0] = (m \cdot r^2)^2 = m^2 \cdot r^4$ $R[1] = m^3 \cdot r^4 \cdot m \cdot r^2 = m^4 \cdot r^6$	$(R[0]/N) = (1)(1) = 1$ $(R[1]/N) = (1)(1) = 1$
1	1	$R[0] = m^6 \cdot r^{10}$ $R[1] = (m^4 \cdot r^6)^2 = m^8 \cdot r^{12}$	$(R[0]/N) = (1)(1) = 1$ $(R[1]/N) = (1)(1) = 1$
0	1	$R[0] = m^{14} \cdot r^{22}$ $R[1] = (m^8 \cdot r^{12})^2 = m^{16} \cdot r^{24}$	$(R[0]/N) = (1)(1) = 1$ $(R[1]/N) = (1)(1) = 1$

TABLE 2: Algorithm 1 executed with FA, where $d_{i+1} \neq d_i$.

In table 2, it can be noted that a modular multiplication in $i-1$ must be performed by two elements with odd exponents to obtain a result with an even exponent and so obtain $(S/N)=1$, which is the key point of the Schmidt's attack. This situation is observed when the modular multiplication $R[1]_{i=2} = R[1]_{i=3} \cdot R[0]_{i=4}$ is calculated after skipping the squaring operation $R[0]_{i=3}$.

The two attacks mentioned above are easy to implement and powerful because they only need to know about the Jacobi symbol in the returned value by the attacked algorithm.

3. PROPOSED ALGORITHM

In this section, a modification of the FV scheme is proposed that is secure against Schmidt's attack, and the behavior of the proposed algorithm is explained.

3.1 Algorithm

In the approach proposed by Schmidt to attack the FV scheme, the idea is not to put a random value z in the execution but to skip a complete squaring operation in the iteration i when the algorithm is being executed. Then, depending on the value of (S/N), it can be determined whether $d_{i+1} = d_i$.

It can be noted that only even intermediate exponents, through an algorithm, can be used to calculate any modular exponentiation. On the basis of this observation, algorithm 4 is proposed. It can be seen that this algorithm begins the register $R[1]$ with an even exponent $R[1] = m \cdot m = m^2$. This even exponent will affect all the calculations through the algorithm, and thus, it will affect the JS of all the intermediate values calculated by algorithm 4. Here, odd values d are considered.

In algorithm 4, it can be seen that the loop is not executed from $n-1$ to 0 but from $n-1$ to 1, because of the behavior of the algorithm; this behavior will be explained in section 3.2. It can be noted that only the value in $R[1]$ was altered, whereas no extra value was placed in $R[0]$.

Algorithm 4 guarantees that when an attacker skips one squaring operation, in any iteration of the loop, he will not be able to obtain any relevant information about the bits of the string of d, because to obtain any information, it is necessary to have in the output value $(S/N)=1$ or $(S/N)=-1$ depending on the value of the attacked bits d_{i+1} and d_i. However, the output value of algorithm 4 will always be $(S/N)=1$ if $(m/N)=1$ and $(S/N)=-1$ if $(m/N)=-1$, regardless of the values of d_{i+1} and d_i.

Algorithm 4 Modified FV scheme

```
1: Input $m \in G$ , $d=(d_{n-1}\dots d_0)_2$
2: Output $s=m^d \in G$
3: $R[0] \leftarrow r$
4: $R[1] \leftarrow m^2 \cdot r$
5: $R[2]=r^{-1}$
6: for $n-1$ to $1$ do
7:       $R[\overline{d_i}] \leftarrow R[\overline{d_i}] \cdot R[d_i] \bmod N$
8:       $R[d_i] \leftarrow R[d_i] \cdot R[d_i] \bmod N$
9:       $R[2]=R[2] \cdot R[2] \bmod N$
10: end for
11: $R[0]=R[0] \cdot m$
12: Return $R[0] \cdot R[2] \bmod N$
```

All the values obtained in the intermediate steps of algorithm 4 have an even exponent, and obviously, all of them are quadratic residues; therefore, they have a JS equal to 1. Now, in line 11 of algorithm 4, it is possible to see that the register $R[0]$ is altered by the operation $R[0]=R[0]_{i=1} \cdot m$, where $R[0]_{i=1}$ is the resulting value of the iteration $i=1$ of the *for* loop (lines 6 to 10 of algorithm 4). All the values calculated through the *for* loop have a JS equal to 1, and therefore, the JS of $R[0]_{i=1}$ is equal to 1. For that reason, the JS of the returned value depends of the JS of m, disregarding completely the values of d_{i+1} and d_i, because if the JS of m is equal to 1, then $R[0]=R[0]_{i=1} \cdot m=(1)\cdot(1)=1$ (considering only JS values), and if the JS of m is equal to -1, then $R[0]=(1)\cdot(-1)=-1$.

As shown in table 2, elements with even exponents (quadratic residues) and with odd exponents (quadratic non-residues) are needed in the intermediate products to deduce the binary string of d. Thus, the proposed countermeasure is a protection against the Jacobi symbol attack, because the execution of algorithm 4 has only even exponents in the intermediate products. This protection is observed in table 3. In this example, it was supposed that $d=39=100111$ and $(m/N)=-1$.

As shown in table 3, all the JS values of the intermediate steps in the algorithm are equal to 1, and it does not matter if $(m/N)=1$ or if $(m/N)=-1$.

3.2 Behavior of the Proposed Algorithm

The modular exponentiation m^d, where $d=\sum_{i=0}^{n-1} d_i 2^i$ and $d_i \in \{0,1\}$, can be represented by

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

If equation (1) is calculated using algorithm 1, it is possible to know that the last iteration of algorithm 1 can be represented by equation (2), which is the correct result of m^d

i	d_i	**Intermediate products**	**Jacobi symbol**
5	1	$R[0] = m^2 \cdot r^2$ $R[1] = (m^2 \cdot r)^2 = m^4 \cdot r^2$	$(R[0]/N) = (1)(1) = 1$ $(R[1]/N) = (1)(1) = 1$
4	0	$R[0] = (m^2 \cdot r^2)^2 = m^4 \cdot r^4$ $R[1] = m^6 \cdot r^4$	$(R[0]/N) = (1)(1) = 1$ $(R[1]/N) = (1)(1) = 1$
3	0	$R[0] = (m^4 \cdot r^4)^2 = m^8 \cdot r^8 = FA$ $R[1] = m^{10} \cdot r^8$	$(R[0]/N) = (1)(1) = 1$ $(R[1]/N) = (1)(1) = 1$
2	1	$R[0] = m^4 \cdot r^4 \cdot m^{10} \cdot r^8 = m^{14} \cdot r^{12}$ $R[1] = (m^{10} \cdot r^8)^2 = m^{20} \cdot r^{16}$	$(R[0]/N) = (1)(1) = 1$ $(R[1]/N) = (1)(1) = 1$
1	1	$R[0] = m^{34} \cdot r^{28}$ $R[1] = (m^{20} \cdot r^{16})^2 = m^{40} \cdot r^{32}$	$(R[0]/N) = (1)(1) = 1$ $(R[1]/N) = (1)(1) = 1$

TABLE 3: Algorithm 4 executed with JS attack where $d_{i+1} = d_i$.

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

Now, it can be supposed that algorithm 4 is executed from d_{n-1} to d_0. Then, the modular exponentiation is represented by

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

The behavior of equation (3) through algorithm 4 is given by equations (4) to (7), where each step represents an iteration and $k = n-1-i$.

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

$$\vdots \qquad \vdots$$

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

$$\vdots \qquad \vdots$$

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

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

Note that equation (2) is very similar to equation (6). Now, if the last squaring and the last multiplication by $m^{2(d_0)}$ of equation (6) are deleted, then equation (8) is obtained

$$m^{2^{n-1}(d_{n-1})+2^{n-2}(d_{n-2})+\cdots+2^1(d_1)} \tag{8}$$

If equation (8) is multiplied by $m^{(d_0)}$, the correct result of the operation m^d has been obtained. Therefore, algorithm 4 is executed from $n-1$ to 1 (the last squaring and the last multiplication by

$m^{2(d_0)}$ are deleted), and it is necessary to multiply by m in line 11 of algorithm 4 (the multiplication by $m^{(d_0)}$ is made, but it is supposed that $d_0 = 1$, and thus, $m^{(d_0)} = m$).

Equations (9) and (10) are given only to show the relationship between the registers of algorithms 1 and 4, where $R[0]_{(o)_i}$ and $R[1]_{(o)_i}$ are the registers of algorithm 1 running from $n-1$ to 0; $R[0]_{(p)_i}$ and $R[1]_{(p)_i}$ are the registers of algorithm 4 running from $n-1$ to 1; and $d_{(p)}$ is a bit of the exponent in algorithm 4 at the iteration $i-1$.

$$\text{If } d_{(p)_{i-1}} = 0, \text{ then} \begin{cases} \mathrm{R}[0]_{(p)_\mathrm{i}} = R[0]_{(o)_{i-1}} \\ \mathrm{R}[1]_{(p)_\mathrm{i}} = R[1]_{(o)_{i-1}} \cdot m \end{cases} \tag{9}$$

$$\text{If } d_{(p)_{i-1}} = 1, \text{ then} \begin{cases} \mathrm{R}[0]_{(p)_\mathrm{i}} = R[0]_{(o)_{i-1}} \cdot m^{-1} \\ \mathrm{R}[1]_{(p)_\mathrm{i}} = R[1]_{(o)_{i-1}} \end{cases} \tag{10}$$

3.3 Expansion of the Algorithm

Up to this point, the discussion has addressed an algorithm that is effective when the keys d are odd, but it is possible to use algorithm 4 for all types of d values, by adding a few lines. The resulting algorithm is given below as algorithm 5.

Algorithm 5 can be used not only with odd keys, given as exponents, but also with even keys. To understand this option, recall that it is necessary to multiply the value m^{d_0} (where d_0 determines if a key is odd or even) by equation (8) to obtain the correct result of m^d, but $d_0 \in \{0,1\}$. If $d_0 = 1$, equation (8) is multiplied by $m^1 = m$, and if $d_0 = 0$, equation (8) is multiplied by $m^0 = 1$. Therefore, the *if statement* in algorithm 5 allows the algorithm to work with any kind of secret key d.

Algorithm 5 Modified FV scheme to counteract JS attack and to work with any exponent

```
1: Input $m \in G$ , $d = (d_{n-1} \ldots d_0)_2$
2: Output $s = m^d \in G$
3: $R[0] \leftarrow r$
4: $R[1] \leftarrow r \cdot m^2$
5: $R[2] = r^{-1}$
6: for $n-1$ to $1$ do
7:         $R[\overline{d_i}] \leftarrow R[\overline{d_i}] \cdot R[d_i] \bmod N$
8:         $R[d_i] \leftarrow R[d_i] \cdot R[d_i] \bmod N$
9:         $R[2] = R[2] \cdot R[2] \bmod N$
10: end for
11: if $d_0 = 1$ then
12:         $R[0] = R[0] \cdot m$
13: end if
12: Return $R[0] \cdot R[2] \bmod N$
```

Algorithm 5 uses more lines than algorithm 4; thus, when the d values are always odd numbers, algorithm 4 is recommended, and when the d values can be either odd or even numbers, algorithm 5 can be used.

4. CHARACTERISTICS OF THE PROPOSED ALGORITHM

The proposed algorithm is highly regular: there is always a multiplication followed by a squaring regardless of the processed bit. The relation between the registers $R[1]/R[0]=m^2$ is invariant throughout the execution of the algorithm.

Table 4 compares some characteristics of the proposed algorithm against the characteristics of other similar algorithms. Table 4 shows the number of registers and the average number of multiplications executed by the proposed algorithm, compared with algorithms derived from the original Montgomery powering ladder and the square-and-multiply algorithms. In table 4, the squarings are considered multiplications; the *if statements* are not considered; and n is the bit length of the exponent.

Algorithms	Number of registers	Average number of multiplications
Square-and-multiply left-to-right	1	1.5*n*
Square-and-multiply right-to-left	2	1.5*n*
Montgomery powering ladder	2	2*n*
Giraud's algorithm	2	2*n*
FV scheme	3	3*n*
Proposed algorithm	3	3*n*

TABLE 4: Comparison of the number of registers and the average number of multiplications executed by algorithms based on the Montgomery powering ladder and the square-and-multiply algorithms.

According to table 4, the proposed algorithm has disadvantages in runtime and number of registers compared with similar algorithms; however, these disadvantages are countered by the security characteristics of the proposed algorithm. Section 4.1 shows the level of security of the proposed technique with respect to other algorithms.

4.1 Security

Simple Power Analysis (SPA) [23] can recognize in a power trace, obtained from a device which executes a cryptographic algorithm, when a bit is equal to 0 and when it is equal to 1 if there are operations that depend on the bit's value being processed. The square-and-multiply algorithm is vulnerable to SPA because it has a conditional branch during its execution. The proposed algorithm does not have conditional operations and is therefore secure against SPA.

Because dummy operations are used in the square-and-multiply always algorithm, it can be attacked with the SEA, which consists of inducing a fault during the execution of the algorithm. If the fault affects a dummy operation ($d_i=0$), the output result will not be altered, but if the fault affects a necessary operation ($d_i=1$), the output result will be altered. Thus, an attacker can determine when a bit equal to 0 was attacked. The proposed algorithm does not have dummy operations that can be attacked and is thus resistant to the SEA.

To break the security of a cryptosystem with *Differential Power Analysis* (DPA) [23], it is necessary to collect many power traces of the same algorithm with different input values and perform a statistical analysis over them. The algorithm proposed by Giraud [6] and the Montgomery powering ladder are vulnerable to DPA, but the value r used by the proposed algorithm helps to avoid DPA.

RDA is an attack that uses two related messages M and M^2, and by observing the relationship between the two messages through the execution of the same algorithm, it can obtain the secret key of the cryptosystem. This attack was developed against the Montgomery powering ladder, but Giraud's algorithm is also vulnerable to it. The FV scheme and the proposed algorithm are resistant to this attack because the random value r breaks the relationship between M and M^2.

Kim and Quisquater showed the possibility of inducing two faults during the same execution of an algorithm [19]: the first fault to corrupt a register and the second fault to avoid an operation (such as a coherence test). Under this scheme, the algorithm proposed by Giraud can be vulnerable to the JS attack proposed by Schmidt because the coherence test will not be performed. Thus, the Giraud's algorithm will not recognize that the relationship between the registers has been lost, and an attacker can calculate the JS of the erroneous value, obtaining useful information[1].

It has been shown that the proposed algorithm is secure against the attack proposed by Schmidt and Medwed, whereas the FV scheme, Giraud's algorithm, and the Montgomery powering ladder are vulnerable against that attack.

As demonstrated in this section, the proposed algorithm offers better security than that offered by the other algorithms mentioned here.

5. COMMENTS

There is a concept that can be used to protect algorithms against this kind of attacks: by changing a quadratic non-residue value into a quadratic residue value (or working only with quadratic residue values through an algorithm, such as the proposed algorithm), it is possible to prevent an attacker from using a JS attack against a cryptographic algorithm.

As examples, the algorithms square-and-multiply right-to-left (SaM RtL) and square-and-multiply left-to-right (SaM LtR) are considered. As stated in section 2.3, Boreale attacked the SaM RtL algorithm (Algorithm 2). In this attack, if the squaring $R[1]_{i-1} = R[1]^2$ in the iteration $i-1$ is corrupted with a value z, where $(z/N) = -1$, and if the value of the bit in the i-th iteration is equal to 1, the JS value $(z/N) = -1$ will affect the operation $R[0] \leftarrow R[0] \cdot R[1] \bmod N$ in the i-th iteration, then $R[0]_i = (1) \cdot (-1) = -1$. (It is supposed that $(m/N) = 1$). Henceforth, the register $R[0]$ will have a JS equal to -1, a value that can be exploited by an attacker.

On the other hand, the SaM LtR (algorithm 6) cannot be attacked using Boreale's attack, because if it is placed an error in any operation $R[0] = R[0] \cdot m$ or $R[0] \leftarrow R[0]^2$ in the i-th iteration such that $(R[0]_i / N) = -1$ (It is supposed that $(m/N) = 1$), the operation $R[0]_{i-1} = R[0]_i^{\,2}$ will convert the JS value $(R[0]_i / N) = -1$ to $(R[0]_{i-1} / N) = 1$ in the next iteration of the algorithm. In other words, the operation $R[0]_{i-1} = R[0]_i^{\,2}$ will convert a quadratic non-residue value into a quadratic residue value, and this process will be repeated in each step of the *for* loop, which will avoid any kind of JS attack because there will be no any JS value that can be used to obtain relevant information about the cryptosystem.

Thus, the SaM LtR is intrinsically secure against JS attacks, because it converts any quadratic non-residue value into a quadratic residue value through its execution.

6. FUTURE WORK

Here, the blinded Montgomery ladder exponentiation algorithm has been protected against the Jacobi symbol attack. The modification of algorithm 1 was developed according to its specific characteristics, and according to the fault model used over it, but each modular exponentiation algorithm in the literature has different characteristics. To extend our results, we will develop forms to protect other algorithms that are vulnerable to the JS attack and that have different characteristics, such as the algorithms *Add only* and *Add always*, which were presented by Marc Joye in [24] and attacked in 2010 by Kim [25].

[1] Dottax et al. have proposed a method to resist the double-fault attack in [26].

Algorithm 6 Square-and-multiply left-to-right

```
1: Input $m \in G$ , $d = (d_{n-1} \ldots d_0)_2$
2: Output $s = m^d \in G$
3: $R[0] \leftarrow 1$
4: for $n-1$ to $0$ do
5:  $R[0] \leftarrow R[0]^2 \bmod N$
6:       if $d_i = 1$ then
7:            $R[0] \leftarrow R[0] \cdot m \quad \bmod N$
8:       end if
9: end for
10: Return $R[0]$
```

7. CONCLUSIONS

In this paper, we have proposed an algorithm that is secure against the attack proposed by Schmidt and Medwed. It has disadvantages in runtime and space compared to similar algorithms, but it also provides a higher level of security than these other algorithms.

Acknowledgments: We wish to thank the referee for carefully reading this paper and for his constructive suggestions. This paper was in part supported by the PACIVE project GC-19 of the FES-C UNAM.

8. REFERENCES

[1] P. Kocher. “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*, 1996, pp. 104-113.

[2] D. Boneh, R. DeMillo and R. Lipton. “On the importance of checking cryptographic protocols for faults.” *In Fumy, W., Ed.: Advances in Cryptology-EUROCRYPT ’97. Volume 1233 of Lecture Notes in Computer Science*, 1997, pp. 37-51.

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

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

[5] M. Joye and S.M. Yen. “The montgomery powering ladder.” *In Cryptographic Hardware and Embedded Systems-CHES 2002, 2523 of Lecture Notes in Computer Science*, 2003, pp. 291-302.

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

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

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

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

[10] M. Boreale. “Attacking right-to-left modular exponentiation with timely random faults.” Fault *Diagnosis and Tolerance in Cryptography, 4236 of LNCS, pp.* 24-35, 2006.

[11] J. M. Schmidt and M. Medwed. “Fault attacks on the montgomery powering ladder”. *Information Security and Cryptology ICISC-2010,* pp. 396-406*,* 2011.

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

[13] C.C. Lu, S.Y. Tseng, and S.K. Huang. “A secure modular exponential algorithm resists to power, timing, c safe error and m safe error attacks.” *In 19th International Conference on Advanced Information Networking and Applications, 2005. AINA 2005, pp.* 151-154.

[14] C.H. Kim and J.J. Quisquater. “How can we overcome both side channel analysis and fault attacks on rsa-crt?.” *Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 21–29, 2007.

[15] A. Boscher, R. Naciri, and E. Prouff. “Crt rsa algorithm protected against fault attacks.” *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems, 4462 of LNCS*, pp.229-243, 2007.

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

[17] A. Boscher, H. Handschuh, and E. Trichina. “Blinded fault resistant exponentiation revisited.” *In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, Workshop on Fault Diagnosis and Tolerance in Criptography - FDTC’09*, 2009, pp. 3-9.

[18] S.M. Yen, W.C. Lien, S.J. Moon, and J.C. Ha. “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*, 2005, pp. 183-195.

[19] C. Kim and J.J. Quisquater. “Fault attacks for crt based rsa: New attacks, new results, and new countermeasures.” *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems, 4462*, pp. 215-228, 2007.

[20] S. Chari, J. Rao, and P. Rohatgi. “Template attacks.” *Cryptographic Hardware and Embedded Systems-CHES 2002, 2523 of Lecture Notes in Computer Science*, 2002, pp. 12–28.

[21] S.M. Yen and M. Joye. “Checking before output may not be enough against fault-based cryptanalysis.” *IEEE Transactions on Computers, 49(9), pp.* 967-970, 2000.

[22] S.M. Yen and C.S Laih. “Common-multiplicand multiplication and its application to public-key cryptography.” *Electronic Letters, 29(17),* pp. 1583-1584, August 1993.

[23] P.C. Kocher, J. Jaffe, and B. Jun. “Differential Power Analysis.” *In Wiener, M., Ed.: Advances in Cryptology-CRYPTO ’99. Volume 1666 of Lecture Notes in Computer Science*, Springer 1999, pp. 388-397.

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

[25] C.H. Kim. “New fault attacks using jacobi symbol and application to regular right-to-left algorithms.*” Information Processing Letters, 110(20)*, pp. 882-886, 2010.

[26] E. Dottax, C. Giraud, M. Rivain, and Y. Sierra. “On second-order fault analysis resistance for CRT-RSA implementations.*” Information Security Theory and Practice. Smart Devices, Pervasive Systems, and Ubiquitous Networks,* pp. 68-83, Springer 2009.