Countermeasure against the Jacobi symbol attack

DAVID TINOCO VARELA
Computational Science Graduate Program
Universidad Nacional Autónoma de México, Campus Cuautitlán.
Av. 1 de Mayo s/n. Col. Atlanta. 54740 Cuautitlán Izcalli, Edo. Mex.
MEXICO
dativa19@comunidad.unam.mx

Abstract: Many physical attack types (Timing attacks, Power consumption attacks, Fault attacks, etc.) have been developed against cryptosystems in the recent years. Indeed there is a real necessity to eliminate the vulnerabilities of the cryptosystems, like CRT-RSA or the Elliptic Curve Cryptosystem, that make them susceptible to those attacks. In 2006 Boreale described a new type of physical attack which is based in the Jacobi symbol concept. In this paper a countermeasure against the Jacobi symbol attack is presented and implemented in two modular exponentiation algorithms to make them immune to such attack.

Key–Words: Cryptography, Security, Modular exponentiation algorithms, Side channel attacks, Jacobi symbol, Embedded devices.

1 Introduction

Kocher [16] 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 get the binary chain by just observing the power traces or the timing consumption in an electronic device such as an oscilloscope, the SCA's are, first of all, used to attack the modular exponentiation(*Add and double* is the analogous function in the Elliptic Curve Cryptosystems, ECC) which is the core operation in cryptosystems like RSA.

SCA's opened a door to a new type of physical attacks, one of those was the *Fault Attack* (FA) proposed by *Bonhe, DeMillo* and *Lipton* [1], the FA's were more aggressive than SCA's because FA's disturb physically the execution of the device which is running up the cryptographic algorithm.

In order to prevent SCA's and FA's many modular exponentiation algorithms have been created, *Coron* [6] gave the first algorithm specifically designed to defeat the SCA's to the world with the *Square-and-Multiply Always algorithm*, but this algorithm was attacked by the denominated *Safe Error Attack*(SEA) [21].

The *Montgomery powering ladder* [12] was a new idea proposed by *Joye* and *Yen* to protect cryptosystems against SCA's and FA's, the algorithm works in a regular form, this is, it doesn't matter what is the value of the bit being processed (0 or 1), the algorithm always will calculate a multiplication followed by a squaring. Montgomery ladder had a good acceptation and attracted the attention of many researchers. *Giraud* [9] modified the Mongomery ladder in order to protect it against FA's , 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".

Montgomery ladder was attacked by the *Relative Doubling Attack* (RDA) [22], a modification of the *Doubling Attack* (DA) [7], but *Fumaroli* and *Vigilant* (FV) [8] added a random value to the Montgomery ladder to blind the exponentiation of the algorithm. The algorithm proposed by FV was secure against SCA, DA, RDA, and in a partial form against FA, but *Kim* and *Quisquater* [15] pointed out that FV scheme has a vulnerability when a FA is induced during the squaring computation of the inverse of the random number r.

A new type of attack was presented by *Boreale* in 2006 [2], his attack uses a combination between FA and SCA, and it is possible to get the binary chain of the secret key d using the *Jacobi symbol* (JS). He used his model in the *Right-to-Left modular exponentiation algorithm* and he proved that the attack is effective even in the presence of message blinding. *Schmidt* and *Medwed* [19] used the Ja-

cobi symbol to create an attack which breaks the security of the Montgomery ladder in its blinded form.

There are more modular exponentiation algorithms ([18], [17], [15], [4], [10], [3]) trying to defeat all the SCA's ([23], [14], [5], [21], [20]) that threaten the security of the cryptosystems, but here we only focuses our attention on the Montgomery ladder algorithm and the Square-and-Multiply Right-to-Left algorithm.

2 Preliminaries

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

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

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

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 $n = p_1 \cdots p_k$ and p's are prime factors. The Jacobi symbol is a generalization of the Legendre symbol.

On the other hand, many modular exponentiation algorithms have been developed, but *Joye* proposed a new kind of algorithm which calculates the modular exponentiation called *Montgomery powering ladder* [12]; his model was based on a different idea to those algorithms designed before it. The principal concept was that

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

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

In 2006 *Boreale* gave a new kind of attack against the modular exponentiation in the binary *Square and Multiply Right-to-Left* algorithm (Algorithm 2) [2]. He put a fault z in $R[1]$ when an squaring is executed in the iteration $i-1$ then, depending of the value of $\left(\frac{S}{N}\right)$, where S is the attacked output value, can be known what value of the bit d_i was attacked. This scheme works assuming that $\left(\frac{m}{N}\right) = 1$, where m is the input value, and its behavior is similar to the *Safe error*: if the bit in the i-th iteration is equal to 0, the fault does not affect the calculus of the JS of $(R[0]_i/N) = 1$, and z is squaring what results in the JS of $(z^2/N) = 1$, but if $d_i = 1$, z affects the register $R[0]_i$ and it can be a value $(R[0]_i/N) = -1$. The value z can be or cannot be a quadratic residue, if z is a quadratic residue the final result will be $(S/N) = 1$, but if it is a quadratic non-residue the result will be $(S/N) = -1$, for this reason the Boreale's attack is a probabilistic model.

Algorithm 1 FV scheme.

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

Algorithm 2 Square and Multiply Right-to-Left.

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

In table 1 it is shown the behavior of the algorithm 2 under the attack described by *Boreale*, in the example it was assumed that $\left(\frac{m}{N}\right) = 1$, $\left(\frac{z}{N}\right) = -1$, and $d = 25 = 11001$.

In 2010 *Schmidt* [19] proposed an attack consisting in giving a message m with $(m/N) = -1$ to the

Algorithm 2 performed with a JS attack, FA in $i-1$ and $d_i = 1$.			
i	d_i	Intermediate products	Jacobi symbol
0	1	$R[0] = m$	$(R[0]/N) = (1) = 1$
		$R[1] = (m)^2 = m^2$	$(R[1]/N) = (1) = 1$
1	0	$R[0] = m$	$(R[0]/N) = (1) = 1$
		$R[1] = (m^2)^2 = m^4$	$(R[1]/N) = (1) = 1$
2	0	$R[0] = m$	$(R[0]/N) = (1) = 1$
		$R[1] = (m^4)^2 = m^8 = z$	$(R[1]/N) = (-1) = -1$
3	1	$R[0] = m \cdot z$	$(R[0]/N) = (1)(-1) = -1$
		$R[1] = (z)^2 = z^2$	$(R[1]/N) = (1) = 1$
4	1	$R[0] = m \cdot z^3$	$(R[0]/N) = (1)(-1) = -1$
		$R[1] = (z^2)^2 = z^4$	$(R[1]/N) = (1) = 1$

Table 1: Algorithm 2 performed with a JS attack.

FV scheme and skipping the operation $R[d_i] = R[d_i]^2$ then, observing the resulting value it can be learned what the value of d_i and d_{i+1} was, if $(S/N) = -1$ then $d_i = d_{i+1}$, the scheme of this attack is given as algorithm 3.

Algorithm 3 Attack proposed in [19].

1: **Ensure** Exponent $d = (d_{n-1}, \cdots, d_0)$ that is used by the device.
2: Set $d_{n-1} = 1$
3: **for** $n-2$ to 0 **do**
4: Choose $m \in Z_N$ with $(\frac{m}{N}) = -1$
5: Calculate S with the ith squaring operation skipped
6: **if** $\left(\frac{S}{N}\right) = -1$ **then**
7: $d_i = d_{i+1}$
8: **else**
$d_i = 1 \oplus d_{i+1}$
9: **end if**
10: **end for**
11: **Return** d

An example of the attack described in the algorithm 3 over the FV scheme can be seen in table 2, in this example it was supposed that $(\frac{m}{N}) = -1$ and $d = 19 = 10011$.

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

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

3 Proposed countermeasure

3.1 Right-to-Left algorithm modified

It is already known that the binary Right-to-Left algorithm is broken by many attacks like SPA and DPA, but here it will only be taken into account the JS attack and it will be given a countermeasure against it.

Algorithm 2 has been modified to obtain a secure algorithm against JS attack and then the algorithm 4 has been obtained .

Algorithm 4 *Square and Multiply Right-to-Left* modified to counteract JS attack.

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

It has been learned that when a z with $(z/N) = -1$ is introduced in the iteration $i-1$ and $d_i = 1$, z will affect $R[0]$ in a way that

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

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

$(R[0]_i/N) = -1$, this is because z is a quadratic non-residue, however, if we can convert the quadratic non-residue into quadratic residue the attack proposed in [2] will not affect the execution of the algorithm because all the calculations involved trough the algorithm 4 would have an JS=1. This is the idea behind the algorithm 4, make any quadratic non-residue into a quadratic residue for what the output value in each execution of the algorithm, regardless of whether there existed or not a *Fault*[1], will be 1.

The operation $R[0] \leftarrow R[0] \cdot R[1]^2 \ mod \ N$ in the algorithm 4 will guarantee that all attacks over $R[1]$ in iteration $i-1$ will not affect the JS of $R[0]_i$ because in iteration i, if $d_i = 1$, the operation $R[1]_{i-1} = z$ is squared belonging in a quadratic residue. For that reason the output always will be 1 if $(\frac{m}{N}) = 1$ and -1 if $(\frac{m}{N}) = -1$. The behavior of the algorithm 4 is observed in the table 3, table 3 has the same input values as table 1.

Algorithm 2 is started with $R[0] = 1$ and $R[1] = m$, and then the values $R[0] = m$ and $R[1] = m^2$ are always obtained if $d_i = 1$ in the first iteration, but in the algorithm 4 it can be noted that the input values are $R[0] = R[1] = m$, and apparently the first iteration has been eliminated, however it can be observed that only the first squaring was eliminated, this means that the first operation $R[1]_{i=0} = R[1]^2$ was skipped. Now it can be observed that the first squaring operation skipped $R[1]_0$ is in the operation $R[0]_i \leftarrow R[0] \cdot R[1]^2_{i-1} \ mod \ N$ in the iteration $i = 1$.

The input values in algorithm 4 are used because it is assumed that the exponent d is an RSA exponent where $d_0 = d_{n-1} = 1$ in its binary representation.

[1]It is considered that m is $\left(\frac{m}{N}\right) = 1$, if m is $\left(\frac{m}{N}\right) = -1$ the countermeasure will guarantee the output value $(S/N) = -1$.

3.2 FV scheme modified

In the threat 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, depending of what is the value of (S/N), it can be determined whether $d_{i+1} = d_i$ or not.

The proposed countermeasure against this attack consists in start $R[0] = 1$ and $R[1] = m \cdot m = m^2$. This countermeasure is presented as algorithm 5.

Algorithm 5 FV scheme modified.

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: $R[1] \leftarrow m^2 \cdot r$
5: $R[2] \leftarrow r^{-1}$
6: **for** $n-1$ to 1 **do**
7: $\quad R[\bar{d}_i] \leftarrow R[\bar{d}_i] \cdot R[d_i] \ mod \ N$
8: $\quad R[d_i] \leftarrow R[d_i] \cdot R[d_i] \ mod \ N$
9: $\quad R[2] \leftarrow R[2] \cdot R[2]$
10: **end for**
11: $R[0] = R[0] \cdot m$
12: **Return** $R[0] \cdot R[2]$

In the algorithm 5, it can be seen that it is not executed the loop from $n-1$ to 0 but it is executed from $n-1$ to 1, this is because of the behavior of the algorithm; such behavior will be explained in section 3.4. It can be noted that only the value in $R[1]$ was

Algorithm 4 performed with a JS attack, FA in $i-1$ and $d_i = 1$.			
i	d_i	Intermediate products	Jacobi symbol.
0	1	$R[0] = m$ $R[1] = m$	$(R[0]/N) = (1) = 1$ $(R[1]/N) = (1) = 1$
1	0	$R[0] = m$ $R[1] = m^2$	$(R[0]/N) = (1) = 1$ $(R[1]/N) = (1) = 1$
2	0	$R[0] = m$ $R[1] = (m^2)^2 = m^4 = z$	$(R[0]/N) = (1) = 1$ $(R[1]/N) = (-1) = -1$
3	1	$R[0] = m \cdot (z)^2 = m \cdot z^2$ $R[1] = (z)^2 = z^2$	$(R[0]/N) = (1)(-1)^2 = 1$ $(R[1]/N) = (1) = 1$
4	1	$R[0] = m \cdot z^2 \cdot (z^2)^2 = m \cdot z^6$ $R[1] = (z^2)^2 = z^4$	$(R[0]/N) = (1)(1) = 1$ $(R[1]/N) = (1) = 1$

Table 3: Algorithm 4 performed with a JS attack.

altered, but in $R[0]$ it was not placed any extra value.

Algorithm 5 guarantees that when an attacker skips one squaring operation in any iteration in the loop, he will not be able to obtain any relevant information about the bits of the chain of d, because to obtain any information it is necessary to have in the output value $(S/N) = 1$ or $(S/N) = -1$ depending of the value of the bits d_{i+1} and d_i, however the output value of the algorithm 5 will always be $(S/N) = 1$ this is because all elements in the algorithm have an even exponent and obviously all of them are a quadratic residues. As it has been seen in table 2, elements with even exponents (quadratic residues) and with odd exponents (quadratic non-residue) are needed to deduce the binary chain of d, for what the proposed countermeasure is a protection against Jacobi symbol attack. This protection is observed in table 4, in this example was supposed that $d = 39 = 100111$ and $(m/N) = -1$.

3.3 Behavior of Right-to-Left algorithm modified

In this section it will be given the explanation of the algorithm 4 and how it is calculating the correct value of the exponentiation modular.

Algorithm 4 begins the loop from d_1 to d_{n-1} and the initial values are $R[0] = m$ and $R[1] = m$, this means that the first square exponentiation has been skipped, in other words, the operation $R[1]_0 = R[1]^2$ has been skipped.

It will be analyzed what "skip the first squaring" means. The explanation begins with the algorithm 2. It is known that $R[1]$ is executed in an independent manner both in the algorithm 2 and in the algorithm 4, it doesn't matter if $d_i = 0$ or if $d_i = 1$, $R[1]_i$ is always squared. The final value of $R[1]$ in the algorithm 2 is

$$R[1]_{last} = m^{2^n}$$

and each value of $R[1]$ in each one of the iterations i of the loop has the form

$$R[1]_i = m^{2^i}.$$

It has been seen that the value of $R[0]$ is modified only when $d_i = 1$, in both algorithms 2 y 4. The modular multiplication performed in i has the form $R[0]_i = R[0] \cdot R[1]_{i-1}$ in the algorithm 2, then it can be substituted by

$$R[0]_i = R[0] \cdot m^{2^{i-1}} \quad (1)$$

and at the end of the loop the equation 2 is given

$$R[0]_{last} = R[0] \cdot m^{2^{n-1}} \quad (2)$$

if the algorithm 2 is applied.

What happens with the algorithm 4?: As told before, in the modified algorithm the first squaring was eliminated, for what at the end of the loop (Executed from d_1 a d_{n-1}) the value of $R[1]$ will be

$$R[1]_{last} = m^{2^{n-1}}$$

and the value in each iteration i is given by

$$R[1]_i = m^{2^{i-1}}$$

How it is known, in the modified algorithm the first squaring is not performed, the loop begins from d_1, $R[0]$ has the form $R[0]_i = R[0] \cdot (R[1]_{i-1})^2$

Algorithm 5 executed with JS attack, where $d_{i+1} = d_i$.			
i	d_i	Intermediate results	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 4: Algorithm 5 executed with JS attack, where $d_{i+1} = d_i$.

through the iterations i, and $R[1]_i = m^{2^{i-1}}$ for what, when $d_i = 1$, $R[0]_i$ is defined by

$$\begin{aligned} R[0]_i &= R[0] \cdot (R[1]_{i-1})^2 = R[0] \cdot (m^{2^{i-1-1}})^2 \\ &= R[0] \cdot m^{2^{i-2} \cdot 2} = R[0] \cdot m^{2^{i-1}} \end{aligned} \quad (3)$$

The equation 3 performs the same operation as equation 1. It was demonstrated that in each iteration, the same value in $R[0]_i$ is obtained in both algorithms 2 and 4. $R[0]$ in the last iteration of the algorithm 4 has the form

$$\begin{aligned} R[0]_{last} &= R[0] \cdot (R[1]_{last-1})^2 = R[0] \cdot (m^{2^{n-2}})^2 \\ &= R[0] \cdot m^{2^{n-2} \cdot 2} = R[0] \cdot m^{2^{n-1}} \end{aligned} \quad (4)$$

which is equal to equation 2.

3.4 Behavior of FV scheme modified

The difference of the algorithm 5 with respect to algorithm 1 is that at the beginning of the algorithm 5, $R[1] = m^2$ instead of $R[1] = m$. First of all some definitions will be given. When the algorithm 1 is executed, a loop running from $n-1$ to 0 is performed and the loop is called *loop* (o), in each iteration i the values of $R[0]$ and $R[1]$ are named $R[0]_{(o)_i}$ and $R[1]_{(o)_i}$ respectively[2]. Now when the algorithm 5 is executed, the registers are $R[0]$ and $R[1]$, but in this case, they will be named $R[0]_{(m)_i}$ and $R[1]_{(m)_i}$, and they will be running up in a loop called *loop* (m).

The behavior of the algorithm 5, started with $R[1] = m^2$, is defined by

$$\text{If } d_{(m)_{i-1}} = 0, \text{ then } \begin{cases} R[0]_{(m)_i} = R[0]_{(o)_{i-1}} \\ R[1]_{(m)_i} = R[1]_{(o)_{i-1}} \cdot m \end{cases} \quad (5)$$

$$\text{If } d_{(m)_{i-1}} = 1, \text{ then } \begin{cases} R[0]_{(m)_i} = R[0]_{(o)_{i-1}} \cdot m^{-1} \\ R[1]_{(m)_i} = R[1]_{(o)_{i-1}} \end{cases} \quad (6)$$

It can be seen in the equation 6 that when $d_{(m)i=0} = 1$ the value of $R[0]_{(m)_{i=1}}$ is equal to $R[0]_{(o)_{i=0}} \cdot m^{-1}$, and it is known that $R[0]_{(o)_{i=0}} = m^d$ because all the modular exponentiation has been done in the loop (o), for what in the algorithm 5 the last iteration must be eliminated, and it can be seen too that in the last iteration, $i = 1$ in the loop (m), the value of $R[0]$ is $R[0]_{(m)_{i=1}} = m^d \cdot m^{-1} = m^{d-1}$ and is for that reason that the line $R[0] = R[0] \cdot m$ must be added to the algorithm 5 to obtain:

$$\begin{aligned} R[0]_{(m)_{i=1}} &= R[0]_{(o)_{i=0}} \cdot m^{-1} \\ &= m^d \cdot m^{-1} \cdot m = m^{d-1+1} \\ &= m^d \end{aligned} \quad (7)$$

The input values in algorithm 5 are given because odd exponents are considered.

3.5 Expansion of the algorithms

Up to here, it has been talking about algorithms which are effective when the exponent values are odd values, but it is possible to use the algorithms 4 and 5 for all type of exponent values, for what it is necessary to add some lines to the modified algorithms, the resulting algorithms are given as algorithm 6 and algorithm 7 respectively.

[2] $R[2]$ is an independent register, for this reason we do not pay attention to it.

Algorithm 6 *Square and Multiply Right-to-Left* modified to counteract JS attack and to work with any exponent.

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

Algorithm 7 FV scheme modified to counteract JS attack and to work with any exponent.

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

The algorithm 6 and the algorithm 7 can be used not only with odd exponent values but also with even exponent values. Those algorithms use some lines more than the algorithms 4 and 5, for this reason the recommendation is that when the exponents are always odd numbers, algorithms 4 and 5 can be used, and when the exponents are odd or even numbers, algorithms 6 and 7 can be used.

3.6 Physical behavior

In this section it will be observed the physical behavior of the algorithms presented in this paper. In the figure 1 it is possible to see that the behavior of the algorithms 1 and 5 are almost the same, in the practice they can be considered equal. On the other hand, the algorithms 2 and 4 are a little different in their behavior, it can be seen in the figure 1 that the algorithm *Square and multiply* modified uses more time in its execution than the algorithm 2. The extra time of the algorithm 4 is caused by the extra operations used when the algorithm executes binary values, of the binary chain of the exponent, equal to 1.

Algorithm 4 executes u squarings more than algorithm 2, where u is the number of 1's in the binary chain of the exponent, for this reason the behavior of the algorithm 4 is given in terms of the number of 1's of the binary chain. This behavior is observed in the figure 2.

It can be seen in the figure 2 the behavior of the algorithm 4 compared with the algorithm 2 where the values of the exponents are 2048-bit numbers. It has been used three cases in the figure 2: the best case ($1\underbrace{0...0}_{2046}1$, only the first and the last bit of the exponent's binary chain are equal to 1), the middle case ($2048/2$ bits with value equal to 1), and the worst case (2048 bits with value equal to 1).

In figure 2 it is possible to see that the difference between the two algorithms is about 1.5 milliseconds in the worst case when 2048 bit numbers are used in the exponent. It can be seen that the algorithm 4 sacrifices a relatively small runtime difference in comparison with the algorithm 2, but the algorithm 4 will be secure against the JS attack, for what the algorithm 4 is a good option to provide security in embedded devices.

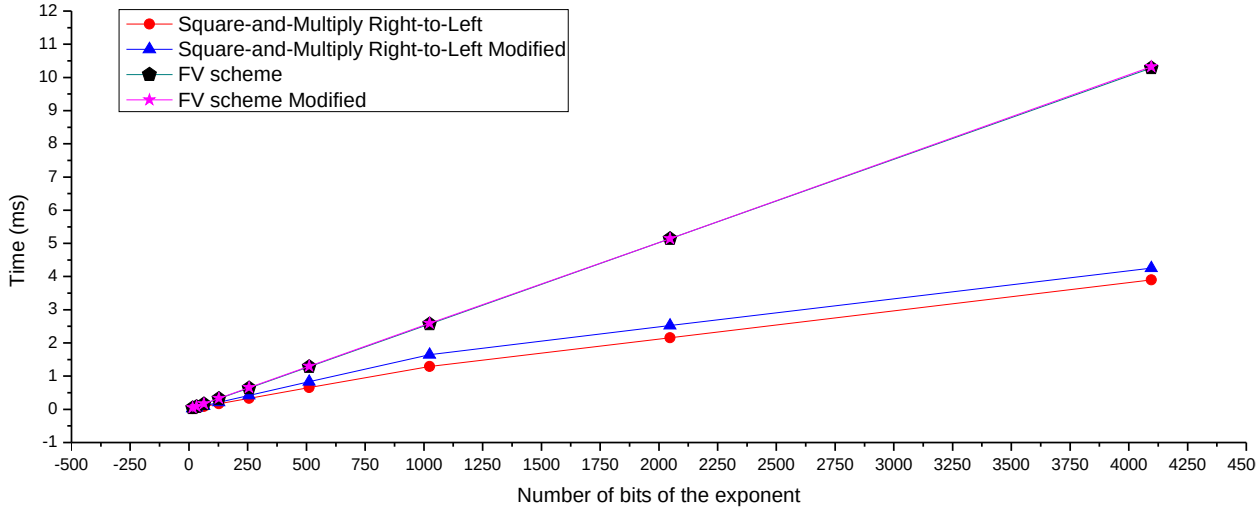

Figure 1: Physical behavior of the algorithms 1 to 4, where the x-axis represents the number of bits of the exponent, and the y-axis represents the runtime of the algorithms given in milliseconds.

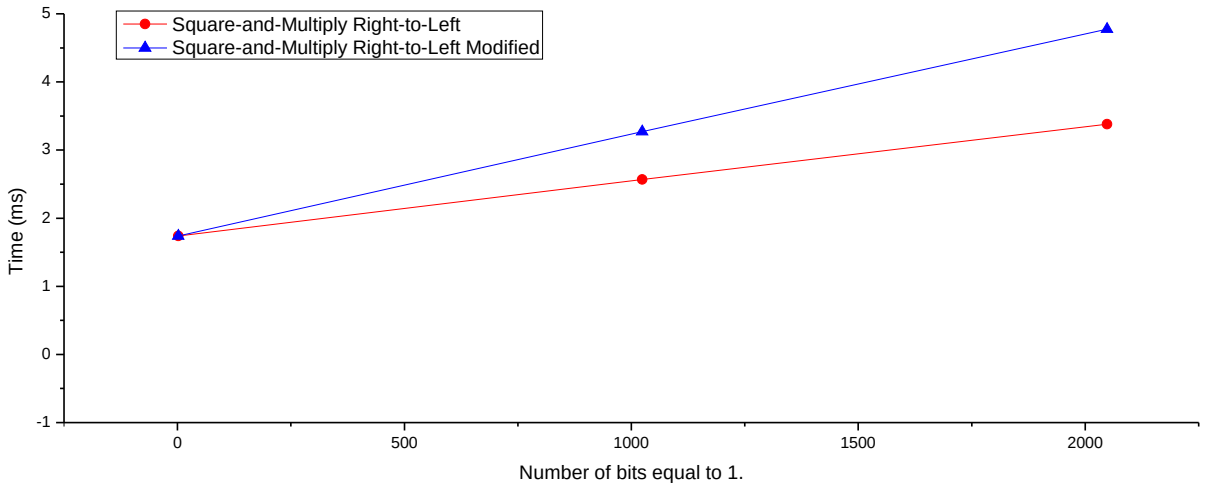

Figure 2: Runtime of the algorithms 2 and 4 in the worst, middle and the best case, executing exponents with 2048 bits.

4 Future work

The proposed countermeasure against the Jacobi symbol attack pointed out in this paper has been implemented under two algorithms which have arithmetic operations that can be evaluated independently, here we have illustrated the basic idea to protect the exponentiation modular in cryptosystems against JS attack, but each exponentiation modular algorithm in the literature has different characteristics for what we will search the form to implement our idea over other algorithms which can be attacked by the JS attack and have different characteristics like the algorithms *Add only* and *Add always* presented by *Marc Joye* in [11] and attacked in 2010 by *Kim* [13].

5 Conclusion

We have proposed a countermeasure to protect two modular exponentiation algorithms against the Jacobi symbol attack, the coutermeasure can be implemented without difficult steps. The attacks mentioned are implemented in a different form and are designed under different models, then we are designed the methods to defeat the attacks specifically to work according to the fault model used in each case, but there is an idea that is essential in both modified algorithms: You need to change a quadratic non-residue value into a quadratic residue value.

We have obtained the physical behavior of the proposed algorithms and we have compared them against the original versions, in the case of the algorithms 1 and 4 there is no difference between their behaviors, but in the case of the algorithms 2 and 4 there is only a little difference in runtime, this increase in runtime of the algorithm 4 is balanced with the security obtained.

References:

[1] 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.* Vol. 1233 of Lecture Notes in Computer Science, 1997, pp. 37-51.

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

[3] 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 Crptography - FDTC'09.* 2009, pp. 3-9.

[4] 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.* Vol. 4462, 2007, pp. 229–243.

[5] 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.

[6] J. S. Coron. 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, 1999, pp. 292–302.

[7] P.A. Fouque and F. Valette. The doubling attack-why upwards is better than downwards. *Cryptographic Hardware and Embedded Systems-CHES 2003.* Vol. 2779 of Lecture Notes in Computer Science, 2003, pp. 269-280.

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

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

[10] 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.

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

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

[13] C.H. Kim. New fault attacks using jacobi symbol and application to regular right-to-left algorithms. *Information Processing Letters.* Vol. 110, No. 20, 2010, pp. 882-886.

[14] 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.* Vol. 4462, 2007, pp. 215-228.

[15] 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.* 2007, pp. 21–29.

[16] P. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. *In Koblitz, N., ed.: Advances in Cryptology-CRYPTO 96.* Vol. 1109 of Lecture Notes in Computer Science, pp. 104-113, Springer, 1996.

[17] 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.* Vol. 2, 2005, pp. 151154.

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

[19] Jörn-Marc Schmidt and Marcel Medwed. Fault attacks on the montgomery powering ladder. *In Springer, editor, 13th Annual International Conference on Information Security and Cryptology, Proceedings, LNCS.*, 2010, in press.

[20] S.M. Yen and M. Joye. Checking before output may not be enough against faultbased cryptanalysis. *IEEE Transactions on Computers.* Vol. 49, No. 9, 2000, pp. 967-970.

[21] 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.* Vol. 2288 of Lecture Notes in Computer Science, 2001, pp. 414–427.

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

[23] S.M. Yen, W.C. Lien, S.J. Moon, and J.C. Ha. Power analysis by exploiting chosen message and internal collisionsvulnerability of checking mechanism for rsa decryption. *Progress in CryptologyMycrypt 2005.* Vol. 3715 of Lecture Notes in Computer Science, 2005, pp. 183-195.