Fault Attacks on the Montgomery Powering Ladder

Jörn-Marc Schmidt[1] and Marcel Medwed[1,2]

[1] Graz University of Technology
Institute for Applied Information Processing and Communications
Inffeldgasse 16a, A–8010 Graz, Austria
joern-marc.schmidt@iaik.tugraz.at
[2] Université catholique de Louvain, Crypto Group, Belgium
marcel.medwed@uclouvain.be

Abstract. Security-aware embedded devices which are likely to operate in hostile environments need protection against physical attacks. For the RSA public-key algorithm, protected versions of the Montgomery powering ladder have gained popularity as countermeasures for such attacks. In this paper, we present a general fault attack against RSA implementations which use the Montgomery powering ladder. In a first step, we discuss under which realistic fault assumptions our observation can be used to attack basic implementations. In a second step, we extend our attack to a scenario, where the message is blinded at the beginning of the exponentiation algorithm. To the best of our knowledge this is the first fault attack on a blinded Montgomery powering ladder.

Keywords: Montgomery Powering Ladder, Fault Attack, Blinded Exponentiation, Quadratic Residue

1 Introduction

In order to judge the security of the practical realization of a cryptographic algorithm, the way it is implemented is as important as the theoretical security of the algorithm itself. This is because an adversary can try to manipulate the computation and reveal secrets from the erroneous output of the device. These so-called fault attacks were first described against public-key schemes by Boneh, DeMillo and Lipton [1]. They showed for example how to recover an RSA secret exponent by disturbing one of the two exponentiations of a CRT-RSA computation. In the following years, further fault attacks on the RSA algorithm were published [2, 3]. In particular, those two works target the square-and-multiply exponentiation algorithm. They iteratively disturb and skip respectively the square operation in order to recover the secret key bit-by-bit.

In practice, a straightforward implementation of the square-and-multiply algorithm is also weak against other side-channel attacks like timing attacks [4] and Simple Power Analysis [5]. Therefore, the square-and-multiply algorithm is often replaced by more regular algorithms like the Montgomery powering ladder. This replacement does not only thwart some side-channel attacks but also

hardens an adversary's life for fault attacks. For instance, in [6] it is stated that higher-order faults (several faults per algorithm invocation) are needed to apply their attack to the Montgomery ladder.

An attack that combines side-channel analysis and fault attacks was presented by Park et at. [7]. In their work, they induce single faults either during the multiplication or during the squaring and check which future intermediate results are affected by looking at the device's power consumption.

Our contribution: In this paper, we present a general fault attack against the Montgomery powering ladder. We show that various fault models can be used to put the attack into practice, such as tampering with the intermediate variables or with the program flow. Moreover, we show how our approach can be extended to blinded implementations. In particular, our attack can defeat the blinded implementation of the Montgomery ladder proposed by Fumaroli and Vigilant [8]. To the best of our knowledge this is the first fault attack on a blinded Montgomery powering ladder.

The remaining paper is organized as follows: Section 2 gives a brief introduction into RSA and the Montgomery powering ladder. The general attack method is detailed in Section 3. Afterwards, Sections 4 and 5 show how to launch the attack in a realistic scenario. Finally, we extend the attack to work on blinded implementations in Section 6. The complexity of the attacks is discussed in Section 7. Conclusion is drawn in Section 8.

2 Preliminaries

In this section, we first discuss the RSA public-key algorithm. Afterwards, we look at the Montgomery powering ladder and its protected version as proposed by Fumaroli and Vigilant [8]. Finally, we revisit the Jacobi symbol.

The security of RSA is based on the hardness of factoring the product of two large primes. Let p and q be such primes, $n = pq$ their product, and $\varphi(\cdot)$ denote Euler's totient function. All computations of the RSA take place in the ring $\mathbf{Z}_n$. The public exponent e is an element of $\mathbf{Z}^*_{\varphi(n)}$, its corresponding secret exponent is $d = e^{-1} \mod \varphi(n)$. Due to this construction, $m = (m^e)^d \mod n$ holds for any $m \in \mathbf{Z}_n$. The owner of the secret key can sign messages by computing $s = m^d \mod n$ or decrypt ciphertexts by calculating $m = c^d \mod n$. By giving away the public key (N, e), he enables everybody else to either verify signatures ($m = s^e \mod n$) or to encrypt messages ($c = m^e \mod n$). In order to omit side-channel attacks on the exponentiation, it is recommended to use regular exponentiation algorithms, which leak as little information as possible about the secret exponent. As mentioned before, one such regular algorithm to compute a modular exponentiation is the Montgomery powering ladder [9]. It is depicted in Algorithm 1.

In each iteration of the ladder, one intermediate is assigned the product of both, the other one is squared. If the current bit is one, R_0 is set to $R_0 \cdot R_1$ and R_1 is squared, and vice versa if the bit is zero. Let $d = (d_{t-1}, \ldots, d_0)_2 =$

Algorithm 1 Montgomery ladder [9]

Require: n, $d = (d_{t-1}, \ldots, d_0)_2$, $m \in \mathbf{Z}_n$
Ensure: $m^d \bmod n$

$R_0 = 1$
$R_1 = m$
for $i = t-1$ **downto** 0 **do**
  $R_{\bar{d}_i} = R_0 \cdot R_1 \bmod n$
  $R_{d_i} = R_{d_i}^2 \bmod n$

end for
return R_0

Algorithm 2 Protected ladder [8]

Require: $d = (d_{t-1}, \ldots, d_0)_2$, $m \in \mathbf{Z}_n$
Ensure: $m^d \bmod n$
$r = rand() \in \mathbf{Z}_n^*$; $R_2 = r^{-1} \bmod n$
$R_0 = r$
$R_1 = r \cdot m \bmod n$
for $i = t-1$ **downto** 0 **do**
  $R_{\bar{d}_i} = R_0 \cdot R_1 \bmod n$
  $R_{d_i} = R_{d_i}^2 \bmod n$
  $R_2 = R_2^2 \bmod n$
end for
return $R_0 \cdot R_2 \bmod n$

$[d_L, d_i, d_T]$. Here, d_i denotes the bit which is currently processed and d_L the already processed (Leading) bits. The remaining (Trailing) bits are denoted by d_T. The intermediates after processing the bit d_i are

$$(R_0, R_1) = \begin{cases} (m^{2 \cdot d_L} \bmod n, \quad m^{2 \cdot d_L + 1} \bmod n) & \text{for } d_i = 0 \\ (m^{2 \cdot d_L + 1} \bmod n, \, m^{2 \cdot d_L + 2} \bmod n) & \text{for } d_i = 1. \end{cases}$$

A basic property of the Montgomery ladder is that the quotient $\frac{R_1}{R_0}$ is constant. This property is important for our attack. In a correct execution of the RSA algorithm this quotient equals m.

In order to achieve further side-channel resistance, Fumaroli and Vigilant suggested a blinded version [8]. Their proposal is depicted in Algorithm 2. It includes a random mask r, which is a factor of both intermediates, R_0 and R_1. Hence, r is squared in each step in both variables. As a consequence, the result includes the factor r^{2^t}. By squaring the inverse of r in every iteration we get r^{-2^t} at the end of the algorithm. Thus the result can be unblinded by calculating $R_0 \cdot R_2 \bmod n$.

To counteract fault attacks, they suggested to include a checksum which is updated at the end of every iteration. If one or more iterations (key bits) are skipped due to a fault attack, the checksum is invalid at the end of the algorithm. This prevents an adversary from tampering with the loop counter. However, this checksum cannot detect our attack. Therefore, it is not included in Algorithm 2.

Finally, we use the Jacobi symbol for the attack on the blinded Montgomery ladder. The Jacobi symbol is a generalization of the Legendre symbol for composite moduli and is defined as

$$\left(\frac{c}{k}\right) = \left(\frac{c}{p_1}\right)^{\alpha_1} \left(\frac{c}{p_2}\right)^{\alpha_2} \cdots \left(\frac{c}{p_k}\right)^{\alpha_k} \quad \text{where } k = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$$

with $\left(\frac{c}{p}\right)$ denoting the Legendre symbol for primes p

$$\left(\frac{c}{p}\right) = \begin{cases} 0 \text{ if } c = 0 \pmod{p} \\ +1 \text{ if } c \neq 0 \pmod{p} \text{ and } \exists x \text{ s.t. } x^2 = c \pmod{p} \\ -1 \text{ otherwise.} \end{cases}$$

The Jacobi symbol can be efficiently computed using the law of quadratic reciprocity (see e.g. [10]), even if the factorization of the modulus is not known. Note that the Jacobi symbol only gives a guarantee for quadratic non-residues by evaluating to -1, but a result of 1 does not imply that c is a quadratic residue modulo k.

3 General Attack Method

As already mentioned, the aim of a fault attack is to deduce information on secrets involved in the internal computation of the device. In most cases, this is done by manipulating the device and analyzing its erroneous output[3]. Normally, an adversary is assumed to have access to the attacked device and can manipulate it. Hence, the inputs of the device are chosen by the adversary.

In order to successfully attack a device, it is important to know the algorithm that is computed internally. Furthermore, it is necessary to make assumptions about the faults that occur and to set up relations between the intermediates processed inside the device and the erroneous output.

In this paper, we discuss such relations for the Montgomery powering ladder. A fault attack on its implementation can exploit a general observation: For two arbitrary values in R_0 and R_1 the algorithm behaves as in Table 1. For $a = 1$

Table 1. The Montgomery powering ladder starting with R_0 and R_1 set to arbitrary values. $(\cdot)^d$ denotes the application of the ladder with the exponent d of length t.

Step	R_0	R_1
	a	b
$=$	a	$a \cdot \frac{b}{a}$
$(\cdot)^d$	$a^{2^t} \cdot (\frac{b}{a})^d$	$a^{2^t} \cdot (\frac{b}{a})^{d+1}$
$=$	$a^{2^t-d} \cdot b^d$	$a^{2^t-d} \cdot b^d \cdot \frac{b}{a}$

and $b = m$ this evaluates directly to m^d. In a fault attack an adversary would set one of the two intermediates to a random (or partially random) value during

[3] Note that for some fault attacks the behavior of the device itself, after a fault is injected, is sufficient [11].

the exponentiation. As before, let $d = (d_{t-1}, \ldots, d_0)_2 = [d_L, d_i, d_T]$. After the computation of d_L, one intermediate is modified to contain a random value z. As a consequence, the intermediate values develop as depicted in Table 2. This basic

Table 2. Fault injection during Montgomery ladder computation.

	Fault in R_0		Fault in R_1	
Step	R_0	R_1	R_0	R_1
	1	m	1	m
$(\cdot)^{d_L}$	m^{d_L}	m^{d_L+1}	m^{d_L}	m^{d_L+1}
Fault	z	m^{d_L+1}	m^{d_L}	z
Output	$m^{(2^i \cdot d_i + d_T)\cdot(d_L+1)} \cdot z^{2^i-(2^i \cdot d_i+d_T)}$		$m^{(2^i-(2^i \cdot d_i+d_T))\cdot d_L} \cdot z^{(2^i \cdot d_i+d_T)}$	

property can be exploited to retrieve the secret exponent. Therefore, the fault must be injected in a way it is predictable. What is left is to reduce the number of unknown bits of the exponent in the equation by exploiting the property of RSA that $d = e^{-1} \pmod{\varphi(n)}$. In the following, we are considering different fault models that are suitable for an attack and show how to proceed in these cases. Furthermore, we discuss how to mount an attack on blinded versions of the algorithm.

4 Fault Model: A Guessable Fault

In the first model, we assume that an intermediate variable of an RSA implementation using the Montgomery powering ladder is modified by a fault at a known point in time. Furthermore, this fault influences the variable in a way that the result is within a limited range that can be searched exhaustively.

In order to inject such a fault, an adversary can use a focused laser beam to flip/set some bits in a register [12]. Another method to cause a fault is to interrupt the loading of a word into a register, e.g. by injecting glitches or spikes [13]. Hence, the size of the fault depends on the word size of the microcontroller. Both methods can control the point in time when the fault is injected very precisely.

In particular, if fault w is injected into R_0 after processing the exponent bits of d_L (a fault in R_1 leads to similar equations), the intermediate variables (R_0, R_1) contain $(m^{d_L} \oplus w, m^{d_L+1})$. As a consequence, the final (erroneous) signature is

$$\tilde{S} = m^{(2^i \cdot d_i + d_T)\cdot(d_L+1)} \cdot (m^{d_L} \oplus w)^{2^{i+1}-(2^i \cdot d_i+d_T)} \pmod{n}.$$

Since we assume that the fault is injected at the beginning of the computation, d_L is small and hence guessable, while d_T is not. However, we can substitute

$d_T + 2^i \cdot d_i = d - 2^{i+1} \cdot d_L$:

$$\tilde{S} = m^{(d-2^{i+1}\cdot d_L)\cdot(d_L+1)} \cdot (m^{d_L} \oplus w)^{2^{i+1}-(d-2^{i+1}\cdot d_L)} \pmod{n}.$$

From this we can eliminate d because we know e and we further know that $ed = 1 \pmod{\varphi(n)}$. Raising $\tilde{S}$ to the power of e delivers an expression in which only w and d_L remain unknown:

$$\tilde{S}^e = m^{(1-2^{i+1}\cdot e\cdot d_L)\cdot(d_L+1)} \cdot (m^{d_L} \oplus w)^{2^{i+1}i\cdot e+2^{i+1}\cdot d_L\cdot e-1} \pmod{n}. \quad (1)$$

Now we can test hypotheses for w and d_L. For correctly guessed values equation (1) must hold.

The observations above allow an iterative attack: Inject an appropriate fault while the first bits of d are processed. Using the public exponent, the result can be transferred into a value that depends on the message and a few unknown bits. Now the hypotheses for the fault and d_L can be tested against this value. A correct hypothesis increases the knowledge about d. The attack is repeated until the whole exponent is known.

Note that the same attack is possible injecting a fault into R_1 by substituting R_1 by $R_1 \oplus w$ in Table 2. If a register that stores either R_0 or R_1 is attacked, both possibilities have to be checked simultaneously.

5 Fault Model: Skipping an Instruction

Another possible fault model is based on a modification of the program flow. Instead of manipulating the data directly by flipping bits, an instruction is not executed. This reduces the overhead generated by guessing the flipped bits, since only the position of the skipped instruction is required, which depends on the point in time the fault is injected[4]. Using the public exponent e the same way as in the previous attack delivers a value which contains only a small part of the unknown secret exponent. In this way, the whole exponent can be determined iteratively.

Let m be a message to be signed using the exponent $d = [d_L, d_i, d_T]$ with d_i the bit that is processed as the squaring is skipped. The resulting equation depends on d_i, because if it is zero, a squaring of R_0 is skipped, while for a one the squaring of R_1 is left out.
First, assume $d_i = 0$. By skipping the squaring, R_0 stays unchanged and R_1 contains the value $m^{2\cdot d_L+1}$. This can be seen as skipping d_i and changing the quotient to $d_L + 1$. Together with the last line of Table 2 and $d = 2^{i+1} \cdot d_L + d_T$ this results in:

$$\begin{aligned} \tilde{S} &= m^{(2^i-d_T)\cdot d_L+(2d_L+1)\cdot d_T} \pmod{n} \\ &= m^{2^i\cdot d_L+d-2^{i+1}\cdot d_L+d_L\cdot(d-2^{i+1}\cdot d_L)} \pmod{n} \\ \Rightarrow \tilde{S}^e &= m^{1+d_L-e\cdot 2^i\cdot d_L\cdot(1+2\cdot d_L)} \pmod{n}. \end{aligned}$$

[4] In this model, we allow the fault injection to be imprecise, since it is possible to check whether the fault is exploitable for our attack.

Table 3. Intermediates of a Montgomery Powering ladder with a skipped squaring instruction while $d_i = 0$ was processed.

Step		R_0	R_1	Quotient
after d_L		m^{d_L}	m^{d_L+1}	m
after d_i		m^{d_L}	$m^{2\cdot d_L+1}$	m^{d_L+1}
next bit	$d_{i+1}=0$	$(m^{d_L})^2$	$m^{3\cdot d_L+1}$	m^{d_L+1}
	$d_{i+1}=1$	$m^{3\cdot d_L+1}$	$m^{4\cdot d_L+2}$	m^{d_L+1}
Erroneous Output		$m^{(2^i-d_T)\cdot d_L+(2d_L+1)\cdot d_T}$		

Table 3 details the content of the intermediate variables and the quotient for a skipped squaring of $d_i = 0$. After the quotient between R_0 and R_1 is changed by the fault, it stays constant for the rest of the computation. For $d_i = 1$, R_1 stays constant and R_0 changes to $2 \cdot d_L + 1$. Together with $d = 2^{i+1} \cdot d_L + 2^i + d_T$, we get:

$$\begin{aligned} \tilde{S} &= m^{d_T\cdot(d_L+1)+(2^i-d_T)\cdot(2d_L+1)} \pmod n \\ &= m^{2^i\cdot(1+d_L\cdot(3+2\cdot d_L))-d_L\cdot d} \pmod n \\ \Rightarrow \tilde{S}^e &= m^{e\cdot 2^i\cdot(1+d_L\cdot(3+2\cdot d_L))-d_L} \pmod n. \end{aligned}$$

The same equations can be set up for a skipped multiplication:

$$\tilde{S}^e = \begin{cases} m^{1-d_l\cdot(1-2^{i+1}\cdot d_l\cdot e)} \pmod n & \text{for } d_i = 0 \\ m^{(2+d_L)\cdot(1-2^{i+1}\cdot e\cdot(1+d_L))} \pmod n & \text{for } d_i = 1. \end{cases}$$

Hence, a Montgomery powering ladder can be attacked by iteratively skipping either squarings or multiplications and calculating the expected values. If they do not match, the fault was not injected in the intended way.

Note that the attack works analogously starting from the least-significant bits of the exponent. Furthermore, the attack can be applied to algorithms that are based on ECC. Moreover, for an ECDSA implementation, guessing a block of several bits for each ephemeral key and building a lattice is also possible, like it is done in [6] for a double-and-add algorithm.

6 Attack on a Blinded Implementation

In order to provide a side-channel secure implementation, Fumaroli and Vigilant proposed a blinded version of the Montgomery ladder [8]. Their suggestion also includes a signature for preventing an adversary from tampering with the loop counter. In this section, we assume the same fault model as in the previous

Algorithm 3 Schematic of the Attack

Require: A device that can be manipulated and uses the blinded Montgomery ladder to produce faulty signatures $\tilde{S}$.
Ensure: The exponent $d = (d_{t-1}, \ldots, d_0)_2$ that is used by the device.
  Set $d_{t-1} = 1$ (leading zeros are neglected)
  for $i = t - 2$ **downto** 0 **do**
    Choose $m \in \mathbf{Z}_n$ with $\left(\frac{m}{n}\right) = -1$
    Calculate $\tilde{S}$ with the ith squaring operation skipped
    if $\left(\frac{\tilde{S}}{n}\right) = -1$ **then**
      $d_i = d_{i+1}$
    else
      $d_i = 1 \oplus d_{i+1}$
    end if
  end for
  return d

one. The only difference is that a precise fault injection is required. On the one hand, each fault that is injected into only one operation i.e. the squaring or the multiplication is not detected by the checksum. On the other hand, such a fault produces an unpredictable output, since a part of the random mask is still included in the return value. This makes a direct guessing of bit chunks as in the previous attacks impossible. But there is still one bit of exploitable information left in the output, namely if the result is a quadratic residue. More precisely, we can compute the Jacobi symbol of the result, which indicates a quadratic non-residue if it is negative[5]. A schematic view of the attack on a blinded implementation is given in Algorithm 3.

Taking a closer look at the result shows that if a squaring is skipped during the processing of Algorithm 2, the result $\tilde{S}$ is

$$\tilde{S} = R_2^{(2^t)} \cdot r^{2^{t-1}+2^{i-1}\cdot u} \cdot m^{\tilde{d}} \text{ with}$$

$$u = \begin{cases} d_T & \text{for } d_i = 0 \\ 2^i - d_T & \text{for } d_i = 1 \text{ and} \end{cases}$$

$$\tilde{d} \cdot e = \begin{cases} 1 + d_L - e \cdot 2^i \cdot d_L \cdot (1 + 2 \cdot d_L) \pmod{\varphi(n)} & \text{for } d_i = 0 \\ e \cdot 2^i \cdot (1 + d_L \cdot (3 + 2 \cdot d_L)) - d_L \pmod{\varphi(n)} & \text{for } d_i = 1. \end{cases}$$

Hence, the result can be split up into an unknown part, which includes the random mask and another one that depends on the input message, on the exponent, and of the position of the fault. Raising the resulting $\tilde{S}$ to the power e cancels the unknown bits of d_T out. If the fault is chosen in a way that only d_i is unknown and d_L is known, the whole message-dependent part of the signature

[5] Note that a positive result does not imply a quadratic residue.

depends on the one bit d_i. Furthermore, it follows that it directly depends on this bit, whether the result is a quadratic residue assuming that m is a quadratic non-residue. This is because the remaining part of the random mask is always a quadratic residue due to its exponent, which is a multiple of two. In detail, if m is chosen with a Jacobi symbol $\left(\frac{m}{n}\right) = -1$, $\tilde{S}$ is a quadratic non-residue with $\left(\frac{\tilde{S}}{n}\right) = -1$, iff $\tilde{d}$ is odd. Moreover, whether $\tilde{d}$ is even or not depends on the last bit of d_L and d_i. Since d_L is known, the knowledge of the Jacobi symbol of $\tilde{S}$ determines d_i. This is because $\varphi(n)$ is always even and d is always odd in the case of RSA. Thus, computing the Jacobi symbol leads to an attack similar to the one presented by Boreale on square-and-multiply [2]. In contrast, our result is not probabilistic, if m can be chosen with a negative Jacobi symbol. Since the loop itself is not manipulated, a checksum cannot prevent this attack.

6.1 Practical Considerations

The possibility of influencing the program flow by means of spike attacks was demonstrated in [3]. In their work, the authors used the resulting fault model for an attack on a square-and-multiply implementation. It turned out that in a practical attack the probability of a successfully injected fault is smaller one. Thus, a method to check whether the output of the device is the intended result or not is favorable if not mandatory. For the unblinded version of the Montgomery ladder, this can be easily done by checking if the output corresponds to one of the desired (precalculated) results. For the blinded version, this is not possible because a multiple of the random mask is still a part of the result. Thus, an adversary cannot tell from the result whether computing the Jacobi symbol delivers information about one bit of the exponent or not. Fortunately, there is another kind of information that makes it possible to recognize a successful fault injection. By measuring the time a computation takes, an adversary can judge whether a whole multiplication was skipped. Thus, the adversary can sweep through the algorithm and identify the positions where multiplications are invoked. Additionally, there is a way of telling the three different multiplications of one loop iteration of the ladder apart: A skipped multiplication and the skipped consecutive squaring respectively yield the same Jacobi symbol. For a skipped squaring of the mask, the result has always a negative Jacobi symbol, since d is odd. This leaves the adversary with the necessary tools for a successful attack.

7 Complexity of the Attacks

Table 4 overviews the efforts the different attacks require. The first two attacks allow to determine the exponent in chunks of several bits. This reduces the number of fault injections but increases the computational effort of the attack. This is because an exponentiation is required for each possible value of such a chunk

Table 4. Required effort for the attacks with t denoting the bit-length of the secret exponent and c the bit-size of the chunks the exponent is recovered in.

Attack Method (presented in Section)	Fault Injections	Exponentiations
Fault of Bit-Size v (4)	t/c	$t/c \cdot 2^{(c+v)}$
Skipping an Instruction (5)	t/c	$t/c \cdot 2^c$
Attack on Blinded Implementation (6)	t	-

to determine the corresponding exponent bits. Hence, it is possible to trade fault injections for computational effort. In particular, if we assume a chunk of bit-size c and an exponent of bit-size t, t/c fault injections are required. For each faulty computation 2^c values have to be guessed and tested with the corresponding formulas, which requires an exponentiation each. In the first attack, the maximum possible bit-size v of the injected fault also needs to be considered. Each unknown bit doubles the number of required tests.

The attack on a blinded implementation of the algorithm recovers the exponent bit-by-bit. Thus, the number of injected faults equals the bit length of the exponent. Since the test involves only the computation of the Jacobi symbol, it does not require extra exponentiations.

8 Conclusion

In this paper, we presented new fault attacks on the Montgomery powering ladder. We demonstrated that our attacks are feasible for two realistic fault models: (1) for random register faults that can be guessed and (2) for a manipulation of the program flow. For both models, we discussed how to determine the secret exponent of an unprotected implementation in bit-chunks. In addition, it is possible to recognize a successful fault injection by the output of the device.

For the latter fault model, we also showed how to mount an attack on a blinded implementation. In this attack, the exponent was recovered bit-wise by measuring the execution time and checking the Jacobi symbol of the output. To the best of our knowledge, this is the first fault attack on a blinded Montgomery ladder.

The presented results show that fault attacks on the blinded Montgomery ladder are possible. All attacks require the adversary to know the plaintext. For the attack on the blinded version, knowing the Jacobi symbol of the plaintext is sufficient.

We conclude that blinding in combination with a loop-checksum does not prevent all fault attacks on the Montgomery powering ladder. Therefore, additional protection by exponent blinding or by a check whether the quotient between the two intermediate variables is correct should be implemented.

9 Acknowledgements

The work described in this paper has been supported in part through the Austrian Science Fund (FWF) under grant number P22241-N23. The information in this document reflects only the authors views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

1. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In Fumy, W., ed.: Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceedings. Volume 1233 of Lecture Notes in Computer Science., Springer (1997) 37–51
2. Boreale, M.: Attacking Right-to-Left Modular Exponentiation with Timely Random Faults. In Breveglieri, L., Koren, I., Naccache, D., Seifert, J.P., eds.: Fault Diagnosis and Tolerance in Cryptography, Third International Workshop, FDTC 2006, Yokohama, Japan, October 10, 2006, Proceedings. Volume 4236 of Lecture Notes in Computer Science., Springer (October 2006) 24–35
3. Schmidt, J.M., Herbst, C.: A Practical Fault Attack on Square and Multiply. In Breveglieri, L., Gueron, S., Koren, I., Naccache, D., Seifert, J.P., eds.: Fault Diagnosis and Tolerance in Cryptography, Fifth International Workshop, FDTC 2008, Washington DC, USA, August 10, 2008, Proceedings, IEEE Computer Society (August 2008) 53–58
4. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Koblitz, N., ed.: Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings. Number 1109 in Lecture Notes in Computer Science, Springer (1996) 104–113
5. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In Wiener, M., ed.: Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Volume 1666 of Lecture Notes in Computer Science., Springer (1999) 388–397
6. Schmidt, J.M., Medwed, M.: A Fault Attack on ECDSA. In Naccache, D., Oswald, E., eds.: Fault Diagnosis and Tolerance in Cryptography, Sixth International Workshop, FDTC 2009, Lausanne, Switzerland September 6, 2009, Procceedings, IEEE-CS Press (September 2009) 93–99
7. Park, J., Bae, K., Moon, S., Choi, D., Kang, Y., Ha, J.: A New Fault Cryptanalysis on Montgomery Ladder Exponentiation Algorithm. In: ACM International Conference Proceeding Series; Vol. 403, Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, ACM Press (2009) 896–899
8. Fumaroli, G., Vigilant, D.: Blinded Fault Resistant Exponentiation. In Breveglieri, L., Koren, I., Naccache, D., Seifert, J.P., eds.: Fault Diagnosis and Tolerance in Cryptography, Third International Workshop, FDTC 2006, Yokohama, Japan, October 10, 2006, Proceedings. Volume 4236 of Lecture Notes in Computer Science., Springer (October 2006) 62–70

9. Joye, M., Yen, S.M.: The Montgomery Powering Ladder. In Goos, G., Hartmanis, J., van Leeuwen, J., eds.: Cryptographic Hardware and Embedded Systems – CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. Volume 2523 of Lecture Notes in Computer Science., Springer (2003) 291–302
10. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. Series on Discrete Mathematics and its Applications. CRC Press (1997) ISBN 0-8493-8523-7, Available online at `http://www.cacr.math.uwaterloo.ca/hac/`.
11. Yen, S.M., Joye, M.: Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. In: IEEE Transactions on Computers. Volume 49 of IEEE Transactions on Computers., IEEE Computer Society (2000) 967–970
12. Skorobogatov, S.P., Anderson, R.J.: Optical Fault Induction Attacks. In Jr., B.S.K., Çetin Kaya Koç, Paar, C., eds.: Cryptographic Hardware and Embedded Systems – CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. Volume 2523 of Lecture Notes in Computer Science., Springer (2003) 2–12
13. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The Sorcerer's Apprentice Guide to Fault Attacks. Cryptology ePrint Archive (`http://eprint.iacr.org/`), Report 2004/100 (2004)