Secret Key Leakage from Public Key Perturbation of DLP-based Cryptosystems

Alexandre Berzati[1,2], Cécile Canovas-Dumas[1], Louis Goubin[2]

[1] CEA-LETI/MINATEC, 17 rue des Martyrs, 38054 Grenoble Cedex 9, France,
{alexandre.berzati,cecile.dumas}@cea.fr
[2] Versailles Saint-Quentin-en-Yvelines University,
45 Avenue des Etats-Unis, 78035 Versailles Cedex, France
Louis.Goubin@prism.uvsq.fr

Abstract. Finding efficient countermeasures for cryptosystems against fault attacks is challenged by a constant discovery of flaws in designs. Even elements, such as public keys, that do not seem critical must be protected. From the attacks against RSA [5,4], we develop a new attack of DLP-based cryptosystems, built in addition on a lattice analysis [26] to recover DSA public keys from partially known nonces. Based on a realistic fault model, our attack only requires 16 faulty signatures to recover a 160-bit DSA secret key within a few minutes on a standard PC. These results significantly improves the previous public element fault attack in the context of DLP-based cryptosystems [22].

Keywords: DSA, exponentiation, fault injection, public modulus, lattice reduction.

1 Introduction

Since the advent of side channel attacks, classical cryptanalysis is no longer sufficient to ensure the security of cryptographic algorithms. In practice, the implementation of algorithms on electronic devices is a potential source of leakage that an attacker can use to completely break a system [23,14,18]. The injection of faults during the execution of cryptographic algorithm is considered as an intrusive side channel method because secret information may leak from malicious modifications of the device's behavior [10,11,7]. In this context, the security of public key cryptosystems [10,11] and symmetric ciphers in both block [7] and stream modes [20] has been challenged.

Recently, some interesting results have been obtained by attacking public key cryptosystems. More precisely, several papers demonstrated that the perturbation of public elements may induce critical flaws in implementations of public key cryptosystems [6,13,22]. Whereas very efficient fault attacks against public elements were elaborated for ECDLP (Elliptic Curve Discrete Logarithm Problem) [6] and IFP (Integer Factorisation Problem)-based algorithms [13,4], DLP (Discrete Logarithm Problem)-based algorithms seem to be less vulnerable: the best known attack against public elements of DLP-based algorithms requires an average of $4 \cdot 10^7$ and $3 \cdot 10^6$ faulty signatures to recover the secret key for respectively a 160-bit DSA [27] and a 1024-bit ElGamal [17]. This amount of faults is a serious drawback to ensure the practicability of this attack [22].

In this paper, we propose a new fault attack against public key elements of DLP-based cryptosystems. It is based on both lattice analysis proposed by P.Q. Nguyen and I.E. Shparlinski [26] to recover DSA public keys from known part of nonces and fault attacks against RSA public keys [5,4]. Under a practical fault model, our presented attack only requires 16 faulty signatures to recover a 160-bit DSA secret key within a few minutes on a standard PC. Hence, this attack significantly improves previous results about perturbation of public elements in the context of DLP-based cryptosystems [22]. Moreover, this performance provides evidence that DLP-based cryptosystems are not more resistant against perturbation of public elements.

The remainder of this paper is organized as follows: Section 2 introduces the notations used throughout the paper for the DSA signature scheme [27] and the different implementations of the exponentiation. Section 3 describes our fault attack against the DSA public modulus for *"Left-To-Right"*-based exponentiations but the attack also works when dual implementations are used. The detailed evaluation of the performance of our new fault attack is provided in Section 4. Finally we conclude in Section 5 about the need for protecting public key elements from faults.

2 Background

2.1 Presentation of the DSA

The *Digital Signature Algorithm*, or DSA, is the American federal standard for digital signatures [27]. The system parameters are composed by $\{p, q, h, g\}$ where p stands for a large prime modulus of size n, q is a l-bit prime such that $q \mid p-1$, h is a hash function and $g \in \mathbb{Z}_p^*$ is a generator of order q. The private key is an integer $x \in \mathbb{Z}_q^*$ and the public key is $y \equiv g^x \bmod p$. The security of DSA relies on the hardness of the discrete logarithm problem in prime finite fields and its subgroups.

Signature. To sign a message m, the signer picks a random $k < q$ and computes:

$$u \leftarrow \left(g^k \bmod p\right) \bmod q \quad \text{and} \quad v \leftarrow \frac{h(m) + xu}{k} \bmod q.$$

The signature of m is the pair: (u, v).

Verification. To check (u, v), the verifier ascertains that:

$$u \overset{?}{\equiv} \left(g^{wh(m)} y^{wu} \bmod p\right) \bmod q, \quad \text{where } w \equiv v^{-1} \bmod q.$$

2.2 Modular exponentiation algorithms

Binary exponentiation algorithms are often used for computing the DSA signatures (see Sect. 2.1). Their polynomial complexity with respect to the input length makes them very interesting to perform modular exponentiations.

The Algorithm 1 describes a way to compute modular exponentiations by scanning bits of d from least significant bits (LSB) to most significant bits (MSB). That is why it is usually referred to as the *"Right-To-Left"* modular exponentiation algorithm.

The dual algorithm that implements the binary modular exponentiation is the *"Left-To-Right"* exponentiation described in Algorithm 2. This algorithm scans bits of the exponent from MSB to LSB and is lighter than *"Right-To-Left"* one in terms of memory consumption.

Algorithm 1: *"Right-To-Left"* modular exponentiation

```
INPUT: $g, k = \sum_{i=0}^{n-1} 2^i \cdot k_i, p$
OUTPUT: $A \equiv g^k \bmod p$
1 : $A$:=1;
2 : $B$:=$g$;
3 : for $i$ from 0 upto $(n-1)$
4 :     if $(k_i == 1)$
5 :         $A := (A \cdot B) \bmod p$;
6 :     end if
7 :     $B := B^2 \bmod p$;
8 : end for
9 : return $A$;
```

Algorithm 2: *"Left-To-Right"* modular exponentiation

```
INPUT: $g, k = \sum_{i=0}^{n-1} 2^i \cdot k_i, p$
OUTPUT: $A \equiv g^k \bmod p$
1 : $A$:=1;
2 : for $i$ from $(n-1)$ downto 0
3 :     $A := A^2 \bmod p$;
4 :     if $(k_i == 1)$
5 :         $A := (A \cdot g) \bmod p$;
6 :     end if
7 : end for
8 : return $A$;
```

3 Fault Attack Against DSA signature Scheme

3.1 Fault Model

Description of the model. The model we have chosen to perform the attack is inspired by the previously used by A. Berzati *et al.* to successfully attack both *"Right-To-Left"* [5] and *"Left-To-Right"* [4] based implementation of standard RSA. We suppose that the attacker is able to inject a transient fault that modifies the public parameter p during the computation of u (see Sect. 2.1). The fault is supposed to affect randomly on byte of p such that the value of the faulty modulus $\hat{p}$ is not known by the attacker, namely:

$$\hat{p} = p \oplus \varepsilon \tag{1}$$

where $\varepsilon = R_8 \cdot 2^i, i \in [\![0; \frac{n}{8} - 1]\!]$ and R_8 is a non-zero random byte value. For the sake of clarity, we assume that the exponentiation algorithm used to implement the first part of the signature (*i.e.* the computation of u) is the *"Left-To-Right"* algorithm (see Sect. 2.2). The attacks also applies for *"Right-To-Left"* based implementations of the DSA signature scheme.

Discussion. This fault model has been chosen for its simplicity and practicability in smart card context, leading to successful applications [19,8,4]. Furthermore, it can be easily adapted to 16-bit or 32-bit architectures.
A large number of fault model have been described in the literature. Most of them are listed and discussed in [9,30]. Hence from these references, the fault model used in [2] seems to be more restrictive than the one used in our analysis since D. Naccache *et al.* consider an attacker that is able to set some bytes of k to a known value (*i.e.* some bytes of k are set to 0). On the contrary, our fault model seems to be stronger than Kim *et al.*'s one [22]. But, although their fault model is easier to practice, the significant number of fault required by the analysis represents a serious drawback. As a consequence, compared to previous work, our new fault attack is a good trade-off between practicability of the fault model and efficiency of the analysis.

3.2 Faulty Execution

This section details a faulty execution of the *"Left-To-Right"* modular exponentiation. We suppose that the fault occurs t steps before the end of the exponentiation. Let $k = \sum_{i=0}^{l-1} 2^i \cdot k_i$ be the binary representation of k and A the internal register value before the modification of p:

$$A \equiv g^{\sum_{i=t}^{l-1} 2^{i-t} \cdot k_i} \bmod p \tag{2}$$

If the first perturbed operation is a square, then the faulty first part of the signature $\hat{u}$ can be written:

$$\begin{aligned}\hat{u} &\equiv \left(\left(\left(\left(A^2 \cdot g^{k_{t-1}}\right)^2 \cdot g^{k_{t-2}}\right)^2 \ldots\right) g^{k_0} \bmod \hat{p}\right) \bmod q \\ &\equiv \left(A^{2^t} \cdot g^{\sum_{i=0}^{t-1} 2^i \cdot k_i} \bmod \hat{p}\right) \bmod q \end{aligned} \tag{3}$$

Obviously, the other part of the signature v is also infected by the fault:

$$\hat{v} \equiv \frac{h(m) + x\hat{u}}{k} \bmod q \tag{4}$$

One can notice from (3) that the perturbation has isolated the t least significant bits of k. The adaptation of the method described in [4] allows an attacker to recover this part of k.

3.3 Extraction of a part of k

The differential fault analysis against the *"Left-To-Right"* implementation of the RSA signature, described in [4], takes advantage of the difference between a correct and a faulty RSA signature to recover a part of the private exponent. This method can not be applied as itself to analyze the DSA faulty signature since k is a nonce. But, by using the properties involved in the DSA signature verification (see Sect. 2.1), the sole knowledge of the faulty DSA signature $(\hat{u}, \hat{v})$ and the input message m may be sufficient to recover the least significant part of the nonce k.

Getting a correct *u*. The correct signature part u can be obtained by using the trick of the DSA signature verification. Indeed, if $\hat{v}$ is invertible in $\mathbb{Z}_q^*$, let $\hat{w} \equiv \hat{v}^{-1} \bmod q$:

$$\begin{aligned}\left(g^{\hat{w}h(m)} \cdot h^{\hat{w}\hat{u}}\right) \bmod p &\equiv g^{\hat{w}(h(m)+x\hat{u})} \bmod p \\ &\equiv g^k \bmod p \\ &\equiv u \end{aligned} \tag{5}$$

The advantage of this method is that it requires only the knowledge of an input message and the corresponding faulty signature $(\hat{u}, \hat{v})$. The only condition to satisfy is that $gcd(\hat{v}, q) = 1$ which is always the case since q is prime and $\hat{v} < q$. Then, the attacker can compare $\hat{u}$ and u for guessing and determining the searched part of k by applying the analysis provided in [4]. Its application to the DSA is detailed below.

Guess-and-determine the part of *k*. The attacker aims to recover the least significant part of k isolated by the fault injection $k_t = \sum_{i=0}^{t-1} 2^i \cdot k_i$ (see also Sect. 3.2). Since the attacker knows that the fault occurs t steps before the end of the exponentiation and that it has randomly modified one unknown byte of p, the attacker tries to guess both k_t and $\hat{p}$. Namely, the attacker chooses a pair of candidates (k'_t, p') and first computes:

$$R_{(k'_t)} \equiv u \cdot g^{-k'_t} \bmod p \tag{6}$$

For the sake of clarity, let us rewrite u obtained with the *"Left-To-Right"* algorithm:

$$\begin{aligned} u &\equiv g^k \bmod p \\ &\equiv \left(A^{2^t} \cdot g^{k_t}\right) \bmod p \end{aligned} \tag{7}$$

By observing (6) and (7) one can notice that when $k'_t = k_t$, $R_{(k'_t)} \equiv A^{2^t} \bmod p$. So, $R_{(k'_t)}$ is expected to be a t-th quadratic residue in $\mathbb{Z}_p^*$. Hence, if $R_{(k'_t)}$ is not a quadratic residue, the attacker can directly discard k'_t. This condition is not sufficient but allows to reduce the number of computations. Since p is prime, the attacker can perform the quadratic residuosity test based on Fermat's Theorem. Hence, $R_{(k'_t)}$ is a quadratic residue if

$$\left(R_{(k'_t)}\right)^{\frac{p-1}{2}} \equiv 1 \bmod p \tag{8}$$

In this case, the attacker can compute the square roots of $R_{(k'_t)}$ using the Tonelli and Shanks algorithm [15]. This step has to be repeated until the quadratic residuosity test fails and at most t times since $R_{(k'_t)}$ is expected to be a t-th quadratic residue. At first sight, one may think that computing t-th quadratic residues of $R_{(k'_t)}$ may return 2^t possible values. In fact, this is not the case since a number randomly chosen in a cyclic group has $2^{\min(s,t)}$ t-th quadratic residues, where s is the bigger power of 2 that divides $p-1$. In the general case, the power s is lower or equal to 4. The purpose of this step is to obtain a candidate value for the internal register $A_{k'_t}$ when the fault was injected.

The next step consists in simulating a faulty end of execution from the chosen candidate pair (k'_t, p') and the previously obtained $A_{k'_t}$. Hence, the attacker computes:

$$u'_{(k'_t,p')} \equiv \left(A_{k'_t}^{2^t} \cdot g^{k_t} \bmod p' \right) \bmod q \tag{9}$$

Finally, to validate his guess for k'_t and p', the attacker checks if the following condition is satisfied:

$$u'_{(k'_t,p')} \equiv \hat{u} \bmod q \tag{10}$$

According to [5,4], the satisfaction of (10) implies that the candidate pair (k'_t, p') is very probably the right one (see App. A) and so, the attacker directly deduces the t-bit least significant part of k.

One can notice that the quadratic residuosity tests discards a majority of k'_t candidates before guessing p'. Hence, the pair of candidate values is not simultaneously, but quite sequentially, searched. So, the practical complexity of this step is smaller than the theoretical one (see Sect. 4).

The main advantage of this analysis compared to the one about the *"Left-To-Right"* implementation of RSA is that $\hat{p}$ has not to be prime for allowing the the attacker to compute square roots. So, only one faulty signature $(\hat{u}, \hat{v})$ may suffice to recover k_t.

According to [5,4], we can also perform the analysis when the first perturbated operation is a multiplication (*i.e.* instead of a square). Moreover the fault model can be extended to the perturbation of a single operation during the exponentiation (*i.e.* such that p is *error-free* for subsequent operations).

This fault analysis does not work only for the *"Left-To-Right"* implementation of the exponentiation but also for variants based on the *"Left-To-Right"* approach such that Fixed/Sliding windows [25] or the SPA-resistant "Square and Multiply Always" [16].

3.4 Extraction of the Key

The purpose of this part is to approximate the DSA secret key x as accurately as possible from public values and the recovered part of nonce. In the context of ElGamal-type signature schemes, previous results demonstrated the possibility for retrieving the secret key from partially known nonces by using lattice reduction [21,26]. This result was later applied in the context of fault attacks [26]. The following parts briefly describe how lattice attack works for obtaining the secret key from previously recovered t-bit least significant part of k. This description is inspired from the work of P.Q. Nguyen and I.E. Shparlinski [26].

Lattice attacks exploit the linearity of the second part of the faulty signature, namely: $\hat{v} \equiv \frac{h(m)+x\hat{u}}{k} \bmod q$. The partial knowledge of the nonce k causes an information leakage from $\hat{v}$ about the private key x. As a consequence, by collecting sufficiently many faulty signatures and recovering corresponding parts of k, the attacker will be able to recover x.

Let k_t be the t-bit recovered part of a nonce k and $r \geq 0$ the unknown part. Then, we have $k = r2^t + k_t$. As previously mentioned, the lattice attack takes advantage of the second part of the faulty signature, that can be written as:

$$x\hat{u} \equiv k\hat{v} - h(m) \bmod q$$

If $\hat{v} \neq 0$, we can also write

$$x\hat{u}\hat{v}^{-1}2^{-t} \equiv \left(k_t - \hat{v}^{-1}h(m)\right)2^{-t} + r \bmod q \tag{11}$$

Let us define the two following elements:

$$\begin{aligned} a &= \hat{u}\hat{v}^{-1}2^{-t} \bmod q \\ b &= \left(k_t - \hat{v}^{-1}h(m)\right)2^{-t} \bmod q \end{aligned}$$

Hence, from (11), we also have:

$$xa - b \equiv r \bmod q \tag{12}$$

Since $0 \leq r \leq q/2^t$, $\exists \lambda \in \mathbb{Z}$ such that:

$$0 \leq xa - b - \lambda q \leq q/2^t \tag{13}$$

And then:

$$|xa - b - q/2^{t+1} - \lambda q| \leq q/2^{t+1} \tag{14}$$

From the equation below, the attacker gets an approximation of xa modulo q by $c = b + q/2^{t+1}$. We can notice that the more t is high and the more accurate is the approximation. But, to apply the lattice attack and determine the secret key x, the attacker needs sufficiently many approximations. Now, suppose that the attacker has collected d faulty DSA signatures $(\hat{u}_i, \hat{v}_i)_{1 \leq i \leq d}$ and recovered for each one k_{i_t}, the t-bit least significant part of the corresponding nonce k_i (see Sect. 3.3). From these values, he can also compute a_i and c_i such that, $\exists \lambda_i \in \mathbb{Z}$:

$$|xa_i - c_i - \lambda_i q| \leq q/2^{t+1} \tag{15}$$

The problem of recovering the secret key x from this set of equations is similar to the hidden number problem introduced by D. Boneh and R. Venkatesan [12]. This problem can be solved by transforming it into a lattice closest vector problem [12,26]. Indeed, consider the lattice L generated by the row vectors of the following matrix:

$$\begin{pmatrix} q & 0 & \cdots & 0 & 0 \\ 0 & q & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & q & 0 \\ a_1 & \cdots & \cdots & a_d & 1/2^{t+1} \end{pmatrix} \in M_{(d+1),(d+1)}(\mathbb{Q}) \tag{16}$$

Then the vector $\alpha = (xa_1 + \lambda_1 q, \ldots, xa_d + \lambda_d q, x/2^{t+1})$ belongs to the lattice since it can be expressed in a linear combination of the row vectors:

$$\alpha = \sum_{i=1}^{d} \lambda_i L_i + x L_{d+1} \tag{17}$$

where L_i stands for the i-th row vector of the matrix. The vector α is also referred as the hidden vector because its last coordinate (*i.e.* $x/2^{t+1}$) directly depends on the hidden number x. From the set of equations obtained with faulty DSA signatures (see Eq. 15), the hidden vector is approximated by the vector $\beta = (c_1, \ldots, c_d, 0)$. β does not belong in the lattice but its coordinates can be computed by the attacker since it relies only on public information and the recovered part of nonce $(k_{it})_{1 \leq i \leq d}$. To find x, the attacker tries to find the closest vector to β that belongs to the lattice L. If the lattice matrix is transformed to an LLL-reduced basis [24], then it is possible to solve an appoximated instance of the closest vector problem in a polynomial time using the Babai's algorithm [3,29]. Once the closest vector is found, the attacker derives from its last coordinate a candidate value for the secret key x'. Finally, to check the correctness of the solution, the attacker has to check if the following condition is satisfied:

$$g^{x'} \equiv y \bmod p \tag{18}$$

The success rate of the lattice attack depends on d, number of faulty signatures to gather, and t, size of nonce recovered. According to [26,2], $d \approx l/t$ faulty signatures may suffice to recover x (*i.e.* l corresponds to the size of q). Our experimental results emphasise this evaluation. Furthermore, one can notice that the attacker can exploit faulty signatures perturbated at different steps t_i before the end of the exponentiation. In this case, the attacker has to collect $d \approx l / \min_i (t_i)$ faulty signatures to succeed in the lattice attack.

Finally, this analysis can be easily adapted to attack *"Right-To-Left*-based implementations of DSA. In this case, the attacker also exploit faults on p that has been injected t steps before the end of the exponentiation to recover most significant bits of k. But, if q is close to 2^{l-1}, the most significant bit of k may often be equal 0. To avoid this loss of information, D. Boneh and R. Venkatesan [12] proposed to use another representation for the MSB referred as most significant modular bits MSMB [26]. By using this representation, the attacker only has to slightly adapt the previous analysis to exploit the recovered most significant parts of random.

3.5 Attack algorithm

Our fault attack against the public parameters of the DSA signature scheme can be dividing in five distinguishable steps that have been presented throughout the paper. This section provides a summary of the attack that lists these different steps.

Step 1: Gather d faulty DSA signatures by modifying the public parameter p at t step before the end of the computation of u. The number of signatures to gather depends on the size of DSA but, in general $d \approx l/t$,

Step 2: For each faulty signature $(\hat{u}_i, \hat{v}_i)$, recover the t-bit part of the corresponding nonce k_i using the fault analysis of Section 3.3,

Step 3: From the set of faulty signatures $(\hat{u}_i, \hat{v}_i)_{1 \leq i \leq d}$ and the parts of nonce $(k_i)_{1 \leq i \leq d}$, the attacker computes the lattice matrix and the public approximation vector β (see Sect.3.4),

Step 4: The attacker applies the LLL-reduction [24] to find a reduce basis for the lattice. Then, he uses the Babai's polynomial algorithm [3] to obtain the closest vector to β that belongs in the lattice.

Step 5: Finally, the attacker extracts a secret key candidate from the last coordinate of this vector and checks if it is the right key (see Eq. (18)).

4 Performance

In order to compare the performance of our brand new fault attack against DSA public parameters, we will give a theoretical evaluation of its performance in terms of fault number and computational complexity.

4.1 Theoretical Evaluation

Fault Number. In this section, we evaluate the number of faults $\mathcal{N}$ required to recover the secret key. According to the model chosen (see Sect. 3.1), each fault injection results to an exploitable faulty output. Hence, since one part of nonce may be obtained from one faulty DSA signature (see sect. A), $\mathcal{N}$ can be approximated by the number of faulty signatures to gather for extracting the key (see sect. 3.4):

$$\mathcal{N} = \mathcal{O}\left(\frac{l}{t}\right) \tag{19}$$

For a 160-bit DSA with $t = 10$ (which is a reasonable choice according to the experimental results in Table 1), 16 faulty signatures may suffice to extract the DSA key. According to [26], the number of faults can be decreased to $\log l$. As a comparison, the best known fault attack against the DSA public elements [22] requires a mean of $4 \cdot 10^7$ faults to succeed in practice (and $2 \cdot 10^8$ in theory). This significant improvement is due to the difference of method employed and also because, in our analysis, each signature modified according to the model can be exploited and so, brings a certain amount of information about the secret key.

Computational complexity. In this section, we aim to evaluate the computational complexity of our fault analysis. According to Section 3.5, the overall complexity $\mathcal{C}$ of the attack can be expressed as:

$$\mathcal{C} = \mathcal{C}_{Lattice\ attack} + \sum_{i=1}^{\mathcal{N}} \mathcal{C}_{extract\ k_t} \tag{20}$$

First, we evaluate the complexity $\mathcal{C}_{extract\ k_t}$ for recovering a t-bit part of nonce k in the case of the *"Left-To-Right"* implementation of the DSA in the theorem below. The proof is given in Appendix B.

Theorem 1. *For a random byte fault assumption against the "Left-To-Right" implementation of a DSA such that p is a n-bit prime and $q|(p-1)$ a l-bit prime, the computational complexity $\mathcal{C}_{extract\ k_t}$ for recovering a t-bit part of the nonce is at most:*

$$\mathcal{C}_{extract\ k_t} = \mathcal{O}\left(2^{5+t} \cdot n^2 \cdot t\right) \ \textit{exponentiations} \tag{21}$$

Concerning the computational complexity of the lattice attack (see Sect. 3.4), when the closest vector approximation algorithm is used, the running time is subexponential in the size of q [26,1]. However, the exploitation of faulty signatures allows to handle quite small lattices (*i.e.* $d \leq 20$ vectors), so that the complexity of the lattice reduction step is negligible with respect to the extraction of all the $(k_t)_i$.

As a consequence, our fault attack provides an algorithm with a running time exponential in the parts of nonce to recover and subexponential in the size of the secret key. The exponential dependency in the number of bits of nonces to recover is not critical since it is a parameter set by the attacker in accordance with his capabilities. These results have been validated experimentally (see Sect. 4.2) with the NTL implementation of the Babai's algorithm.

4.2 Experimental Results

In order to evaluate the practicability of our fault attack, we have implemented the attack algorithm described in Section 3.5 using the C++ NTL Library [29] against a 160-bit DSA (*i.e.* $n = 1024$ bits and $l = 160$ bits). We have generated faulty DSA signatures from a random message by simulating faults according to the model (see Sect. 3.1). For the lattice attack, we have used as a lattice basis reduction algorithm the NTL implementation of Schnorr-Euchner's BKZ algorithm [28] and the NTL implementation of the Babai's algorithm [3] to solve the approximated instance of CVP. The experimental results detailed below were obtained on a Intel Core 2 Duo at 2.66 GHz with the `G++` compiler. First, we have estimated the time to extract parts of the nonce, for different values of t. The computation of the average time to extract the t least significant part of a nonce was obtained from 100 measures. These results for different values of t are presented in Table 1. The obtained results highlight the exponential dependency in t of the execution which emphasises our complexity analysis. As an example,

Table 1. Average time to extract t bits of nonce for a 160-bit DSA

t	4 bits	5 bits	6 bits	8 bits	10 bits
Average time	16 s	33 s	1 min	4 min 30 s	17 min

for $t = 10$ bits, it takes two hours for recovering all of the $(k_t)_i$ on a dual-core PC and a few second to recover the private key with the Lattice Attack [21,26]. But, these performances depend on the amount of k_t bits an attacker is able to recover by analyzing faulty signatures. As shown in Table 1, the choice of the number of bits of nonce to recover is a tradeoff between the time of execution and the number of faults. So this parameter has to be carefully chosen in function of the attacker's resources. Finally, one can advantageously notice that since

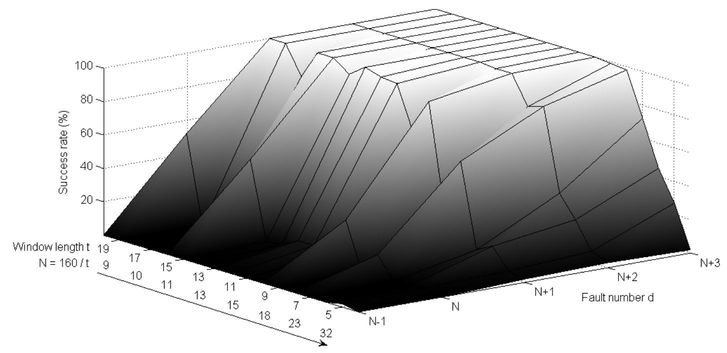

Fig. 1. Success rate of the lattice attack in function of window length t and fault number d

recovering a part of nonce only requires the analysis of one faulty signature, the attacker can recover part of multiple nonces in parallel. Thus, the attacker can optimize the fault analysis in terms of execution time.

Then, we have evaluated the performance of the lattice attack. In the condition of our experiments, it takes a few seconds for recovering the 160-bit secret key from 10-bit parts of nonces extracted from 16 signatures. Hence, the analysis algorithm is practicable even on a standard PC. The success of the attack depends on the window length t that determines the precision of the approximation and the number of faulty signatures d that are used for building the lattice. The figure 1 presents the number of times when the attack succeeds in function of t and d and shows that if d equals to the recommended value $\mathcal{N} = 160/t$ the approximation may not be sufficient when t is small. For example, if the attacker chooses to recover 6 bits of nonce because it only takes around 1 minute, with $160/6 = 26$ faults the attack only succeeds one time out of 100, but with 29 faults the gain is increased to 45%. More t and d are high, more the attack has chance of success, but increasing t takes a longer time and increasing d makes a larger lattice. Moreover reducing t implies to increase the number of faults d. So the choice of these parameters is also a tradeoff between time, ressource and fault number.

5 Conclusion

The methods used in the literature to attack some cryptosystems by perturbing public key elements can be divided into two classes. In the first class, one can modify public elements before the computation, such that the algebraic properties of the finite fields are changed and the system becomes weaker. In the second class, the perturbation can come up during the execution, splitting the computation into two parts so as to isolate a small part of the key. The DLP-based algorithm is not an exception and this paper described a practical attack against DSA and El Gamal signature schemes. This attack belongs to second class and does not require the knowledge of a correct signature. Partial values of nonces are retrieved thanks to a guess-and-determine strategy and then the secret key is derived from lattice reductions. The used fault model is the classical byte modification or any other model allowing the guess of a value. The simulation of the attack has shown its efficiency as it only requires 16 faulty signatures to recover a 160-bit DSA secret key within a few minutes on a standard PC. Moreover the simulation results confirm the complexity analysis and give some decision factor for the choice of the parameters.

This attack underlines that it is essential to protect public elements against fault attacks, for instance redundancy or infective techniques. The power of the lattice reduction techniques shows that even a small leakage of information can reveal secret information, even if it does not seem sufficient at first sight. Therefore, lattice reduction algorithms must be also be seriously taken in account in the context of fault attacks.

References

1. M. Ajtai, R. Kumar, and D. Sivakumar. A Sieve Algorithm for the Shortest Lattice Vector Problem. In *ACM Symposium on Theory on Computation (STOC 2001)*, pages 601–610, 2001.
2. F. Armknecht and W. Meier. Fault Attacks on Combiners with Memory. In B. Preneel and S. E. Tavares, editors, *Selected Areas in Cryptography (SAC 2005)*, volume 3897 of *Lecture Notes in Computer Science*. Springer, 2006.
3. L. Babai. On Lovász lattice reduction and the nearest point problem. *Combinatorica*, 6:1–13, 1986.
4. A. Berzati, C. Canovas, J-G. Dumas, and L. Goubin. Fault Attacks on RSA Public Keys: Left-To-Right Implementations are also Vulnerable. In M. Fischlin, editor, *RSA Cryptographer's Track (CT-RSA 2009)*, volume 5473 of *Lecture Notes in Computer Science*, pages 414–428, San Francisco (USA), 2009. Springer.

5. A. Berzati, C. Canovas, and L. Goubin. Perturbating RSA Public Keys: an Improved Attack. In *Cryptographic Hardware and Embedded Systems (CHES 2008)*, Lecture Notes in Computer Science, Washington DC (USA), 2008. Springer-Verlag.
6. I. Biehl, B. Meyer, and V. Müller. Differential Fault Attacks on Ellitic Curve Cryptosystems. In M. Bellare, editor, *Advances in Cryptology (CRYPTO 2000)*, volume 1880 of *Lecture Notes in Computer Science*, pages 131–146. Springer-Verlag, 2000.
7. E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *Advances in Cryptology (CRYPTO 1997)*, 1997.
8. J. Blömer and M. Otto. Wagner's Attack on a secure CRT-RSA Algorithm Reconsidered. In L. Breveglieri, I. Koren, D. Naccache, and J-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography (FDTC 2006)*, volume 4236 of *Lecture Notes in Computer Science*, pages 13–23. Springer-Verlag, 2006.
9. J. Blömer, M. Otto, and J-P. Seifert. A New CRT-RSA Algorithm Secure Against Bellcore Attack. In *ACM Conference on Computer and Communication Security (CCS 2003)*, pages 311–320. ACM Press, 2003.
10. D. Boneh, R.A. DeMillo, and R.J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In W. Fumy, editor, *EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 1997.
11. D. Boneh, R.A. DeMillo, and R.J. Lipton. "On the Importance of Eliminating Errors in Cryptographic Computations". *Journal of Cryptology*, 14(2):101–119, 2001.
12. D. Boneh and R. Venkatesan. Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes. In *Advances in Cryptology (CRYPTO 1996)*, volume 1109 of *Lecture Notes in Computer Science*, pages 129–142. Springer-Verlag, 1996.
13. E. Brier, B. Chevallier-Mames, M. Ciet, and C. Clavier. Why One Should Also Secure RSA Public Key Elements. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems (CHES 2006)*, volume 4249 of *Lecture Notes in Computer Science*, pages 324–338. Springer-Verlag, 2006.
14. D. Brumley and D. Boneh. Remote Timing Attacks are Practical. In *12th Usenix Security Symposium*, pages 1–14, 2003.
15. H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer-Verlag New-York Inc., 1993.
16. J-S. Coron. Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES 1999)*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 1999.
17. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.
18. K. Gandolfi, C. Mourtel, and F. Olivier. Electormagnetic Analysis: Concrete Results. In *Cryptographic Hardware and Embedded Systems (CHES 2001)*, volume 2162, pages 251–261. Springer-Verlag, 2001.
19. C. Giraud. DFA on AES. In V. Rijmen, H. Dobbertin, and A. Sowa, editors, *Fourth Conference on the Advanced Encryption Standard (AES4)*, volume 3373 of *Lecture Notes in Computer Science*, pages 27–41. Springer-Verlag, 2005.
20. J. Hoch and A. Shamir. Fault Analysis of Stream Ciphers. In M. Joye and J-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems (CHES 2004)*, volume 3156 of *Lecture Notes in Computer Science*, pages 240–253. Springer, 2004.
21. N. A. Howgrave-Graham and N. P. Smart. Lattice Attacks on Digital Signature Schemes. *Design, Codes and Cryptography*, 23:283–290, 2001.

22. C.H. Kim, P. Bulens, C. Petit, and J.-J.Quisquater. Fault Attaks on Public Key Elements: Application to DLP-Based Schemes. In S.F. Mjølsnes, S. Mauw, and S.K. Katsikas, editors, *European PKI workshop Public Key Infrastructure (EuroPKI 2008)*, volume 5057 of *Lecture Notes In Computer Science*, pages 182–195. Springer, 2008.
23. P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Advances in Cryptology (CRYPTO 1999)*, volume 1666, pages 388–397. Springer-Verlag, 1999.
24. A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring Polynomials with Rational Coefficients. *Mathematische Annalem*, 261(4):515–534, 1986.
25. A. J. Menezes, P. C. Van Oorschot, S. A. Vanstone, and R. L. Rivest. Handbook of Applied Cryptography, 1997.
26. P. Q. Nguyen and I. E. Shparlinski. The Insecurity of the Digital Signature Algorithm with Partially Known Nonces. *Journal of Cryptology*, 15(3):151–176, 2002.
27. National Institute of Standards and Technology (NIST). FIPS PUB 186-2: Digital Signature Standard (DSS), January 2000.
28. C. P. Schnorr and M. Euchner. Lattice Basis Reduction: Improved practical algorithms and solving subset sum problems. In *Math. Programming*, volume 66, pages 181–199, 1994.
29. V. Shoup. Number Theory C++ Library (NTL).
30. D. Wagner. Cryptanalysis of a provably secure CRT-RSA algorithm. In *Proceedings of the 11th ACM Conference on Computer Security (CCS 2004)*, pages 92–97. ACM, 2004.

A False-Acceptance Probability

This section provides a detailed analysis of the probability that a wrong pair (k_t', p') satisfies (18). Hence, the false-acceptance probability can be modeled as:

$$\begin{aligned} &\Pr\left[(10) \text{ is satisfied} \mid (k_t', p') \neq (k_t, \hat{p})\right] \\ \Leftrightarrow &\Pr\left[(10) \text{ is satisfied} \mid (k_t' \neq k_t \text{ or } p' \neq \hat{p})\right] \end{aligned} \tag{22}$$

For the sake of clarity, let us rewrite the previous equation:

$$\Pr\left[\mathrm{A} \mid \mathrm{B}\right]$$

where A denotes the event "(10) is satisfied" and B denotes the event "$k_t' \neq k_t$ or $p' \neq \hat{p}$". This probability is quite difficult to evaluate since the two events are not independent. But, using the theorem of conditional probabilities we also have:

$$\begin{aligned} \Pr\left[\mathrm{A} \mid \mathrm{B}\right] &= \frac{\Pr\left[\mathrm{A} \cap \mathrm{B}\right]}{\Pr\left[\mathrm{B}\right]} \\ &= \frac{\Pr\left[\mathrm{A}\right] - \Pr\left[\mathrm{A} \cap \bar{\mathrm{B}}\right]}{\Pr\left[\mathrm{B}\right]} \\ &< \frac{\Pr\left[\mathrm{A}\right]}{\Pr\left[\mathrm{B}\right]} \end{aligned} \tag{23}$$

The equation (10) belongs in $\mathbb{Z}_q{}^*$, so, the probability that (10) is satisfied is:

$$\Pr\left[\mathrm{A}\right] = \frac{1}{q-1} \tag{24}$$

Knowing that k_t is a t-bit value and that the model chosen for $\hat{p}$ is a random byte fault model, we can also evaluate $\Pr\left[\mathrm{B}\right]$:

$$\begin{aligned}\Pr\left[\mathrm{B}\right] &= \Pr\left[k'_t \neq k_t\right] + \Pr\left[p' \neq \hat{p}\right] \\ &= \frac{t-1}{t} + \frac{P-1}{P}, \text{ where } P = \frac{n(2^8-1)}{8}. \end{aligned} \tag{25}$$

The values of k_t and p' are independent, that why the term of intersection is null in the previous expression. From these partial results, we can deduce that:

$$\begin{aligned}\Pr\left[\mathrm{A} \mid \mathrm{B}\right] &< \frac{1}{q-1}\left(\frac{1}{\frac{2^t-1}{2^t} + \frac{P-1}{P}}\right) \\ &< \frac{1}{q-1}\left(\frac{2^tP}{P(2^t-1)+2^t(P-1)}\right) \\ &< \frac{1}{q-1} \end{aligned} \tag{26}$$

where $P = \dfrac{n(2^8-1)}{8}$.
Hence, the false-acceptance probability is bounded by:

$$0 \leq \Pr\left[(10) \text{ is satisfied} \mid (k'_t, p') \neq (k_t, \hat{p})\right] < \frac{1}{q-1} \tag{27}$$

As a consequence, this probability is negligible usual values of q (*i.e.* q is a 160-bit value for a 160-bit DSA).

B Proof of the Theorem 1

According to the analysis described in Section 3.3, the attacker has to guess-and-determine both the faulty modulus $\hat{p}$ and k_t. Hence, according to the random byte fault model, the attacker has to test at most $\frac{n}{8}\cdot\left(2^8-1\right)$ possible values for $\hat{p}$ and 2^t for k_t. Moreover, for each candidate pair, the attacker may have to perform some quadratic residuosity tests and depending on the result, apply the Tonelli and Shanks algorithm to obtain square roots. At most, the computation of square roots will require to perform t tests followed by the computation of square roots. Since the complexity of a quadratic residuosity test is one exponentiation and the complexity of the Tonelli and Shanks algorithm is $\mathcal{O}\left(n\right)$ exponentiations, the complexity of the nonce extraction is:

$$\begin{aligned}
\mathcal{C}_{extract\ k_t} &= \mathcal{C}_{candidate\_pairs} \cdot \mathcal{C}_{square\_roots} \\
&= \mathcal{O}\left(\frac{n}{8} \cdot \left(2^8 - 1\right) \cdot 2^t \cdot t \cdot n\right) \\
&= \mathcal{O}\left(2^{t+5} \cdot n^2 \cdot t\right) \text{ exponentiations } \square
\end{aligned} \tag{28}$$