Lattice-based Fault Attacks on DSA - Another Possible Strategy

Tomáš Rosa

trosa@ebanka.cz

eBanka, a.s.
Václavské náměstí 43
109 00 Prague 1, Czech Republic, EU

Abstract

We start by describing a realistic DSA signing procedure which tries to defeat fault attacks by employing an implicit verification of every signature made. Only correct signatures are returned from the procedure. We then show that, despite looking robust on a first glimpse, such a procedure cannot be regarded as being automatically resistant against fault attacks. We demonstrate this by showing a realistic fault attack that passes undetected by the procedure. Furthermore, it is even somehow accelerated and hidden thanks to it. The attack described in this paper can also have a real use. We emphasize, however, that instead of breaking a concrete implementation of DSA an effort was mainly made to deliver an existential proof of that we shall not rely solely on a paradigm of implicit verification when we design a defence against fault attacks.

Keywords: DSA, fault attack, implicit verification, lattice, cryptanalysis.

1 Introduction

It is well known cryptanalytical result that the security of a DSA [7] private key strongly depends on statistical properties of temporary nonces (i.e. Numbers-used-ONCE; usually denoted as *k*) used for a particular signature generation. Such a nonce must have a uniform distribution on a certain interval and must be kept secret. Otherwise, an undesirable subliminal side channel is created that enables a consecutive leakage of the private key information in every signature made. Having collected enough such signatures, an attacker can recover the whole private key in a doable time on a general personal computer or a workstation. In 2002, Nguyen and Shparlinski presented a theoretically stable and practically very fruitful approach [11] to private key recovering which employs a lattice-based solution of a Nguyen's variant of the hidden number problem (HNP) introduced in 1996 by Boneh and Venkatesan [3]. They results show, for instance, that we can recover the whole private key knowing only as few as the lowest three bits of each nonce for only 100 signatures made.

In 2004, the paper of Naccache, Nguyen, Tunstall, and Whelan was made public on IACR's ePrint (eprint.iacr.org) [10]. It connects the results obtained by Nguyen and Shparlinski together with a vulnerability to a fault injection observed for a certain kind of smartcard. Naccache et al. demonstrated that it was possible to use the fault injection to substitute known values for the lowest bytes of each nonce *k*. Using the aforesaid theory, they were able to recover the whole private key from 38 signatures made (actually, only 30 ones of them were selected for the computation). Besides the others, it is interesting to observe that this is such a kind of fault attack that cannot be prevented simply by checking each signature for faults by verifying its validity using a public key. Videlicet, every signature made in this way is valid. So, despite seeming robust on a first look, it turns out that this countermeasure is totally useless against this attack. In this article, we go further this way. We present such a lattice-based fault attack on the DSA scheme that becomes even more dangerous when the device under attack behaves according to the following scenario: Using the public key, the device checks every signature made whether it is valid or not. Only valid signatures can be read from the device. Furthermore, the module restarts the signing procedure automatically, until a valid signature is computed or the number of attempts is out of a predefined boundary. Since designers of such a module would probably require certain level of robustness and independence, we may reasonably assume that the device would allow even hundreds of attempts before it blocks. Another possible scenario includes an application that calls a cryptographic module using a data link which is not error-free. This application would again test resulting signatures for their validity and it would allow repeating the computation several times until a valid signature is received. We show that there exists a fault attack based on a private key modification (only public parameters are modified) which enables the private key leakage in every signature made in the aforesaid way. We then use a slight modification of the lattice-base approach by Nguyen and Shparlinski [11] to recover the whole private. Basing on the theory and experimental results presented in [11] and [10], we can estimate that the attack would require practically feasible amounts of signatures, time, and memory space.

The rest of the paper is organized as follows: In §2, we define an expected behaviour of an attacked module together with the modification of public parameters that creates a hidden fault side channel. In §3, we describe how to employ lattice-based technique of [11] to recover the value of DSA private key basing on the information obtained from the side channel constructed in §2. General countermeasures are then discussed in §4 and finally, we conclude our observations in §5.

2 The Core of the Fault Attack

2.1 The Signing Algorithm with an Implicit Verification

Let (p, q, g) denote DSA public parameters according to [7]: p, q are primes, such that $2^{1023} < p < 2^{1024}$, $2^{159} < q < 2^{160}$, $q \mid p - 1$, and g is a generator of a cyclic multiplicative subgroup $\boldsymbol{G}$ of $\boldsymbol{Z}_p^*$ of order $|\boldsymbol{G}| = q$. Furthermore, let x be a private key, $x \in \boldsymbol{Z}$, $0 < x < q$, and let y be a public key, $y = g^x \bmod p$. We assume that a cryptographic module employing the implicit verification strategy (possibly as a countermeasure against fault attacks) would behave according to the following algorithm.

Algorithm 1. Signing a message using DSA with implicit verification.

Input: Message to be signed m, private key x, public key y, public parameters (p, q, g), repeat boundary B.

Output: Signature (r, s) or FAILURE.

Computation:

```
1.  Let $i = 1$.
2.  Choose an integer nonce $k$ at random, such that $0 < k < q$.
3.  Compute $r = (g^k \bmod p) \bmod q$.
4.  Compute $s = (h(m) + rx)k^{-1} \bmod q$, where $kk^{-1} \equiv 1 \pmod q$ and $h$ denotes the hash function SHA-1[6].
5.  If $r = 0$ or $s = 0$ then go to 2.
6.  Compute $u = h(m)s^{-1} \bmod q$, where $ss^{-1} \equiv 1 \pmod q$.
7.  Compute $v = rs^{-1} \bmod q$.
8.  Compute $w = (g^u y^v \bmod p) \bmod q$.
9.  If $w = r$ then return ($r$, $s$).
10. $i \leftarrow i + 1$
11. If $i > B$ then return FAILURE.
12. Go to 2.
```

■

As we can see, the algorithm describes formally what a programmer would do naturally if she was asked to implicitly verify every signature made whether it is valid or not before letting it go out from a cryptographic module. Steps 2 to 4 cover the signature generation, while steps 5 to 9 do the signature verification. Both parts are written according to [7]. Another thing that would the programmer do naturally in such a situation is to employ an automatic repeat function which would retry the signing operation several times before the algorithm echoes a failure to a calling process. This repeating is driven by the boundary denoted as B which sets the maximum number of signing procedure re-runs. Note that for a small value of the boundary (circa $B \leq 20$), such an algorithm can also originate due to a user activity: The user, for instance, wants to send a signed e-mail, while the device says that there is something wrong about a signing module. We can reasonably expect that she would try to sign her message several times before she gives it up. The more eager the user is the higher B we get.

2.2 Embedding the Fault Side Channel

On a first glimpse, Algorithm 1 described above can be regarded as being resistant against fault attacks, since no faulty signature can leave perhaps the innermost place of the cryptographic module. Such a reasoning which could be inspired by typical symptoms of fault attacks on RSA (c.f. [2], [8], [13], [15]), can, however, be terribly misleading here. An example of fault attack that passes undetected in such a situation can be found in [10]. In the following text, we describe a fault attack that can be even right allowed thanks to relying on such a "fault tolerant" algorithm.

Let d be an integer, such that $d \mid p - 1$ and $\gcd(d, q) = 1$. Furthermore, let β be an integer, $1 < \beta < p$, of order $\mathrm{ord}(\beta) = d$ in $\boldsymbol{Z}_p^*$. Now, we will observe what happens if an attacker substitutes the value of $g' = g\beta \bmod p$ in place of g in Algorithm 1. Such a change can be theoretically possible since g is a part of public parameters whose protection architects often tend to underestimate. For instance, in the CryptoAPI subsystem of the MS Windows platform, there is a function CryptSetKeyParam with the parameter KP_G reserved for such a purpose [9]. It is left up to designers of cryptographic modules how to implement this function and whether to allow such modifications at all. There is, however, no warning about how dangerous this functionality can be. Therefore, we may reasonably assume that at least some architects will allow the attacker to freely change the value of g. Several problems with integrity of a key material were also identified by Clulow for the PKCS#11 security standard platform [5]. There was also a successful attack based on DSA public parameters modification described by Klíma and Rosa in [13]. We shall, therefore, fully anticipate the possibility of such a modification when we discuss security aspects of a particular signing procedure.

Now, let us denote r', s', and v' the variables from Algorithm 1 computed for a substituted value of $g' = g\beta \bmod p$. We can write:

$$r' = (g^k\beta^k \bmod p) \bmod q,$$

$$s' = (h(m) + r'x)k^{-1} \bmod q,\ kk^{-1} \equiv 1 \pmod q.$$

Since here, the situation differs in that which value of the generator g is used in step 8 of the algorithm. We denote A the case where the original value g is used here. It occurs when the module stores the public parameters independently in both records of the private and public keys. If the programmer aimed for getting independent verification of the signature, this will be the most probable case. We then denote as B the situation where the substituted value of g' is used in step 8. We show that the fault attack is possible for both variants A and B. Let us denote w_A' and w_B' the values computed according to variants A and B, respectively. We can write:

$$w_A' = (g^k \bmod p) \bmod q = w,$$

$$w_B' = (\beta^u g^k \bmod p) \bmod q.$$

Basing on the aforesaid equations, we can rewrite the condition the signature (r', s') must pass in step 9 as:

$$\text{case A: } w_A' = r' \Leftrightarrow (g^k \bmod p) \bmod q = (g^k\beta^k \bmod p) \bmod q,$$

$$\text{case B: } w_B' = r' \Leftrightarrow (\beta^u g^k \bmod p) \bmod q = (g^k\beta^k \bmod p) \bmod q.$$

Since we can neglect an influence of "inner" collisions in the mapping $\varphi(k) = (g^k\beta^k \bmod p) \bmod q$ (c.f. [4], [14]), we can claim that with a probability close to 1 the following conditions are necessary and sufficient to release the signature (r', s') in step 9:

$$\text{case A: } \beta^k \bmod p = 1,\ \text{i.e. } k \equiv 0 \pmod d, \qquad \text{(CA)}$$

$$\text{case B: } \beta^u \bmod p = \beta^k \bmod p,\ \text{i.e. } k \equiv u' \pmod d,\ \text{where } u' = h(m)(s')^{-1} \bmod q. \qquad \text{(CB)}$$

Therefore, in both cases, we get nontrivial direct information on k whenever Algorithm 1 releases a signature pair (r', s'). This a vital side channel that we can use to recover the whole value of the private key using a slightly modified approach from [11].

2.3 The Amount of Information We Can Obtain

Definition 1. Binary equivalent of partial information.

Let k, d be integers, $d > 0$. We say that we get a partial information on k with a binary equivalent $l = \log_2 d$ if we get a value of $a = k \bmod d$.

■

Obviously, the higher the binary equivalent of information obtained from a side channel defined in §2.2 is the better information on k we get and the less signatures will be necessary to recover the private key. However, we can also see that the higher the binary equivalent is the higher the boundary B (c.f. Algorithm 1) must be to allow the algorithm to release a "correct" signature in step 9. Basing on the conditions (CA) and (CB) we can estimate that the signature is released in step 9 with a probability $\Pr = d^{-1} = 2^{-l}$ in both cases. Therefore, we can estimate that the mean value of the number of passes through the main loop of Algorithm 1 is $E = d = 2^l$ with a variance $D = \sigma^2 = d(d-1) = 2^l(2^l - 1)$. We shall, therefore, ensure at least that $B \geq d$, while safer would be (using a common rule of thumb also known as "six sigma") $B \geq d + 3*d^{1/2}(d-1)^{1/2}$.

F	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
median	4.0	4.58	5.58	6.39	7.23	8.1	9.02	9.87
mean	4.46	5.18	6.27	7.13	8.06	9.01	10.02	10.99

Table 1: Basic statistics of possible information leakage.

Another obvious condition that governs how much information we get is that $d \mid p - 1$. To demonstrate this aspect, we made several experiments to find the largest d that divides the value of $p - 1$ for a randomly generated instances of DSA. The process of generation of p was made exactly according to [7]. Table 1 shows median and mean value of a random variable X, such that $x = \max\{ \log_2 n : n \mid p - 1, n = \Pi_{(i)} p_i^{e_i}, p_i \leq F \}$, where F is a an upper bound of prime factors in n. We did the measurements for $F = 2^3, 2^4, \ldots, 2^{10}$ using 100 000 of DSA instances generated randomly according to [7]. Basing on the results obtained, we may expect it to be possible to find a suitable value of d for a general DSA instance.

3 Lattice-based Computation of the Private Key

Recall that since the method described in [11] is focused on a bit-oriented partial information, it describes the partial knowledge a of the nonce k in a form

$$k = b*2^l + a, \text{ where } a = k \bmod 2^l. \qquad \text{(A1)}$$

It is easy to observe that the main idea of [11] also applies for a partial knowledge a in a form:

$$k = b*d + a, \text{ where } a = k \bmod d,\ d \in \boldsymbol{Z}, d > 0, \gcd(d, q) = 1. \qquad \text{(A2)}$$

A sketch of the proof can be obtained by setting $l = \log_2 d$ and observing that there is no need for l to be in $\boldsymbol{Z}$ in the central computation from [11]. Note that l is a binary equivalent of d (c.f. Definition 1). We can expect the private key recovering based on (A2) to be as effective as the computation based on (A1) for a particular value of the binary equivalent.

Let (r', s') be a signature value computed with a substituted value of the generator g'. Furthermore, let $h(m)$ be an SHA-1 hash code of a message the signature belongs to. Using the approach of (A2) together with the relations from either (CA) or (CB), we get the following partial information on k denoted as a:

$$\text{case CA: } a = 0,$$

$$\text{case CB: } a = (h(m)(s')^{-1} \bmod q) \bmod d.$$

Now, we compute the coefficients t and u, such that:

$$t = rs^{-1}d^{-1} \bmod q, \text{ where } ss^{-1} \equiv 1 \pmod q \text{ and } dd^{-1} \equiv 1 \pmod q,$$

$$u = [(a - h(m)s^{-1})d^{-1}] \bmod q + q/2d,\ u \in \boldsymbol{Q}.$$

Recall that since $b < q/d$, (t, u) is an approximation of the private key x satisfying:

$$| xt - u |_q \leq q/2d,$$

$$\text{where } | y |_q = \min_{c \in \mathbf{Z}} | y - cq | = \min \{ y \bmod q, q - (y \bmod q) \}.$$

Now, we collect N such pairs (t_i, u_i) for N signatures (r_i', s_i') computed by Algorithm 1. The value of N can be estimated using the binary equivalent of a particular d chosen by an attacker. Basing on the experimental results described in [11] and [10], we can say, for instance, that if $l \geq 8$ then it should suffice to set $N = 27$ (c.f. Table 1 in [10]). For $l \geq 4$, $N = 70$ should be sufficient in 90% of attacks [11].

Having collected N pairs $(t_1, u_1), \ldots, (t_N, u_N)$, we construct a full-rank lattice $L(q, d, t_1, \ldots, t_N)$ of a dimension $\dim(L) = N + 1$ with the basis $\boldsymbol{B} = (\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_N, \boldsymbol{b}_{N+1})$, where $\boldsymbol{b}_i \in \boldsymbol{Q}^{N+1}$, $\boldsymbol{b}_1 = (q, 0, \ldots, 0)$, $\boldsymbol{b}_2 = (0, q, 0, \ldots, 0)$, …, $\boldsymbol{b}_N = (0, 0, \ldots, q, 0)$, $\boldsymbol{b}_{N+1} = (t_1, t_2, \ldots, t_N, 2^{-1}d^{-1})$. Furthermore, we prepare a rational vector $\boldsymbol{u} = (u_1, \ldots, u_N, 0)$. Using an appropriate randomized polynomial-time algorithm for solving *approximate closest vector problem* ([11], [12]), we find a vector $\boldsymbol{v} \in L$ satisfying $\| \boldsymbol{v} - \boldsymbol{u} \| \leq f(\dim(L)) \min_{\boldsymbol{a} \in L} \| \boldsymbol{a} - \boldsymbol{u} \|$, where $\| \cdot \|$ is the Euclidean norm and f, $f(\dim(L)) \leq 2^{\dim(L)/4}$, is a parameter given by the particular method chosen. The concrete function $f(\dim(L)) = 2^{\dim(L)/4}$ corresponds with the Babai's Nearest plane algorithm [1] with an adjusted definition of the LLL-reduced lattice basis [3]. It was shown in [11] that with a high probability we get $\boldsymbol{v}$ satisfying $v_{N+1} = (x + \gamma)2^{-1}d^{-1}$, where $\gamma \equiv 0 \pmod q$. From here, we can recover the private key now as:

$$x = 2dv_{N+1} \bmod q.$$

4 Countermeasures

An obvious countermeasure would be to prevent a modification of the private key together with the public parameters used during a signature generation. We shall not, however, discuss this kind of countermeasures, since we did not strive to show that a modification of these values can lead to a successful attack on the private key. This result was already well known for a while (c.f., for instance, [2], [13], [15]). According to this subject, we only note that checking the integrity of private key parameters *before* the computation starts may not be enough, since an attacker can force their change right during the computation (c.f., for instance, [10]). Let us, however, return to the central motif of the paper. The thing we attacked is actually quite different, since we attacked mainly on a paradigm of automatically using implicit verification of digital signatures as a robust countermeasure against fault side channels. The main remedy is, therefore, to use such an approach very carefully. At first, we must ensure that there is no automatic repeat of the signing procedure in case of a faulty result. If an error is detected then the cryptographic module shall enter a “service needed” state in which it rejects any further signing attempts. An administrator shall then reload the device completely including the records of public parameters, a private key, and a public key. If the faulty behaviour persists then the module must undergo a deep repair procedure.

5 Conclusion

We saw not only that implicit verification cannot be regarded as a robust countermeasure against fault attacks, since there are strategies that pass undetected by it. We also saw that there are attacks which can be even right allowed thanks to relying on the “power” of implicit verification process. Therefore, despite being a bit paradoxical on a first glimpse, we shall use this approach very carefully in a cryptographic modules design. Of course, this is not to say that we shall not use it at all. We just want to emphasize that we must not rely *solely* on this approach and that we have to design and implement it properly. The caution mainly addresses the phase of a design verification in which we shall check every possible attack scenario to see whether our implementation can resist it or not. Otherwise, the situation about fault attacks can become even worse, since they might become somehow hidden and accelerated.

References

[1] Babai, L.: On Lovász' Lattice Reduction and the Nearest Lattice Point Problem, *Combinatorica*, 6:1-13, 1986.

[2] Boneh, D., DeMillo, R.-A., and Lipton, R.-J.: On the Importance of Checking Cryptographic Protocols for Faults, in *Proc. of EUROCRYPT '97*, pp. 37-51, Springer-Verlag, 1997.

[3] Boneh, D., and Venkatesan, R.: Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes, in *Proc. of CRYPTO '96*, pp. 129-142, Springer-Verlag, 1996.

[4] Brown, D.-R.-L.: Generic Groups, Collision Resistance, and ECDSA, *IEEE 1363 report*, February 2002.

[5] Clulow, J.: On the Security of PKCS #11, in *Proc. of CHES 2003*, pp. 411-425, Springer-Verlag, 2003.

[6] FIPS PUB 180-1: Secure Hash Standard (SHA-1), *National Institute of Standards and Technology*, January 2001.

[7] FIPS PUB 186-2: Digital Signature Standard (DSS), *National Institute of Standards and Technology*, January 27, 2000, update: October 5, 2001.

[8] Joy, M., Lenstra, A.-K., and Quisquater, J.-J.: Chinese Remaindering Based Cryptosystems in the Presence of Faults, *Journal of Cryptology*, Vol. 12, No. 4, pp. 241-245, Autumn 1999.

[9] Microsoft CryptoAPI, *MSDN Library*, October 2001.

[10] Naccache, D., Nguyen, P.-Q., Tunstall, M., and Whelan, C.: Experimenting with Faults, Lattices and the DSA, in *Proc. of Public Key Cryptography – PKC'05*, pp. 16-28, Springer-Verlag, 2005.

[11] Nguyen, P.-Q., and Shparlinski, I.-E.: The Insecurity of the Digital Signature Algorithm with Partially Known Nonces, *Journal of Cryptology*, Vol. 15, No. 3, pp. 151-176, Springer-Verlag, 2002.

[12] Nguyen, P.-Q., and Stern, J.: The Two Faces of Lattices in Cryptology, in *Proc. of Cryptography and Lattices – CALC'01*, pp. 146-180, Springer-Verlag, 2001.

[13] Klíma, V., and Rosa, T.: Attack on Private Signature Keys of the OpenPGP format, PGP$^{(TM)}$ Programs and Other Applications Compatible with OpenPGP, *IACR ePrint archive*, 2002/076, http://eprint.iacr.org, 2001

[14] Rosa, T.: On Key-collisions in (EC)DSA Schemes, *CRYPTO 2002 Rump Session*, IACR ePrint archive 2002/129, Santa Barbara, USA, August 2002.

[15] Rosa, T.: Modern Cryptology – Standards Are Not Enough, *Ph.D. Thesis*, 2004.