A Novel Fault Attack Against ECDSA

Alessandro Barenghi*, Guido Bertoni†, Andrea Palomba* and Ruggero Susella†
*Dipartimento di Elettronica e Informazione, Politecnico di Milano, 20133 Milano (MI), Italy
Email: barenghi@elet.polimi.it, palomba@elet.polimi.it
†STMicroelectronics, 20041 Agrate Brianza (MB), Italy
Email: guido.bertoni@st.com, ruggero.susella@st.com

***Abstract*—A novel fault attack against ECDSA is proposed in this work. It allows to retrieve the secret signing key, by means of injecting faults during the computation of the signature primitive. The proposed method relies on faults injected during a multiplication employed to perform the signature recombination at the end of the ECDSA signing algorithm. Exploiting the faulty signatures, it is possible to reduce the size of the group of the discrete logarithm problem warranting the security margin up to a point where it is computationally treatable. The amount of faulty signatures requested to perform the attack is relatively small, ranging from 4 to a few tenths. The key retrieval can be applied to any key length, like those standardised by NIST, including the ones mandated for top secret documents by NSA suite B. The required post processing of the obtained faulty values is practical on a common consumer grade desktop.**

The procedure does not rely on any particular structure of the employed curve and may easily be extended to the regular DSA based on modular arithmetics.

I. Introduction

The need to provide a sound and secure way to warrant the authenticity of digital contents is nowadays a growing requirement for modern computing systems. This need is fulfilled by cryptographic digital signature protocols. Among them, the most innovative and standardised cryptosystem able to provide a signature scheme for digital contents is represented by the Elliptic Curve Digital Signature Algorithm (ECDSA), which has been standardised by both NIST [1] and IEEE. In particular, thanks to the high security warranties, it has also been recommended to sign top secret information in the NSA Suite B. The ECDSA cryptosystem is composed of two primitives: a signing and a signature verification algorithm. The signing algorithm is able to produce an authenticated token, the digital signature, employing a secret value known only to the signer, while the signature verification algorithm checks for the authenticity of the signature, employing only publicly known values. The retrieval of the secret key by an attacker allows him to forge valid signatures at will, thus voiding any authenticity warranty provided by the scheme. Therefore, the security margin of the ECDSA cryptosystem relies on the difficulty of deriving the secret value relying only on the knowledge of the publicly available parameters and the digital signature. In a scenario where the devices performing the signature may be seized by an attacker, it is important that the secret needed in order to build correct signatures cannot be extracted from the device holding it. A common practical scenario for an attack is represented by an attacker gaining access to a signing token containing the secret key, such as a smart card, and willing to duplicate it, before returning the original one to the legitimate owner. The open literature provides few examples of a secret key retrieval attacks. The authors of [2] rely on both a particular arithmetical representation of the values involved in the algorithm and the possibility of inducing single bit faults in a specific bit of a single value with a rather strict time accuracy. In [3] an attack is presented relying on a fault on the modulus used for the computations, but needs a few thousands faulty results to be successful against a system employing a small-sized curve. Eventually, in [4] an instruction skip fault is used to recover some bits of the nonce used during the signature. A few tenths of faulty results are again sufficient for a small-sized curve.

This work will present a fault based attack able to retrieve the full secret key, for all the standardised lengths, through the injection of a single bit fault, without any restriction on the specific position of the bit in the word. The attack is practically viable for all the real-world word lengths of the target architecture. The remainder of this work is organized as follows: Section II provides the mathematical background on elliptic curve cryptography and the fault model assumed by our attack, Section III describes the new attack technique in detail and IV provides an analysis of the practical means needed to carry out the attack. Finally, Section V summarises our contribution.

II. Mathematical background and fault model

A. Preliminaries on Elliptic Curves

The domain of operation of the ECDSA algorithm is the set of integers modulo p denoted usually as $\mathbb{Z}_p$. For the sake of clarity, in the remainder of the paper the $\bmod p$ notation will be omitted and the modular reductions will be implicitly assumed for all the coordinates of the points of the curve. The key mathematical object employed to build a computationally hard problem in the ECDSA cryptosystem is a particular algebraic group employing as a support the points of an elliptic curve defined over $\mathbb{Z}_p$. Given a field $(\mathbb{Z}_p, +, \cdot)$, where p is a large prime number, an elliptic curve $\mathbb{E}$ is represented by the set of points with coordinates over $\mathbb{Z}_p \times \mathbb{Z}_p$ where the following relation holds:

$$y^2 = x^3 + a_4 x + a_6,$$

and it is denoted either as $\mathbb{E}[\mathbb{Z}_p]$ or, simply as $\mathbb{E}$. The set of points $\mathbb{E}$ is used as a support for a commutative group $(\mathbb{E}, +)$, where the $+$ operator denotes the so-called *point addition* operation. The *point addition* operation between two points of the curve $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ is suitably defined to be an associative and commutative operation. The identity element of the operation is a particular point, known as *point at infinity* denoted as $\mathcal{O}$, while the inverse of the point $P = (x_P, y_P)$ is defined as $-P = (x_P, -y_P)$. The order n of the group $(\mathbb{E}, +)$ lies within the bounds stated by Hasse's theorem [5], i.e.:

$$p + 1 - 2\sqrt{p} \leq n \leq p + 1 + 2\sqrt{p}. \quad (1)$$

We remark that, for all the curves over $\mathbb{Z}_p$ standardized by NIST, n is prime. Since the group $(\mathbb{E}, +)$ is cyclic, a point

G of maximum order is selected as the default *generator* and standardized together with the other curve parameters for protocols definitions.

In order to construct the one-way function which will be exploited in order to build the ECDSA cryptosystem, the group law is used to build an external operation, called *scalar multiplication*.

Given an integer $k \in \mathbb{Z}$, and a point $P \in \mathbb{E}$ the scalar multiplication operation is defined as

$$[k]P = \underbrace{P + P + \ldots + P}_{k \text{ times}}.$$

i.e. as the iterated sum of a point. The result of $[0]P$ is defined as $\mathcal{O}$, and it follows from the definition of the operation that $[-k]P = -[k]P$. It is thus possible to regard the group $(\mathbb{E}, +)$ where the curve has prime order n as isomorphic to $(\mathbb{Z}_n, +)$ through the map $k \leftrightarrow [k]P$. The operation on the elliptic curve employed as a trapdoor function is the aforementioned point scalar multiplication. It is in fact possible to compute efficiently the scalar multiplication operation through the use of a double and add strategy [6], which has a complexity of $O(\log(k))$ point additions. On the other hand, given a point $Q \in \mathbb{E}$ and a generator of the additive group over the curve P, it is computationally hard to find the value of k such that $[k]P = Q$. This problem is known in open literature as the Elliptic Curve Discrete Logarithm Problem (ECDLP) and the best algorithms known at present time belong to the $O(n)$ complexity class (with n the order of the curve).

The ECDSA suite relies on the hardness of the ECDLP problem in order to produce a signature token from a secret value k, which represents an ephemeral key held only during the signature procedure duration. This protocol for digital signatures is derived from the classic DSA [1], through substituting the discrete logarithm problem over a number field with the one constructed over the $(\mathbb{E}, +)$ curve points group. The signing algorithm of ECDSA is described in Algorithm 1.

Algorithm 1 ECDSA Signature Generation

Input: curve parameters $(\mathbb{E}, G)$, private key d, message m
Output: signature $\mathcal{S}$
1: $e \leftarrow \text{hash}(m)$
2: $k \leftarrow \text{random} \in [1, n-1]$
3: $P \leftarrow [k]G$
4: $r \leftarrow x_P \bmod n$
5: $s \leftarrow (e + rd)/k \bmod n$
6: **return** $\mathcal{S} = (r, s)$

In particular, the signature generation algorithm (Algorithm 1) produces the signature token S, taking as input the definition of the group $(\mathbb{E}, +)$ together with a default generator $G \in \mathbb{E}$, the private key parameter $k \in \mathbb{Z}_p$ and the message of which authenticity must be warranted m. In order to build the signature, the algorithm at first obtains a hashed version e of the message m (Line 1) and a non zero random number (Line 2), smaller than the order of the curve. Subsequently, the point scalar multiplication between the random number and the generator G is performed (Line 4) and the x coordinate of the resulting point is divided by the order of the curve n and stored in r. In case $r = 0$ the procedure is re-run with a different random number until a non-zero r is obtained. Finally the signature is computed through combining together the hash of the message, the value obtained through the point scalar multiplication and the extracted random k (Line 5). The signature token S is represented by the pair (r, s). It is particularly important to choose a cryptographically strong random number for k and never reuse it: it is trival in fact to extract the value of the secret key d if two different signatures are computed with the same random k.

In order to check if an ECDSA signature is valid, the verifier is provided with the public key $Y = [d]G$, where the secret value d is protected by the computational hardness of the ECDLP. The verifier proceeds to compute the message hash e and compares the received r value with $[e/s]G + [r/s]Y$. If the two quantities match, the provided signature is valid.

In order to attack this scheme this paper will be considering a single bit flip fault model. The fault is assumed to be transient and to be hitting a single multiplication as detailed in the next section. The required fault model can be practically realised either with cheap equipment through tampering with either the supply voltage of the device, as reported in [7], or through the proper employment of a laser fault injection station, which allows a greater precision in the injection, depending on the targeted device [8]. Both these methods have been shown to be widely practical and are recognised as a realistic fault injection methodology.

III. Attacking Technique Details

In this section we present the details of the attack development. Starting from the analysis of the computation that is carried out in faulty conditions, we develop a method to recover information on some internal data that is not supposed to be known outside the device. Namely, after the analysis of one faulty result it will be possible to come to know one of the words of either the key d or the intermediate value $h = e + rd$. A collection of some of these values, coming from different faulty results, will then be used to reconstruct the whole secret key d.

We will, at first, analyse the propagation of the error caused by the fault to the final signature result, and show a fast method to recover secret data from the analysis of one single signature. We will subsequently deal with possible issues which arise from the fact that the actual fault position might be unknown. Finally, we will show how to recombine the results obtained from the analysis of different signatures to obtain the whole secret key d.

A. Error Propagation

We analyse in this section the error propagation during the signature computation, so as to derive an expression for the final faulty signature. The two modular multiplications in the signature recombination (Algorithm 1, Line 5) are usually performed in the most straightforward way: an integer multiplication is followed by a modular reduction step. We will now focus on the integer multiplication step, showing later that the modular reduction does not influence the results we obtain. Multiple precision integer multiplication may be implemented in a number of ways: we choose to tackle the *operand scanning* method since it is widely used and it has been adopted as the method of choice in the OpenSSL Toolkit.Nonetheless the results do not change when different multiplication algorithms are used. Algorithm 2 reports the

Algorithm 2 Operand Scanning Integer Multiplication

Input: $a = (a_{\sigma-1} \cdots a_1 a_0)_{2^w}$
$b = (b_{\sigma-1} \cdots b_1 b_0)_{2^w}$
Output: $c = a \cdot b = (c_{2\sigma-2} \cdots c_1 c_0)_{2^w}$
1: $c_i \leftarrow 0 \; \forall i \in [0, 2\sigma - 2]$
2: **for** i in 0 to $\sigma - 1$ **do**
3: $u \leftarrow 0$
4: **for** j in 0 to $\sigma - 1$ **do**
5: $(u, v)_{2^w} \leftarrow c_{i+j} + a_j b_i + u$
6: $c_{i+j} \leftarrow v$
7: **end for**
8: $c_{i+\sigma} \leftarrow u$
9: **end for**
10: **return** c

operand scanning multiplication algorithm for an architecture with w-bit wide words.We will denote with σ the number of architectural words needed to represent a multiple precision number.

The algorithm is basically formed by two nested loops during which each pair of words of the operands are multiplied and the result is added to a partial product. It can be noted that Line 5 is the only one actually performing a computation, taking in input only two input words at a time. We assume that the single bit transient fault described in Section II alters one of the two words during an iteration of the operand scanning loop. For the sake of clarity, we will denote the iteration hit by the fault through its pair of loop indexes $(\hat{\imath}, \hat{\jmath})$.

In other words, the device will compute the partial product employing the value $\tilde{a}_{\hat{\jmath}}$ in place of $a_{\hat{\jmath}}$ where the two values satisfy $\tilde{a}_{\hat{\jmath}} = a_{\hat{\jmath}} \oplus 2^{\mu}$ where μ is smaller than the word length w. The difference between the correct and faulty value may also be seen arithmetically as $\tilde{a}_{\hat{\jmath}} = a_{\hat{\jmath}} \pm 2^{\mu}$, in order to elaborate the difference between the correct and faulty result of the multiplication. The error induced in the partial product $(u, v)_{2^w}$ may be quantified as:

$$\begin{aligned} \widetilde{(u, v)}_{2^w} &= c_{\hat{\imath}+\hat{\jmath}} + \tilde{a}_{\hat{\jmath}} b_{\hat{\imath}} + u \\ &= c_{\hat{\imath}+\hat{\jmath}} + (a_{\hat{\jmath}} \pm 2^{\mu}) b_{\hat{\imath}} + u \\ &= c_{\hat{\imath}+\hat{\jmath}} + a_{\hat{\jmath}} b_{\hat{\imath}} \pm 2^{\mu} b_{\hat{\imath}} + u \\ &= (u, v)_{2^w} \pm 2^{\mu} b_{\hat{\imath}} \end{aligned} \tag{2}$$

Propagating the effect of the additive fault in the partial product up to the final result, taking care of assigning the proper weight to the faulty word we obtain:

$$\begin{aligned} \tilde{c} &= c \pm b_{\hat{\imath}} 2^{\mu} 2^{(\hat{\imath}+\hat{\jmath})w} \\ &= c \pm b_{\hat{\imath}} 2^{\mu+(\hat{\imath}+\hat{\jmath})w}. \end{aligned} \tag{3}$$

As represented in the equation, the additive error on the result depends only on a w-bit word $b_{\hat{\imath}}$ of an operand and the position μ where the fault did happen. As stated previously, it can be seen from the previous equation that this error is not disturbed by a modular reduction step following the integer multiplication, since the operations work over the same finite field $(\mathbb{Z}, +, \cdot)$ as the modular multiplication.

We will now consider the multiplications performed by the signature generation algorithm in order to compose the signature. In particular, when the signature is actually composed combining the message and the secret key (Algorithm 1, Line 5) two modular multiplications are performed: namely, the ones yielding rd and $(e + rd)k^{-1}$. We will analyse the information leaked from the fault injection in both of them.

Taking into consideration the first multiplication, assume the value r is the one affected by a fault. For (3), the result of the operation, according to the previous derivations, can thus be expressed as

$$\widetilde{rd} = rd \pm d_{\hat{\imath}} 2^{\mu+(\hat{\imath}+\hat{\jmath})w} \tag{4}$$

Notice that in this case, the additional term depends directly on a word of the secret key d, shifted by a factor dependent on where and when the fault altered the computation. Employing the faulty value of rd in the subsequent signature recombination steps, we obtain a faulty value for s, $\tilde{s}$, satisfying

$$\begin{aligned} \tilde{s} &= (e + \widetilde{rd})k^{-1} \\ &= \left(e + rd \pm d_{\hat{\imath}} 2^{\mu+(\hat{\imath}+\hat{\jmath})w}\right) k^{-1}, \end{aligned}$$

and, through distributing the multiplication by k^{-1},

$$\tilde{s} = s \pm d_{\hat{\imath}} 2^{\mu+(\hat{\imath}+\hat{\jmath})w} k^{-1}. \tag{5}$$

It can be seen that the value of the faulty signature differs from the correct one by a value directly depending on the secret key. When the fault alters the value of a word of d instead of r during the multiplication, thus causing a leakage of a word of r, it is possible for the attacker to check if the recovered word is correct, through comparing it to the words composing r. Moreover, since r changes at each execution, it is unlikely that a word of the secret key d and a word of r match for more than one computation.

We will now consider the second multiplication, i.e. $s = (e + rd)k^{-1}$ as the one affected by the fault, denoting $h = e + rd$ for readability, a fault on k^{-1} yields the result

$$\tilde{s} = s \pm h_{\hat{\imath}} 2^{\mu+(\hat{\imath}+\hat{\jmath})w}. \tag{6}$$

In this case the final result carries information about $h = e + rd$, which is related to the secret key and two other values known to the attacker. Again, if the variable hit by the fault is h in place of k^{-1} the result is different and it will be necessary to set the result apart.

We would like to remark that, if the non-fault affected word of any partial product is equal to zero, the effect of the injected fault will be nullified and there will be no difference in the outputs. However, this case is particularly rare and it's sufficient to repeat the fault injection since the random salt will be avoiding the repetition of such cases.

B. Discrete Logarithm

We will now present the method to recover the value of the words $d_{\hat{\imath}}$ or $h_{\hat{\imath}}$ relying on the faulty signature $\tilde{s}$.

The most simple method which can be used in case the difference from the correct signature depends on $h_{\hat{\imath}}$ (Equation 6), is to try all the possible values of $h_{\hat{\imath}}$ and $(\hat{\imath}, \hat{\jmath}, \mu)$. The correctness of the guess on the value of $h_{\hat{\imath}}$ may be checked verifying the corrected signature $\tilde{s} \pm h_{\hat{\imath}} 2^{\mu+(\hat{\imath}+\hat{\jmath})w}$: if the corrected signature passes the verification stage, the correct guess is detected. However, this procedure is particularly computationally intensive and will not work if the error depends on both $d_{\hat{\imath}}$ and k^{-1} (Equation 5), since the possible hypotheses for the value of k are too many to allow an exhaustive search.

Being the size of k the same as the secret key d, this would imply a computational effort equivalent to find the key through a brute-force attack.

In order to devise a less computationally intensive and universally working recovery strategy, some preliminary considerations must be made. First of all, although the point $[k]G$ calculated in algorithm 1 is not known outside the device, it is possible to make hypotheses on its coordinates since the value $r = x_{[k]G} \bmod n$ is publicly known. It is indeed possible to state the following

Statement 1. *Let r be the x coordinate of an unknown point $P \in \mathbb{E}$, reduced modulo $ord(\mathbb{E}) = n$. Given a value of r, there are only either two or four points belonging to the curve such that $r = x_P \bmod n$*

Proof: The coordinate x_P is an element of the base field $\mathbb{Z}_p$, implying that $0 \leq x_P < p$. We may express the modular reduction r has undergone as $x_P \bmod n = x_P - \lambda n$,for some non-negative integer λ. We will now prove that either $\lambda = 0$ or $\lambda = 1$, that is, for a single reduced value $r = x_P \bmod n$ there are only two possible values of x_P. We recall that the value of the curve order n is bound by Hasse's theorem (1) to be $p+1-2\sqrt{p} \leq n \leq p+1+2\sqrt{p}$. We may thus distinguish three cases, depending on whether n is bigger or smaller than p. The case $n = p$ is not encountered in practice, since elliptic curves with such a property are provably cryptographically weak [9]. If $n > p$ holds, then there is no reduction going on (i.e. $\lambda = 0$) since x_P is smaller than p. If $n < p$ holds, a the value of the x coordinate lies in one of the following intervals: $[1, n-1]$ or $[n, p-1]$. If the coordinate lies in the first interval, the reduction is not performed, i.e. $r = x_P$. In the second case, it must be noted that from Hasse's Theorem, it follows that $n \geq p+1-2\sqrt{p}$. Since $p+1-2\sqrt{p} > p/2$ holds for every prime p, the modulus n will also be greater than $p/2$, consequently $p < 2n$. This in turn implies that x_P is smaller than $2n$, thus the reduction modulo n will at most subtract n (i.e. $\lambda = 1$ at most). It has been shown that the modular reduction of the coordinate $x_{[k]G}$ leads to $r = x_P - \lambda n$ for $\lambda \in \{0, 1\}$ only. It is thus possible to obtain at most two valid x coordinates given a value of r, depending on the fact that the modular reduction did actually subtract something or not. Substituting the two possible values for x_P in the Weierstrass equation (II-A) describing the curve, each of the given x_P may correspond to either 0 or 2 points, depending on whether or not the value $x^3+a_4x+a_6$ is a quadratic residue modulus p. Since r is correctly derived from an actual scalar multiplication P performed by the algorithm, either r or $r + n$ must yield a quadratic residue, and two points $P = (x_P, y_P)$ and $-P = (x_P, -y_P)$ will be bound to the value. The number of points which can be bound to a generic x coordinate reduced modulo n is thus 2 or 4. ∎

This result implies that it is possible for an attacker to derive a very small number of hypotheses on the value of the point $P = [k]G$ obtained as a result of the point scalar multiplication of the ECDSA signature algorithm, since the value of the reduced x coordinate is sent along in the signature token.

We will now elaborate the faulty signature equations obtained from the previous sections in order to expose how it is possible to recover words of the secret key d from the leakage. Starting from the case where the leaked word directly depends on d, i.e. the fault hit the value r during the rd multiplication, we can obtain:

$$\begin{aligned}\tilde{s} &= s \pm d_i 2^{\mu+(\hat{\imath}+\hat{\jmath})w} k^{-1} \\ &= \frac{e+rd}{k} \pm \frac{d_i 2^{\mu+(\hat{\imath}+\hat{\jmath})w}}{k}\end{aligned}$$

Solving the equation for k yields:

$$k = \frac{e+rd}{\tilde{s}} \pm \frac{d_i 2^{\mu+(\hat{\imath}+\hat{\jmath})w}}{\tilde{s}}$$

Using all the scalar values in the equations as scalar coefficients in a point scalar multiplication, employing as a base point the curve generator G we obtain:

$$[k]G = \left[\frac{e}{\tilde{s}}\right] G + \left[\frac{r}{\tilde{s}}\right] [d]\, G \pm \left[\frac{d_i 2^{\mu+(\hat{\imath}+\hat{\jmath})w}}{\tilde{s}}\right] G$$

Finally, by substituting the value of the public key $Y = [d]G$ in the former equation yields:

$$[k]G = \left[\frac{e}{\tilde{s}}\right] G + \left[\frac{r}{\tilde{s}}\right] Y \pm \left[\frac{d_i 2^{\mu+(\hat{\imath}+\hat{\jmath})w}}{\tilde{s}}\right] G \qquad (7)$$

Observing this last result, we notice that every value, except for the last summand and the value of the scalar k are fully known. However, through employing the result proven before in Statement 1, it is possible to obtain at most 4 values of the actual $[k]G$ employing the publicly known r value, thus we will substitute the point $[k]G$ with all the possible valid ones R. Proceeding now to solve the equation in the only unknown left, we obtain:

$$\pm \left[\frac{d_i 2^{\mu+(\hat{\imath}+\hat{\jmath})w}}{\tilde{s}}\right] G = R - \left[\frac{e}{\tilde{s}}\right] G - \left[\frac{r}{\tilde{s}}\right] Y \qquad (8)$$

Analogously, it is possible to derive from the effects of a fault on the multiplication combining hk^{-1}, where $h = (e + rd)$:

$$\tilde{s} = \frac{e+rd}{k} \pm \frac{k h_i 2^{\mu+(\hat{\imath}+\hat{\jmath})w}}{k} \qquad (9)$$

thus leading to:

$$\pm \left[\frac{h_i 2^{\mu+(\hat{\imath}+\hat{\jmath})w}}{\tilde{s}}\right] R = R - \left[\frac{e}{\tilde{s}}\right] G - \left[\frac{r}{\tilde{s}}\right] Y \qquad (10)$$

It is possible to write a unified form for both equations, through indicating with ρ the leaked word, be it either d_i or h_i, and with P_1 and P_2 two points of the curve computable from known values. Through indicating as $\theta = \mu+(\hat{\imath}+\hat{\jmath})w$ the position of the faulty bit, where μ indicates the fault position within a word of the computing architecture, while w indicates the word length.

$$[\rho] \left(\left[2^{\theta}\right] P_1 \right) = \pm P_2 \qquad (11)$$

This equation shows that the value of the leaked word ρ is obtainable through solving a reduced complexity ECDLP. In particular, it is important to notice that the value of ρ is confined within the bounds of the values representable in a single word of the computer architecture computing the ECDSA algorithm. This implies that, contrarily to the value of the hidden random number k, which is at most as large as the curve order (192-521 bits for NIST standard curves [1]), the value of w is bound to be 8 to 32 bits long in most of the cases. Due to the limited range of values of ρ it becomes practically

feasible to solve the ECDLP problem in the previous equations using one of the well known algorithms. We chose to employ the *baby-step giant-step* algorithm [5] for our simulations since it can be efficiently adapted to the special case of computing many logarithms employing the same base point. Since the two known points depend on the guess R made before, in order to retrieve a leaked word it is necessary to compute either two or four discrete logarithms for each retrieved value of ρ, if the position of the fault θ is known (as in the case of clock glitch based fault injections). In case it is not possible to know a-priori the position of the fault it is still possible to compute a logarithm for each possible guess for θ. As shown in the experimental evaluation section, this is still within the computational power available with a common desktop PC.

C. Fault Position Recognition

We will now analyse the result of the logarithm computation to understand how to use it for the subsequent reconstruction procedure. To this end, we need to discern the position when the faulty bit in the computation was inserted.

Basically, a logarithm computation cannot distinguish among $(2^z\rho)2^\theta$ and $\rho 2^{\theta+z}$. We rewrite ρ as $\rho = \rho' 2^\tau$ where ρ' is taken to be odd and $\tau \geq 0$ is called a *shift coefficient*. Being ρ' uniquely determinable by the logarithm computation it is possible to write the following

$$\rho 2^\theta = \rho' 2^{\tau+\mu+(\hat{\imath}+\hat{\jmath})w}$$

The problem of knowing the correct ρ is therefore reduced to the problem of guessing the correct τ.

It can be noted that $\rho < 2^w$ implies $\tau \leq w - \lceil \log_2 \rho' \rceil$. In other words, said n_h and n_l the amount, respectively, of the leading and trailing 0's in the binary expression of ρ, the amount of admissible shift coefficients is given by $1+n_h+n_l$. Finally, we note that for any given hypothesis for τ the value $\mu = \theta - \tau \bmod w$ and the sum $(\hat{\imath} + \hat{\jmath}) = (\theta - \mu - \tau)/w$ are uniquely determined. The latter will be needed by the key reconstruction procedure.

As special case, let us note that if $\rho \geq 2^{w-1}$ and $\rho \bmod 2 = 1$ hold at the same time there is no uncertainty and it is possible to state that $\tau = 0$. Indeed, if the retrieved words are uniformly distributed in $[1, 2^w - 1]$, this case is not so rare, having a probability approximately $1/4$. We can furthermore give an estimation of the expected uncertainty about the alignment of the word ρ that will have to be considered by the attacker.

Considering ρ as a discrete random variable following a uniform distribution $\mathcal{U}[1, 2^w - 1]$, we can model the probability of every bit of ρ being equal to zero as a Bernoulli distributed random variable with parameter $1/2$. The amount of leading and trailing 0's of the word may therefore be approximated with a geometric distribution with parameter 0.5, thus resulting in a single leading and a single trailing zero on average. The expected value of the total amount of admissible shift coefficients τ for a word is thus 3, i.e. $0 \leq \tau < 3$.

D. Key Reconstruction Techniques

We will now present how to use the values retrieved by applications of the logarithm search explained in section III-B to a number of faulty signatures in order to reconstruct the whole secret key d. At this point the attack procedure splits into two different attack procedures, depending on whether the fault hit the first or the second multiplication during the actual composition of the message and the secret key during step 5 of Algorithm 1.

In case the fault injection technique does not allow to know into which multiplication the fault has been injected (i.e. the fault injection is not timed), the faults may hit any of the 4 different operands involved in the multiplications, i.e. the key d, the mask r, the inverse of the nonce k^{-1} and the combined data $h = e+rd$. It is possible to identify and discard the values belonging to the words of r, since this value is known, through checking the retrieved word against the words composing it. Since the secret key d is the only value of the four which is not changing throughout different runs of the algorithm, it is therefore possible to identify the words of the key as they are, with high probability, the only ones to appear more than once if a uniform distribution of the faults is considered. On the other hand, the remaining words, belonging to either k^{-1} or h will need a more complex procedure in order to allow the attacker to reconstruct the secret key. Namely, it will be necessary to repeat the reconstruction procedures making guesses for each of those words.

Willing to tackle the issue of reconstructing d in case the retrieved words belong directly to it, it is needed to understand the correct order in which those must be put back together. In order to do so, the attacker needs now to recover the loop iteration $\hat{\imath}$ which represents the position of the retrieved word $\rho = d_{\hat{\imath}}$ within the original value. As previously mentioned, it is possible to obtain the sum of the loop and word index where the fault happened $\alpha = (\hat{\imath} + \hat{\jmath})$. Such value belongs to $[0, 2\sigma - 1]$. Since only the sum of the two values is known to the attacker, one way is to retrieve words leaking from each of the $2\sigma - 1$ distinct positions. Consider in the two cases for which $\alpha = 0$ and $\alpha = 2\sigma - 2$ there is no ambiguity on the value of $\hat{\imath}$ and it is thus possible to recognize the words d_0 and $d_{\sigma-1}$. This in turn allows to disambiguate if a discovered word for $\alpha = \hat{\imath} + \hat{\jmath} = 1$ is the result of an error falling in $\hat{\imath} = 0, \hat{\jmath} = 1$ or in $\hat{\imath} = 1, \hat{\jmath} = 0$. It is possible to iterate the disambiguation procedure, employing the newly discovered values, until all the words of the secret exponent are correctly placed in their location. In case fewer than the necessary faults are available, it is possible to check all the possible permutations of the still ambiguous words through computing the public key corresponding to the alleged value of d and verifying the equality with the known public key.

The second key reconstruction procedure assumes that the collected words come from the second multiplication of the signature computation instead, i.e. $(e+rd)k^{-1}$. We will restrict the analysis to the case $\hat{\imath} = \alpha = 0$ for the sake of clarity, though it is possible to extend the analysis also to the other values of α. Since the word which is retrieved from one faulty signature is h_0 where $h = e + rd$, it is possible to write $e + rd \bmod n = \xi 2^w + h_0$, where ξ represents the unknown part of h. While a single equation of the previous form does not allow the recovery of d, employing a collection of faults to write different equations of the same form allows to formulate the key retrieval problem as a Hidden Number Problem and solve the system eventually finding d. The relationship between the DSA family of protocols and the HNP has been deeply analysed in [10], [11].

To address the word alignment issue described in section III-C two main ways to proceed are possible. The actual choice of an attacker should again be the combination of the two that

TABLE I
NUMBER OF FAULTS AND COMPUTATION TIME REQUIRED TO RETRIEVE THE WHOLE SECRET KEY

Architecture Word Size	DLP Precomputation Time [s] P-192	P-521	DLP Time [s] P-192	P-521	Number of Faults P-192	P-521	HNP Time [s] P-192	P-521	Total Time [s] P-192	P-521
8	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$1.2 \cdot 10^{-3}$	28	115*	0.80	180	2.2	250
16	$< 10^{-3}$	0.030	0.036	0.017	13	37	0.34	6.7	83	330
32	1.8	7.7	1.3	5.9	7	17	0.092	0.97	$1.5 \cdot 10^3$	$51 \cdot 10^3$
64	$120 \cdot 10^3$	$500 \cdot 10^3$	$107 \cdot 10^3$	$446 \cdot 10^3$	4	10	0.05	0.45	$56 \cdot 10^6$	$220 \cdot 10^6$

leads to the best performance, considering the cost of the word retrieval. The first way is to try the lattice attack more than once, using every possible alignment of the words. A different approach is to discard some retrieved ρ's, keeping only the certain ones, i.e. the words of the type $(1 \cdots 1)_2$. We know that r and thus $e + rd$ change at every execution, and the retrieved word h_i can be considered to be evenly distributed in $[1, 2^w - 1]$, so that the discussed type of word appears with probability $\approx 1/4$.

IV. FEASIBILITY ANALYSIS

In this section we will evaluate the computational requirements and the number of faults needed in order to recover fully an ECDSA signature key depending on the different levels of security and target architecture word size. The required computations were performed employing a C implementation of the baby-step giant-step algorithm and Sage [12], on a Gentoo Linux x86_64 running on an Intel Core i7 920 endowed with 12 GB of DDR3 RAM. All the timings are taken on a single core, fully serial implementation of both the DLP and HNP solvers. Table I reports the running times required to fully retrieve the whole secret key depending on the word size of the targeted device architecture and the level of security chosen. The table reports the upper and lower bound curves indicated by NIST as the standard ones to be employed. As it can be seen from the table, the required number of faults is below 40, except for the 8-bit case, since every fault leaks significantly less informative content. The table shows the number of required faults to solve the HNP with probability ≈ 1. The only noticeable exception happens in the unlikely case a P-521 curve is used on 8-bit architectures, slightly more than 115 faults are required to provide a 50% chance of recovering the key. However, it is practically viable to collect thousands of faults if fault injection techniques such as voltage brown-outs [13] or clock glitches without damaging the target device. The required computation times for a whole attack, taking into account the need to repeat multiple discrete logarithm computations are reported in the last column of table. As reported, it is practical to recover the secret key of the ECDSA algorithm within minutes for 8- and 16-bit architectures, while the recovery takes a couple of hours for 32-bit ones. For the sake of completeness, also timings for 64-bit architectures are reported, although they are not yet present in the embedded systems environment. In this case, the time requirements scale up to a few years. However, all the mentioned attacks may be fully parallelized (since there are no dependencies among the solutions of the different DLPs), achieving easily an order of magnitude of speedup through the use of general purpose parallel co-processors such as the recently available OpenCL compliant video cards. Such an improvement would bring down the breaking time to only a few months, thus allowing a single attacker to break ECDSA even when the highest security grade curves are employed on a 64-bit platform.

V. CONCLUSIONS

In this paper a novel attack against the Elliptic Curve DSA has been presented. This attack targets the signature recombination part of the algorithm, a point that was not yet analysed by any known fault attack on ECDSA. The proposed attack is able to fully recover the secret signing key through the injection of single bit transient faults, which can be injected with low cost equipment and without damaging the device under attack [14], [15]. The discovery of the key succeeds because the fault injection allows an attacker to reduce the ECDLP at the basis of ECDSA to a reduced size version which is computationally solvable in short time. The post processing times are compatible with common consumer grade desktop PCs for all the curves standardised by NIST in FIPS-186, and in general it does not depend on any particular feature of the chosen curves. It may also be possible to extend our attack technique to the regular DSA, since it shares the signature recombination methodology with ECDSA.

REFERENCES

[1] *FIPS-186-3: Digital Signature Standard (DSS)*, National Institute of Standards and Technology (NIST) Std., 2009.

[2] J. Blömer, M. Otto, and J.-P. Seifert, "Sign change fault attacks on elliptic curve cryptosystems." in *Workshop on Fault Diagnosis and Tolerance in Cryptography*. Prentice Hall, 2006, pp. 36–52.

[3] M. Kara-Ivanov, E. Iceland, and A. Kipnis, "Attacks on authentication and signature schemes involving corruption of public key (modulus)," in *Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE Computer Society, 2008, pp. 108–115.

[4] J.-M. Schmidt and M. Medwed, "A fault attack on ecdsa," in *6th Workshop on Fault Diagnosis and Tolerance in Cryptography - FDTC 2009, Proceedings*, D. Naccache and E. Oswald, Eds. Verlag IEEE-CS Press, 2009, pp. 93 – 99.

[5] J. H. Silverman, *The arithmetic of elliptic curves*, ser. Graduate Texts in Mathematics. New York: Springer-Verlag, 1992, vol. 106, corrected reprint of the 1986 original.

[6] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot, *Handbook of Applied Cryptography*, 1st ed. Boca Raton, FL, USA: CRC Press, Inc., 1996.

[7] A. Barenghi, G. Bertoni, E. Parrinello, and G. Pelosi, "Low voltage fault attacks on the RSA cryptosystem," in *Fault Diagnosis and Tolerance in Cryptography*. IEEE Computer Society, 2009, pp. 23–31.

[8] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370–382, Feb. 2006.

[9] R. Avanzi, H. Cohen, and G. Frey, *Handbook of elliptic and hyperelliptic curve cryptography*. Chapman & Hall/CRC, 2006.

[10] P. Q. Nguyen and I. E. Shparlinski, "The insecurity of the elliptic curve digital signature algorithm with partially known nonces," *Des. Codes Cryptography*, vol. 30, no. 2, pp. 201–217, 2003.

[11] N. A. Howgrave-Graham and N. P. Smart, "Lattice attacks on digital signature schemes," *Des. Codes Cryptography*, vol. 23, no. 3, pp. 283–290, 2001.

[12] W. Stein *et al.*, *Sage Mathematics Software (Version 4.6.1)*, The Sage Development Team, 2011, `http://www.sagemath.org`.

[13] A. Barenghi, G. M. Bertoni, L. Breveglieri, M. Pellicioli, and G. Pelosi, "Low Voltage Fault Attacks to AES," in *HOST*, 2010.

[14] F. Amiel, C. Clavier, and M. Tunstall, "Fault Analysis of DPA-Resistant Algorithms," in *FDTC*, 2006, pp. 223–236.

[15] J.-M. Schmidt and M. Hutter, "Optical and EM Fault-Attacks on CRT-based RSA: Concrete Results," in *Austrochip 2007*, J. W. Karl C. Posch, Ed. Verlag der Technischen Universität Graz, 2007, pp. 61 – 67.