GLV/GLS Decomposition, Power Analysis, and Attacks on ECDSA Signatures With Single-Bit Nonce Bias

Diego F. Aranha[1], Pierre-Alain Fouque[2], Benoît Gérard[3,4], Jean-Gabriel Kammerer[3,5], Mehdi Tibouchi[6], and Jean-Christophe Zapalowicz[7]

[1] Institute of Computing, University of Campinas, `dfaranha@ic.unicamp.br`
[2] Université de Rennes 1 and Institut Universitaire de France, `fouque@irisa.fr`
[3] DGA–MI, Rennes
[4] IRISA, `benoit.gerard@irisa.fr`
[5] IRMAR, Université de Rennes 1, `jean-gabriel.kammerer@m4x.org`
[6] NTT Secure Platform Laboratories, `tibouchi.mehdi@lab.ntt.co.jp`
[7] Inria, `jean-christophe.zapalowicz@inria.fr`

Abstract. The fastest implementations of elliptic curve cryptography in recent years have been achieved on curves endowed with nontrivial efficient endomorphisms, using techniques due to Gallant–Lambert–Vanstone (GLV) and Galbraith–Lin–Scott (GLS). In such implementations, a scalar multiplication $[k]P$ is computed as a double multiplication $[k_1]P + [k_2]\psi(P)$, for ψ an efficient endomorphism and k_1, k_2 appropriate half-size scalars. To compute a random scalar multiplication, one can either select the scalars k_1, k_2 at random, hoping that the resulting $k = k_1 + k_2\lambda$ is close to uniform, or pick a uniform k instead and decompose it as $k_1 + k_2\lambda$ afterwards. The main goal of this paper is to discuss security issues that may arise using either approach.

When k_1 and k_2 are chosen uniformly at random in $[0, \sqrt{n})$, $n = \mathrm{ord}(P)$, we provide a security proofs under mild assumptions. However, if they are chosen as random integers of $\lfloor \frac{1}{2}\log_2 n \rfloor$ bits, the resulting k is slightly skewed, and hence not suitable for use in schemes like ECDSA. Indeed, for GLS curves, we show that this results in a bias of up to 1 bit on a suitable multiple of $k \bmod n$, and that this bias is practically exploitable: while lattice-based attacks cannot exploit a single bit of bias, we demonstrate that an earlier attack strategy by Bleichenbacher makes it possible. In doing so, we set a record by carrying out the *first ECDSA full key recovery using a single bit of bias.*

On the other hand, computing k_1 and k_2 by decomposing a uniformly random $k \in [0, n)$ avoids any statistical bias, but the decomposition algorithm may leak side-channel information. Early proposed algorithms relied on lattice reduction and exhibited a significant amount of timing channel leakage. More recently, constant-time approaches have also been proposed, but we show that they are amenable to power analysis: we describe a template attack that can be combined with classical lattice-based attacks on ECDSA to achieve full key recovery on physiscal devices.

Keywords: Elliptic Curve Cryptography, GLV/GLS Method, Bleichenbacher's ECDSA Attacks, Side-Channel Analysis

1 Introduction

The GLV/GLS techniques. Many record implementations of elliptic curve cryptography in software, including, most recently, works such as [25,5,9], rely on elliptic curves endowed with fast endomorphisms, as constructed by the methods due to Gallant–Lambert–Vanstone (GLV) [13], Galbraith–Lin–Scott (GLS) [12], and generalizations thereof. In such implementations, the fast endomorphism ψ on the elliptic curve $E/\mathbb{F}_q$ is used to speed up full size scalar multiplications $[k]P$ by computing them as multi-exponentiation $[k_1]P + [k_2]\psi(P)$, where k_1 and k_2 are roughly half of the size of k. Indeed, on a prime order subgroup of $E(\mathbb{F}_q)$, ψ acts by multiplication by some constant λ, and thus, for a generator P of that subgroup, we have $[k_1]P + [k_2]\psi(P) = [k_1 + k_2\lambda]P$.

In order to compute random scalar multiplications with those techniques, two types of approaches have been considered, as far back as in the earliest presentations of the GLV method (such as Gallant's talk at ECC'99 [**?**]).

On the one hand, k_1 and k_2 can simply be chosen uniformly at random in a suitable half-length interval. This approach, which we call the *recomposition technique* (since k is "recomposed" as $k = k_1 + k_2\lambda$), results in a very simple implementation, and has been used in several implementation records including [25], but Gallant expressed concerns about possible biases in the resulting scalar k. Such concerns have been partially vindicated by some numerical evidence provided by Brumley and Nyberg [7], who also described a relatively general way to choose intervals for k_1 and k_2 so that the resulting choice of k is in fact secure (in the sense that it has high entropy). However, the Brumley–Nyberg method is a bit cumbersome, and no attack so far has been demonstrated against arbitrary half-length uniform choices of k_1 and k_2, so that the security picture is somewhat unclear.

On the other hand, one can also pick k at random and subsequently deduce half-length values k_1 and k_2, which eliminates concerns regarding possible biases in the distribution of k. This *decomposition technique* usually relies on lattice reduction in dimension 2 (or equivalently, continued fractions, a generalized Euclidean algorithm, etc.), as originally described in the GLV paper [13], and is significantly more computationally demanding than recomposition. Simplifications of this method have later been proposed (particularly in [26]), as well as higher-dimensional generalizations [23] to tackle decompositions involving several endomorphisms (as recently used in [29,14] for instance).

ECDSA attacks. The success of GLV/GLS method in implementations lately makes it desirable to reconsider these decomposition and recomposition techniques from a security viewpoint. We do so in this paper in the context of ECDSA signatures, one of the most widely deployed elliptic curve cryptographic schemes, and an interesting target for the cryptanalyst (like other Schnorr-like signature schemes) due to its sensitivity to biases in the distribution of nonce values, as demonstrated by the powerful attack due to Howgrave-Graham and Smart [15] based on lattice reduction techniques, which breaks (EC)DSA when some of the most significant bits of the nonces are known. This attack was analyzed in further details by Nguyen and Shparlinski [21,22] and carried out in practice in many contexts, including against physical devices (see e.g. [20,6] for some examples). The basic idea is to express the key recovery problem as an instance of the Hidden Number problem (HNP), which reduces to the closest vector problem (CVP) in a suitable lattice. Since CVP is tractable in low-dimensional lattices, many practical instances of ECDSA can be broken depending on key size and the number of leaked nonce bits. The largest problem instance broken so far is the case of 2-bit nonce leaks on 160-bit curves, tackled by Liu and Nguyen [17] using the most advanced known techniques for lattice reduction (BKZ 2.0 [8]). Breaking 2-bit leaks on 256-bit curves, or 4-bit leaks on 384-bit curves seems currently out of reach (see the discussions in [8,19]).

In any case, there is a hard limit to what can be achieved using lattice reduction: due to the underlying structure of the HNP lattice, it is impossible to attack (EC)DSA using a single-bit nonce leak with lattice reduction. In that case, the “hidden lattice point” corresponding to the HNP solution will not be the closest vector even under the Gaussian heuristic (see [24]), so that lattice techniques cannot work. To break this “lattice barrier”, the only known alternate attack is an algorithm due to Bleichenbacher [3] which predates the attack of Howgrave-Graham and Smart, but was generally considered of mostly theoretical interest until it was recently revisited by De Mulder *et al.* [19] to attack 384-bit curves. Bleichenbacher devised his attack to demonstrate a vulnerability in DSS at the time, in which DSA nonces were generated by picking a random value of ℓ_n bits, where ℓ_n is the bit length of the group order n, and then to reduce it modulo n. Bleichenbacher showed that the resulting bias could be exploited in a very interesting way, obtaining a key recovery using about 2^{41} signatures and about 2^{47} time and 2^{41} memory complexities. At that time, it was not possible to mount this attack and only simulations on reduced numbers were possible and the paper was never published.

In the first stage, Bleichenbacher’s algorithm reduces the signatures from 160 bits to say 40 bits using linear combinations of the original signatures and then, during a second phase, a Discrete Fourier Transform is used to recover the most significant bits of the secret key. The bias of the reduced signatures is higher than the bias of the original signatures, that’s the reason why Fourier technique is needed to extract this information. This algorithm is very similar to Blum, Kalai and Wasserman algorithms [4,16] for solving LPN and LWE problems. For 384-bit order, the first stage of Bleichenbacher original attack is not sufficiently efficient to reduce the signatures and more advanced techniques based on LLL and BKZ are needed if the number of leaked bits is high enough [19]. The modification of the first stage is not possible if less than one bit of nonces is available and we turn back to Bleichenbacher’s original attack which requires a high number of signatures.

Our contributions. Our first contribution is the first implementation of Bleichenbacher’s attack against ECDSA with a single-bit on nonce bias. We carry out this attack on the standardized SECG P160 R1 elliptic curve. On this 160-bit curve, we use 2^{33} ECDSA signatures, and achieve a full key recovery in a few hours of wall-clock time on a 64-core workstation. The most time-consuming part of the attack is the first phase, in which a sorting algorithm is executed several times. This is the first key recovery from a single bit of bias, which paves the way to new applications. We stress again that this record cannot be achieved using lattice reduction techniques based on HNP problem, since even if the HNP lattice satisfies the Gaussian heuristic, a condition for finding the hidden lattice point is that the number of known bits of the nonce must be greater than $\log_2(\sqrt{\pi e/2}) \approx 1.0471$ (hence at least 2) [24], irrespective of the underlying lattice reduction algorithm.

As a second contribution, we show a security proof for the recomposition method on curves obtained by the quadratic GLS method once the values k_1 and k_2 are uniformly distributed in the interval $[0, \sqrt{n})$, where n is the prime group order. We prove that the statistical distance between this distribution and the uniform distribution in $[0, n)$ is negligible. Furthermore, if k_1 and k_2 are taken at random in a small interval of the form $[0, 2^m)$, where $m = \lfloor \frac{1}{2} \log_2 n \rfloor$, the bias on the distribution on k used in Bleichenbacher's attack is negligible. However, we show that the bias of the distribution on tk where t is the trace is sufficiently large and a Bleichenbacher's attack allows to recover the secret key. We also implement this attack and the complexities are similar to the previous part.

Finally, we study the decomposition technique proposed in GLV with the implementation described by Park *et al.* in [26]. To this end, we propose a very efficient side-channel attack that uses the leakage on the multiplication in order to recover some of the least significant bits of the nonces. Consequently, we can thus use lattice techniques to recover the secret key.

2 Preliminaries

2.1 Bias definition and properties

The measurement of the bias of random variables represents a significant part of our analyses. We thus recall the definition of the bias which was proposed by Bleichenbacher in [3].

Definition 1. *Let X be a random variable over $\mathbb{Z}/n\mathbb{Z}$. The bias $B_n(X)$ is defined as*

$$B_n(X) = E(e^{2\pi i X/n}) = B_n(X \bmod n),$$

where $E(X)$ represents the mean.
Similarly, the sampled bias of a set of points $V = (v_1, \cdots, v_L)$ in $\mathbb{Z}/n\mathbb{Z}$ is defined by

$$B_n(V) = \frac{1}{L} \sum_{j=0}^{L-1} e^{2\pi i v_j/n}.$$

The bias as defined above presents some useful properties we recall in Lemma 1.

Lemma 1. *Let $0 < T \leq n$ be a bound and X, Y random variables uniformly distributed on the interval $[0, T-1]$.*

(a) If X is uniformly distributed on the interval $[0, n-1]$, then $B_n(X) = 0$.
(b) If X and Y are independent, then $B_n(X+Y) = B_n(X)B_n(Y)$.
(c) $B_n(-X) = \overline{B_n(X)}$ where $\overline{a}$ denotes the conjugate of a.
(d) $B_n(X) = \frac{1}{T}\left|\frac{\sin(\pi T/n)}{\sin(\pi/n)}\right|$ and $B_n(X)$ is real-valued with $0 \leq B_n(X) \leq 1$.
(e) Let a be an integer with $|a|T \leq n$ and $Y = aX$, then $B_n(Y) = \frac{1}{T}\frac{\sin(\pi a T/n)}{\sin(\pi a/n)}$

2.2 ECDSA signature generation

ECDSA is a NIST standard and we describe the signature generation in Algorithm 1.

Algorithm 1 ECDSA signature. P is a base point of order n and $H : \{0,1\}^* \to [0, n-1]$ is a cryptographic hash function. The private key is an element $x \in \mathbb{Z}/n\mathbb{Z}$ and the public key is denoted by (p, n, H, P, Q) with $Q = [x]P$.

```
1: function Sign_ECDSA($m$)
2:     $k \xleftarrow{\$} [0, n-1]$
3:     $(u, v) \leftarrow [k]P$
4:     $r \leftarrow u \bmod n$; if $r = 0$ then goto step 2;
5:     $s \leftarrow k^{-1}(H(m) + rx) \bmod n$; if $s = 0$ then goto step 2;
6:     return $(r, s)$
7: end function
```

3 Bleichenbacher's Attack on single bit bias

In this part, we present our results on an ECDSA signature generation scheme where the nonce k is 1-bit biased. We demonstrate that an attack proposed some years ago by Bleichenbacher can succeed in retrieving the secret key in about 2^{37} time and 2^{33} memory complexities given 2^{33} signatures, for 160-bit order. This attack was initially focusing on the DSA signature generation scheme but can be applied without any modification to ECDSA we consider in this paper.

The main idea consists in using the fact that the nonces k_j are chosen from a biased random variable $\mathbf{K}$, *i.e.* k are not randomly and uniformly generated on $[0, n-1]$. Because the values k_j are biased and linked with the secret key x by the equations which are used for the signature computations, these signatures, correctly manipulated, also present a bias which will only be significant for the correct value of x. In other words the bias plays the role of the distinguisher in this attack.

Obviously, for cryptographic sizes, evaluating the bias for all values in $[0, n-1]$ is impractical. However, Bleichenbacher observed that it is possible to "broaden the peak" of the bias in such a way that, with a value close the correct value of x, the bias will remain significant. Thus the bias computations can be performed on a more sparse set of candidates thanks to the Fast Fourier Transform. In return, it requires a non-negligible work on the signatures which reduces the bias, and the attack returns an approximation of the secret key, *i.e.* its most significant bits. The attack can be iterated to retrieve more bits of the secrets and as soon as sufficiently many bits of x are known, Pollard's lambda method [27] can be used to derive the remaining bits. Algorithm 2 presents the main steps of the attack.

Algorithm 2 Bleichenbacher's attack given S ECDSA signatures. The parameters S, ℓ and ι have to be chosen accordingly to the bias.

```
Require: $S$ biased ECDSA signatures $(r_j, s_j)$ computed using a single secret key $x$.
Ensure: The $\ell$ most significant bits of $x$.
 1: Preprocessing
 2: for $j = 0$ to $S - 1$ do
 3:     $h_j \leftarrow H(m_j) \cdot s^{-1} \bmod n$
 4:     $c_j \leftarrow r_j \cdot s_j^{-1} \bmod n$
 5: end for
 6: Reduction of the $c_j$ values (Sort-and-Difference Algorithm)
 7: $A \leftarrow [(c_j, h_j)]_{0 \leq j \leq S-1}$
 8: for $i = 1$ to $\iota$ do
 9:     Sort $A$ by the $c_j$ values                          ▷ $c_j \leq c_{j+1}$
10:     for $j = 0$ to $S - \iota$ do
11:         $A[j] \leftarrow A[j+1] - A[j]$         ▷ $A[j] = (c_{j+1} - c_j, h_{j+1} - h_j)$
12:     end for
13: end for
14: Only keep the pairs $(c_j, h_j)$ such that $c_j < 2^\ell$
15: Denote by $L$ the number of such pairs
16: Bias computation using the inverse FFT
17: $Z \leftarrow (0, \cdots, 0)$ a vector of size $2^\ell$
18: for $j = 0$ to $L - 1$ do
19:     $Z_{c_j} \leftarrow Z_{c_j} + e^{2\pi i h_j / n}$
20: end for
21: $W \leftarrow \mathrm{iFFT}(Z)$   ▷ Inverse FFT computation. The output is also a vector of
    complex numbers.
22: Find the value $m$ such that $|Z_m|$ is maximal
23: return $msb_\ell(mn/2^\ell)$
```

3.1 Attack analysis

We first explain why the bias can serve as a distinguisher and while doing so explain the goal of the preprocessing phase, as it was done in [19], for the sake of completeness. For that purpose, consider S ECDSA signatures (r_j, s_j) with biased nonces k_j. We have the following relation due to step 5 of Algorithm 1:

$$k_j = H(m_j)s_j^{-1} + r_j s_j^{-1} x \bmod n \quad \text{for} \quad 0 \leq j \leq S-1.$$

Now let $h_j = H(m_j)s_j^{-1} \bmod n$ and $c_j = r_j s_j^{-1} \bmod n$. Then the set $\{h_j + c_j x\}_{j=0}^{S-1} = \{k_j\}_{j=0}^{S-1}$ will show a significant nonzero sampled bias. Moreover, for any $w \neq x$, the sampled bias from $V_w = \{h_j + c_j w\}_{j=0}^{S-1}$ will be relatively small. Since h_j and c_j are publicly computable, we thus have a way to determine the correct value of x by testing all the value $w \in [0, n-1]$.

To have a practical test, we have to broaden the peak of the bias such that values of w close to the correct value x will also show a significant bias. The peak will be broad if the c_j are relatively small. More precisely, by denoting 2^ℓ a bound such that $0 \leq c_j < 2^\ell$, then we can find an approximation of x by evaluating the sampled bias of 2^ℓ evenly-spaced values of w between 0 and $n-1$.

The reduction of the c_j, second phase in Algorithm 2, can be done using a sort-and-difference algorithm. From S pairs (c_j, h_j), we first sort them according to their first element. Then we subtract each c_j from the next largest one and we take the differences of the corresponding h_j as well. We thus obtain a list of $S-1$ pairs (c'_j, h'_j) whose values c'_j are on average $\log(S)$ bits smaller. More details about the analysis of this reduction are given later. This reduction algorithm can be repeated in order to achieve the bound 2^ℓ: once the MSB of x are known, one can rewrite the system and attack the next top bits, by integrating the learnt MSB into the c_j as was done in [19].

Now let $w_m = mn/2^\ell$, with $m \in [0, 2^\ell - 1]$, be 2^ℓ evenly-spaced values between 0 and $n-1$. For sake of clarity, we keep the notation (c_j, h_j) for the reduced pairs with $c_j < 2^\ell$ and we consider having L such pairs. Then

$$\begin{aligned} B_n(V_{w_m}) =& \frac{1}{L}\sum_{j=0}^{L-1} e^{2\pi i (h_j + c_j mn/2^\ell)/n} = \sum_{t=0}^{2^\ell-1}\left(\frac{1}{L}\sum_{\{j|c_j=t\}} e^{2\pi i h_j/n}\right) e^{2\pi i t m/2^\ell} \\ =& \sum_{t=0}^{2^\ell-1} Z_t e^{2\pi i t m/2^\ell} \end{aligned}$$

with $Z_t = \frac{1}{L}\sum_{\{j|c_j=t\}} e^{2\pi i h_j/n}$. $B_n(V_{w_m})$ can be viewed as the inverse Fast Fourier Transform of the vector $Z = (Z_0, \cdots, Z_{2^\ell-1})$. Thus the multiple bias computations can be performed very efficiently using the FFT. From Step 17 to 20 in Algorithm 2, we compute this vector Z. Step 21 outputs a vector of the sampled bias for the 2^ℓ candidates, *i.e.* $\text{iFFT}(Z) = (B_n(V_{w_0}), B_n(V_{w_1}), \cdots, B_n(V_{w_{2^\ell-1}}))$. Finally, the value of $w_m = mn/2^\ell$ with the largest sampled bias should share its ℓ most significant bits with the secret key x.

Choosing the parameters. We first give some properties which will help to define the parameters for the attack. We can estimate the sampled bias for a wrong candidate w_m, *i.e.* a value w_m which do not share some most significant bits with the secret key x. More precisely, it can be shown that for w_m either significantly larger or significantly smaller than x, we have $B_n(V_{w_m}) \approx \frac{1}{\sqrt{L}}$, which corresponds to the average distance from the origin for a random walk on the complex plane.

The second property concerns the c_j reduction phase and gives a relation between the number of signatures S and the number of reduced pairs L.

Proposition 1. *Consider S ECDSA signatures of the form (c_j, h_j) and $\gamma \in \mathbb{Z}$. The percentage of signatures (c'_j, h'_j) after the first application of the sort-and-difference algorithm such that $c'_j < 2^{\log q - \log S + \gamma}$ can be approximated by $1 - e^{-2^\gamma}$.*

Lemma 2. *Let $X_1, \ldots, X_N$ be N independent uniformly distributed random variables over $[0,1]$, and for all i, denote by $X_{(i)}$ the i-th order statistic of the X_j's (namely, $X_{(i)}$ is the i-th smallest among the X_j's). Then, the random variables $Y_i = X_{(i+1)} - X_{(i)}$ for $i = 1, \ldots, N-1$ are identically distributed, and all follow the beta distribution $B(1, N)$, of probability density function (hereafter pdf) $f(t) = N \cdot (1-t)^{N-1}$. As a result, for any constant $\alpha > 0$, we have $\Pr[Y_i \leq \alpha/N] = 1 - e^{-\alpha} + O(1/N)$.*

Proof. Indeed, a standard formula [10, 2.2.1] expresses the joint pdf of $X_{(i)}$ and $X_{(i+1)}$ as:

$$f_{i,i+1}(u,v) = \begin{cases} \frac{N!}{(i-1)!(N-i-1)!} u^{i-1}(1-v)^{N-i-1} & \text{for } 0 \leq u \leq v \leq 1, \\ 0 & \text{otherwise.} \end{cases}$$

Hence, the pdf f_i of Y_i is given by:

$$f_i(t) = \int_0^{1-t} f_{i,i+1}(u, u+t)dt \quad \text{for } t \in [0,1].$$

The change of variable $u = (1-t)w$ gives:

$$\begin{aligned} f_i(t) &= (1-t) \int_0^1 f_{i,i+1}\big((1-t)w, (1-t)w + t\big) dw \\ &= c(1-t) \int_0^1 (1-t)^{i-1} w^{i-1} (1 - w - t + wt)^{N-i-1} dw \\ &= c(1-t)^i \int_0^1 w^{i-1} (1-t)^{N-i-1} (1-w)^{N-i-1} dw \\ &= c(1-t)^{N-1} \int_0^1 w^{i-1} (1-w)^{N-i-1} dw, \end{aligned}$$

where $c = \frac{N!}{(i-1)!(N-i-1)!}$. In particular, we have $f_i(t) = c'(1-t)^{N-1}$ for some constant c' and all $t \in [0,1]$, and since $\int_0^1 f_i = 1$, we must have $f_i(t) = N(1 -$

$t)^{N-1} = f(t)$ as required. As a result, we obtain:

$$\Pr\left[Y_i \le \frac{\alpha}{N}\right] = \int_0^{\alpha/N} N(1-t)^{N-1}dt = 1 - \left(1 - \frac{\alpha}{N}\right)^N$$
$$= 1 - \exp\big(N \cdot (-\alpha/N + O(1/N^2))\big) = 1 - e^{-\alpha} + O(1/N).$$

This concludes the proof. □

As an example consider a modulus n of size 160. Starting from 2^{40} ECDSA signatures, after one iteration of the sort-and-difference algorithm, about 86.5% of them will have a value $c'_j < 2^{121}$. The percentage drops to 22.1% if we consider only those ones with a value $c'_j < 2^{118}$. Note that this proposition is only true for the first iteration of the algorithm where we really can consider variables as uniformly random and independently distributed. Clearly they are not after this: if after the first round variables were uniformly distributed, the ratio between $\gamma = -2$ and $\gamma = 1$ would be $0.125 = 1/2^3$ where it is ≈ 0.255. Sadly, it appears that the ratio progress in our disfavor when we want to iterate, *i.e.* the ratio after ι iterations is less than $(1 - e^{-2^\gamma})^\iota$. We thus do not have a lower bound. However the ratio can be experimentally determined and Table 1 gives an overview for different values of γ up to 6 iterations.

γ	-2	-1	0	1	2
1st iteration	0.22	0.39	0.63	0.86	0.98
2nd iteration	0.031	0.12	0.36	0.75	0.94
3rd iteration	$3.2\ 10^{-3}$	0.025	0.17	0.64	0.89
4th iteration	$3.0\ 10^{-4}$	$4.6\ 10^{-3}$	0.069	0.53	0.84
5th iteration	$2.0\ 10^{-5}$	$6.7\ 10^{-4}$	0.022	0.40	0.79
6th iteration	$2.8\ 10^{-6}$	$9.5\ 10^{-5}$	$6.5\ 10^{-3}$	0.28	0.73

Table 1. Experimental ratio between the ECDSA signatures of the form (c'_j, h'_j) such that $c'_j < 2^{\log n - \iota\cdot(\log S+\gamma)}$, and the S initial signatures, after ι iterations of the sort-and-difference algorithm.

Given S signatures, we have to choose a pair (γ, ι) such that $\log n - \iota \cdot (\log S + \gamma) = \ell$ is sufficiently small to perform a FFT in $2^\ell \log \ell$ time and 2^ℓ memory complexities. The algorithm complexity is $O\big(S\log(S) + \ell\log(\ell)\big)$. Now a verification is necessary to be sure that this set of parameters will give a successful attack. Indeed denote by $B_n(\mathbf{K})$ the initial bias which is fully determined by the number of most (or least) significant bits of the k_i which are known or set to zero (see Table 2 for some values). From properties (b) and (c) of the Lemma 1, each iteration of the sort-and-difference algorithm reduces the bias by raising it to the square of its norm (assuming that the variables are independant): indeed, let X, Y be uniformly distributed and independent random variables on $[0, n-1]$, then $B_n(X) = B_n(Y)$ and $B_n(X - Y) = B_n(X)\overline{B_n(Y)} = |B_n(X)|^2$. The final bias is then approximated by $|B_n(\mathbf{K})|^{2^\iota}$. Thus the following inequality holds since $B_n(V_{w_m}) \approx 1/\sqrt{L}$:

$$|B_n(\mathbf{K})|^{2^\iota} \gg 1/\sqrt{L},$$

where L represents as before the number of reduced pairs (c_j, h_j) with $c_j < 2^\ell$. Using Table 1 which gives the ratio L/S for different choices of pairs (γ, ι), we obtain a relation between S, ι, ℓ and n.

Note that contrary to previous reports in the literature [19,3], we do not need to center the k_j around 0. Indeed sort-and-difference algorithm performs only subtractions and does not mix subtractions and additions as is common with lattice reduction or generalized birthday algorithms.

b	1	2	3	4	5
$B_n(\mathbf{K})$	0.6366198	0.9003163	0.9744954	0.9935869	0.9983944

Table 2. Some values of bias for large n, when b most (or least) significant bits of k are known, using Property (d) of Lemma 1.

3.2 Implementation

We successfully implemented the attack. As our target, we chose the SECG P160 R1 curve, published in 2000 by the SECG consortium [28] and still considered secure. We fixed the most significant bit in the nonces and checked (with the help of the secret) that we indeed got the expected bias: ≈ 0.63662. Our C++ implementation was based on the RELIC toolkit (using its provided plain C integer arithmetic) [1] and FFTW [11]. We parallelized it in a straightforward manner (including (quick)sorting phases) and tested it on a multicore machine.

We generated 2^{33} signatures and performed 4 sort-and-difference reduction phases. 450 millions (which is 52.5%) of our initial 2^{33} signatures had their c_j reduced down to 32 bits, as was expected from table 1. The bias after 4 reduction steps was 0.000743558 which is slightly greater than the expected $0.63662^{2^4} \approx 0.00072792$. We then computed a FFT on 32 bits (we selected the reduced c_j smaller than 2^{32}). The best candidate had a score approximately 35% greater than the second. Both corresponding MSB of the secret differed only by the 31st and 32nd most significant bits. The 3rd and 4th candidates were also very close to the two first ones, with score approximately 1/3 of the best candidate. Then, there was a number of random values with maximal score approximately 1/6 of the best one. We repeated the experiment several (5) times and got similar results, always finding at least the 30 MSB of the secret with the best candidate. We couldn't repeat it more because of the high computational resources involved.

The total memory used by the signatures and FFT tables was slightly more than 1 terabyte. To recover 32 bits of the private key, the attack took approximately 1150 CPU-hours, most of it being data exchange, which we can decompose as follows:

– 70%: parallelised quicksort (the most memory-intensive phase)
– 18%: signature generation (approximately 250 to 430 kilocycles per signature depending on the CPU, excluding hash computations)
– 10%: candidate selection and FFT table preparation
– $\approx$ 1%: the FFT itself.

We did not use more parallelizable sorts like Batcher odd-even mergesort [2] but this would clearly be the next thing to do from a performance perspective.

Next steps of the attack to recover the following bits of the secret were done as in [19]. Basically, it amounts to a replay of algorithm 2 on the initial signatures, putting the previously found MSB of the secret into h_j. Write the private key x as $x_0 2^m + x_1$ where x_0 is the recovered m MSB at the first round. Then $(h_j) + (c_j)x = (h_j + c_j x_0 2^m) + (c_j x_1)$ and we want to recover the MSB of x_1. We proceed as in the first round, except that we now keep the c_j that are smaller than $2^{\ell+m}$ instead of 2^ℓ (thus when $\ell = m$ we just have to stop the reduction one iteration earlier). Then we build the FFT table as $Z[c_j/2^m] = Z[c_j/2^m] + e^{2\pi i h'_j/n}$. The FFT recovers the next most significant chunk of bits of the secret key. The computation restart makes it necessary to go back from the initial signatures, but there's no need to keep them in memory during the reduction. In practice we had barely enough memory to keep them, but in order to reduce memory usage they should either be stored on disk and retrieved to iterate the secret recovery, or tracked down through the reduction and rebuilt afterwards.

In practice, it is advisable to take a small security margin and reinject only 30 bits of the computed MSB of the secret to account for small variations of sampling around the peak. In any case, if we recurse with a wrong secret, the FFT will not detect any peak. Experiments indeed showed no peak in this case, with the highest score not being statistically different from the other ones. This paves the way for a time/memory tradeoff: suppose the hardware is limited in memory and can only work on (say) 2^{31} signatures and 2^{30} FFT size instead of the 2^{33} needed for attacking 160 bits with 4 iterations with the previous algorithm. We first reduce the c_j from 160 to 40 bits with 4 reductions as usual. We then simply guess the 10 MSB of the secret and build 2^{30}-sized FFT tables accordingly. The guess will be correct on the only one FFT among the 2^{10} which shows a significant peak. Since FFTs are particularly efficient, much more than sorting, this is of practical importance. Alternatively, if it's possible to compute 2^{41} signatures, we can select only the expected $1/2^{10}$ fraction of signatures whose corresponding c_j have their 10 MSB already zeroes, that is to say that have 150 bits instead of 160 and can be reduced to 30 with 4 iterations. Finally, since the FFT table takes less memory than the signatures (a complex number occupies 16 bytes whereas a signature requires at least 40), we could improve the attack further by either carrying out several FFTs in parallel when guessing some bits of the secret, or by increasing the size of the FFT table slightly (with a corresponding increase of the selection bound on c_j). This would have two advantadges. Firstly, it would improve the sampling around the peak and reduce the uncertainty. Secondly, the bound increase implies that some signatures would be selected after the third round of reduction instead of the fourth, thus having a much better bias and hopefully revealing more precise information about the secret.

Our experiments targeted a 160-bit curve, but it should be pointed out that larger curves are susceptible to this attack as well. Roughly speaking, one can carry out the key recovery attack with 1-bit nonce bias on an N-bit curve in time $\approx 2^{N/5 \log_2(N/5)}$ and memory $\approx 2^{N/5}$. For example, a 256-bit curve can be attacked in time $\approx 2^{58}$ and memory $\approx 2^{52}$: generate 2^{52} signatures, perform 4 reduction steps (removing $4 \cdot 51 = 204$ bits on approximately 53% of the data), keep signatures with c_j less than 2^{52} and carry out the FFT on a table of size 2^{52}. One signature is $64 = 2^6$ bytes, so that the total memory needed for the attack is 2^{18} terabytes of storage, which corresponds to 65536 of today's 4 TB disks. This does not appear to be out of reach of well-funded adversaries.

4 Security analysis of the recomposition technique

The results presented so far had no direct connection with GLV/GLS curves. We now turn to such curves, and first discuss in this section the security of what we called the "recomposition technique" for GLV/GLS coefficients (namely, choose k_1 and k_2 uniformly at random in some interval $[0, K)$ to obtain $k = k_1 + k_2\lambda \bmod n$), whereas the next section will focus on the "decomposition technique".

To fix ideas, we consider an elliptic curve E obtained by the quadratic GLS method over a prime field [12, §2.1]. In other words, there is an elliptic curve E_0 over the prime field $\mathbb{F}_p$ such that E is the quadratic twist of E_0 over $\mathbb{F}_{p^2}$. If we denote by $p + 1 - t$ the order of $E_0(\mathbb{F}_p)$ (where t is bounded as $|t| \leq 2\sqrt{p}$ by the Hasse–Weil theorem), the order n of $E(\mathbb{F}_{p^2})$ satisfies:

$$n = (p-1)^2 + t^2. \tag{1}$$

We assume that this order n is prime, which is the main case of interest. Then, E is endowed with an efficient endomorphism ψ (obtained by conjugating the Frobenius map with the twisting isomorphism) which acts on the cyclic group $E(\mathbb{F}_{p^2})$ by multiplication by

$$\lambda \equiv t^{-1}(p-1) \pmod{n}. \tag{2}$$

In particular, $\lambda^2 \equiv (p-1)^2/t^2 \equiv -t^2/t^2 \equiv -1 \pmod{n}$.

In this setting, we first prove in §4.1 that if k_1 and k_2 are chosen uniformly at random in $[0, \sqrt{n})$, then $k = k_1 + k_2\lambda$ is statistically close to uniform in $\mathbb{Z}/n\mathbb{Z}$, so that such a choice of (k_1, k_2) can be used securely in any cryptographic protocol (and in particular ECDSA). On the other hand, we show in §4.2 that if k_1 and k_2 are chosen in $[0, 2^m)$ where $m = \lfloor \frac{1}{2} \log_2 n \rfloor$ instead, then $k = k_1 + k_2\lambda$ may not be close to uniform anymore, and we show that a variant of Bleichenbacher's attack can apply. In §4.3, we describe an implementation of that attack on a 160-bit GLS curve, similar to the attack of §3.

4.1 A secure choice of (k_1, k_2)

Let E be a curve of prime order n over $\mathbb{F}_{p^2}$ obtained by the quadratic GLS method as above. In view of (1), we have:

$$(p-1)^2 \le n \le (p-1)^2 + (2\sqrt{p})^2 = (p+1)^2,$$

and the inequalities are in fact strict, since n is prime. Thus, we have $p - 1 < \sqrt{n} < p + 1$, and it follows that the distribution of $k = k_1 + k_2\lambda$ for (k_1, k_2) uniform in $[0, \sqrt{n})^2$ is statistically close to the distribution of the same k for (k_1, k_2) uniform in $[0, p-1)^2$. We will thus concentrate on the latter, and show that it is close to uniform in $\mathbb{Z}/n\mathbb{Z}$ using the following lemma.

Lemma 3. *The following map is injective.*

$$\begin{aligned} F\colon [0, p-1)^2 &\longrightarrow \mathbb{Z}/n\mathbb{Z} \\ (k_1, k_2) &\longmapsto k_1 + k_2\lambda. \end{aligned}$$

Proof. Consider two distinct pairs $(k_1, k_2) \neq (k_1', k_2')$ such that $F(k_1, k_2) = F(k_1', k_2')$. We have:

$$\begin{aligned} (x - x') + (y - y')\lambda \equiv 0 \pmod{n} \\ (x - x')^2 \equiv \lambda^2 (y - y')^2 \pmod{n} \\ (x - x')^2 + (y - y')^2 \equiv 0 \pmod{n}, \end{aligned}$$

since $\lambda^2 \equiv -1 \pmod{n}$. Thus, the positive integer $(x-x')^2 + (y-y')^2$ is divisible by n, and it is also smaller than $2(p-1)^2 < 2n$, so we must have $(x - x')^2 + (y - y')^2 = n$. In other words, $(x - x')^2 + (y - y')^2$ is a decomposition of n as a sum of two squares. Now it is well-known that, as a prime number, n has at most one decomposition as a sum of two squares up to order and sign (see e.g. [18, §3.6]), and $(p-1)^2 + t^2$ is one such representation. As a result, we must have either $x - x' = \pm(p-1)$ or $y - y' = \pm(p-1)$, and neither is possible since those difference are bounded by $p - 2$ in absolute value. Hence, F is injective as required. $\square$

Theorem 1. *The distribution of the values $k = k_1 + k_2\lambda$ for (k_1, k_2) uniform in $[0, p-1)^2$ is statistically close to the uniform distribution on $\mathbb{Z}/n\mathbb{Z}$. More precisely, the statistical distance:*

$$\Delta_1 = \sum_{k \in \mathbb{Z}/n\mathbb{Z}} \left| \Pr\left[k = k_1 + k_2\lambda \ ; \ (k_1, k_2) \overset{\$}{\leftarrow} [0, p-1)^2\right] - \frac{1}{n} \right|$$

is given by $\Delta_1 = 2t^2/n$, which is negligible.

Proof. Indeed, since the function F above is injective by Lemma 3, the probability $\Pr\left[k = k_1 + k_2\lambda \ ; \ (k_1, k_2) \overset{\$}{\leftarrow} [0, p-1)^2\right]$ is equal to $1/(p-1)^2$ for each of the $(p-1)^2$ points in the image of F, and 0 for each of the $n-(p-1)^2 = t^2$ points outside of that image. Therefore:

$$\varDelta_1 = (p-1)^2 \cdot \left|\frac{1}{(p-1)^2} - \frac{1}{n}\right| + t^2 \cdot \left|0 - \frac{1}{n}\right| = 1 - \frac{(p-1)^2}{n} + \frac{t^2}{n} = \frac{2t^2}{n}$$

as required. This is bounded above by $8p/(p-1)^2$, which is indeed negligible. □

Remark 1. Theorem 1 means that it is secure, in any ECC protocol instantiated over the GLS curve E, to sample random scalars k by picking k_1 and k_2 uniformly in $[0, p-1)$, or equivalently $[0, \sqrt{n})$.

As we can see, the proof relies on the particular arithmetic properties of the quadratic GLS method (mainly the fact that $\lambda = \sqrt{-1}$ in $\mathbb{Z}/n\mathbb{Z}$), so that the result does not readily extend to different settings, like the GLV method on a curve of CM discriminant -3. And indeed, in that case, Brumley and Nyberg have provided evidence that choosing (k_1, k_2) uniformly in $[0, \sqrt{n})$ may not yield a close to uniform distribution for k [7, Example 3]. They suggest an alternate approach to select intervals to choose k_1 and k_2 from and still achieve high entropy in a more general setting, but since the quadratic GLS method is one of the most used variants of GLV/GLS, we believe Theorem 1 is of significant practical interest.

4.2 Breaking insecure choices of (k_1, k_2) with Bleichenbacher's attack

In the quadratic GLS setting, we have just seen that choosing (k_1, k_2) uniformly in $[0, \sqrt{n})^2$ yields a close-to-uniform distribution of $k = k_1 + k_2\lambda$. However, we can reasonably suspect that if we choose k_1 and k_2 uniformly in $[0, 2^m)$, $m = \lfloor \frac{1}{2} \log_2 n \rfloor$ (i.e. uniform bitstrings of length just under half of the size of n), the distribution of k will no longer be uniform. This is not immediately visible on the bias, however.

Indeed, if we let $T = 2^m$ and define K_1, K_2 as independent uniform random variables over $[0, T)$ and K as the random variable in $\mathbb{Z}/n\mathbb{Z}$ given by $K = K_1 + K_2\lambda$, we have, by Lemma 1:

$$B_n(K) = B_n(K_1) \cdot B_n(\lambda K_2) = \frac{1}{T}\left|\frac{\sin(\pi T/n)}{\sin(\pi/n)}\right| \cdot \frac{1}{T}\left|\frac{\sin(\pi\lambda T/n)}{\sin(\pi\lambda/n)}\right|.$$

The first factor is very close to 1, but the second factor is usually negligible. For example, on the 160-bit GLS curve (3) below, we have $T = 2^{79}$ and $B_n(\lambda K_2) \approx 1.52/T$. As a result, Bleichenbacher's attack does not apply directly to this setting in general.

However, since $\lambda \equiv t^{-1}(p-1) \pmod{n}$, we claim that there is a significant bias on the values $t \cdot k$. Indeed, we have:

$$\begin{aligned}
B_n(tK) &= B_n(tK_1) \cdot B_n\big((p-1)K_2\big) \\
&= \frac{1}{T}\Big|\frac{\sin(\pi t T/n)}{\sin(\pi t/n)}\Big| \cdot \frac{1}{T}\Big|\frac{\sin(\pi(p-1)T/n)}{\sin(\pi(p-1)/n)}\Big| \\
&= \frac{1}{T}\,\frac{\pi t T/n + O((tT/n)^3)}{\pi t/n + O((t/n)^3)} \cdot \frac{1}{T}\,\frac{|\sin(\pi(p-1)T/n)|}{\pi(p-1)/n + O(((p-1)/n)^3)} \\
&= \Big(1 + O\big((tT/n)^2 + (p/n)^2\big)\Big) \cdot \Big|\frac{\sin(\pi(p-1)T/n)}{\pi(p-1)T/n}\Big|.
\end{aligned}$$

The big-O in the first factor is negligible since $tT/n = \Theta(p^{1/2} \cdot p/p^2) = \Theta(p^{-1/2})$ and $p/n = \Theta(p^{-1})$. On the other hand, $(p-1)T/n \approx T/\sqrt{n}$ is roughly between 0.5 and 1 depending on how close n is to a power of two. Thus, the bias is significant in general, and is maximal when $(p-1)T/n$ is smallest (close to 1/2), which happens when n is just under a power of two. The bias $B_n(tK)$ is then close to $1/(\pi \cdot 1/2) = 2/\pi \approx 0.637$.

It is then straightforward to adapt Bleichenbacher's attack to this setting by targetting the values $t \cdot k$ instead of k. We can then break ECDSA signatures that use nonces of the form $k = k_1 + k_2\lambda$ above using that variant. An implementation of that attack is discussed in the next subsection.

4.3 Implementation of Bleichenbacher's attack in the GLS setting

We carry out the attack described above on the 160-bit GLS curve E defined as follows. Over the 80-bit prime field[8] $\mathbb{F}_p$, $p = 255 \cdot 2^{72} + 1$, we define $E_0\colon y^2 = x^3 - 3x/23 + 104$. Then, the elliptic curve E is the quadratic twist of E_0 over $\mathbb{F}_{p^2} = \mathbb{F}_p(\sqrt{23})$, namely:

$$E\colon y^2 = x^3 - 3x + 104 \cdot \sqrt{23}^{\,3} \text{ over } \mathbb{F}_{p^2}. \tag{3}$$

The order of $E_0(\mathbb{F}_p)$ is $p + 1 - t$ for $t = 776009485427$, and $E(\mathbb{F}_{p^2})$ is of prime order $n = (p-1)^2 + t^2$. The theoretical value of the bias $B_n(tK)$, computed using the exact formula above, is then ≈ 0.634.

We performed the recovery of 32 MSB of a private key as in section 3.2. We computed 2^{33} signatures and unrolled the attack on $(tc_j \bmod n, th_j \bmod n)$ instead of (c_j, h_j). We checked the bias and obtained ≈ 0.634116 which is close to the theory. In practice the attack took about 2000 CPU-hours, with 56% for the signature generation, 37% for the four sort-and-difference reduction steps, 5% for the candidate selection and FFT table preparation and less than 0.5% for the FFT itself. In wall-clock time terms, except for the signature generation which took (much) longer, other phases were identical as 3.2. We attribute this unexpected increase in signing time to threshold effects: for example, representing elements on a prime field with $\approx 2^{160}$ elements needs only 3 64-bit words, whereas a on $\mathbb{F}_{p^2}$ we needed $4 \times 2 = 8$ words.

[8] This is an example of "optimal prime field" (OPF). See e.g. [30].

5 Security analysis of the decomposition technique

In this section, we analyze the security of algorithms for computing the decomposition of the nonces used in the GLV method from a side-channel analytic perspective. Many techniques have been proposed, including [13,26]. The original GLV method [13] based on LLL reduction of a lattice that depends on the nonce k, and variants thereof, have an execution time that depends on k, and are therefore vulnerable to timing attacks.

Therefore, we examine the security of a potentially more secure approach, the Park *et al.* [26] decomposition technique, using more involved power analysis technique.

5.1 Decomposition Algorithm

Park *et al.* provide an alternative decomposition to the GLV paper [13] which reduces the theoretical bound for the decomposition using the theory of μ-Euclidian algorithm and is a little bit faster. The algorithm requires two short and independent vectors v_1 and v_2 of the two-dimensional lattice $L = \{(x, y) : x + y\lambda = 0 \bmod n\}$. We can find these vectors during a precomputation time using the Gauss reduction. The algorithm consists in finding a vector in the lattice $L = \mathbb{Z}v_1 + \mathbb{Z}v_2$ that is close to $(k, 0)$ using linear algebra. Then, (k_1, k_2) is determined by the equation:

$$(k_1, k_2) = (k, 0) - (\lfloor b_1 \rceil v_1 + \lfloor b_2 \rceil v_2),$$

where $(k, 0) = b_1 v_1 + b_2 v_2$ is an element of $\mathbb{Q} \times \mathbb{Q}$.

Algorithm 3 Decomposition technique of Park *et al.* in [26]

```
Require: $k \approx n$, the shortest vectors $v_1 = (x_1, y_1), v_2 = (x_2, y_2)$
Ensure: $(k_1, k_2)$ such that $k = k_1 + k_2\lambda (\bmod n)$
1: $D = x_1 y_2 - x_2 y_1, a_1 = y_2 k, a_2 = -y_1 k$
2: $z_i = \lfloor a_i / D \rceil$ for $i = 1, 2$
3: $k_1 = k - (z_1 x_1 + z_2 x_2)$, $k_2 = z_1 y_1 + z_2 y_2$ return $(k_1, k_2)$
```

The decomposition technique depicted in Algorithm 3 makes many computations involving the sensitive nonce k. Particularly, the computation of a_1 (*resp.* a_2) is based on a multiplication of the nonce k by y_2 (*resp.* y_1) which is assumed to be known since it is a precomputed value obtained from public parameters using a deterministic algorithm.

Suppose now that we obtain the knowledge of the least significant byte of ℓ nonces $k_1, \cdots, k_\ell$. The best strategy for finding the secret key x consists in performing classical lattice attacks as proposed in [15,21,22]. For a 160-bit modulus, the lattice attack works consistently for $\ell \gtrsim 27$. However the side-channel attack may sometimes fail, *i.e.* the returned byte of some k_j can be a wrong value. Thus, by denoting $0 < c < 1$ the confidence rate, the side-channel attack has to be performed on $m > \lceil 27/c \rceil$ signatures. Then:

- Select 27 signatures at random among them.
- Perform the attack using these signatures.
- If the attack fails, goto the first step.

The probability of success at each iteration of the lattice attack is $\binom{m\cdot c}{27}/\binom{m}{27}$. As an example, suppose we obtain $m = 200$ signatures, and can guess the least significant byte with 90% accuracy ($c = 0.9$). Then the probability of success of the lattice attack is about 4.7% and 21 lattice reductions have to be performed on average. Since LLL reductions are cheap, much lower success probabilities are tractable as well.

In the following, we discuss the side-channel attack that aims at recovering the first byte of the nonce targeting the two aforementioned multiplications. We present the attack in the particular case of a 8-bit implementation (that corresponds to the device we used in experiments). Note that this attack may also work for 16-bit implementation but in this case the computational cost will be larger and the success rate smaller.

5.2 Side-Channel Attack on this implementation

The details of the attack highly rely on the way the multiplication is implemented. Depending of the underlying algorithm, the attack may be more or less difficult. We present here the attack corresponding to the implementation we target but we will discuss adaptations to different algorithms. The multiplication we target is a schoolbook multiplication with the nonce being scanned in the outter loop. Algorithm 4 outlines the implementation of such multiplication for ℓ_n-bit nonces and $\ell_{n/2}$-bit b.

Algorithm 4 Multiplication $v = kb$ of $k = \sum_{i=0}^{\ell_n/8} k_i 2^{8i}$ times $b = \sum_{i=0}^{\ell_{n/2}/8} b_i 2^{8i}$

```
Require: $\ell_n$-bit $k$ and $\ell_{n/2}$-bit $b$ two integers, $v = 0$
Ensure: $v = k \times b$

1: $v \leftarrow 0$
2: for $i = 0$ to $i < \ell_n/8$ do
3:     $c_0 \leftarrow 0$
4:     for $j = 0$ to $j < \ell_{n/2}/8$ do
5:         $v_{i+j} = (k_i \times b_j + c_j)$ & 0xFF
6:         $c_{j+1} = (k_i \times b_j + c_j) \gg 8$
7:     end for
8: end for   return $v$
```

The idea is to take profit of all operations involving the first nonce-byte in the inner loop to recover its value. This can be done by propagating a probability distibution from an operation to another and updating it with the corresponding leakages. Since the nonce bits have to be recovered using a single trace (the nonce is randomly generated for each signature) we place ourselves in the context of a profiled attack. The application of such an attack in a non-profiled setting is left as an open question.

Template Attack on One Step. One step of the inner loop consists in a multiplication of the first byte of the nonce k_0, a byte of the auxiliary input b_i and the carry c_i. This results in a value v_i and a new carry c_{i+1}. We may obtain leakages for each of these variables. We denote by capital letters the output distributions of template exploitation corresponding to small letter variables. For instance, after processing the leakage corresponding to c_i, the attacker gets a distribution

$$C_i = \big(\Pr(c_i = 0), \Pr(c_i = 1), \ldots, \Pr(c_i = 255)\big).$$

Since these variables may be manipulated more than once during the computation, different leakage points may be combined by multiplicating probabilities then normalizing the resulting distribution. More precisely, let $l_1, l_2, \ldots, l_l$ be leakages corresponding to variable k_0, then the distribution K_0 obtained from these leakages is computed as

$$\Pr(k_0 = x) = \frac{1}{Z} \prod_{j=1}^{l} \Pr(k_0 = x | l_j),$$

where the normalizing coefficient Z is given by $\sum_x \prod_{j=1}^{l} \Pr(k_0 = x | l_j)$.

Propagating and Updating Distribution. Let us now discuss how to take profit of all the leakages of the inner loop to gain information on the byte k_0. The main idea is to gather all the information from all variables of a given step i into distribution K_0 and C_{i+1} then do the same at step $i + 1$ using the newly updated distributions. From a probabilistic point of view we should compute the joint distribution of variables of step i then compute marginalized distributions K_0 and C_{i+1}. The following algorithm updates the distributions K_0 and C_{i+1} according to the distributions of variables b_i, v_i and c_i.

The attacker starts with using Algorithm 5 for the first step. Then she uses the newly updated distributions K_0 and C_1 and the initial distributions B_1, V_1 and C_2 as inputs of Algorithm 5 and so on ... At the end, the attacker gets the final distribution K_0 from which she can derive the most likely value of the least significant bit (or more).

References

1. D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. `http://code.google.com/p/relic-toolkit/`.
2. K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 307–314, New York, NY, USA, 1968. ACM.

Algorithm 5 Information propagation for one step of the multiplication inner loop.

Require: distributions K_0, B_i, V_i, C_i and C_{i+1}
Ensure: K'_0 and C'_{i+1} updated distribution

```
1: $K'_0 = (0, 0, \dots, 0)$
2: for $0 \le k, b, c < 256$ do
3:     $2^8 \cdot u + v \leftarrow k \times b + c$
4:     $K'_0(k) \leftarrow K'_0(k) + K_0(k) \cdot B_i(b) \cdot C_{i-1}(c) \cdot V_i(v) \cdot C_i(u)$
5:     $C'_{i+1}(u) \leftarrow C'_{i+1}(u) + K_0(k) \cdot B_i(b) \cdot C_i(c) \cdot V_i(v) \cdot C_{i+1}(u)$
6: end for
7: return $K'_0 / \sum_k K'_0(k)$ and $C'_{i+1} / \sum_u C'_{i+1}(u)$
```

3. D. Bleichenbacher. On the generation of one-time keys in DL signature schemes. *Presentation at IEEE P1363 Working Group meeting*, 2000.
4. A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, 2003.
5. J. W. Bos, C. Costello, H. Hisil, and K. Lauter. High-Performance Scalar Multiplication Using 8-Dimensional GLV/GLS Decomposition. In G. Bertoni and J.-S. Coron, editors, *CHES*, volume 8086 of *Lecture Notes in Computer Science*, pages 331–348. Springer, 2013.
6. B. B. Brumley and R. M. Hakala. Cache-Timing Template Attacks. In M. Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 667–684. Springer, 2009.
7. B. B. Brumley and K. Nyberg. On Modular Decomposition of Integers. In B. Preneel, editor, *AFRICACRYPT*, volume 5580 of *Lecture Notes in Computer Science*, pages 386–402. Springer, 2009.
8. Y. Chen and P. Q. Nguyen. BKZ 2.0: Better Lattice Security Estimates. In D. H. Lee and X. Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011.
9. C. Costello, H. Hisil, and B. Smith. Faster Compact Diffie-Hellman: Endomorphisms on the x-line. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2014.
10. H. A. David and H. N. Nagaraja. *Order Statistics*. Wiley, 2003.
11. M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on “Program Generation, Optimization, and Platform Adaptation”.
12. S. D. Galbraith, X. Lin, and M. Scott. Endomorphisms for faster elliptic curve cryptography on a large class of curves. *J. Cryptology*, 24(3):446–469, 2011.
13. R. P. Gallant, R. J. Lambert, and S. A. Vanstone. Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms. In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200. Springer, 2001.
14. A. Guillevic and S. Ionica. Four-Dimensional GLV via the Weil Restriction. In K. Sako and P. Sarkar, editors, *ASIACRYPT (1)*, volume 8269 of *Lecture Notes in Computer Science*, pages 79–96. Springer, 2013.
15. N. Howgrave-Graham and N. P. Smart. Lattice Attacks on Digital Signature Schemes. *Des. Codes Cryptography*, 23(3):283–290, 2001.

16. É. Levieil and P.-A. Fouque. An Improved LPN Algorithm. In R. D. Prisco and M. Yung, editors, *SCN*, volume 4116 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2006.
17. M. Liu and P. Q. Nguyen. Solving BDD by Enumeration: An Update. In E. Dawson, editor, *CT-RSA*, volume 7779 of *Lecture Notes in Computer Science*, pages 293–309. Springer, 2013.
18. H. McKean and V. Moll. *Elliptic curves: function theory, geometry, arithmetic*. Cambridge University Press, 1999.
19. E. D. Mulder, M. Hutter, M. E. Marson, and P. Pearson. Using Bleichenbacher's solution to the hidden number problem to attack nonce leaks in 384-bit ECDSA: extended version. *J. Cryptographic Engineering*, 4(1):33–45, 2014.
20. D. Naccache, P. Q. Nguyen, M. Tunstall, and C. Whelan. Experimenting with Faults, Lattices and the DSA. In S. Vaudenay, editor, *Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 16–28. Springer, 2005.
21. P. Q. Nguyen and I. Shparlinski. The Insecurity of the Digital Signature Algorithm with Partially Known Nonces. *J. Cryptology*, 15(3):151–176, 2002.
22. P. Q. Nguyen and I. Shparlinski. The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces. *Des. Codes Cryptography*, 30(2):201–217, 2003.
23. P. Q. Nguyen and D. Stehlé. Low-Dimensional Lattice Basis Reduction Revisited. In D. A. Buell, editor, *ANTS*, volume 3076 of *Lecture Notes in Computer Science*, pages 338–357. Springer, 2004.
24. P. Q. Nguyen and M. Tibouchi. Lattice-Based Fault Attacks on Signatures. In M. Joye and M. Tunstall, editors, *Fault Analysis in Cryptography*, Information Security and Cryptography, pages 201–220. Springer, 2012.
25. T. Oliveira, J. López, D. F. Aranha, and F. Rodríguez-Henríquez. Two is the fastest prime: lambda coordinates for binary elliptic curves. *J. Cryptographic Engineering*, 4(1):3–17, 2014.
26. Y.-H. Park, S. Jeong, C. H. Kim, and J. Lim. An Alternate Decomposition of an Integer for Faster Point Multiplication on Certain Elliptic Curves. In D. Naccache and P. Paillier, editors, *Public Key Cryptography*, volume 2274 of *Lecture Notes in Computer Science*, pages 323–334. Springer, 2002.
27. J. M. Pollard. Kangaroos, monopoly and discrete logarithms. *J. Cryptology*, 13(4):437–447, 2000.
28. C. Research. Standards for efficient cryptography, SEC 1: Elliptic curve cryptography, September 2000. Version 1.0.
29. B. Smith. Families of Fast Elliptic Curves from $\mathbb{Q}$-curves. In K. Sako and P. Sarkar, editors, *ASIACRYPT (1)*, volume 8269 of *Lecture Notes in Computer Science*, pages 61–78. Springer, 2013.
30. E. Wenger and J. Großschädl. An 8-bit AVR-based elliptic curve cryptographic RISC processor for the Internet of Things. In *MICRO Workshops*, pages 39–46. IEEE Computer Society, 2012.