Side-Channel Analysis of Weierstrass and Koblitz Curve ECDSA on Android Smartphones*

Pierre Belgarric[1,4], Pierre-Alain Fouque[2], Gilles Macario-Rat[1], and Mehdi Tibouchi[3]

[1] Orange Labs, Issy-les-Moulineaux, France
gilles.macariorat@orange.com
[2] Université de Rennes 1 and Institut universitaire de France, Rennes, France
pierre-alain.fouque@ens.fr
[3] NTT Secure Platform Laboratories, Tokyo, Japan
tibouchi.mehdi@lab.ntt.co.jp
[4] HP Labs, HP Inc., Bristol, United Kingdom
pierre.belgarric@hp.com

Abstract. In this paper, we study the side-channel resistance of the implementation of the ECDSA signature scheme in Android's standard cryptographic library. We show that, for elliptic curves over prime fields, one can recover the secret key very efficiently on smartphones using electromagnetic side-channel and well-known lattice reduction techniques. We experimentally show that elliptic curve operations (doublings and additions) can be distinguished in a multi-core CPU clocking over the giga-hertz. We then extend the standard lattice attack on ECDSA over prime fields to binary Koblitz curves. This is the first time that such an attack is described on Koblitz curves. These curves, which are also available in Bouncy Castle, allow very efficient implementations using the Frobenius operation. This leads to signal processing challenges since the number of available points are reduced. We investigate practical side-channel, showing the concrete vulnerability of such implementations. In comparison to previous works targeting smartphones, the attacks presented in the paper benefits from discernible architectural features, like specific instructions computations or memory accesses.

1 Introduction

Side-Channel Analysis is an important set of techniques allowing to recover secret information. Isolation breaches are exploited during the execution of a sensitive algorithm [26,27]. Various sources of leakage can be used, such as physical ones (e.g., power consumption [27], electromagnetic emanations, or execution timing [26]), or microarchitectural ones (e.g., cache state or branch prediction).

Physical side-channels have been used for more than 15 years to assess the security of smartcards, ASIC and FPGA. Security vulnerabilities have been a real concern for embedded devices like smartcards that hold sensitive data and can be accessed by an adversary. These integrated circuits were thought to hold and protect only a few applications. But the upcoming of smartphones allowed all kinds of applications to be run on a unique mobile device, which was thought to be a mobile computer rather than a generalized smartcard. As a consequence, the hardware is not designed to be protected against physical attacks. This problem has been studied for many years by mobile operators to protect private data on these devices. Mobile operators standardized the SIM card which is used in many countries and is built to prevent any leakage of information. This chip is still used in today's phones. But the quantity of data processed nowadays is increasing exponentially, leading to a dead-end when considering the computing limitations of SIM cards and the latency of communication with smartphone hardware.

* An extended abstract appeared in the proceedings of CT-RSA 2016. This is the full version.

Sensitive applications are now developed on smartphones and software security vulnerability is an important issue. However, if the cryptographic library is not protected against physical attacks, the secret keys can be extracted and data protection becomes useless.

Our Contributions. With this evolution in mind we assess the security of Android smartphones against electromagnetic analysis. We show that the standard implementation of elliptic curve cryptography, which has been provided since the version 4.4 of the Android operating system, is not protected against these attacks and that the manipulated secret key can be extracted using a few hundreds of measurements. Many issues remained in the related literature [24,42,1,33] in order to mount a real and practical attack on mainstream libraries running on smartphones. No article address the security of widely used library such as Bouncy Castle and actual implementation. For instance, in [33], authors show that we can distinguish square and multiplication in the usual square-and-multiply algorithm. However, since in Bouncy Castle the implementation uses a sliding windows algorithm, this information is not sufficient to recover the secret key. Here, we show that on real implementation that calls this library we can recover the secret key.

On the hardware side, modern smartphone processors have interesting features which make physical attack harder: many cores, fast clock (GigaHertz, while smartcards are clocked at around 20 MegaHertz), and the leaking parts of the circuit under focus are integrated into hundred millions of transistors. This makes the leaking signal much harder to acquire and interpret. Moreover, Android is a rich OS that use many threads running concurrently and the software is executed in an applicative virtual machine (cf. appendix B). Thus the abstraction layers induce many system activities and it is not really easy to get the full trace during cryptographic computation. Previous work mainly focused on simpler processors and OSes, with the noticeable exceptions of Genkin *et al.*'s works [19,18] and Zajic and Prvulovic's experiments [41]. Nevertheless, in the two first papers, exponentiations were not observed, and in the third paper, no cryptographic algorithm was evaluated. A more detailed related work is given in Appendix A.

On a cryptanalytical viewpoint, implementations that were previously attacked on general-purpose devices, processed each bit independently. In order to have efficient cryptographic codes, sliding window algorithms are used in Bouncy Castle, and it is no more possible to mount the attacks described in related work. This explains the use of the lattice-based technique which only uses the last iterations of the trace. We can detect the last bits since we are able to identify a specific pattern that ends the computation. These attacks can be used even though we do not have the whole electromagnetic (EM) curve: with windowing algorithms, we cannot distinguish between the additions of different precomputed values and multi-threading can interrupt the double-and-add algorithm with different operations. Even in these difficult scenarios, we are able to identify the number of zero bits at the beginning or at the end of ECDSA nonces, leading to a successful lattice-based cryptanalysis.

Furthermore, the security of the windowing algorithm on Koblitz curves has not been investigated yet. Arithmetic on such curves is very efficient on hardware, and it has recently been shown that the new carryless vector instructions make these curves also appealing in software. It raises new signal processing challenge since the Frobenius endomorphism, which plays a role in the Koblitz curve setting analogous to doublings in standard scalar multiplications, is a very efficient operation, and is implemented through precomputed tables in Bouncy Castle. These operations are successfully monitored through EM side-channel. Lattice-based cryptanalysis has also been modified to address the specificities of these curves. In Bouncy Castle, the implementation of elliptic curves uses affine coordinates, but our attack can still be applied on other coordinates system such as Jacobian or

lambda [31,38] coordinates if the most significant bits of the nonces leak. Indeed, we learn these bits since we can distinguish the addition and double (frobenius in the case of binary curves) operations. Being able to distinguish them depends on their actual implementations, but in any coordinates systems, the internal operations are usually rather different and timing or power consumption are different if no careful protection are added.

We implement two EM side-channel attacks on smartphones running Android standard ECDSA implementations. We recover the private key using very few signatures either on prime field curves or on Koblitz ones. In the first attack, defined over prime field, we show that, even on systems as complex as smartphones, it is possible to distinguish exponentiation operations via EM side-channels. It allows to recover the least significant bits of the nonces during the execution of the sliding window exponentiation algorithm. Then, we conduct classical lattice-based cryptanalysis. The second attack is new and is an adaptation of the lattice-based attack in the case of Koblitz curves. In addition to this new technique, the efficient Frobenius operation is retrieved via EM side-channel. It allows to break these specific kind of curves even on complex devices.

Concurrent work. Similar results to the ones presented in this paper have been obtained, concurrently and independently, by Genkin et al. [17]. A synthesis of our results is in preparation.

Organization of the paper. In section 2, we describe some background on Android security and elliptic curve over prime field and binary field and their implementations in Bouncy Castle. In section 3, we present how we acquire and process the signal. In section 4, we show how we can recover the secret for prime field and binary curves, and discuss possible countermeasures.

2 Background on Elliptic Curve Cryptography

The security of elliptic curve cryptography is based on the computational complexity of the discrete logarithm problem over the additive group of points of an elliptic curve. This problem is stated as follows: given P and Q two points such that $Q = k \cdot P$, finding k is difficult when the group order is a large prime. Let P be a publicly known generator point and a scalar k in the finite field. Efficient algorithms allow to compute a new point $Q = k \cdot P$. Here, we work with prime and binary curves. The arithmetic used to compute with Jacobian coordinates on prime field curves and affine coordinates for binary curve, and the exact implementations used in Bouncy Castle with NAF and TNAF representation is detailed in appendix C. Computations are done on large integers, using the *BigInteger* class. In Android, the class functions ultimately bind to native ones through the *JNI*. These native functions are implemented in an *OpenSSL* class.

Prime Field Elliptic Curve. An elliptic curve can be defined over some finite field $\mathbb{K}$ of characteristic different from 2 and 3 by its short Weierstrass equation $E(\mathbb{K})$ which is the set of points on:

$$E : y^2 = x^3 + ax + b, \tag{1}$$

where $a, b \in \mathbb{K}$ and the points $(x, y) \in \mathbb{K} \times \mathbb{K}$ are solution of equation (1). To serve as a neutral element, a point at infinity (∞) is added to the other points to form a group. The addition of two points, needed to efficiently compute $k.P$, is defined for two points $P_1 = (x_1, y_1) \in E(\mathbb{K})$ and $P_2 = (x_2, y_2) \in E(\mathbb{K})$ by the new point $P_3 = (x_3, y_3) \in E(\mathbb{K})$ (see [21]): $P_3 = (\lambda^2 - x_1 - x_2, \lambda(x_1 - x_3) - y_1)$, where $\lambda = (y_1 - y_2)/(x_1 - x_2)$ if $P_1 \neq P_2$ and $\lambda = (3x_1^2 + a)/(2y_1)$ if $P_1 = P_2$.

The computation of these new coordinates requires to compute inversion which is time consuming. Consequently, the elliptic curve points are represented in Jacobian coordinates in Bouncy Castle.

To reduce the number of additions, the nonces are represented in NAF and scalar multiplication is performed using a sliding window implementation.

Koblitz Curve. Koblitz curves are anomalous binary curves defined over $\mathbb{F}_2$ and considered over the extension field $\mathbb{F}_{2^m}$. The advantage of these curves is that scalar multiplication algorithms can avoid using point doublings and are very efficient on hardware. Recently, carryless instructions have been added to general processors which makes binary curves appealing as well for software [38]. In the case of Koblitz curve, it is shown in [3], that such curves are competitive. They have been discovered by Koblitz [25], efficient algorithms have been proposed by Solinas [40] and treated formally in [21]. Their equations have the following form $E_a(\mathbb{F}_{2^m})$:

$$y^2 + xy = x^3 + ax + 1, \text{ and } a = 0 \text{ or } 1. \tag{2}$$

The interest of these curves resides in some tricks in the arithmetic of point calculus. The Frobenius map $\tau : E_a(\mathbb{F}_{2^m}) \to E_a(\mathbb{F}_{2^m})$ is defined as

$$\tau(\infty) = \infty, \text{ and } \tau(x, y) = (x^2, y^2).$$

It can be efficiently computed because squaring in $\mathbb{F}_{2^m}$ is inexpensive since it consists in adding a bit to zero between each bit of the binary representation of an element and then reducing it modulo the polynomial defining the finite field. It is known that

$$(\tau^2 + 2)P = \mu\tau(P) \text{ for all } P \in E_a(\mathbb{F}_{2^m}),$$

where $\mu = (-1)^a$. Hence, the Frobenius map can be seen as a complex number τ satisfying $\tau^2+2 = \mu\tau$ so that $\tau = (\mu+\sqrt{-7})/2$. Then, the ring of quadratic integers $\mathbb{Z}[\tau]$ generated by τ has a well-defined scalar multiplication operation on points in $E_a(\mathbb{F}_{2^m})$ via the Frobenius endomorphism. As a result, we can efficiently carry out the scalar multiplication by an integer k if we can find a compact representation of k in the form $\sum_{i=0}^{l-1} k_i\tau^i$ as an element of $\mathbb{Z}[\tau]$, with $k_i \in \{-1, 0, 1\}$. One such representation is the τ-non adjacent form, or TNAF, representation of the integer k. There are efficient algorithms to compute it (see [21]). Finally, since $\tau^2 = \mu\tau - 2$, every element $\alpha \in \mathbb{Z}[\tau]$ can be written in canonical form as $\alpha = a_0 + a_1\tau$ where $a_0, a_1 \in \mathbb{Z}$. The implementation of Bouncy Castle in order to represent an integer in WTNAF representation (TNAF representation with window) is recalled in appendix C.

ECDSA. The ECDSA signature scheme has been standardized by NIST in [36] and allows to sign any message m using two scalars (r, s) such that r is the abscissae of $k \cdot P$ and s is computed as $s = (rx + h)/k \bmod q$, where q is a large prime, $h = H(m)$ and x is the signer's ECDSA secret key.

3 Signal Processing

In this section, we explain the experimental setup used to acquire the signal. The acquisition bench is described in appendix F. We present how we synchronize the signal and we show how to distinguish doubling and addition operations. We observe that the number of multiplications is different for doubling and addition, the time intervals between these multiplications being a characteristic of each operation. Then, we explain some particular issue according to the Bouncy Castle code. Finally, we show that the multiplications, corresponding to a decrease in signal energy, are used in a different CPU mode than the other executed instructions. It may possibly explain the observed leakage.

3.1 Synchronizing the acquisitions

In [1], the oscilloscope is triggered at acquisition time through SD Card communication. The voltage of one of the data pins is monitored while a message is sent to the card. There are a few issues with that method. The SD Card is not used in all the smartphones. It is problematic to easily evaluate all devices. The mechanical base is not the same on all the platforms and much of them are difficult to access. The time is not very stable between the communication on the SD card and the beginning of the processing of interest. It is not an issue for so-called *horizontal* attacks (where the leakage patterns are a function of time) where only one trace is required, but for *vertical* attacks, it is important to have a stable and generic synchronizing signal. Finally, the phone is dismounted and a wire is melted on each evaluated phones.

To address these problems, we trigger on USB channel, which is the only standard I/O on smartphones. We send 120 bytes equal to 0 on the channel just before cryptographic computation. Low-pass filtering the USB physical signal gives a good approximation of a square signal, because the high frequencies of the succession of fronts are filtered. The pattern is clearly visible on figure 1, while sniffing the USB voltage signal. The oscilloscope triggers on a wide enough square pattern. Similarly a message can be sent just after the cryptographic processing to surround the interesting leakage in time. Other signals with the same values could transit on the channel triggering the oscilloscope on a wrong pattern. The probability of such an occurrence is low, and experimentally the problem did not occur during our experiments.

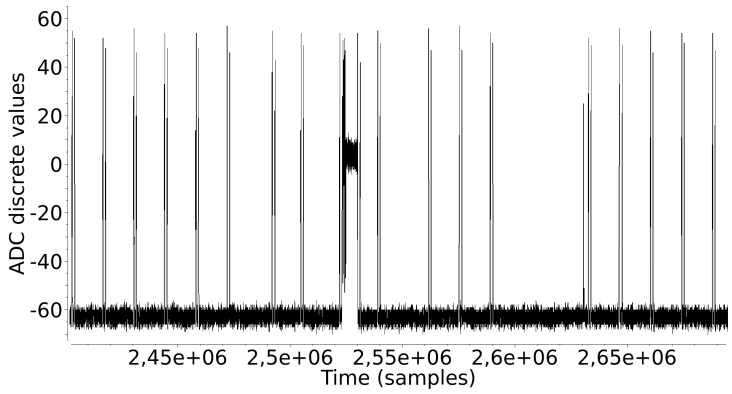

Fig. 1. USB voltage: synchronization message pattern sent on USB channel before the signature.

There is still significant jitter between oscilloscope triggering and the beginning of cryptographic computation. To improve the acquisitions, a "sleep" operation was added just before the sensitive computation. The CPU does not consume power during that period. It is easily detectable on EM signals as can be seen on figure 3 (a). There are other time periods where the processor is idle. We forced this state to be long enough in order to discriminate it with other idle states between USB pattern and cryptographic computations.

The coupling of USB channel pattern with CPU idle state (Figure 2) leads to a precise synchronization stage. The jitter is only a few instructions long, which is very interesting, especially for investigations of *Differential Power Analysis*.

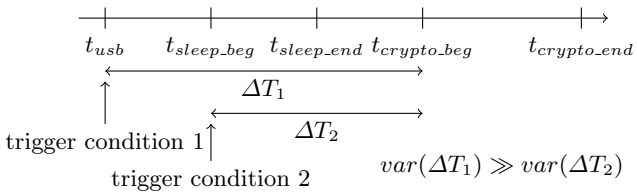

Fig. 2. triggering sequence: first USB pattern (less false positives); second sleep state (less variance between adc triggering and algorithm start).

3.2 Energy variations - Leakage frequencies

Zooming on EM signal of figure 3 (a), there are time locations when the AC absolute magnitude decreases, characteristic of signal energy variations. In signal processing, the energy of a signal is given by the integration over time of its squared absolute values: $E_s = \int_{-\infty}^{\infty} |x(t)|^2 \mathrm{d}t$.

To locally evaluate the signal energy around a point in time, the integral is computed on a window centered on that point. It is equivalent to convoluting a square window centered on that point, and summing the values of the convoluted signal. Applied to all signal points, the output signal is a low-pass filtered signal of the original one. This filter has some drawbacks. The sharp edges of the square window involve important ripples in the frequency domain. Alternatively, we used a FIR (Finite Impulse Response) filter weighted with a Hamming window. The cutting frequency was taken at 50kHz, a value giving a good SNR ratio. Then a high-pass filter was applied to the signal. As a consequence, the signal was band-pass filtered around the frequency band of compromission [2,15].

High energy variations are visible on the filtered signal (figure 3 (b)). They happen during signature computation as we show later. Energy variations during the computation of sensitive values has long been of interest in the field of computer security. In the particular case of ECDSA, being able to differentiate the leakage patterns of the doubling and addition operations is a big security threat, because the flow of operations is directly linked to secret data.

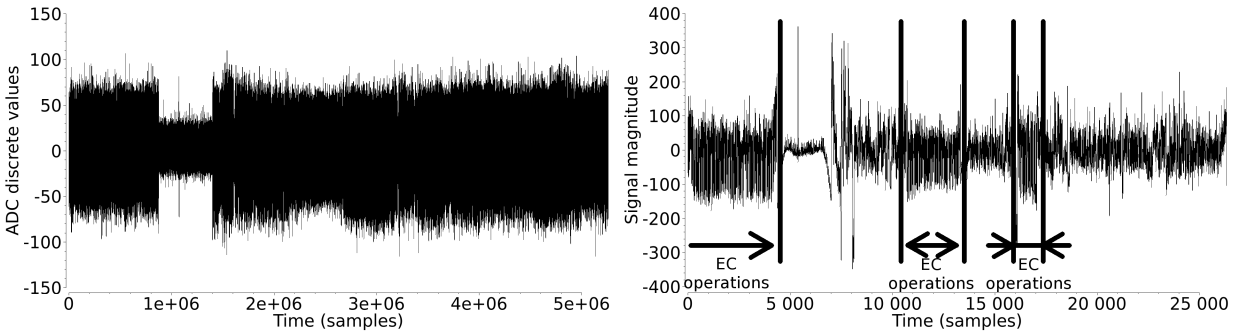

Fig. 3. (a) Measured signal: Noisy curve, visible period when processor is idle (*Qualcomm MSM 8225*) – (b) After signal filtering: higher energy variations during scalar multiplication (less time samples because of subsampling - *Qualcomm MSM 8225*)

Distinguishing EC operations patterns. The evaluation of a white box scalar multiplication, with a known scalar, and Bouncy Castle's doubling and addition implementations, allows the discrimination of the two operations patterns (figure 4 (a)). Each operation is characterized by a specific set of low power peaks, defined by the number of peaks and the timing intervals between successive peaks.

If the number of operations to extract is low enough, a manual observation is possible, as is the case for the cryptanalysis presented in section 4 where a few hundreds of operations are needed.

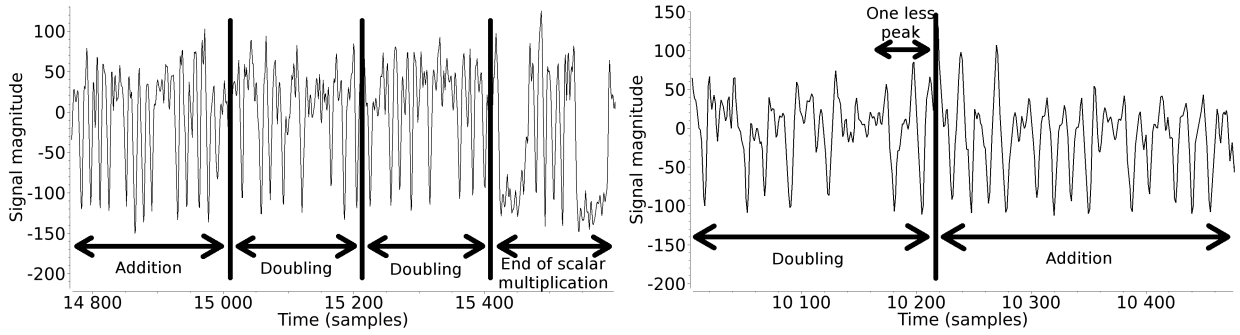

Fig. 4. (a) doubling and addition leakage patterns (*Qualcomm MSM 8225*) – (b) pattern of a doubling operation preceding an addition operation (*Qualcomm MSM 8225*)

The regularity of the peaks is compared to the code of both the doubling and the addition operations. Considering the doubling implementation (algo. 1), the number of multiplications is the same as the number of peaks in the doubling leakage pattern. The number of additions and subtractions between successive multiplications ($\{3, 0, 1, 3, 6, 1, (1)\}$), which is plotted on figure 5 (b), evolves similarly to the timing intervals on figure 4 (b).

An interesting part of algorithm 1 is the block condition in line 14, which is executed if the operation is followed by another doubling. If it is followed by an addition, the block is not executed, and so, there is one less modular multiplication at the end of the function. This is clearly visible on the doubling pattern preceding the addition on figure 6 (a). This explains the parentheses surrounding the last value of the list.

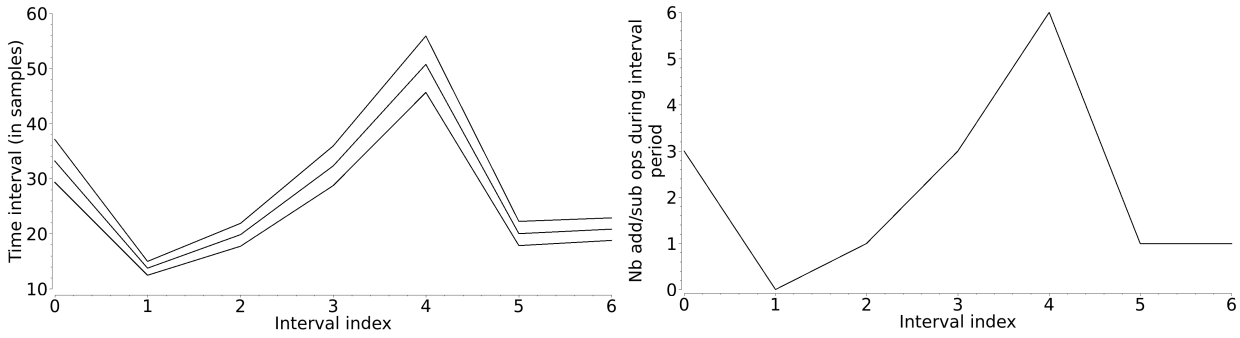

Fig. 5. (a) Mean and standard deviation of doubling operation time intervals – (b) Number of basic operations between multiplications in double BC source code (algo. 1)

The addition sums a precomputed point to an intermediate one during exponentiation. The precomputed points have their coordinate Z set to one. It leads to computation simplifications since the field operations involving this value, its square, or its cubic value, do not need to be computed. If we consider the point P1 to be precomputed in addition algorithm 2, the conditional blocks executed if the bit length of Z1 is different from one, are never computed. As a consequence, the number of additions and subtractions between successive multiplications gives the list $\{0, 0, 0, 2, 0, 0, 0, 4, 0, 1, 0, 0\}$,

plotted on figure 6 (b). It has the same look as the curve on figure 5 (a). The same conclusions may be drawn from the Android debugger *DDMS* as described in Appendix E.

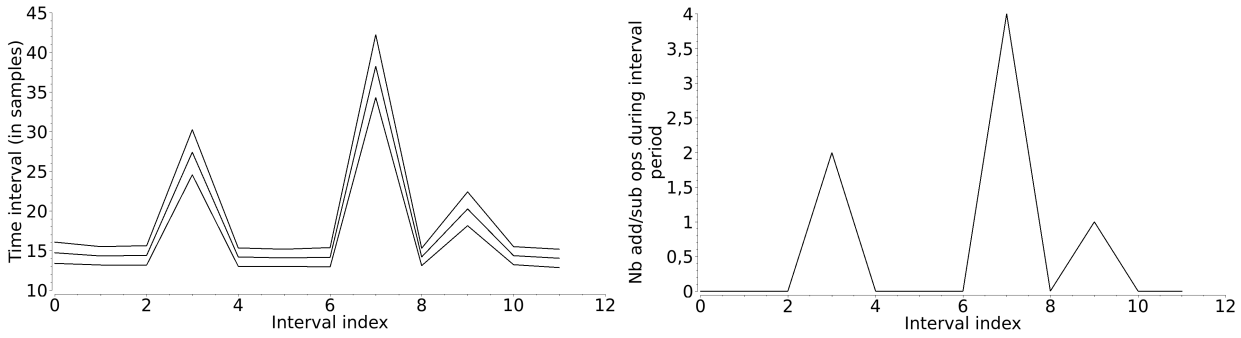

Fig. 6. (a) Mean and standard deviation of addition operation time intervals – (b) number of basic operations between multiplications in add BC source code (algo. 2)

Attacker's strength considerations. The *Qualcomm MSM 8225* processor, clocked at 1.2GHz, leaks in a frequency range which is under 50kHz. This relatively low frequency can be explained by leaking operations executing during multiple clock ticks. An analog-to-digital converter with a sampling frequency of a few hundreds of kilo-hertz, allows to mount the attack with low investment costs. In the paper, the measurements were obtained by decreasing our oscilloscope bandpass cutting frequency to the minimum available one (20MHz) and choosing a sampling frequency of 50MHz. It is small in comparison to the smartphone's CPU clock frequency.

Contrary to the works of Genkin *et al.* [19,18], our attack is not subject to system interruptions (fig. 3 (b)). In fact in their paper, the frequency contents of exponentiation vary with computed values. If the OS cuts the processing in different chunks, the frequency spectrum of exponentiation will be greatly affected. Consequently, the specificities of the inputs will not be discernible with their method.

A possible explanation for the leakages. Field multiplications are computed with the Java class *BigInteger*. These class functions ultimately bind to the native class *NativeBN* through the JNI. The native methods call binary code in shared library. Disassembling the library of interest, the machine code is executed in ARM mode during multiplication, contrarily to usual THUMB-2 mode for other instructions, e.g. addition instruction. Looking at the ARM reference manual for Thumb-2 [6], multiplication instructions are one of the few which have distinct features in ARM mode and in THUMB-2 mode. In particular, conditional flags can be modified in ARM mode, which is important for vectorial operations. It may explain this change of the CPU mode, and consequently the difference observed in the leakage. However, the impact on the leakage is difficult to establish.

One track that may be explored is the way integer pipelines are implemented. However, documentation is not always accessible. For example, the ARM Cortex-A8 architecture (which is not targeted in the paper) implements two ALUs, but only one implements a multiplier (see [5]). Consequently, depending on how the processor is able to fill both of the ALUs (e.g. because of data dependences or the number of successive multiplication in the program) may affect the amount of processing done at a given time. Similar design choices for the targeted processors may explain some leakage variations.

4 Lattice Attack on ECDSA

Monitoring EM radiation during EC scalar multiplication, it may be possible to recover the succession of doublings and additions. With Left-to-Right scalar multiplication (algorithm 3), this information is sufficient to recover the private key from a single signature. However, this approach does not work against Bouncy Castle, which implements the efficient "window NAF" algorithm. A side-channel attacker cannot distinguish which of several precomputed points is added at each iteration. On the other hand, the number of zeros between successive additions (i.e. the number of doublings minus one) *can* be recovered using Simple Power Analysis. In particular, the number of doublings following the last addition reveals the number of zeros in the least significant bit positions (because the LSB of a window is always 1). Using that information, one can mount a full key-recovery attack using well-known lattice-based techniques.

Indeed, in ECDSA and other Schnorr-like signature schemes, an attacker who obtains sufficiently many signatures for which he knowns the least significant or most significant few bits of the random nonces k can recover the private signing key. Recovering this key from the signatures and the known bits of the nonces reduces to an instance of Boneh and Venkatesan's hidden number problem (HNP). The best-known variant of this attack is due to Howgrave-Graham and Smart (and was later revisited and made more precise by Nguyen and Shparlinski), and uses lattice reduction to solve the underlying HNP instance. It is recalled in section 4.1 below. In particular, it yields a key-recovery attack against physical implementations of ECDSA signatures in which the side-channel leakage of scalar multiplication can be used to reveal the least or most significant bits of the nonce.

However, the side-channel attack does not typically apply to ECDSA signatures on Koblitz curves. The scalar multiplication on such curves is normally carried out using the τ-adic expansion of the nonce k. Therefore side-channel leakage can at best reveal the top or bottom bits of that τ-adic expansion, which do not determine the top or bottom bits of (the binary representation of) k itself.

In section 4.2, we describe how a similar attack can be mounted in the setting of Koblitz curves nonetheless. More precisely, we show that the top (or bottom) bits of the τ-adic expansion of the nonce can also be used to recover the signing key. The problem it reduces to, can be seen as a higher-dimensional generalization of HNP that can also be solved using lattice reduction.

4.1 ECDSA over prime fields

From previous section, we have shown that we are able to visualize the inner structure of NAF representation of the secret nonce k involved in the computation of an ECDSA signature. Formally if $k = \sum_i \alpha_i 2^i$ is such a NAF representation of the secret k, then one can determine the positions i for which the NAF digit α_i is valid, otherwise said, is not zero. Although the values of the digits α_i are unknown, this gives us a large amount of information. In particular, it is sufficient to exploit the known position of the last digit: let ℓ be the position of the last digit in the NAF representation of k, then we know that the last ℓ digits in the *binary* representation of k, are a one, followed by $d-1$ zeros.

Knowing the bits of the nonces, we can reduce the problem of recovering the secret key x to solving the HNP, which can be described as follows: given (t_i, u_i) pairs of integers such that

$$|xt_i - u_i|_q \leq q/2^{\ell+1},$$

where ℓ denotes the number of bits we recover, x denotes the hidden number we are looking for and $|\cdot|_q$ denotes the distance to $q\mathbb{Z}$, *i.e.* $|z|_q = \min_{a\in\mathbb{Z}} |z - aq|$. Such problem can be cast as a Closest Vector Problem (CVP) in a lattice and the LLL algorithm can be used to solve it in practice. We refer to appendix D and [35] for details. The main advantage of this technique is that the number of signatures required is usually very small, but it cannot be used when the number of known bits is too small. Indeed, in the case of 2 known bits for a 160-bit modulus, Liu and Nguyen used BKZ 2.0 to solve such lattice as the dimension becomes quite high for lattice algorithms [28]. Following the steps used in [23,34,8] we are able to perform the recovery of the signer's ECDSA secret key. In our case we chose the NIST P-256 elliptic curve. Using the method described in [8], we choose to solve the HNP problem using the Shortest Vector Problem (SVP) on some lattice. Therefore by building an adequate matrix and reducing it using the BKZ algorithm, we find a vector, one of whose coordinates reveals the secret key.

Experimental results. To estimate how many signatures we need to process the attack with a high probability of success, we first performed simulated signatures and solved the problem with a Sage (version 6.2) BKZ algorithm implementation. We want to use as much information as we can and we use the technique developed in [8] to this end. Usually, the lattice takes only signatures that have at least ℓ bits and remove the other ones. Here, we want to extract as much information as it is possible and so we put on the diagonal the number of bits we recover. As in [8], we made experiments by varying the minimum value z of the parameter ℓ of the signatures selected to join the computation of the attack. And then we discovered as a rule of thumb, that for a 256-bit secret key, and a probability of success being nearly 100%, the number of selected signatures should be above $\frac{200}{z}$, and therefore statistically, the total number of signatures to be processed should be above $\frac{200}{z}2^z$. As the complexity of the attack increases with the dimension of the matrix, we found that the best compromise was $z = 2$. Therefore, we processed approximately 500 signatures from which we selected only those for which ℓ was 2 or above, and they were 115 of such. As expected, the SVP attack gave us the secret key in less than five minutes on a common desktop.

4.2 New attack on Koblitz ECDSA

Consider a Koblitz curve E with a subgroup $\mathbb{G}$ of large prime order q, and let τ be the eigenvalue of the Frobenius endomorphism of E acting on $\mathbb{G}$, seen as a quadratic integer (depending on E, we have $\tau = \frac{\pm 1+\sqrt{-7}}{2}$). Suppose that we are given t ECDSA signatures (r_i, s_i) in $\mathbb{G}$, with random nonces k_i for which the top coefficients of some (signed) τ-adic expansion is known (the attack would work similarly for the bottom coefficients). Without loss of generality (up to the obvious affine transformation), we may assume that these known bits are all zero, so that the k_i's can be written in the form:

$$k_i = k_{i,0} + k_{i,1}\tau + \cdots + k_{i,\ell-1}\tau^{\ell-1} \in \mathbb{Z}[\tau]$$

where the coefficients $k_{i,j}$ belong to $\{-1, 0, 1\}$, and ℓ is some fixed integer length (the difference between the maximum length of the τ-adic expansions and the number of known zero nonce bits). Moreover, we can decompose k_i in the form $k_i = u_i + v_i\tau$ where u_i, v_i are the rational integers given by $v_i = (k_i - \overline{k_i})/\sqrt{-7}$ and $u_i = k_i - v_i\tau$. Due to the fact that $|\tau| = \sqrt{2}$ (which is crucial for our attack), it is easy to see that both u_i and v_i satisfy a bound of the form $O(\sqrt{2}^\ell)$, and in particular, there exists a constant $c > 0$ such that $u_i^2 + v_i^2 \leq c \cdot 2^\ell$ for all k_i. A discussion of how to estimate the constant c in cases of interest is provided in appendix G.

Now for each signature (r_i, s_i), if we denote by h_i the hash value of the corresponding message, the ECDSA verification equation ensures that $k_i s_i \equiv h_i + x r_i \bmod q$, which we can rewrite as

$$x \equiv A_i u_i + \tau A_i v_i + B_i \mod q \tag{3}$$

in terms of the known constants $A_i = s_i/h_i$ and $B_i = -r_i/h_i$ in $\mathbb{Z}/q\mathbb{Z}$. Note that, in view of the bound on $u_i^2 + v_i^2$, (u_i, v_i) is contained in a disc of radius $\sqrt{c \cdot 2^\ell}$ centered at the origin, and the right-hand side of (3) can thus take $\big(1 + o(1)\big)\pi c \cdot 2^\ell$ distinct values at most. As a result, as soon as $\ell < \log_2(q/\pi c)$, each such equation should reveal some information about x, and we should be able to recover x when t is large enough, much in the same way as in the HNP setting.

We show how this can be done with lattice reduction (at least heuristically, although in principle the rigorous approach of Nguyen–Shparlinski can be extended to this setting as well). Let the vector $\boldsymbol{u} = (u_1, \ldots, u_t, v_1, \ldots, v_t, w) \in \mathbb{Z}^{2t+1}$, where w is chosen as $\lfloor \sqrt{c \cdot 2^{\ell-1}} \rfloor$. Since $\|\boldsymbol{u}\| \leq \sqrt{t \cdot c \cdot 2^\ell + w^2} \leq \sqrt{c(t+1/2)} \cdot 2^{\ell/2}$, its norm is bounded. Equation (3) can be rewritten in vector form as:

$$x \equiv \langle \boldsymbol{A}_i, \boldsymbol{u} \rangle \mod q$$

where $\boldsymbol{A}_i = (0, \ldots, 0, A_i, 0, \ldots, 0, \tau A_i, 0, \ldots, 0, B_i/w) \bmod q \in \mathbb{Z}^{2t+1}$ has three nonzero components in positions i, $t+i$ and $2t+1$. In particular, $\boldsymbol{u}$ is orthogonal modulo q to each of the vectors $\boldsymbol{A}_1 - \boldsymbol{A}_2, \boldsymbol{A}_2 - \boldsymbol{A}_3, \ldots, \boldsymbol{A}_{t-1} - \boldsymbol{A}_t$ and it is short. We can therefore hope to recover it using lattice reduction.

More precisely, consider the lattice $L \subset \mathbb{Z}^{2t+1}$ of vectors that are orthogonal modulo q to each $\boldsymbol{A}_i - \boldsymbol{A}_{i+1}$, $i = 1, \ldots, t-1$, and whose last component is a multiple of w. L is the kernel of the obvious linear map $\mathbb{Z}^{2t+1} \to \mathbb{Z}/w\mathbb{Z} \times (\mathbb{Z}/q\mathbb{Z})^{t-1}$, and that map is surjective with overwhelming probability (since the vectors $\boldsymbol{A}_i$ themselves are linearly independent modulo q with overwhelming probability on the choice of the randomness in signature generation). Therefore, L is full rank and its volume is given by $\mathrm{vol}(L) = \#(\mathbb{Z}^{2t+1}/L) = \#\mathbb{Z}/w\mathbb{Z} \times (\mathbb{Z}/q\mathbb{Z})^{t-1} = wq^{t-1}$. If the vector $\boldsymbol{u} \in L$ is significantly shorter than the shortest vector length predicted by the Gaussian heuristic (namely $\sqrt{\frac{2t+1}{2\pi e}} \cdot \mathrm{vol}(L)^{1/(2t+1)}$), we should be able to recover $\boldsymbol{u}$ as the shortest vector in L (up to sign) using lattice reduction. This condition can be written as:

$$\sqrt{c(t+1/2)} \cdot 2^{\ell/2} \ll \sqrt{\frac{2t+1}{2\pi e}} \cdot \big(wq^{t-1}\big)^{1/(2t+1)}$$

or equivalently:

$$\ell \lesssim \log_2(q/c\pi e) - \frac{1}{t} \cdot \log_2\left(q\sqrt{2\pi e}\right)$$

which means that recovery is possible for t large enough when $\ell \lesssim \log_2(q/c\pi e)$ (which is quite close to the "information theoretic" bound mentioned above!), and in that case, the condition on t for recovery becomes:

$$t \gtrsim \frac{\log_2\left(q\sqrt{2\pi e}\right)}{\log_2(q/c\pi e) - \ell}. \tag{4}$$

We find that this condition is well-verified in practice, and once $\boldsymbol{u}$ is recovered, it is clearly straightforward to find the signing key x.

Finally, we mention that, to obtain a short basis of L in practice, we use standard orthogonal lattice techniques: we apply lattice reduction to the lattice generated by the rows of the matrix of

dimension $3t$ written by blocks as:

$$\begin{pmatrix} \kappa q & & 0 & \mathbf{0} \\ & \ddots & & \vdots \\ 0 & & \kappa q & \mathbf{0} \\ \kappa(\boldsymbol{A}_1 - \boldsymbol{A}_2) & \cdots & \kappa(\boldsymbol{A}_{t-1} - \boldsymbol{A}_t) & \boldsymbol{I} \end{pmatrix} \boldsymbol{W}$$

where the $\boldsymbol{A}_i$'s are column vectors, $\boldsymbol{I}$ is the identity matrix of dimension $2t+1$, κ is a suitably large constant, and $\boldsymbol{W}$ is the diagonal matrix $\mathrm{diag}(1, \ldots, 1, w)$ to account for the divisiblity condition on the last coefficient of vectors in L.

Experimental results. We implemented our attack in Sage using BKZ-25 lattice reduction, and tested it against the NIST K-163 Koblitz curve, which has a group order of 162 bits, with random unsigned Koblitz expansions. Experimental results are collected in Table 1. As can be seen from that table, the condition on the number t of required signatures is very consistent with (4) with $c \approx 0.30$ (as discussed in appendix G). It is easy to attack up to 6 bits of bias.

Bits of bias ($\log_2 q - \ell$)	9	8	7	6
Predicted t (Eq. (4))	22	25	30	36
Experimental t	21	25	31	39
Lattice dimension	63	75	93	117
CPU time (s)	2.4	4.7	17	102

Table 1. Implementation of our new attack against Koblitz curve K-163, using Sage's BKZ-25, run on single core of a Core i5-3570 CPU at 3.4 GHz.

Practical SCA. We show on fig. 7 (a) and 7 (b) that the Frobenius operation is distinguishable on *Qualcomm MSM 7225*. On fig. 7 (a), there are five Frobenius in the first succession of operations and four in the two others. Comparatively, there is one addition of points between each succession of Frobenius. The ratio of timing execution between addition and doubling is worse on prime field (see fig. 4 (a)). The Frobenius on Koblitz curves is implemented with pre-computed tables in *Bouncy Castle 1.50*. Thus, the leakage observed is different from the arithmetic implementations observed on prime field. The twofold repetition of pattern leakages in each Frobenius method is linked to the affine coordinate representation of elliptic curve points (algorithm 5).

5 Use Case: Bitcoin Wallet

We present a significant use case, namely the Bitcoin crypto-currency [32,10], where our cryptanalysis is of practical interest. A Bitcoin address is associated with an ECDSA key pair over the prime field elliptic curve Secp256k1[5] The knowledge of the private key allows to spend the money stored at that address, and in the case of Bitcoin clients on Android smartphones, that key may be recoverable using an attack of the form considered in this paper. Indeed, Android wallet apps are typically lightweight clients using a so-called Simplified Payment Verification (SPV) mode, and usually based on *bitcoinj*, a Java implementation of that lightweight mode. The core cryptography of the library

[5] The proceedings version of this paper incorrectly describes that curve as a Koblitz curve, which it is not. Our prime field attack applies directly nonetheless, or with a minor tweak in the case of implementations using the special GLV endomorphism that that curve is endowed with. See also [4].

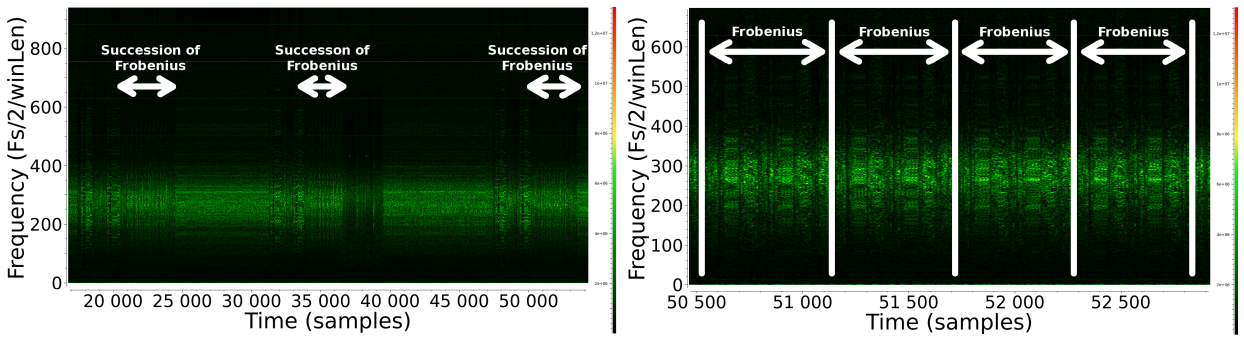

Fig. 7. (a) Succession of Frobenius and one addition between them (STFT, window length = 16000pts, Hamming window, *Qualcomm MSM 7225*) – (b) Zoom on a succession of four Frobenius operations (STFT, window length = 16000pts, Hamming window, *Qualcomm MSM 7225*)

relies on Bouncy Castle. Thus, the practical leakages observed in this paper may raise concerns for many Bitcoin users.

A concrete attack scenario could involve a malicious NFC reader at a shop where the Victim goes a few dozen times and pays with Bitcoin stored in its smartphone. This reader could improve our lab synchronization through legitimate contactless channel. In addition, the reader would contain a hidden EM probe, thus monitoring a signature each time the Victim comes to the shop. Such an attack scenario is still theoretical but the difficulty to catch the Attacker after theft evidence may motivate malevolent actors. Therefore, in order for crypto-currencies to become a sound technology for smartphone payment, we recommend that their implementations should integrate side-channel countermeasures as proposed e.g. in appendix H.

References

1. Aboulkassimi, D., Agoyan, M., Freund, L., Fournier, J., Robisson, B., and Tria, A. Electromagnetic analysis (EMA) of software AES on Java mobile phones. In *WIFS* (2011), IEEE, pp. 1–6.
2. Agrawal, D., Archambeault, B., Rao, J. R., and Rohatgi, P. The EM side-channel(s). In *CHES* (2002), B. S. K. Jr., Ç. K. Koç, and C. Paar, Eds., vol. 2523 of *LNCS*, Springer, pp. 29–45.
3. Aranha, D. F., Faz-Hernández, A., López, J., and Rodríguez-Henríquez, F. Faster implementation of scalar multiplication on Koblitz curves. In *LATINCRYPT* (2012), A. Hevia and G. Neven, Eds., vol. 7533 of *LNCS*, Springer, pp. 177–193.
4. Aranha, D. F., Fouque, P., Gérard, B., Kammerer, J., Tibouchi, M., and Zapalowicz, J. GLV/GLS decomposition, power analysis, and attacks on ECDSA signatures with single-bit nonce bias. In *ASIACRYPT* (2014), P. Sarkar and T. Iwata, Eds., vol. 8873 of *LNCS*, Springer, pp. 262–281.
5. ARM. The ARM architecture — with a focus on v7A and Cortex-A8. Presentation support.
6. ARM. *ARM Architecture Reference Manual — Thumb-2 supplement.*
7. Balasch, J., Gierlichs, B., Reparaz, O., and Verbauwhede, I. DPA, bitslicing and masking at 1 GHz. In *CHES* (2015), T. Güneysu and H. Handschuh, Eds., vol. 9293 of *LNCS*, Springer, pp. 599–619.
8. Benger, N., van de Pol, J., Smart, N. P., and Yarom, Y. "Ooh Aah... Just a Little Bit" : A Small Amount of Side Channel Can Go a Long Way. In *CHES* (2014).
9. Boneh, D., and Venkatesan, R. Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes. In *CRYPTO* (1996), N. Koblitz, Ed., vol. 1109 of *LNCS*, Springer, pp. 129–142.
10. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J. A., and Felten, E. W. SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. In *Security and Privacy* (2015), IEEE Computer Society, pp. 104–121.
11. Chevallier-Mames, B., Ciet, M., and Joye, M. Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. *IEEE Trans. Computers 53*, 6 (2004).

12. Cohen, H., and Frey, G. *Handbook of elliptic and hyperelliptic curve cryptography*. CRC Press, 2005.
13. Cohen, H., Miyaji, A., and Ono, T. Efficient elliptic curve exponentiation using mixed coordinates. In *ASIACRYPT* (1998), K. Ohta and D. Pei, Eds., vol. 1514 of *LNCS*, Springer, pp. 51–65.
14. Coron, J.-S. Resistance against differential power analysis for elliptic curve cryptosystems. In *CHES* (1999), Ç. K. Koç and C. Paar, Eds., vol. 1717 of *LNCS*, Springer, pp. 292–302.
15. Gebotys, C. H., Ho, S., and Tiu, C. C. EM analysis of rijndael and ECC on a wireless java-based PDA. In *CHES* (2005), J. R. Rao and B. Sunar, Eds., vol. 3659 of *LNCS*, Springer, pp. 250–264.
16. Genkin, D., Pachmanov, L., Pipman, I., and Tromer, E. Stealing keys from PCs using a radio: Cheap electromagnetic attacks on windowed exponentiation. In *CHES* (2015), T. Güneysu and H. Handschuh, Eds., vol. 9293 of *LNCS*, Springer, pp. 207–228.
17. Genkin, D., Pachmanov, L., Pipman, I., Tromer, E., and Yarom, Y. ECDSA key extraction from mobile devices via nonintrusive electromagnetic attacks. IACR Cryptology ePrint Archive, 2016. http://eprint.iacr.org/.
18. Genkin, D., Pipman, I., and Tromer, E. Get your hands off my laptop: Physical side-channel key-extraction attacks on PCs. In *CHES* (2014), L. Batina and M. Robshaw, Eds., vol. 8731 of *LNCS*, Springer, pp. 242–260.
19. Genkin, D., Shamir, A., and Tromer, E. RSA key extraction via low-bandwidth acoustic cryptanalysis. In *CRYPTO* (2014), J. A. Garay and R. Gennaro, Eds., vol. 8616 of *LNCS*, Springer, pp. 444–461.
20. Giraud, C., and Verneuil, V. Atomicity improvement for elliptic curve scalar multiplication. In *CARDIS* (2010), D. Gollmann, J. Lanet, and J. Iguchi-Cartigny, Eds., vol. 6035 of *LNCS*, Springer, pp. 80–101.
21. Hankerson, D., Vanstone, S., and Menezes, A. *Guide to Elliptic Curve Cryptography*. Springer Professional Computing. Springer, 2004.
22. Hasan, M. A. Power analysis attacks and algorithmic approaches to their countermeasures for Koblitz curve cryptosystems. *IEEE Trans. Computers 50*, 10 (2001).
23. Howgrave-Graham, N., and Smart, N. P. Lattice attacks on digital signature schemes. *Des. Codes Cryptography 23*, 3 (2001).
24. Kenworthy, G., and Rohatgi, P. Mobile device security: The case for side-channel resistance. Tech. rep., Cryptography Research Inc, 2012.
25. Koblitz, N. CM-curves with good cryptographic properties. In *CRYPTO* (1991), J. Feigenbaum, Ed., vol. 576 of *LNCS*, Springer, pp. 279–287.
26. Kocher, P. C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO* (1996), N. Koblitz, Ed., vol. 1109 of *LNCS*, Springer, pp. 104–113.
27. Kocher, P. C., Jaffe, J., and Jun, B. Differential power analysis. In *CRYPTO* (1999), M. J. Wiener, Ed., vol. 1666 of *LNCS*, Springer, pp. 388–397.
28. Liu, M., and Nguyen, P. Q. Solving BDD by enumeration: An update. In *CT-RSA* (2013), E. Dawson, Ed., vol. 7779 of *LNCS*, Springer, pp. 293–309.
29. Longa, P. Accelerating the scalar multiplication on elliptic curve cryptosystems over prime fields. Master's thesis, School of Information and Engineering, University of Ottawa, Canada, 2007.
30. Longo, J., Mulder, E. D., Page, D., and Tunstall, M. SoC it to EM: elecromagnetic side-channel attacks on a complex system-on-chip. In *CHES* (2015), T. Güneysu and H. Handschuh, Eds., vol. 9293 of *LNCS*, Springer, pp. 620–640.
31. López, J., and Dahab, R. Improved algorithms for elliptic curve arithmetic in GF(2^n). In *SAC* (1998), S. E. Tavares and H. Meijer, Eds., vol. 1556 of *LNCS*, Springer, pp. 201–212.
32. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system, 2009.
33. Nakano, Y., Souissi, Y., Nguyen, R., Sauvage, L., Danger, J.-L., Guilley, S., Kiyomoto, S., and Miyake, Y. A pre-processing composition for secret key recovery on android smartphone. In *WISTP* (2014), D. Naccache and D. Sauveron, Eds., vol. 8501 of *LNCS*, Springer, pp. 76–91.
34. Nguyen, P. Q., and Shparlinski, I. The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Des. Codes Cryptography 30*, 2 (2003).
35. Nguyen, P. Q., and Tibouchi, M. Lattice-based fault attacks on signatures. In *Fault Analysis in Cryptography*, M. Joye and M. Tunstall, Eds., Information Security and Cryptography. Springer, 2012, pp. 201–220.
36. NIST. FIPS PUB 186–3: Digital Signature Standard. Tech. rep., NIST, July 2013.
37. Okeya, K., Takagi, T., and Vuillaume, C. Efficient representations on Koblitz curves with resistance to side channel attacks. In *ACISP* (2005), C. Boyd and J. M. G. Nieto, Eds., vol. 3574 of *LNCS*, Springer, pp. 218–229.
38. Oliveira, T., López, J., Aranha, D. F., and Rodríguez-Henríquez, F. Lambda coordinates for binary elliptic curves. In *CHES* (2013), G. Bertoni and J. Coron, Eds., vol. 8086 of *LNCS*, Springer, pp. 311–330.
39. Rondepierre, F. Revisiting atomic patterns for scalar multiplications on elliptic curves. In *CARDIS* (2013), A. Francillon and P. Rohatgi, Eds., vol. 8419 of *LNCS*, Springer, pp. 171–186.

40. Solinas, J. A. Efficient arithmetic on Koblitz curves. *Des. Codes Cryptography 19*, 2/3 (2000).
41. Zajic, A., and Prvulovic, M. Experimental demonstration of electromagnetic information leakage from modern processor-memory systems. *IEEE Transactions on Electromagnetic Compatibility 56*, 4 (2014).
42. Zenger, C., Paar, C., Lemke-Rust, K., Kasper, T., and Oswald, D. SEMA of RSA on a smartphone. Tech. rep., Ruhr–Universität Bochum, October 2011.

A Related work

Side-channel attacks on more powerful devices than smartcards have been investigated since 2005 and the introductory work published by Gebotys *et al.* on a PDA [15]. In 2011, a DPA attack was lead against AES on Java EE mobile phone [1]. In 2012, Kenworthy and Rohatgi demonstrated in [24] that side-channel attack on smartphone is a real threat: they show that using a single RSA, ECC EM trace, it is possible to recover the secret key. They also show that the pattern of AES is easily seen, but they do not propose an attack on AES. A similar bachelor work has been performed by Zenger *et al.* in [42] in case of the RSA cryptosystem.

In 2014, three major papers have been published about side-channel vulnerabilities on PCs. The first one [41] studies the propagation of information from PCs through electromagnetic emanations. The work evaluates the propagation of periodic processing through covert channels. The article does not investigate the possibilities to lead a cryptanalysis. Nevertheless, the study of exponentiation pseudo-periodicity under their experimental framework would be interesting. The two other papers [19,18] attack double-and-add always RSAs. Their cryptanalyses are based on chosen ciphertexts decyphering, an adaptive one [19,18] (as many numbers of ciphertexts needed as the number of bits of the secret), and a non-adaptive one [18] (only one ciphertext needed). Both of the attacks exploit a performance trick: a high number of zeros in the data being squared leads to a decrease of the number of operations computed. This trend can be discernable through diverse side-channels: acoustic [19], potential of PC chassis [18], IO ground [18], EM emanations [18], supply power [18]. The authors improved their attack to sliding-window and fixed-window exponentiations [16]: using chosen ciphertexts, special intermediate values make frequency leakages vary over time, revealing the index of pre-computed points accessed. The attack succeeds with as few as 8 decryptions, but in comparison, our attack does not rely on chosen inputs. In [33], Nakano *et al.* describe a Simple Electromagnetic Emanation Analysis on a smartphone in order to demonstrate their ability to recover the secret keys from a single execution trace using an information signal processing technique combining time and frequency analysis.

In 2015, the DPA feasibility against AES running on giga-hertz CPU was investigated. In [30], Longo *et al.* target sequential large tables (OpenSSL AES in the ARM ISA) and SIMD (in the NEON ISA) software implementations and a cryptographic co-processor, all of the three implementations running on a 1GHz ARM Cortex-A9 processor. Their work highlights the need for good leakage detection tools. Balasch *et al.* [7] investigate a bitsliced implementation of the AES and succeed their attack using clear leakages. In both [30] and [7], the authors study EM radiations from capacitors that partly handle the power supply of the SOC. [7] accurately calls the measuring method *contactless power measurement*. In fact, due to the macroscopic size of our probe (a diameter of a few millimeters), and leakages usually sharpen at some edges of the SOC, we think that the radiations we measure are in fact radiated from power circuitry inside the SOC.

B Android Smartphone Architecture

All the experiments have been done on smartphones running the Android system upon ARM processors.

ARM processors. They are predominant in the smartphone market. It is mainly due to their power consumption efficiency. They are built upon a RISC architecture and their memory management is a register-register one.

Two processors have been investigated, both designed by Qualcomm. The first one is a *Qualcomm MSM 8225*, designed upon an *ARM Cortex-A5* architecture with the ARMv7 ISA. It is a dual-core processor, developed under a 45nm technology, and its frequency goes up to 1.2GHz. The second one is a *Qualcomm MSM 7225*. It is based on an *ARM11* architecture (*ARMv6* instruction set), built upon a 65nm process. Its maximal frequency is 511MHz. Both of the CPUs studied in the article are 32-bit ones.

ARM processors implement multiple instruction sets. The main one is the *ARM* instruction set, which instructions are 32-bit long. To improve density of code in embedded devices, *Thumb*, a 16-bit instruction set, was introduced in 1994. It provides a subset of *ARM* ISA for common use instructions. *Thumb-2*, an extension of *Thumb* with 32-bit instuctions, was developed to allow almost all of code to be developed in *Thumb* mode. Actually, *ARM* and *Thumb* instruction sets do not provide independent implementations for similar instructions. In decoding stage, Thumb instructions are mapped to equivalent *ARM* instructions. The presence of these two instruction sets is of interest in our paper, since as for as we know, the leakages observed happen when executing either in *ARM* mode or in *Thumb* mode.

Android OS. Android is the most popular operating system in the smartphone field, with a market share of about 80% in 2015.

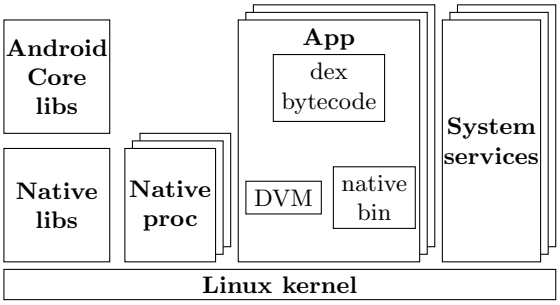

Fig. 8. Android Architecture

An overview of Android architecture is represented in fig 8. The OS is built upon a Linux kernel, even if it cannot be considered as a Linux distribution, in part because of its application management. Applications are run in an applicative virtual machine, the *Dalvik VM*. App creation is done by forking a native process called *Zygote*. This allows efficient startup and memory optimizations, in particular because of the Linux kernel copy-on-write (COW) mechanism. Memory is shared between original and new processes until the latter one writes on it, leading to dynamic content copy.

Dalvik VM bytecode is obtained from the JAVA language. However, it is different from the one compiled for the *Java Virtual Machine (JVM)*, to meet embedded hardware constraints. In fact,

for a given Java code, the memory footprint of the Dalvik bytecode is generally smaller than the JVM one. Nevertheless, its computing efficiency is often worse, leading to improvements over time from Google. Initially being an interpreter, Dalvik evolved to implement a Just-in-time compiler (JIT). The parts of the code which are foreseen to be executed many times are compiled at run-time. There is a loss of computing resources during this dynamic compilation, but execution from native code then involves better performances. In the paper, we investigate implementations running on Dalvik VM either with JIT or without it. For efficiency reasons, some parts of the application can be developed in C language. They are natively compiled at development time and linked through the *Java Native Interface (JNI) API*.

Cryptographic development is identical for Dalvik VM and JVM. The Java standard provides an API, allowing cryptographic use independently of primitive implementations. It makes software compliant to any library conforming with the API, and thus enables the use of different implementations. On the cryptographic developers side, a library compliant to the API is easily usable by any Java developer. It can also be upgraded with backward compatibility restrictions.

On Android, the standard cryptographic provider is *Bouncy Castle*. The version evaluated in the paper is *1.50*.

C Bouncy Castle version 1.50

Prime field EC implementations Arithmetic operations over prime field $GF(p)$ are the (modular) addition, subtraction, multiplication and inversion of elements. Using these basic field operations, different algorithms exist to add and double elliptic curve points. They depend on optimization trade-off between memory, timing execution or processing power.

Modified Jacobian coordinates and Operations. The point $P = (x, y)$ defined in affine coordiantes, has Jacobian coordiantes (X, Y, Z), where $x = X/Z$ and $y = Y/Z$ and $Z \neq 0$. All the Jacobian coordinates defined as (r^2X, r^3Y, rZ^2) with $r \in \mathbb{K}^*$ represent the same point $(X : Y : Z)$, named a representative point [21], and they define equivalent classes. Doubling and addition of points defined with Jacobian coordinates can be computed without modular inversion.

In *Bouncy Castle 1.50*, modified Jacobian coordinates are used as proposed in [13]. The quadruple (X, Y, Z, aZ^4) represents the affine point $(X/Z^2, Y/Z^3)$. Doubling is implemented with 4 multiplications and 5 squares (Algorithm 1), and addition with 13 multiplications and 6 squares (Algorithm 2).

Scalar multiplication implementations. The most vulnerable part of ECDSA is scalar multiplication. Several algorithms exist to compute the point $Q = k.P$. We detail two of them.

Left-to-right double and add is common. Each bit of the scalar k is processed successively as shown in pseudo-code algorithm 3. Distinguishing double and add operations through side-channel allows to find the secret scalar, consequently breaking ECDSA in one signature.

Window non-adjacent form (*window NAF* or *wNAF*) is the standard implementation in *Bouncy Castle 1.50*. It allows better performance than *left-to-right double and add* algorithm. Each addition is performed over a multi-bits subset of the scalar rather than over a unique bit, reducing the number of field operations. The binary representation of the scalar is transformed into a wNAF representation, which is a succession of values in the set $S = \{0, \pm 1, \pm 3, \ldots, \pm(2^{w-1} - 1)\}$.

Definition 1. *A non-adjacent form (NAF) of a positive integer k is an expression $k = \sum_{i=0}^{n-1} k_i 2^i$ where $k_i \in \{-1, 0, 1\}$, $k_{n-1} \neq 0$, and no two consecutive digits k_i are nonzero. The length of the NAF is n. The NAF of an integer k is denoted $NAF(k)$ or $(k_{n-1}, \ldots, k_0)_{NAF}$.*

Algorithm 1 Doubling implementation in basic operations over Modified Jacobian coordinates in Bouncy Castle library

Input: *Point* $P_1 = (X_1, Y_1, Z_1, W_1)$ and boolean W
Output: *Point* $P_3 = (X_3, Y_3, Z_3, W_3)$

function ModifiedJacobianDoubling(W, P_1)
  $X1sq \leftarrow X_1 * X_1$
  $M \leftarrow ((X_1sq + X_1sq) + X_1sq) + W_1$
  $Y_1sq \leftarrow Y_1 * Y_1$
  $T \leftarrow Y_1sq * Y_1sq$
  $temp \leftarrow X_1 + Y_1sq$
  $temp_1 \leftarrow ((temp * temp) - X_1sq) - T$
  $S \leftarrow temp_1 + temp_1$
  $X_3 \leftarrow (M * M) - (S + S)$
  $temp_2 \leftarrow T + T$
  $temp_3 \leftarrow temp_2 + temp_2$
  $\_8T \leftarrow temp_3 + temp_3$
  $Y_3 \leftarrow (M * (S - X_3)) - \_8T$
  if $W = true$ **then**
    $temp_4 \leftarrow \_8T * W_1$
    $W_3 \leftarrow temp_4 + temp_4$
  end if
  if $Z_1.bitLen = 1$ **then**
    $temp_5 \leftarrow Y_1$
  else
    $temp_5 \leftarrow Y_1 * Z_1$
  end if
  $Z_3 \leftarrow temp_5 + temp_5$
  return $ECPoint.Fp(X_3, Y_3, Z_3, W_3)$
end function

Koblitz Curve implementations **τ-adic expansion and windowing methods.** One trick that speeds up scalar multiplication is the use of the Frobenius morphism denoted by ϕ_2, which sends each point (x_1, y_1) to (x_1^2, y_1^2) and ∞ to itself. Not surprisingly, ϕ_2 is a morphism over the curve. Its characteristic polynomial is $X^2 - \mu X + 2$, where $\mu = (-1)^{1-a}$. Let τ be one of the two conjugate roots of this polynomial. Using the relation $2 = \mu\tau - \tau^2$, every integer η can be written as its τ-adic expansion: $\eta = \sum_{i=0}^{l-1} r_i\tau^i$ where $r_i \in \{0, 1\}$. In analogy with binary expansion, one can also define τ-adic non adjacent form: $\eta = \sum_{i=0}^{l-1} r_i\tau^i$ with the extra conditions $r_i \in \{0, \pm 1\}$ and $r_i r_{i+1} = 0$ for all i. It is also possible to generalize windowing methods to τ-adic expansion. For w a parameter greater than 1, the width-w τ-adic expansion in non-adjacent form is $\eta = \sum_{i=0}^{l-1} r_i\tau^i$ where

- each r_i is 0 or $\pm\alpha_u$ where $\alpha_u \equiv u(\mod \tau^w)$ for some odd $u \in [1, 2^{w-1} - 1]$
- $r_{l-1} \neq 0$
- among any w consecutive coefficients, at most one is nonzero.

In a straightforward manner we get from the τ-adic expansion of the scalar, the following multiplication algorithm.

D Lattice-based attack on ECDSA

Using the ℓ least significant bits of k (the attack also works with the most significant bits), we can write $k = 2^\ell(k \gg \ell) + \mathrm{lsb}_\ell(k) = 2^\ell b + \mathrm{lsb}_\ell(k)$ for some integer $b \geq 0$. We then get from $xr = sk - h \bmod q$:

$$xr \cdot 2^{-\ell}s^{-1} = b - h \cdot 2^{-\ell}s^{-1} + \mathrm{lsb}_\ell(k) \cdot 2^{-\ell} \bmod q.$$

Algorithm 2 Addition implementation in basic operations over Modfied Jacobian coordinates in Bouncy Castle library (simplified to useful conditional blocks)

Input: *Points* $P_1 = (X_1, Y_1, Z_1, W_1)$ and $P_2 = (X_2, Y_2, Z_2, W_2)$
Output: *Point* $P_3 = (X_3, Y_3, Z_3, W_3)$

1: **function** ModifiedJacobianAdd(P_1, P_2)
2: **if** $Z_1.bitLen \neq 1$ **then**
3: $Z_1sq \leftarrow Z_1 * Z_1$
4: $U_2 \leftarrow Z_1sq * X_2$
5: $Z_1cubed \leftarrow Z_1sq * Z_1$
6: $S_2 \leftarrow Z_1cubed * Y_2$
7: **end if**
8: **if** $Z_2.bitLen \neq 1$ **then**
9: $Z_2sq \leftarrow Z_2 * Z_2$
10: $U_1 \leftarrow Z_2sq * X_1$
11: $Z_2cu \leftarrow Z_2sq * Z_2$
12: $S_1 \leftarrow Z_2cu * Y_1$
13: **end if**
14: $H \leftarrow U_1 - U_2$
15: $R \leftarrow S_1 - S_2$
16: $Hsq \leftarrow H * H$
17: $G \leftarrow Hsq * H$
18: $V \leftarrow Hsq * U1$
19: $X_3 \leftarrow ((R * R) + G) - (V + V)$
20: $Y_3 \leftarrow ((V - X_3) * R) - (S_1 * G)$
21: $Z_3 \leftarrow H$
22: **if** $Z_1.bitLen \neq 1$ **then**
23: $Z_3 \leftarrow Z_3 * Z_1$
24: **end if**
25: **if** $Z_2.bitLen \neq 1$ **then**
26: $Z_3 \leftarrow Z_3 * Z_2$
27: **end if**
28: $W \leftarrow Z_3sq * Z_3sq$
29: $a_4 \leftarrow -(curve.A)$
30: $a_4neg \leftarrow -a_4$
31: **if** $a_4neg.bitLen < a_4.bitLen$ **then**
32: $W \leftarrow -(W * a_4neg)$
33: **else**
34: $W \leftarrow W * a_4$
35: **end if**
return ECPoint.Fp (X_3, Y_3, Z_3, W_3)
36: **end function**

Algorithm 3 Left-to-Right double and add wNAF algorithm

Input: scalar k in wNAF $k_0, \ldots, k_n$ and precomputed points $\{P, \pm[3]P, \pm[5]P, \ldots, \pm[2^w - 1]P\}$
Output: *Point* $Q = kP$

1: **function** ScalarMultiplication(k, P)
2: $Q = \infty$
3: **for** i from n downto 0 **do**
4: $Q = 2 \cdot Q$
5: **if** $k_i \neq 0$ **then** $Q = Q + [k_i]P$
6: **end if**
7: **end for**
8: **return** Q
9: **end function**

Algorithm 4 Left-to-Right double and add τ-wNAF

Input: scalar k in τ-wNAF $k_0, \ldots, k_n$ and precomputed points $\{P, \pm[\alpha_3]P, \pm[\alpha_5]P, \ldots, \pm[\alpha_{2^w-1}]P\}$
Output: *Point* $Q = kP$

```
1: function ScalarMultiplication(k, P)
2:     Q = ∞
3:     for i from n downto 0 do
4:         Q = φ_2(Q)
5:         if k_i ≠ 0 then Q = Q + [k_i]P
6:         end if
7:     end for
8:     return Q
9: end function
```

Algorithm 5 Point doubling implementation in basic operations over affine coordinates in *Bouncy Castle 1.50* library

Input: *Points* $P_1 = (X_1, Y_1)$
Output: *Point* $P_3 = (X_3, Y_3)$

```
1: function Tau(P_1)
2:     X_3 = X_1^2
3:     Y_3 = Y_1^2
4:     return P_3
5: end function
```

Algorithm 6 Point Addition implementation in basic operations over affine coordinates in *Bouncy Castle 1.50* library

Input: *Points* $P_1 = (X_1, Y_1)$
1: $P_2 = (X_2, Y_2)$
Output: *Point* $P_3 = (X_3, Y_3)$

```
2: function Tau(P_1, P2)
3:     if X_1 == X_2 then
4:         if Y_1 == Y_2 then
5:             return P_1.twice()
6:         end if
7:         return infinity
8:     end if
9:     sumX = X_1 + X_2
10:    L = (Y_1+Y_2)/sumX)
11:    X_3 = L^2 + L + sumX + curveA
12:    Y_3 = L * (X_1 + X_3) + X_3 + Y_1
13:    return P_3
14: end function
```

Now let t and u two values which can be computed from known or retrieved information, such as:

$$t = r2^{-\ell}s^{-1} \bmod q$$

and

$$u = -h \cdot 2^{-\ell}s^{-1} + \mathrm{lsb}_\ell(k)2^{-\ell} \bmod q.$$

The inequality $b < q/2^\ell$ can be expressed in terms of t and u as:

$$0 \leq xt - u \bmod q < q/2^\ell.$$

Therefore, if we denote by $|\cdot|_q$ the distance to $\mathbb{Z}/q\mathbb{Z}$, i.e. $|z|_q = \min_{a\in\mathbb{Z}} |z - aq|$, we have:

$$|xt - u - q/2^{\ell+1}|_q \leq q/2^{\ell+1},$$
$$|xt - v/2^{\ell+1}|_q \leq q/2^{\ell+1},$$

where v is the integer $2^{\ell+1}u + q$. Given a number of faulty signatures (r_i, s_i) of various messages, say d of them, the same method yields pairs of integers (t_i, v_i) such that

$$|xt_i - v_i/2^{\ell+1}|_q \leq q/2^{\ell+1}. \tag{5}$$

The goal is to recover x from this data. The problem is very similar to the hidden number problem considered by Boneh and Venkatesan in [9], and is approached by transforming it into a lattice closest vector problem.

More precisely, consider the $(d+1)$-dimensional lattice L spanned by the rows of the following matrix:

$$\begin{pmatrix} 2^{\ell+1}q & 0 & \cdots & 0 & 0 \\ 0 & 2^{\ell+1}q & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & 2^{\ell+1}q & 0 \\ 2^{\ell+1}t_1 & \cdots & \cdots & 2^{\ell+1}t_d & 1 \end{pmatrix}$$

Inequality (5) implies the existence of an integer c_i such that:

$$|2^{\ell+1}xt_i - v_i - 2^{\ell+1}c_iq| \leq q. \tag{6}$$

Now note that the row vector, called *hidden vector*,

$$\mathbf{c} = (2^{\ell+1}xt_1 + 2^{\ell+1}c_1q, \cdots, 2^{\ell+1}xt_d + 2^{\ell+1}c_dq, x)$$

belongs to L and $\mathbf{c}$ is very close to the row vector $\mathbf{v} = (v_1, \cdots, v_d, 0)$. Indeed, by (6), the distance from $\mathbf{c}$ to $\mathbf{v}$ is bounded as:

$$\|\mathbf{v} - \mathbf{c}\| \leq q\sqrt{d+1}.$$

We thus have a CVP to solve. In practice, we use an embedding technique to reduce CVP to SVP. This technique consists in computing the $(d+2)$-dimensional lattice L' spanned by the rows of the matrix

$$\begin{pmatrix} L & 0 \\ \mathbf{v} & 1 \end{pmatrix}$$

The row vector $(\mathbf{v} - \mathbf{c}, 1)$ is short, belongs to L' and we hope this is the shortest vector of L'. This assumption implies a condition on the required number of signatures depending on the parameter ℓ and the modulus. An estimate which makes it possible to recover the private key is:

$$d \gtrsim \frac{n}{\ell - \log_2 \sqrt{\pi e/2}}.$$

The above estimate is heuristic, but it is possible to give parameters for which attacks of this kind can be proved rigorously [34].

E Android debugger analysis

The links between the leakage patterns and the source code are further established with the analysis of a part of the scalar multiplication in the Android debugger *DDMS*. Selecting the arithmetic multiplication, the occurence of this function has the same regularity as the ones observed on the measurements 9.

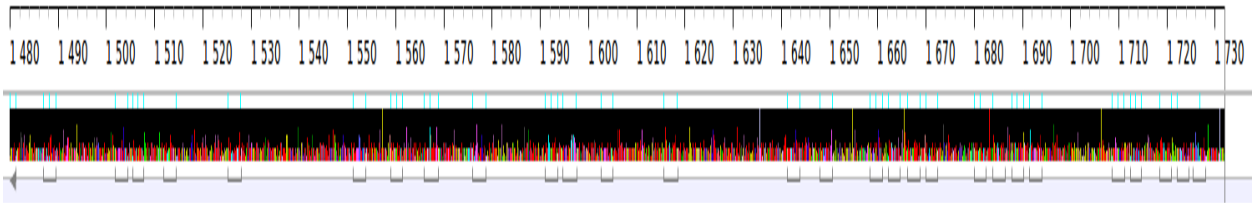

Fig. 9. Debugger trace during a subset of scalar multiplication: The processing of modular multiplication is underlined on the timeline.

F Measurement setup

Acquisition bench. The experimental setup, depicted in figure 10, is built upon three main devices: a computer to manage the acquisition process; an oscilloscope to convert analog signals into digital ones; the smartphone running the cryptographic algorithm. The computer and the smartphone communicate cryptographic data (messages, signatures) and a part of the triggering signal via USB. An electrical probe, plugged on the oscilloscope, measures this triggering signal on wire *D+*. The oscilloscope measures the magnetic field radiated by the smartphone processor via a magnetic probe, which is a loop of wire. This signal, after being digitized, is sent to the computer through a local network, where it can be observed and processed.

Communication with the smartphone. The communication is based over a client-server architecture, the smartphone being the server: see figure 11. The communication protocol is TCP/IP. Android Debug Bridge (ADB) links one of the PC ports to one of the smartphone ports. This action is called *ADB forwarding*. The messages arriving on one of the two ports, are sent to the other one through USB channel. Three main pieces of software are necessary to establish a communication between a PC software (client) and an Android device or an emulator application (server): the *ADB client*, the *ADB server* and the *ADB Deamon (ADBD)*.

ADB daemon runs on the Android device. It catches and interprets the messages arriving on the USB port, and makes the necessary operations in consequence. In our case, it receives messages coming from the PC TCP port (linked via ADB), and redirects them to the TCP server port (linked via ADB). On the computer side, *ADB server* manages the USB communication with the smartphone (TCP ports if the device is emulated on the computer). To establish a link with the *ADB server*, an *ADB client* instance must be created. It is linked to the *ADB server* through TCP port *5037* on localhost. *ADB server* enables the binding of *ADB server* to a local TCP socket. Then the client connects to that socket and commands are sent to the smartphone in a shell command style.

At the physical level, the bits packets are transmitted through the *Non-return to Zero Inverted (NRZI)* protocol. The state of the data signal changes when a bit at 0 is transmitted and remains at

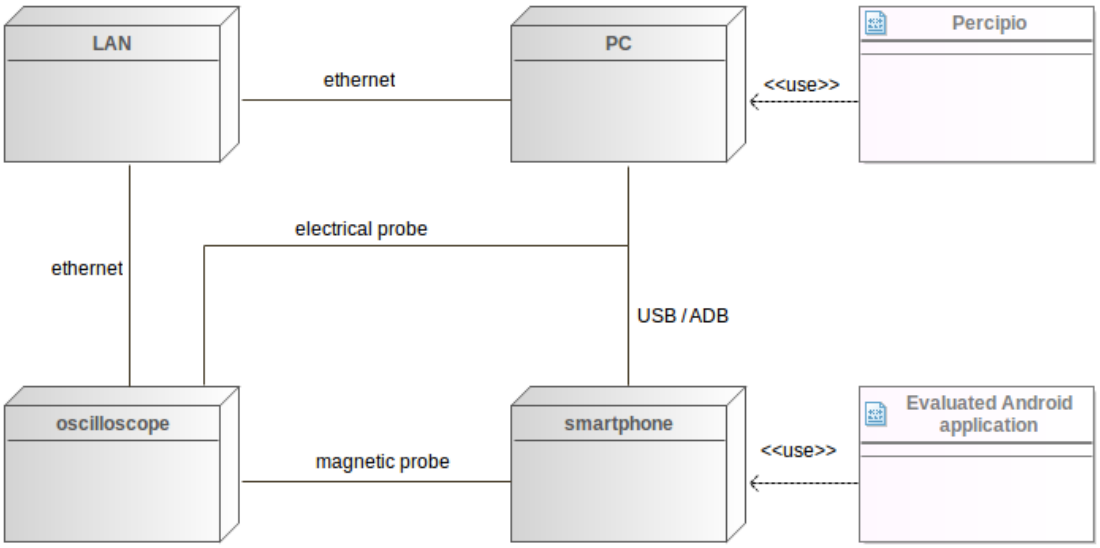

Fig. 10. Measurement bench architecture - Percipio is the name of the control software that we developed for our experiments.

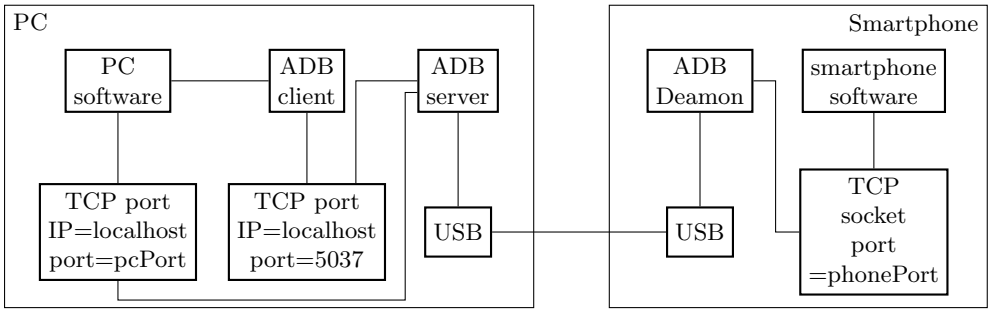

Fig. 11. USB communication architecture between a smartphone and a computer

the same physical state when the value of the bit transmitted is 1. A bit stuffing at 0 it added to the signal when six successive bits at 1 are sent in a packet. The physical signal is the differential voltage of the wires D+ and D- to improve analog quality.

Cryptographic texts are transmitted via this communication architecture. A part of the resynchronization process is also done through the transmission of a particular pattern, which is detected by measuring the data voltage.

G Estimating the Koblitz constant

In this section, we consider signed τ-adic expansions $k = k_0 + k_1\tau + \cdots + k_{\ell-1}\tau^{\ell-1} = u + v\tau \in \mathbb{Z}[\tau]$ for some fixed ℓ ($k_i \in \{-1, 0, 1\}$), and describe how to evaluate the constant c such that $u^2 + v^2 \leq c \cdot 2^\ell$ for all k.

Regard k as a complex number $k = \rho \cdot e^{i\theta}$. Since $\tau = \frac{\mu+\sqrt{-7}}{2}$, the real and imaginary part of k can be written as:

$$\rho\cos\theta = u + \frac{\mu}{2}v \quad \text{and} \quad \rho\sin\theta = \frac{\sqrt{7}}{2}v,$$

hence:

$$v = \rho \cdot \frac{2}{\sqrt{7}} \sin\theta$$
$$u = \rho \cdot \Big(\cos\theta - \frac{\mu}{\sqrt{7}} \sin\theta \Big)$$

and if we introduce the angle ϕ such that $\cos\phi = \sqrt{7/8}$, $\sin\phi = \mu/\sqrt{8}$, we obtain:

$$u = \rho \cdot \sqrt{\frac{8}{7}} \cos(\theta + \phi).$$

Thus, we have:

$$\begin{aligned} \frac{u^2+v^2}{\rho^2} &= \frac{4}{7}\sin^2\theta + \frac{8}{7}\cos^2(\theta+\phi) \\ &= \frac{6}{7} - \frac{2}{7}\cos(2\theta) + \frac{4}{7}\cos(2\theta+2\phi) \\ &= \frac{6}{7} + \frac{2}{7}\Re\Big(e^{2i\theta}\cdot(2e^{2i\phi}-1)\Big). \end{aligned}$$

On the other hand:

$$\big|2e^{2i\phi}-1\big|^2 = (2e^{2i\phi}-1)(2e^{-2i\phi}-1) = 5 - 4\cos(2\phi) = 5 - 8\cos^2\phi + 4 = 2.$$

As a result:

$$\frac{6-2\sqrt{2}}{7}\rho^2 \le u^2+v^2 \le \frac{6+2\sqrt{2}}{7}\rho^2. \tag{7}$$

Therefore, to obtain an upper bound for u^2+v^2, it suffices to obtain one for ρ (although the resulting estimate might not be completely tight). Now we have:

$$\begin{aligned} \rho &= \big|k_0 + k_1\tau + \cdots + k_{\ell-1}\tau^{\ell-1}\big| \\ &= |\tau|^{\ell-1}\cdot\big|k_{\ell-1} + k_{\ell-2}\tau^{-1} + \cdots + k_0\tau^{1-\ell}\big| \\ &\le 2^{(\ell-1)/2}\cdot|z_{\max}| \end{aligned}$$

where $z_{\max} \in \mathbb{C}$ is an element of maximum absolute value in the compact set $Z = \{\sum_{i=0}^{\infty} a_i/\tau^i \mid (a_i) \in [-1,1]^{\mathbb{N}}\}$. A convexity argument shows that the coefficients a_i for $z_{\max}$ are all ±1, and that $z_{\max}$ is unique up to sign, which can be fixed by imposing $a_0 = 1$. That unique value is then easily found by the obvious greedy algorithm:

$$z_{\max} = \begin{cases} 1 - \tau^{-1} - \tau^{-2} + \tau^{-3} - \cdots & \text{if } \mu = -1 \\ 1 + \tau^{-1} - \tau^{-2} - \tau^{-3} - \cdots & \text{if } \mu = 1 \end{cases}$$

and satisfies $|z_{\max}| = 2.4695\ldots$ in both cases. As a result:

$$u^2+v^2 \le \frac{6+2\sqrt{2}}{7}\cdot 2^{\ell-1}|z_{\max}|^2$$

and we can thus take:

$$c = \frac{3+\sqrt{2}}{7}|z_{\max}|^2 \leq 3.85. \tag{8}$$

Using that estimate, it follows the attack of Section 4.2 against ECDSA on Koblitz curves can be applied when the bias exceeds $\log_2(3.85\pi e) \approx 5.04$ bits, compared to $\log_2 \sqrt{\pi e/2} \approx 1.05$ for the usual prime field attack. It would thus seem to require about 4 bits of extra bias to get the same lattice dimension.

In practice, however, that conclusion is overly pessimistic. Indeed, the bound (8) itself is already pessimistic for any given ℓ, due to the fact that the worst case value for u^2+v^2 will usually not be close to the upper bound of (7) (for example, the argument of $\tau^{\ell-1} \cdot z_{\max}$ will not be such that it maximizes $(u^2+v^2)/\rho^2$). But more importantly, what really matters for the attack of Section 4.2 is the norm of vector $\boldsymbol{u}$, which depends not on an upper bound on u^2+v^2, but on the average of $u_i^2+v_i^2$ over a relatively large number t of k_i's. By the central limit theorem, that average is roughly normally distributed, with mean the expected value of u^2+v^2 over the possible choices of k. The expected value will then depend on the precise way k is generated, so it is best evaluated using simulations. We carried out such simulations for values of ℓ between 100 and 300, and for k with either uniform coefficients in $\{-1,0,1\}$, uniform coefficients in $\{0,1\}$ or truncated TNAFs, and found that the average of $(u^2+v^2)/2^\ell$ almost always fell between 0.54 and 0.59 in the first case, and between 0.28 and 0.32 in the latter two cases. Therefore, taking:

$$c \approx \begin{cases} 0.57 & \text{for random signed expansions} \\ 0.30 & \text{for random bit expansions/TNAFs} \end{cases}$$

should lead to better lattice dimension estimates for our attack, and this is well verified in practice. In particular, the attack in fact requires a bias roughly exceeding $\log_2(0.57\pi e) \approx 2.3$ bits in the former case, and $\log_2(0.30\pi e) \approx 1.4$ in the latter two.

H Countermeasures

Since our attack is a SEMA-based (Simple Electro-Magnetic Analysis) one, it is possible to avoid the leakage of this information using many algorithms that have been proposed for the smart card industry. For instance, it is possible to use the well-known double-and-add always algorithm. An addition is performed whatever the value of the bit and therefore the scalar multiplication would show a regular succession of patterns, preventing an attacker from finding the values of the bits. Other solutions are possible. The countermeasure using atomicity patterns is one of them and was proposed by Chevallier-Mames, Ciet and Joye in 2004 [11]. It consists in writing the different elliptic curve operations with the same pattern of suboperations. Inspired from this paper [11], different formulas that are more efficient, or more suitable for particular scalar multiplications, have been proposed [29,20,39].

The previous countermeasures are applicable only for the prime curve. If they are used for the Koblitz curve, then the benefits of using Frobenius maps are lost and it would be better to use general binary curves combined with the Montgomery's ladder. Some countermeasures for Koblitz curves were given by Hasan [22]. The use of zero-free tau-adic expansions [37] would probably be the best choice.

Another countermeasure that thwarts the side-channel vulnerability is the randomization of the private exponent as described by Coron in [14]. A random scalar is chosen and added to the nonce

in a way that the multiplication of the base point with the new scalar gives the same resulting point as would have been obtained with the original nonce. As the succession of EC operations is different, it prevents an attacker from finding a part of the nonce, and so to lead a lattice attack. A good choice for the size of the scalar is 32 bits. The consequence is the extension of the nonce by this size.

Finally, Bouncy Castle 1.51 decides to implement fixed-base comb method as defined in section 9.3 of Cohen *et al.* [12]. This technique avoids to obtain consecutive bits at the most or least significant bit positions and bypasses our attacks.