A Side Journey To Titan

Thomas Roche, Victor Lomné, Camille Mutschler, Laurent Imbert

A Side Journey To Titan

Revealing and Breaking NXP's P5x ECDSA Implementation on the Way

Thomas Roche[1], Victor Lomné[1], Camille Mutschler[1,2], and Laurent Imbert[2]

[1]NinjaLab, Montpellier, France
[2]LIRMM, Univ. Montpellier, CNRS, Montpellier, France

Abstract

The *Google Titan Security Key* is a FIDO U2F hardware device proposed by Google (available since July 2018) as a two-factor authentication token to sign in to applications such as your Google account. In this paper, we present a side-channel attack that targets the *Google Titan Security Key* 's secure element (the NXP A700x chip) by the observation of its local electromagnetic radiations during ECDSA signatures. This work shows that an attacker can clone a legitimate *Google Titan Security Key*. As a side observation, we identified a novel correlation between the elliptic curve group order and the lattice-based attack success rate.

1 Introduction

Hardware security keys for two-factor authentication are the recommended alternatives to SMS-based or app-based two-factor authentication using a smartphone. These security keys are based on the FIDO U2F standard initially developed by Google and Yubico and now administered by the FIDO Alliance. Security-wise, their strength resides in the use of secure microcontrollers (or *secure elements*) for the manipulation of cryptographic secret keys. The secure element must safely generate, store and use a user-unique secret to prove its legitimacy to a remote server during login in. The FIDO U2F standard is based on ECDSA signature over the NIST P-256 elliptic curve [29].

In this paper we study the security of the *Google Titan Security Key* [12] and show that its secure element, the `NXP A700x` chip, is susceptible to side-channel attack (through the observation of its local ElectroMagnetic – EM – activity). This allows, given physical access to a *Google Titan Security Key* during about 10 hours, to retrieve a user-specific secret key (there is one key for each remote account) and therefore to clone the security key.

To understand the NXP ECDSA implementation, find a vulnerability and design a key-recovery attack, we had to make a quick stop on *Rhea* (`NXP J3D081` JavaCard smartcard). This product looks very much like the `NXP A700x` chip and uses the same cryptographic library. *Rhea*, as an open JavaCard platform, gives us more control to study the ECDSA implementation.

The vulnerability allows an attacker, using a non-supervised machine learning mechanism, to gather several bits randomly scattered over the ephemeral key of the ECDSA signature scheme. She can then use a lattice-based attack to exploit this information in a key-recovery attack using a few thousands of ECDSA observations. Contrary to most lattice-based attacks with partial knowledge of the nonces reported in the literature, the known bits are not the leading bits of the nonces.

Surprisingly, the attack is much more efficient than expected in terms of data complexity. This observation led us to a finding of independent interest, relating the success rate of these attacks to the order of the elliptic curve. We believe that this observation opens new directions in the theoretical understanding of (Extended) Hidden Number Problem solvers.

The vulnerability was acknowledged by Google and the chip manufacturer NXP (we assigned CVE-2021-3011). It is present in few other security keys and various NXP JavaCards products[1] (all based on similar secure elements).

The contributions presented in this paper include:

- a teardown / PCB analysis of the *Google Titan Security Key*, and the identification of an NXP open Javacard product that shares a very similar secure element, presented in Section 2;
- the use of side-channel analysis to reverse-engineer the implementation of the cryptographic primitives and to reveal their countermeasures (see Section 3);
- the discovery of a previously unknown vulnerability in the (previously unknown) implementation (see Section 4);
- the exploitation of this vulnerability with a custom lattice-based attack to fully recover an ECDSA private

[1]The full list of identified products is here: https://ninjalab.io/a-side-journey-to-titan/

key from the *Google Titan Security Key* (see Section 5);

- an original observation that seems to link together the success rate of lattice-based attacks on ECDSA and the order of the elliptic curve, and its consequences regarding the success rate of lattice-based attacks on structured-order elliptic curves such as NIST P-256 (see Section 6);
- several countermeasures that could be implemented in order to mitigate the proposed attack (see Section 7).

2 Preliminaries

In this Section, we introduce the public information available for the FIDO U2F protocol and the physical analysis of the *Google Titan Security Key*. We also present the preparation process for EM based side-channel analysis.

2.1 Product Description

The *Google Titan Security Key* is a hardware FIDO U2F (universal second factor) device. It provides a complement to the login/password authentication mechanism, in order to sign in to a Google account, or any other web applications supporting the FIDO U2F protocol.

The *Google Titan Security Key* is available in three versions, as depicted in Figure 1.

Figure 1: *Google Titan Security Key* - Left: version with micro-USB, NFC and BLE interfaces - Middle: version with USB type A and NFC interfaces - Right: version with USB type C interface

2.2 FIDO U2F Protocol

The FIDO U2F protocol, when used with a hardware FIDO U2F device like the *Google Titan Security Key*, works in two steps: *registration* and *authentication*. Three parties are involved: the *relying party* (*e.g.* the Google server), the *client* (*e.g.* a web browser) and the *U2F device*. Let us briefly summarize how the different messages are constructed and exchanged. For more details, see [9].

Registration

1. The FIDO *client* first contacts the *relying party* to obtain a challenge. It then constructs the `registration request message`, made of the challenge and application parameters and sends it to the *U2F device*.
2. The *U2F device* creates a new ECDSA keypair in response to the `registration request message`, and answers the `registration response message`, which contains the user's public key, a key handle (which may contains the encrypted private key), an attestation certificate, and an ECDSA signature on P-256 over the application and challenge parameters, the key handle and the public key.
3. Finally, the FIDO *client* sends the `registration response message` back to the *relying party*, which stores the different fields for later authentications.

Authentication

1. The FIDO *client* contacts the *relying party* to obtain a challenge and constructs the `authentication request message`, made of a control byte (specifying whether or not the *U2F device* should enforce user presence), the challenge parameter, the application parameter and a key handle. Then sends it to the *U2F device*.
2. If the *U2F device* succeeds to process/sign the `authentication request message`, it answers the `authentication response message`, made of a user presence byte indicating whether user presence was verified or not, a counter on 4 bytes that is incremented each time the *U2F device* performs an U2F authentication and an ECDSA signature on P-256 (over the application parameter, the user presence byte, the counter and the challenge parameter).
3. Finally, the FIDO *client* sends the `authentication response message` back to the *relying party*, which can then verify the ECDSA signature using the public key obtained during registration.

2.3 An Attack Scenario on FIDO U2F

From the study of the FIDO U2F protocol, one can imagine the following attack scenario that requires the adversary to get physical access to the victim's U2F device during a limited time frame without the victim noticing (step 2):

1. the adversary steals the login and password of a victim's application account protected with FIDO U2F (*e.g.* via a phishing attack);
2. thanks to the stolen victim's login and password (for a given application account), the adversary can get the

corresponding *client data* and *key handle*. She can then send many authentication requests to the U2F device while performing side-channel measurements[2];

3. the adversary quietly returns the U2F device to the victim;

4. the adversary performs a side-channel attack on the measurements, and succeeds in extracting the ECDSA private key linked to the victim's application account;

5. the adversary can sign in to the victim's application account without the U2F device, and without the victim noticing. In other words the adversary created a clone of the U2F device for the victim's application account. This clone will give access to the application account as long as the legitimate user does not revoke its second factor authentication credentials.

Note that the *relying party* might use the counter value to detect cloned U2F devices and then limit (but not totally remove) the attack impact (see Section 7.2 for more details).

2.4 *Google Titan Security Key* Teardown

Once plugged into a computer's USB port, `lsusb` outputs: `Bus 001 Device 018: ID 096e:0858 Feitian Technologies, Inc.`

As a matter of fact, the company who designed the *Google Titan Security Key* is Feitian [8]. Indeed Feitian proposes generic FIDO U2F security keys, with customization for casing, packaging and related services.

2.4.1 Removing the Casing

We first performed a teardown of the USB type A version of the *Google Titan Security Key*. The plastic casing is made of two parts which are strongly glued together. We used a hot air gun in order to soften the white plastic and we easily separated the two casing parts with a scalpel.

If done carefully, this easy procedure allows to preserve intact the Printed Circuit Board (PCB). An interesting future work could be to find a way to open the *Google Titan Security Key* casing without damaging the two plastic parts, so that it can be re-assembled after the attack.

2.4.2 PCB Analysis

In Figure 2, we display the back of the *Google Titan Security Key* PCB, where the different circuits are soldered. The Integrated Circuit (IC) package markings allow to guess the IC references:

- the first IC (in green in Figure 2) is a general purpose microcontroller from NXP, the LPC11u24 from the LPC11U2x family [30]. It acts as a router between the USB and NFC interfaces and the secure element;

- the second IC (in red in Figure 2) is a secure authentication microcontroller also from NXP, the `A7005a` from the `A700x` family [25]. It acts as the secure element, generating and storing cryptographic secrets and performing cryptographic operations (we validated this hypothesis by probing electric signals between the two ICs while processing an `authentication request message`).

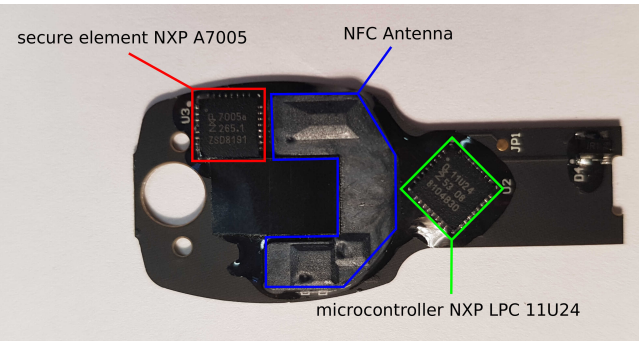

Figure 2: *Google Titan Security Key* PCB, with annotated main parts

2.4.3 NXP A7005a Package Opening

Opening the `NXP A7005a` epoxy package necessitated a wet chemical attack. We protected the PCB with some aluminium tape and dropped some hot fuming nitric acid on the `NXP A7005a` package until the die was revealed (see [11, Chapter 2] for a survey on IC package opening techniques).

The result is shown in Figure 3. With the device still alive, we can then proceed with the EM side-channel measurements.

2.5 Matching the *Google Titan Security Key* with other NXP Products

The FIDO U2F protocol does not allow to extract the ECDSA secret key of a given application account from a U2F device. This is a limitation of the protocol which, for instance, makes it impossible to transfer the user credentials from one security key to another. If a user wants to switch to a new hardware security key, a new registration (i.e. a new ECDSA key pair) is required for every application account.

From a security point of view, this limitation is also a strength as it prevents creating a clone and represents an obstacle for side-channel reverse-engineering. With no control whatsoever on the secret key, understanding the details of a highly secured implementation (let alone attacking) can

[2] it might be limited to several billions of requests, the counter being encoded on 4 bytes

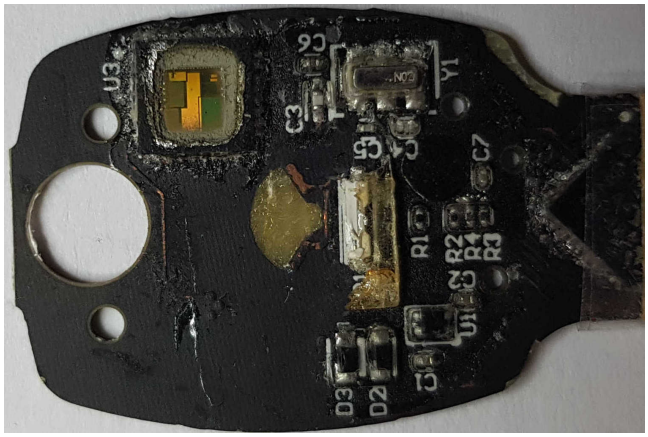

Figure 3: *Google Titan Security Key* PCB, with `NXP A7005a` die visible after wet chemical attack of its package

prove cumbersome. We had to find a workaround to study the implementation in a more convenient setting.

2.5.1 NXP A700x Datasheet Analysis

The `NXP A700x` public datasheet [25] provides the following interesting informations:

- it runs the NXP's JavaCard Operating System called JCOP, in version JCOP 2.4.2 R0.9 or R1 (JavaCard version 3.0.1 and GlobalPlatform version 2.1.1);
- technological node is 140 μm;
- CPU is `Secure_MX51`;
- 3-DES and AES hardware co-processors;
- public-key cryptographic co-processor is NXP `FameXE`;
- RSA available up to 2048 bits and ECC available up to 320 bits.

The `NXP A7005a` RSA and ECC key length limitations, JCOP version and technological node indicate that this is not a very recent chip.

2.5.2 Similarities with other NXP Products

With the information gathered from the `NXP A700x` datasheet and its IC optical analysis, we tried to identify similar NXP products for which we could have more control on the ECDSA operations. In fact, several NXP JavaCard platforms share the `NXP A700x`'s characteristics. They are all based on NXP P5x chips.

The NXP P5x secure microcontroller family is the first generation of NXP secure elements, also called `SmartMX` family [31]. It has the exact same characteristics as the `NXP A700x`. Furthermore the NXP P5x secure microcontroller family is Common Criteria (CC) and EMVCo certified (last CC certification found in 2015).

We went through the public data that can be found online and figured out that several NXP JavaCard smartcards are based on P5x chips. Thanks to BSI and NLNCSA CC public certification reports[3], we were able to compile a (non-exhaustive) list of NXP JavaCard smartcards based on P5x chips.

We selected the product `NXP J3D081` (CC certification report BSI-DSZ-CC-0860-2013) since its characteristics were the closest to those of `NXP A700x` (JCOP 2.4.2 R2, JavaCard 3.0.1 and GlobalPlatform 2.2.1). We named it *Rhea*, in reference to the second largest moon of Saturn, right after *Titan*.

Open JavaCard products, like *Rhea*, are generic platforms that allow developers to load their own applications (a JavaCard *applet*) on the smartcard. The JavaCard OS takes care of low level interactions with the hardware and offers high level APIs for the applets. Hence, an applet needs to comply with the JavaCard OS API independently of the underlying hardware.

On *Rhea*, the JavaCard OS happens to follow JavaCard 3.0.1 specifications [32]. We developed and loaded a custom JavaCard applet allowing us to freely control the JavaCard ECDSA signature engine on *Rhea*. At this point, we were able to upload the long term ECDSA secret keys of our choice, perform ECDSA signatures and verifications.

2.6 Side-Channel Observations

2.6.1 Side-Channel Setup

In order to perform EM side-channel measurements, we used the following side-channel analysis hardware setup (global cost is about US $12,000):

- Langer ICR HH 500-6 near-field EM probe with an horizontal coil of diameter 500μm and a frequency bandwidth ranging from 2MHz to 6GHz [20];
- Thorlabs PT3/M 3 axes (X-Y-Z) manual micromanipulator with a precision of 10μm [37];
- Pico Technology PicoScope 6404D oscilloscope, with a 500MHz frequency bandwidth, sampling rate up to 5GSa/s, 4 channels and a shared channel memory of 2G samples [34].

For triggering the side-channel measurements, we proceeded as follows:

- for the side-channel measurements performed on *Rhea*, we used a modified commercial smartcard reader where

[3] https://www.bsi.bund.de/EN/Topics/Certification/certified_products/Archiv_reports.html

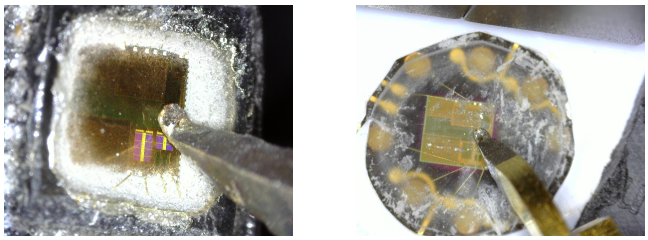

Figure 4: EM Probe Positions on *Titan* (left) and *Rhea* (right)

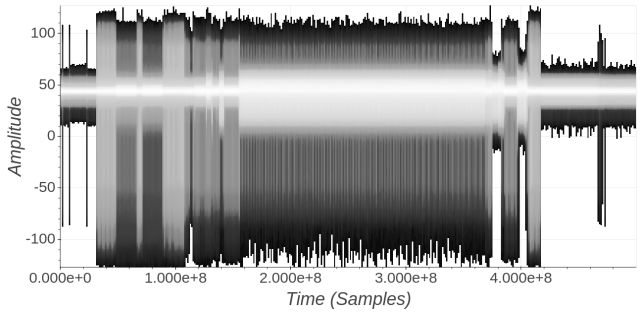

Figure 5: *Titan* ECDSA Signature EM Trace

we tapped the I/O line so we could trigger on the sending of the APDU command;

- for the side-channel measurements performed on *Titan*, we used the triggering capabilities of our oscilloscope to trigger on a pattern present at the beginning of the EM activity of the command processing the `authentication request message`.

2.6.2 First Side-Channel Observations on *Titan* and *Rhea*

Figure 4 depicts the spatial position of the EM probe above the die of the *Google Titan Security Key* `NXP A7005a` and the die of *Rhea*. In Figures 5 and 6, we give the EM activities observed during *Titan*'s `authentication request message` ECDSA signature, and during the processing of the APDU command launching the ECDSA signature available in the JavaCard cryptographic API of *Rhea*.

The similarities between EM activities on *Titan* and *Rhea* confirm our hypothesis that the implementations are very similar. Note that the spatial probe positioning is sensitive to get a clear signal with sharp peaks, but the picture taken for *Rhea* (Figure 4 left) proved sufficient to replay the probe positioning on *Titan*.

3 Reverse-Engineering the ECDSA Algorithm

The reverse-engineering of the ECDSA signature and verification algorithms presented in this section was conducted on *Rhea* as we had full control on the inputs, in particular the private key d.

3.1 ECDSA Signature Algorithm

3.1.1 Basics about the ECDSA Signature Algorithm

Let us briefly recall the ECDSA signature algorithm and introduce the necessary notations. We work on an elliptic curve E defined over the finite field $\mathbb{F}_p$, and denote by $G_{(x,y)}$ a point on E of large prime order q. The ECDSA signature algorithm [17] takes as inputs the hash of the message m to be signed $h = H(m)$, and a secret key d. It outputs a pair (r,s) computed as follows:

1. randomly pick a nonce k in $\mathbb{Z}/q\mathbb{Z}$
2. scalar multiplication[4] $Q_{(x,y)} = [k]G_{(x,y)}$
3. denote by r the x-coordinate of Q : $r = Q_x$
4. compute $s = k^{-1}(h + rd) \bmod q$

Observe that since *Rhea* allows us to choose the secret key d, we can easily compute the nonce value k used to produce any signature (r,s) for any given message $h = H(m)$.

3.1.2 Matching the Algorithm to the Side-Channel Traces

Figure 6 presents a full EM trace of the ECDSA signature at sampling rate 2.5GSa/s. (The whole execution time is approximatively 73ms.) Our first goal was to identify the different steps of the ECDSA algorithm on the trace.

After an initialization phase, where ECDSA inputs are processed and stored, the first step is to randomly generate the nonce k and the z coordinate of G in projective coordinates. The call to a pseudo-random number generator (PRNG) is clear in the identified area: there are 48 calls to the PRNG to generate a 256-bit random and the PRNG re-initializes itself every 60 calls. There must also be at least two modular multiplications in this step to get G in projective coordinates. Also, the nonce k is recoded in the form required by the scalar multiplication algorithm (we give more details in Section 3.3).

The next block corresponds to the scalar multiplication itself. This is the longest operation in ECDSA and its stable iterative process stands out clearly.

The last four blocks are composed of two modular inversions ($z^{-1} \bmod p$ to get $r = Q_x$ and $k^{-1} \bmod q$), the hash of

[4] In a secure implementation, this is usually done on randomized projective coordinates $G_{(x,y)} \to G_{(xz \bmod p, yz \bmod p, z)}$ with z a fresh random from $\mathbb{F}_p$ (see e.g. [6]).

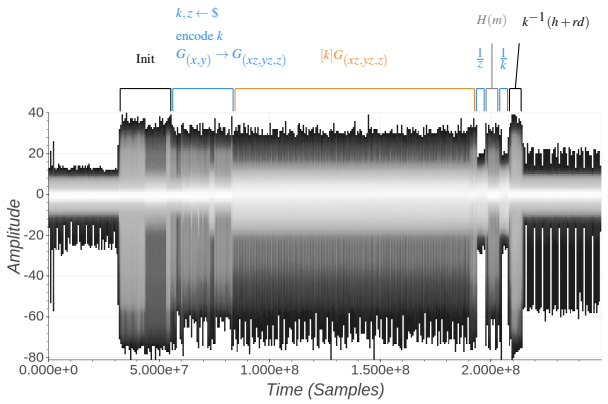

Figure 6: *Rhea* EM Trace - ECDSA Signature (P-256, SHA-256)

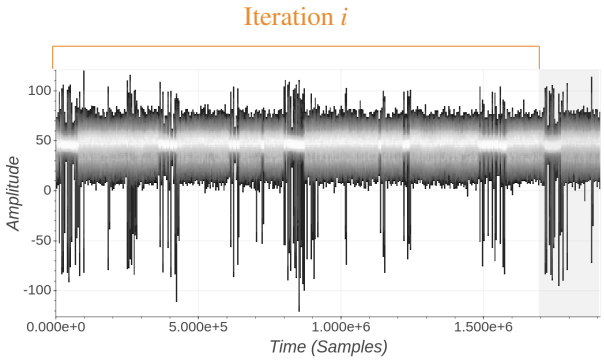

Figure 7: *Rhea* EM Trace - ECDSA Signature (P-256, SHA-256) - Scalar Multiplication Single Iteration

the message m and the final computation of s with two modular multiplications and one addition. We infer the ordering depicted in Figure 6 but we do not have strong arguments to show that these operations are actually performed in this order. It is worth mentioning that the overall process is pretty similar to what was observed in [26]. The authors were working on a P5 chip with an older version of the NXP cryptographic library.

3.1.3 Studying the Scalar Multiplication Algorithm

In side-channel analysis, there are many ways to attack an ECDSA implementation. In fact, any leakage inside one of the previously mentioned operations involving the nonce or the secret key could potentially lead to an attack. In the literature, the most studied operation is the scalar multiplication. Let us have a closer look.

By observing many signature traces, we observed that the scalar multiplication step takes approximatively 43 ms, and more importantly that each scalar multiplication consists of exactly 128 iterations (*i.e.* the repetition of the same signal pattern). Figure 7 displays a single iteration at sampling rate 5GSa/s. We observed that some parts of the traces vary slightly from one iteration to another (probably due to a random delay countermeasure). The iteration length is then not perfectly stable but it takes roughly 340μs, which corresponds to about 1.7M samples at sampling rate 5GSa/s.

We concluded for a constant time algorithm based on some sort of *Double&Add Always* implementation. In particular, the implementation does not skip the leading zero bits of the scalar as in [5], or more recently [16, 24]. In order to find a vulnerability we needed a better understanding of the implementation. To this end, we analyzed the ECDSA signature verification algorithm.

3.2 ECDSA Signature Verification Algorithm

As mentioned before, one great advantage of working on *Rhea* is the possibility to run the ECDSA signature verification algorithm which does not involve any secret. Countermeasures are therefore useless and developers often downgrade or suppress them in order to reduce the execution time. For reverse engineering however, such a countermeasure downgrade is a windfall. It provides the opportunity to learn a lot about the implementation and its countermeasures.

The core of an ECDSA signature verification is a double-scalar multiplication of the form $[k_1]G + [k_2]P$, where G is the curve base point and P the signatory's public key. It instantly appeared that this operation was implemented as two separate scalar multiplications (followed by a final point addition), whose traces looked similar to those observed for the signature algorithm. Our analysis revealed however that it is not constant time: the *Double&Add Always* implementation is replaced by a simple *Double&Add* implementation and the leading zero bits of the scalar are skipped. Also, an expensive pre-computation step is visible before one of the two scalar multiplications. This pre-computation step looks like a scalar multiplication where the scalar is a power of 2 (*i.e.* it is made of *Double* operations only). Finally, the manipulation of the point at infinity can be easily spotted in the side-channel signal.

These observations on the verification algorithm led us to draw the following hypothesis for the signature algorithm:

- each iteration is constituted of a *Double* and a *Add* operation;
- each iteration handles two bits of the scalar, starting with the most significant bit;
- the scalar is not blinded;
- the point at infinity is never manipulated;
- the scalar multiplication requires the pre-computed value $[2^{\lceil l/2 \rceil}]G$ (where l is the bit-length of the scalar) that is hard coded into the chip.

3.3 High-Level NXP Scalar Multiplication Algorithm

There are many ways to implement a scalar multiplication algorithm but the costly pre-computation observed in the previous section, together with the number of subsequent iterations and the fact that there is a single doubling operation for each addition clearly suggests a *comb* method (see [21]) of width 2.

To compute $[k]G$, the scalar $k = (k_1, \ldots, k_l)_2$ of even length l[5] is first encoded as $\tilde{k} = (\tilde{k}_1, \ldots, \tilde{k}_{l/2})$ where $\tilde{k}_i$ is a 2-bit value obtained by concatenation of k_i and $k_{l/2+i}$ such that $\tilde{k}_i = 2k_i + k_{l/2+i}$.

A *comb* implementation of width 2, requires the pre-computation of the curve points $G_1 = G$, $G_2 = [2^{l/2}]G_1$ and $G_3 = G_1 + G_2 = [2^{l/2} + 1]G_1$.

From the above analysis, our first and best guess for the scalar multiplication algorithm is given in Algorithm 1.

Algorithm 1: Scalar Multiplication Algorithm used in Signature Operation

Input : $(\tilde{k}_1, \ldots, \tilde{k}_{129})$, the encoded scalar
Input : G_0, G_1, G_2, G_3, G_4, the pre-computed points
Output : $[k]G$

$S \leftarrow G_1$
for $i \leftarrow 2$ **to** 129 **do**
  $S \leftarrow [2]S$
  if $\tilde{k}_i > 0$ **then**
    $S \leftarrow S + G_{\tilde{k}_i}$
  else
    $Dummy \leftarrow S + G_0$
if $\tilde{k}_1 = 0$ **then**
  $S \leftarrow S - G_4$
else
  $Dummy \leftarrow S - G_4$
Return : S

In Algorithm 1, *Dummy* represents a register or memory address which will not be read and therefore stores useless computation results, G_0 is any point on the elliptic curve, $G_1 = G$ (the elliptic curve base point), $G_2 = [2^{129}]G_1$, $G_3 = G_1 + G_2$ and $G_4 = [2^{128}]G_1$.

Since G_0 is solely used for the dummy additions, it could be any point on the curve, it could even change over time. Most likely $G_0 \in \{G_1, G_2, G_3, G_4\}$, since these points are already computed.

In Algorithm 1, the binary form of k is of length $l = 258$. This means that at least two extra leading zero bits are added to k. The purpose of this trick is to ensure that $\tilde{k}_1$ is either 0 or 1. In the former case however, the initialization of S should be the point at infinity. In order to avoid this, $\tilde{k}_1$ is forced to value 1. It is corrected by the last operations in Algorithm 1 assuming the point G_4 is also stored during the pre-computation step (in addition to G_2 and G_3). This process is confirmed by the presence of an *Add* operation following the scalar multiplication sequence of *Double&Add* iterations.

[5] k may be padded with 0s if necessary

4 A Side-Channel Vulnerability

As explained in the previous section, each signature on *Rhea* allows us to deduce the nonce k from the chosen private key d. Therefore, we could look for statistical dependencies between the side-channel traces and the nonce values, more exactly the encoded digits $\tilde{k}_i$.

The research of sensitive leakage is a tedious task where many interdependent parameters have strong influence and should be set correctly for success. In the next section, we investigate these parameters and show how we eventually managed to find a sensitive leakage. Section 4.1 sums-up several months of work tainted with failed attempts and disillusionment. The details given in Sections 4.1.1 to 4.1.3 can be skipped at first-reading. Section 4.2 provides precise information about the sensitive leakage. Section 4.3 shows how that leakage helped to better understand the scalar multiplication implementation.

4.1 Searching for Sensitive Leakage

Our statistical side-channel analysis started by the acquisition of the EM radiations of the *Rhea* chip during 1000 ECDSA executions (we eventually needed 4000 acquisitions for the attack to be successful). Each trace was then split into 128 sub-traces corresponding to the point doubling and point addition operations inside the main loop of Algorithm 1. We thus ended-up with 1000×128 sub-traces (one per iteration).

As mentioned in Section 3.1.3, the sub-traces are not perfectly synchronized (certainly due to a random delay countermeasure). This means that, at time sample t, two sub-traces do not exactly capture the EM signal related to the same underlying computations. They have to be re-aligned in order for us to estimate any statistical dependency between the EM signal and the encoded nonce digits.

4.1.1 Preliminary Acquisition Setup

This whole process necessitate to choose some acquisition parameters:

- choice of EM probe: we started with a Langer ICR HH 250-75 near-field EM probe with an horizontal coil of diameter 250 μm and a frequency bandwidth ranging from 0.5 MHz to 2 GHz [19].
- EM probe position: we selected a position where the *Double* and *Add* operations were easily distinguishable and the EM signal had a large amplitude.

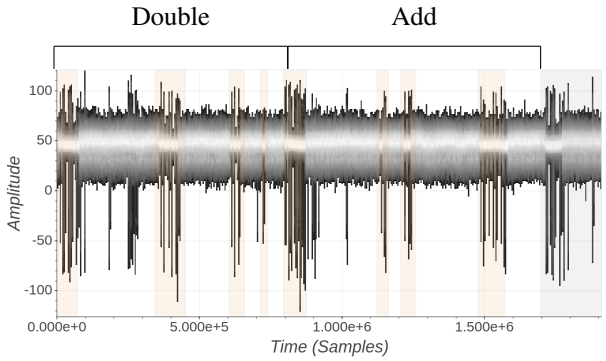

Figure 8: *Rhea* EM Trace - ECDSA Signature (P-256, SHA-256) - Scalar Multiplication Single Iteration

- sampling rate: we choose the highest sampling rate given by our oscilloscope (5 GSamples/s). The drawback is that we then had to handle large traces: about 1.7M samples per sub-trace.

4.1.2 Traces re-alignment

Let us now see how we managed to re-align our 1000×128 sub-traces. In Figure 8, we display a sub-trace with 8 identified sections (in orange) where the execution time seems to randomly vary from one sub-trace to another. Over all sub-traces, we observed that the length of these sections vary by a factor of 2. Our hypothesis is that an elementary random delay countermeasure is applied (by repeating or not some of the computations). The rest of the sub-traces, i.e. the 8 interleaved sections, show a small jitter which very likely comes from the internal clock natural jitter.

These observations led us to try to re-align each of the 16 sections independently. We started with the orange sections showing a random delay countermeasure assuming that they were more likely to hide worthwhile information. As we shall see next, this was clearly not our best bet.

We decided to skip the first section as it was not clear if it was the start of the current iteration or the end of the preceding one. Unfortunately, for each of the seven other orange sections, the SNR analysis resulting from the 1000×128 re-aligned sections did not show any significant leakage.

We then considered the 8 other sections without random delay countermeasure. In each of these sections, the signal is mostly composed of small consecutive EM peaks that we detected and re-aligned. However, the peak detection was too noisy. Some peaks were overlooked and some signal interferences were erroneously identified as signal peaks. Again, these re-alignments did not give us any interesting results.

At this point, we had more questions than answers: is the acquisition setup correct? Was our trace re-alignment procedure correct? Do we have enough traces to observe a sensitive leakage? Was there any sensitive leakage at all?

We modified the EM probe position and adapted some previous re-alignments (those that seemed to give the best SNR results) with no success.

In a last attempt, we focused our attention on the two orange sections at the beginning of the *Double* and the *Add* operations. These parts of the traces reflect the activity of the crypto library which sets the different register addresses before launching the operations. We finally captured a weak sensitive leakage located on a large EM signal peak (one of the peaks with large amplitude that we can see on Figure 8 at the very beginning of the *Add* operation). Note that we did not explicitly exploited these peaks during our first attempts because we based our re-alignment procedure on the peaks belonging to the random delay countermeasure.

This first positive result lead us to perform a last experiment relying on the systematic detection of the EM signal peaks with large amplitude over the whole sub-trace. It turned out that four of these peaks (located during the *Double* operation) bear a strong sensitive leakage, much stronger than the weak leakage observed before. In Figure 9, we show the area where a sensitive leakage was eventually detected. Figure 10 focuses on the four signal peaks that bear the sensitive leakage inside that area.

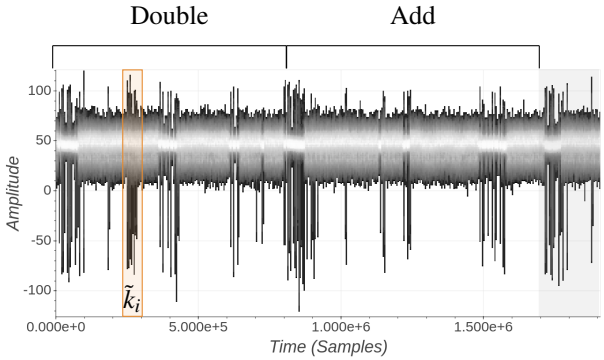

Figure 9: *Rhea* EM Trace - ECDSA Signature (P-256, SHA-256) - Sensitive Leakage Area

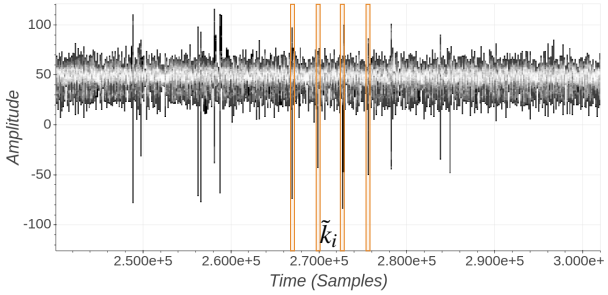

Figure 10: *Rhea* EM Trace - ECDSA Signature (P-256, SHA-256) - Sensitive Leakage

Inputs	random messages, constant key
# operations	4000
Length	100 ms
Sampling rate	5G Sa/s
# Samples/trace	500 M
Channel conf.	DC 50 ohms, ±50 mV
File size	2 TB
Acq. time	≈ 4 hours

Table 1: SCA acquisition parameters for *Rhea*

4.1.3 Final Acquisition Setup

Based on this success, we checked various EM probe positions and even changed our EM probe itself (for the Langer ICR HH 500-6, see Section 2.6.1) to improve the signal strength. The final acquisition setup details are provided in Table 1. A picture of the probe position is shown in Figure 4.

4.2 A Sensitive Leakage

Figure 11 (first sub-figure) depicts 1000 superposed traces after re-alignment, where only 400 samples were kept around each of the four identified signal peaks. As mentioned before, to evaluate the statistical relations between the re-aligned traces and the encoded scalar digits, we computed the *Signal-To-Noise Ratio* (SNR). As stated in [22]: *"The SNR is quantifying how much information is leaking from a point of a power trace. The higher the SNR, the higher is the leakage"*. More precisely, each of the 4000×128 re-aligned traces are classified with respect to the corresponding 2-bit digit $\tilde{k}_i$. We then end up with four sets of traces. For each set s and at each time sample t, we estimated the traces mean $\mu_s(t)$ and variance $v_s(t)$. The SNR computed independently for each time sample t is obtained by:

$$\mathtt{SNR}(t) = \frac{Var(\mu_s(t))}{E(v_s(t))},$$

where $Var(\mu_s(t))$ is the estimated variance over the four estimated means and $E(v_s(t))$ is the estimated mean of the four estimated variances.

In the second sub-figure of Figure 11, we plotted the SNR results for the four sets ($\tilde{k}_i \in \{0,1,2,3\}$). The best SNR value is ≈ 0.53. Clearly the amplitude of the side-channel traces is strongly related to the sensitive values $\tilde{k}_i$[6].

[6]If the side-channel traces amplitude at time sample t is not related to encoded nonce digits, the respective SNR value should tends toward 0 as the number of traces increases (as the signal variance ($Var(\mu_s(t))$) itself tends to 0). This is what happens for most of the traces time samples (see Figure 11, second sub-figure). However, at some specific time samples (where SNR peaks are visible), the SNR converges toward a non-null value. This means that $Var(\mu_s(t))$ itself converges toward a non-null value and therefore the side-channel traces at time sample t are significantly different for the four different encoded nonce digits values.

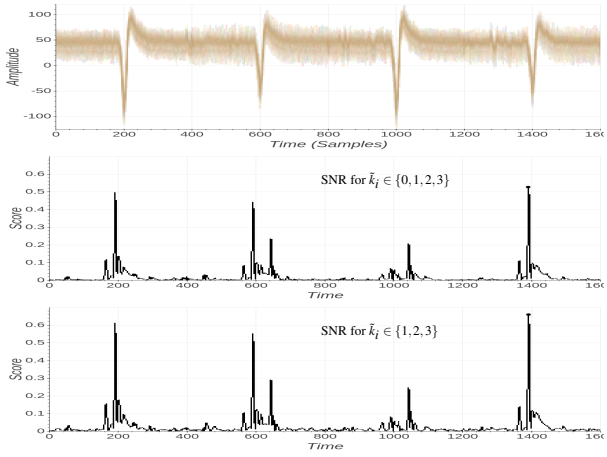

Figure 11: *Rhea* EM Trace - ECDSA Signature (P-256, SHA-256) - SNR results (y-axis range $[0,0.7]$)

Our first guess on the scalar multiplication algorithm (Algorithm 1) did not completely disclose the value taken by G_0, apart from the fact that it is not the point at infinity. In fact G_0 could be any point on the elliptic curve; but it is most likely chosen in $\{G_1, G_2, G_3, G_4\}$. Besides, G_0 could change from one iteration to another. Therefore, we estimated the SNR without considering the cases $\tilde{k}_i = 0$. The corresponding sub-traces were simply discarded from the SNR computations. In the third sub-figure of Figure 11, we can observe a significant increase of the SNR to ≈ 0.65. These results tend to confirm that G_0 takes varying values among G_1, G_2 and G_3 only. Using standard noise reduction techniques, based on filtering and principal component analysis, we managed to further improve the SNR to 0.78.

Let us go a bit further in the understanding of the leakage. Considering only the sub-traces where $\tilde{k}_i \neq 0$, we estimated the leakage strength with respect to the two bits of $\tilde{k}_i$ considered independently.

To do so we used the Welch T-Test [39]. Given two univariate data sources the T-Test tells us whether one can reject the null hypothesis with confidence, *i.e.* whether the two data sources are far enough from two independent sources. We performed two independent T-Tests. For the first test, the data sources are the sub-traces that correspond to $\tilde{k}_i = 1$ and $\tilde{k}_i = 3$ respectively, for which the lsb (of $\tilde{k}_i$) is equal to 1. This allows to test the msb of $\tilde{k}_i$. Similarly, we collected T-Test results for the sub-traces corresponding to $\tilde{k}_i = 2$ and $\tilde{k}_i = 3$ respectively which leave the msb constant; hence testing the lsb of $\tilde{k}_i$.

A T-Test score was computed for each time sample independently. The results are depicted in Figure 12. These scores clearly show that the two bits of $\tilde{k}_i$ do not leak at the same

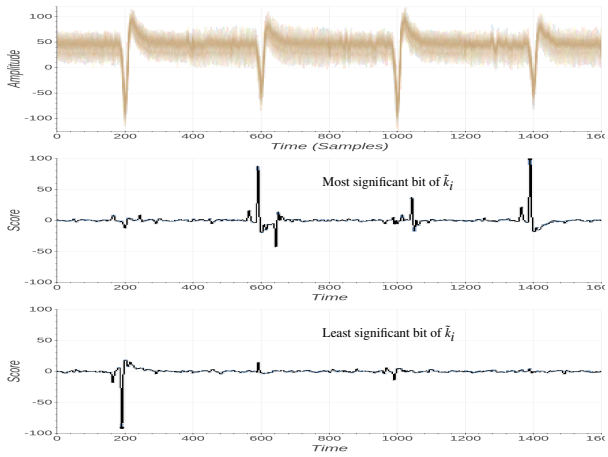

Figure 12: *Rhea* EM Trace - ECDSA Signature (P-256, SHA-256) - T-Test results (y-axis range $[-100, 100]$)

time. Furthermore, we can clearly see that the most significant bit of $\tilde{k}_i$ shows a strong leakage for the last three peaks, whereas the lsb's strongest leakage is mainly located on the first peak.

4.3 Improving our Knowledge of the NXP's Scalar Multiplication Algorithm

As explained in the previous section, we removed the sub-traces corresponding to the case $\tilde{k}_i = 0$ as they seem to deteriorate our SNR computation. Our hypothesis is that, when $\tilde{k}_i = 0$, the developers decided to randomly choose (at each iteration) a point from the available pre-computed points (G_1, G_2, G_3).

To try validate our hypothesis, we designed the following experiment based on supervised Expectation-Maximization clustering (to this end, we use the `GaussianMixture` class from `Scikit-learn` Python library [33]).

The idea is simple. We used the many sub-traces that are correctly labeled when $\tilde{k}_i \neq 0$ to train our clustering algorithm (*i.e.* to precisely define the three clusters using maximum likelihood). We were then able to match the un-labeled sub-traces (*i.e.* those corresponding to $\tilde{k}_i = 0$) by finding the closest cluster, *i.e.* by identifying the value j such that $G_0 = G_j$ for this iteration. The Expectation-Maximization clustering is a multivariate process. It uses multi-dimensional data (*i.e.* our sub-traces with several time samples) to infer multivariate Gaussian distributions from these samples. To ease this work, we had to remove some useless time samples (*i.e.* time samples for which the signal was not strongly related to the sensitive variable $\tilde{k}_i$). The overall process is summarized below:

1. Reduce all sub-traces to the time samples where the SNR is larger than a specific threshold (since the optimal threshold choice is not known a priori, we applied the process for different threshold values until it gave consistent results).
2. With the sub-traces corresponding to $\tilde{k}_i \neq 0$, estimate the three cluster centers for $\tilde{k}_i = 1, 2$ and 3 respectively.
3. For each labeled sub-trace, find the closest center. This phase allows controlling the clustering success rate.
4. Finally, for each un-labeled sub-trace (*i.e.* $\tilde{k}_i = 0$), find the closest center.

The matching phase revealed that about half of the un-labeled sub-traces matched the $\tilde{k}_i = 1$ case, while the other half were equally divided between the cases $\tilde{k}_i = 2$ and $\tilde{k}_i = 3$.

The above observation was validated by our next experiment. We created two sets of sub-traces. In the first set, we put the $\tilde{k}_i = 0$ sub-traces. The other set contained a mix of sub-traces with $\tilde{k}_i \neq 0$, with half of them corresponding to $\tilde{k}_i = 1$ and the rest equally divided between $\tilde{k}_i = 2$ and $\tilde{k}_i = 3$. The T-Test evaluation between these two sets could not reject the null hypothesis (no T-Test peak is visible and the best T-Test absolute value is less than 3[7]), hence confirming the Expectation-Maximization experiment results.

In the improved version of the scalar multiplication algorithm presented in Algorithm 2, we have $G_0 = G_1 = G$ (the elliptic curve base point), $G_2 = [2^{129}]G_1$, $G_3 = G_1 + G_2$ and $G_4 = [2^{128}]G_1$.

Since $G_0 = G_1 = G$, one can check that the $Dummy \leftarrow S + G_{rand}$ addition is operated on G_1 half the time and on G_2 or G_3 the rest of the time. We would like to emphasize that this algorithm is only our interpretation of the real algorithm implemented on *Rhea* that might differ slightly. Details of the real implementation are not our concern here, a high-level understanding of the countermeasures is good enough.

5 A Key-Recovery Attack

In this section, we detail the process that resulted in the full recovering of the private keys embedded into the NXP's secure components of both *Rhea* and *Titan*. Our attack consists of two main steps: we first exploited the vulnerability observed in Algorithm 2 to recover some zero bits of the nonces with very high confidence level. Then, from this partial knowledge on the nonces, we applied a lattice-based attack by reducing our problem to an instance of the Extended Hidden Number Problem (EHNP). We present these two phases in the next sections.

[7] A more formal analysis, following *e.g.* [40], is possible to interpret the T-Test results and estimate the error probability of having an undetected leakage. Here, we do not need such a fine grain analysis, the T-Test results do not show the significant peaks found in prior experiments. We can then safely conclude that the two sets of sub-traces, selected as we did, behave very much alike.

Algorithm 2: Improved Version of Scalar Multiplication Algorithm used in Signature Operation

Input : $\{\tilde{k}_1, \cdots, \tilde{k}_i, \cdots, \tilde{k}_{129}\}$: The encoded scalar
Input : G_0, G_1, G_2, G_3, G_4: The pre-computed points
Output: $[k]G$: The scalar multiplication of scalar k by point G

```
// Init Register S to the point G(= G_1)
S ← G_1
for i ← 2 to l_k/2 do
    S ← [2]S
    rand ← random element from {0,1,2,3}
    if k̃_i > 0 then
        S ← S + G_{k̃_i}
    else
        Dummy ← S + G_rand
if k̃_1 = 0 then
    S ← S − G_4
else
    Dummy ← S − G_4
Return : S
```

5.1 Recovering Scalar Bits from the Observed Leakage

As seen in Section 4, Algorithm 2 leaks non-uniform information whenever the 2-bit encoded digit $\tilde{k}_i$ is zero. We recall that $\tilde{k}_i$ is obtained from the binary representation of k as $\tilde{k}_i = 2k_i + k_{129+i}$. When $\tilde{k}_i = 0$, our analysis confirmed that Algorithm 2 stores the result of the addition $S + G_0$ into a dummy register, with G_0 chosen at random in $\{G_1, G_2, G_3\}$ with respective probability $1/2$, $1/4$, $1/4$. Let $(\hat{k}_i)_i$ denote the sequence of digits recovered from the observed leakage on $\tilde{k}_i$ in a noise free scenario. From the above observation, we have $\hat{k}_i \in \{1,2,3\}$. Let us first examine the case $\hat{k}_i = 1$. With probability $1/4$, the observed value matches the correct value $\tilde{k}_i = 1$, in which case G_1 is correctly added to S. But it may also correspond to the case where $\tilde{k}_i = 0$ and G_1 was randomly chosen to perform the dummy addition, which occurs with probability $1/4 \times 1/2 = 1/8$. In total, we have $P(\hat{k}_i = 1) = 3/8$. The overall analysis for $\hat{k}_i = 1,2,3$ is summarized in Table 2.

Table 2 provides crucial information on the bits of k. In particular, we remark that $\hat{k}_i = 1$ implies $k_i = \text{msb}(\tilde{k}_i) = 0$. Similarly, $\hat{k}_i = 2$ implies $k_{129+i} = \text{lsb}(\tilde{k}_i) = 0$.

As seen in Section 4, T-Test results on carefully re-aligned sub-traces around four EM signal peaks (See Figure 12) gave us very precise time samples where the encoded digits are leaking. Testing the 2 bits of $\tilde{k}_i$ separately also revealed more leakage points for $k_i = \text{msb}(\tilde{k}_i)$ than for $k_{129+i} = \text{lsb}(\tilde{k}_i)$. Therefore, we first focused our analysis on the leakage arising from $\text{msb}(\tilde{k}_i)$. In practice, we used 4000×128 sub-traces on *Rhea* that we carefully filtered out by selecting time samples for which the T-Test was greater than some threshold, in absolute value. We then used unsupervised clustering to classify these sub-traces into two distinct subsets in order to differentiate the cases $k_i = 0$ and $k_i = 1$. For this step, we used the Expectation-Maximization algorithm (`Scikit-learn` GaussianMixture class[8]). If the classification is successful, the number of sub-traces in each subsets should match the respective probabilities given by Table 2, i.e. $P(k_i = 0) = 3/8$, $P(k_i = 1) = 5/8$. This was indeed the case for some T-Test threshold values. Nonetheless, since we were able to deduce nonce values from the private key of our experiments on *Rhea*, we could precisely evaluate the matching success rate for $k_i = 0$. Table 3 summarizes the matching success rates for $k_i = 0$ on the 4000×128 sub-traces of *Rhea* for various threshold values. For a threshold t, we give the resulting sub-traces length (# points) after samples selection and signal processing, the probability of success when a sub-trace is sent to the set $k_i = 0$ and the overall number of sub-traces labeled $k_i = 0$ over the 4000×128 sub-traces. More precisely, the clustering algorithm will choose two cluster centers (*i.e.* two multivariate Gaussian distributions) and output, for each sub-trace, the probability of fitting each cluster. We call confidence level the probability for a sub-trace to fit the cluster corresponding to $k_i = 0$. We ran several experiments on *Rhea*'s traces with various threshold values. For the second phase of the attack, we selected the 109714 sub-traces obtained with $t = 11$ for which the clustering algorithm's confidence level is equal or greater than 95% (highlighted in blue in Table 3). At the end of this first part of the attack, we have thus acquired with very high probability roughly $109714/4000 \approx 27.5$ bits per nonce (all located on the upper half of the nonce since they relate to $\text{msb}(\tilde{k}_i)$). The second phase of the attack presented in the next section consists in recovering the unknown part of each nonce in order to deduce the secret key d.

We proceeded similarly with $\text{lsb}(\tilde{k}_i)$ in the hope to gather even more knowledge about the nonces. However, as mentioned before (see Figure 12), the side-channel leakage related to $\text{lsb}(\tilde{k}_i)$ is significantly weaker than the one related to $\text{msb}(\tilde{k}_i)$ and our matching success rates seemed not

$\hat{k}_i$	$P(\hat{k}_i)$	$\tilde{k}_i$	$(k_i k_{129+i})$
1	3/8	1	(01)
		0	(00)
2	5/16	2	(10)
		0	(00)
3	5/16	3	(11)
		0	(00)

Table 2: Information on Scalar Bits from Noise Free Sensitive Leakage

[8] Exact parameters are `GaussianMixture(n_components=2, covariance_type='tied')`

t	sub-trace length	success rate (%)	# sub-traces
10	697	99.0	110054
11	650	99.0	109714
12	591	99.0	108451
13	554	99.0	106990
14	520	99.1	106691
15	484	99.1	105911

Table 3: Results of the clustering algorithm with minimum confidence level set to 0.95.

good enough. We hence decided to drop this (too) noisy information.

To summarize, we will target only bits with value 0 (as value 1 might hide the randomization of a dummy operation) and only in $\mathrm{msb}(\tilde{k}_i)$ since the sensitive leakage happens to be stronger there (in comparison to $\mathrm{lsb}(\tilde{k}_i)$).

5.2 Lattice-based Attack with Partial Knowledge of the Nonces

In [15], Howgrave-Graham and Smart exploited lattice reduction algorithms in order to recover (EC)DSA private keys from the knowledge of only a few bits per nonce. This work was followed by many others that improved the understanding of so-called lattice-based attacks and/or successfully applied variants to practical settings (see *e.g.* [1–3, 5, 7, 13, 14, 16, 23, 24, 26–28, 36, 38]). All these attacks work as follows:

1. Run N ECDSA signatures and record the inputs $h_i = h(m_i)$, the outputs (r_i, s_i) and the known information $\hat{k}_i$ on the nonce k_i. We denote by u_i the unknown part of k_i so that $k_i = \hat{k}_i + u_i$ (**warning:** contrary to the previous sections where k_i denoted the i-th bit of k, we shall now use the subscript notation where k_i designates the nonce of the i-th signature, where $i = 1, \ldots, N$).

2. Rewrite the ECDSA equations $s_i = k_i^{-1}(h_i + r_i d) \bmod q$ (see Section 3.1), as linear equations of the form $A_i u_i + B_i d \equiv C_i \pmod q$, involving the secret key d and the u_i's for $i = 1, \ldots, N$.

3. Build a lattice $\mathcal{L}$ that contains the vector $v = (u_1, u_2, \ldots, u_N)$ (in practice, this vector often contains some extra elements).

4. If the known part $\hat{k}_i$ of k_i is sufficiently large, then the norm of v is small and one can expect to find v by solving an instance of the Shortest Vector Problem (SVP) in $\mathcal{L}$.

As shown in [15], this attack amounts to finding a solution to the so-called *Hidden Number Problem* (HNP) introduced in [4]. The literature mostly considers the case where the known part consists of some of the most significant bits of each nonce. However, a more general setting sometimes referred to as the *Extended Hidden Number Problem* (EHNP), allows the known part to be a sequence of several blocks of consecutive known bits scattered all over the nonce. In this case, the unknown u_i is a vector whose elements are the unknown sections of each nonce. We note $u_i = (u_{i,1}, u_{i,2}, \ldots)$. This more general setting did not draw much attention (important papers are [13–15, 27]) but led to practical attacks nonetheless, mainly in the specific case of w-NAF implementations of the scalar multiplication [7, 23]. Our attack also relies on this Extended version of the HNP.

Following [15], the ECDSA equations $s_i = k_i^{-1}(h_i + dr_i) \bmod q$ can be rewritten as $k_i = A_i d - B_i \pmod q$, with $A_i = s_i^{-1} r_i$ and $B_i = -s_i^{-1} h_i$. If the most significant bits of $A_i d$ and B_i coincide, or equivalently if $A_i d - B_i$ is small, then one can build a lattice $\mathcal{L}$ such that the closest vector to $v = (B_1, \ldots, B_N, 0)$ in $\mathcal{L}$ reveals the nonces $k_1, \ldots, k_N$, hence the private key d. This situation corresponds to the HNP and the solution is obtained by solving an instance of the Closest Vector Problem (CVP). A common variant makes it possible to reduce the problem to an instance of the Shortest Vector Problem (SVP) in $\mathcal{L}$. In general, this so-called embedding technique (due to [18]) provides a better probability of success.

In our case, the known part of the nonces does not correspond to the most significant bits of k_i. Instead, we have

$$k_i = \hat{k}_i + \sum_{j=1}^{\ell_i} u_{i,j} 2^{\lambda_{i,j}}, \tag{1}$$

where the bits that form the known part $\hat{k}_i$ split the nonces k_i into ℓ_i unknown parts $u_{i,j}$.

For the lattice reduction algorithm (LLL or BKZ) to be successful, the side-channel acquisition phase should provide enough information on the nonces. Notably, over all recorded signatures, the number of known bits should be large enough. It was commonly assumed that this number must be larger than the bitlength of the secret[9]. Yet, it is worth mentioning that very recently, M. Albrecht and N. Heninger managed to slightly break this so-called *information theoretic limit* [1] using a sieve algorithm (and, with less success, an enumeration algorithm). Moreover, and at the price of some rather expensive computations, they showed that 3 known bits by nonce are sufficient in practice to solve HNP when most recent attacks necessitated at least 4 known bits [16, 24].

Based on the above observations and after a few experiments on *Rhea*, we opted for a strategy that we detail in Section 5.3. As explained next, we filtered out the 4000 recorded signatures in order to keep only those for which the known part $\hat{k}_i$ was a block of 5 consecutive zero bits so that

$$k_i = u_{i,2} 2^{\lambda_i} + 0 \times 2^{\mu_i} + u_{i,1} \tag{2}$$

[9] i.e. the bitlength of the group order: 256 in our case.

where $\lambda_i = \mu_i + 5$ is the index of the most significant unknown part of k_i. The least significant part $u_{i,1}$ has index 0, i.e. it coincides with the least significant bits of k_i.

We then used equation 2 to build a lattice that contains a short vector whose elements include the unknown parts $u_{i,1}, u_{i,2}$. Using this information, it was then easy to reconstruct the nonces k_i, notably k_1, and therefore the private key d.

We applied several optimizations to increase both the efficiency and probability of success of the attack. In particular, we removed the secret key d from the equations, we used the already mentioned embedding technique, and we adapted the trick presented in [27] and recalled in [1, 24] that consists of shifting the interval of the unknown parts $u_{i,j}$ from $[0, U_{i,j}]$ to $[-U_{i,j}/2, U_{i,j}/2]$ to the case EHNP. The details of our optimization and lattice construction are given in Appendix A.

5.3 Touchdown on *Rhea*

As seen in Section 5.1, we recorded input and output data on 4000 ECDSA signatures. Our pruning process and parameters allowed us to select 109714 sub-traces (iterations) corresponding to a zero bit with very high probability (99%). This represents an average of ≈ 27.5 known zero bits per nonce over the 4000 signatures. We also know that these zero bits are all located in the upper half of the nonces (see Section 4). Unfortunately, the vast majority of this information is not easily exploitable. Indeed, an elementary, yet rather conservative equation from [13] tells us that in the case of EHNP, a known block of less than three consecutive bits is not helping. In fact, it is deteriorating the success rate by increasing the lattice dimension for no gain. According to [13], there should be at least three (resp. two) known blocks of 3 bits (resp. 4 bits) per nonce for the attack to be successful. Thus, after a few experiments, we decided to seek nonces containing a single block of at least five consecutive zero bits. We ended-up with 180 nonces, out of which only 5 included a wrongly estimated known block.

In simulation, with such a configuration and using LLL for the lattice reduction, 80 error free signatures are enough to get about 50% chances to find the secret. Based on these simulations, we completed the attack on *Rhea* using a brute-force strategy: we randomly selected 80 nonces among the 180 available to define the lattice and run the reduction algorithm until the secret key was found.

Using LLL, each trial attack with 80 signatures took about 100 seconds to complete (on a 3,3GHz Intel Core i7, with 16GB RAM). Eventually, the secret key was recovered after only a few tens of trials.

In the purpose of completeness, we provide in Appendix B the attack success rate estimations in simulation with the BKZ algorithm (for various block sizes). As expected, BKZ offers much better results than LLL, even allowing us to consider 4-bit known blocks instead of 5-bit blocks, significantly decreasing the overall attack data complexity[10].

5.4 Touchdown on *Titan*

We launched the attack on the *Google Titan Security Key* following the exact same trajectory. First, we did our best to locate the EM probe at the same spatial position and with the same orientation (see Figure 4). We acquired 6000 side-channel traces during the execution of the U2F `authentication request` command (details of the acquisition campaign are similar to *Rhea*'s, see Table 1, but for the number of acquisitions and then for the acquisition time that took about 6 hours).

Re-alignment, samples selection and signal processing We applied exactly the same process than for *Rhea* (the same four signal peaks were clearly visible). Once re-aligned around the four signal peaks, we used the T-Test results from *Rhea* to select the time samples and we applied the same signal processing on the sub-traces.[11]

Unsupervised clustering Again, we applied the same Expectation-Maximization algorithm than for *Rhea*. As mentioned earlier, we were optimistic about the correctness of the clustering process since the sizes of the two output clusters were proportional to the expected ratios $(3/8, 5/8)$. We then brute-forced the T-Test threshold for time samples selection and eventually selected $t = 8$ (for *Rhea* it was $t = 11$). After signal processing and samples selection, the sub-trace length with this threshold was 854.

Pruning and nonces selection We chose the highest confidence level that preserved sufficiently many nonces with 5 or more consecutive zeros. Since we had more traces than for *Rhea*, we were able to increase the confidence level to 0.98. We ended-up with 156 nonces with a block of at least 5 consecutive zero bits.

Key recovery attack We ran our EHNP solver on random subsets of size 80 among the 156 selected nonces. The attack was successful after only a few tens of attempts.

Post analysis From the secret key, we can compute the values of the nonces and verify that, among the 156 selected nonces, 7 were erroneous. The attack was then a little more challenging than for *Rhea* but still possible. Again, as shown

[10] The use of a sieve algorithm, as in [1], would certainly improve further these results.

[11] By reusing *Rhea*'s T-Test results for selecting the time samples for *Titan*, we assumed that *Rhea* and *Titan* share the same clock frequency and instructions order. These are not strong hypotheses since the clock frequency can be easily checked and the NXP cryptographic library version seems to be the same on both devices.

in Appendix B, the use of BKZ with medium or large block size would do the work with much fewer nonces.

Time required to replay the attack Once the attacker get hold of the *Titan* device, it should take less than 10 hours to replay the side-channel acquisition: 2 hours for preparing the device, 1 hour for preparing the side-channel acquisition setup, 6 hours for the side-channel acquisition and 1 hour for repackaging the device. After returning the device to the victim, the key recovery can then be performed offline in less than one day.

6 A Crucial Observation

During the post analysis, we ran a lot of simulations on various instances of the EHNP. In particular, we observed that the success rate of the attack, and the minimum number of signatures required to reach a given success rate, differ between the contexts of *Rhea* / *Titan* and that of random instances of the EHNP.[12] We made the following crucial observation:

The success rate of the attack increases when the positions (bits) covered by the known blocks of the nonces correspond to positions where the group order is either all-zeros or all-ones. As a consequence, the number of signatures required to complete the attack can be greatly reduced in this case.

We realized that the *Rhea* / *Titan* implementation of the scalar multiplication with the *comb* method, together with the fact that the observed leakage on the most significant bits of the nonces are easier to infer, were crucial in the success of our attack. Indeed, all the blocks of 5 consecutive zero bits were located between the bit indices 129 and 250 (assuming index 0 corresponds to the lsb). And since the elliptic curve used in the FIDO U2F protocol has a structured order q, the bits of q at these positions consist of large sequences of zeros and ones.

To illustrate this, we conducted the following experiment: for every possible bit index i ranging from 1 to 250, we ran the attack[13] with the 5-bit known blocks set at index i for all the nonces.

The success rates of these 250 attacks is plotted in red in Figure 13. On the same figure, the dashed blue curve corresponds to the function δ_q, defined as follows:

$$\delta_q(i) = \begin{cases} 1 & \text{if } q \text{ has 5 consecutive 0s or 1s at bit position } i \\ 0 & otherwise \end{cases}$$

The order of the curve NIST P-256 used for our attack contains 2 runs of 5 consecutive ones in its lower part, namely at indices 26 and 108.[14] The higher part is decomposed into 3 long runs of ones and zeros. The first 2 peaks on Figure 13 exactly correspond to the two runs of ones in the lower part. Then starting at index 128 (first long run of ones), the success rate reaches 100% except when the 5-bit window meets the transitions between the runs of ones and that of zeros. The correlation between the success rate and the bit values at these exact locations clearly indicate that there exists a strong correlation between $\delta_q(i)$ and the attack success rate for a known block at position i.

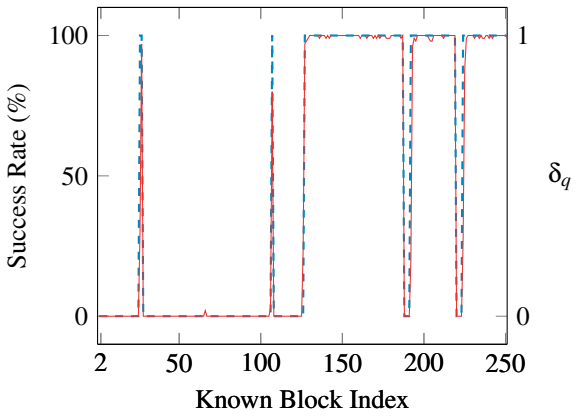

Figure 13: Comparison between the success rate of EHNP with 60 nonces and a single known block of 5 bits at a given bit index (in red) and δ_q (in blue, dashed)

To the best of our knowledge, this phenomena has not been observed before and we believe it opens new directions of research.

First of all, it tends to show that structured elliptic curves, for which the order contains large sequences of zeros or ones happen to be more vulnerable to lattice-based attacks than unstructured elliptic curves. Interestingly enough, we already know in other contexts that these elliptic curves are not the best choice as far as side-channel analysis is concerned as they require more expensive countermeasures (see *e.g.* [35]). It is worth mentioning that structured elliptic curves is a very common choice in real-world protocols (like FIDO or Bitcoin).

At this point, we do not have any theoretical explanation for this observation. We know that finding a short vector in a lattice necessitates that vector to be sufficiently short relative to the lattice volume (at least, this is how these lattice-based attacks were theoretically explained in the first place [15]). However, we did not observe major differences in the norms of the short vector solution or the lattice volume in the favorable[15] and less favorable cases. Hence, to understand the influence of the elliptic curve order on the difficulty to solve SVP in the EHNP lattice, a deeper exploration of the inner

[12] on the same curve and where each nonce contains a single block of 5 consecutive zero bits enclosed by two unknown parts.

[13] Using 60 nonces and lattice reduction algorithm BKZ with block size 25

[14] In fact, at index 26, the order contains a run of 6 consecutive ones. Thus to be precise we should have said "3 runs of 5 consecutive ones at indices 26, 27 and 108".

[15] the so-called favorable case is when the known block position i corresponds to $\delta_q(i) = 1$.

structure of the lattice is needed. The question remains thus open.

We believe that a clear understanding of this surprising behaviour might be a key to improve lattice-based attacks on ECDSA. Indeed, if we understand the structural difference between the favorable and the generic cases, one might be able to adapt the lattice structure in the general case and significantly improve the attack success rate.

7 Attack Mitigations

Several measures can be implemented to thwart the proposed attack, at different levels.

7.1 Hardening the NXP P5x Cryptographic Library

Straightforward ways for hardening the NXP P5x cryptographic library:

- blinding of the scalar. This does not remove the sensitive leakage but makes the attack much harder (as shown in [13]). For instance, by addition of a random factor of the curve order (the bit length of the random number should be at least half the bit length of the curve order);
- re-randomizing the table lookup of precomputed points in the comb implementation at each new access and hence completely remove the sensitive leakage.

7.2 Use the FIDO U2F Counter to Detect Clones

As explained in section 8.1 of [10], the counter *may* be used as a signal for detecting cloned U2F devices. Thus if a *relying party* of an application protected with FIDO U2F receives a cryptographically correct `authentication response message`, but with a counter value smaller or equal to the previous counter value recorded, it means that a clone of the U2F device has been created and used. Then the *relying party* should not validate the authentication request, and lock the account.

This countermeasure would reduce the usability of the clone to a unique time after giving the security key back to the legitimate user. Once the clone has been used (say one month after the attack), the account will be locked by the next access from the legitimate user.

Note that this protection would have to be implemented by each *relying party*, independently of the FIDO U2F device.

References

[1] Martin R. Albrecht and Nadia Heninger. On Bounded Distance Decoding with Predicate: Breaking the "Lattice Barrier" for the Hidden Number Problem. Cryptology ePrint Archive, Report 2020/1540, 2020. https://eprint.iacr.org/2020/1540.

[2] Alejandro Cabrera Aldaya, Cesar Pereida Garcia, and Billy Bob Brumley. From A to Z: Projective Coordinates Leakage in the Wild. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):428–453, Jun. 2020.

[3] Naomi Benger, Joop van de Pol, Nigel P. Smart, and Yuval Yarom. "Ooh Aah... Just a Little Bit" : A Small Amount of Side Channel Can Go a Long Way. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *LNCS*, pages 75–92. Springer, 2014.

[4] Dan Boneh and Ramarathnam Venkatesan. Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 129–142, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[5] Billy Bob Brumley and Nicola Tuveri. Remote Timing Attacks Are Still Practical. In Vijay Atluri and Claudia Díaz, editors, *Computer Security - ESORICS 2011, Leuven, Belgium, September 12-14, 2011. Proceedings*, volume 6879 of *LNCS*, pages 355–371. Springer, 2011.

[6] Jean-Sébastien Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *LNCS*, pages 292–302. Springer, 1999.

[7] Shuqin Fan, Wenbo Wang, and Qingfeng Cheng. Attacking OpenSSL Implementation of ECDSA with a Few Signatures. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1505–1515, New York, NY, USA, 2016. Association for Computing Machinery.

[8] Feitian. Feitian website. https://www.ftsafe.com. [online; accessed 1-June-2021].

[9] FIDO Alliance. FIDO U2F Raw Message Formats. https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-raw-message-formats-v1.2-ps-20170411.html. [online; accessed 1-June-2021].

[10] FIDO Alliance. Universal 2nd Factor (U2F) Overview. https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-overview-v1.2-ps-20170411.pdf. [online; accessed 1-June-2021].

[11] Friedrich Beck. *Integrated Circuit Failure Analysis: A Guide to Preparation Techniques*. John Wiley & Sons, 1998.

[12] Google. Google Titan Key. https://cloud.google.com/titan-security-key/. [online; accessed 1-June-2021].

[13] Dahmun Goudarzi, Matthieu Rivain, and Damien Vergnaud. Lattice Attacks Against Elliptic-Curve Signatures with Blinded Scalar Multiplication. In Roberto Avanzi and Howard M. Heys, editors, *Selected Areas in Cryptography - SAC 2016, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers*, volume 10532 of *LNCS*, pages 120–139. Springer, 2016.

[14] Martin Hlavác and Tomás Rosa. Extended Hidden Number Problem and Its Cryptanalytic Applications. In Eli Biham and Amr M. Youssef, editors, *Selected Areas in Cryptography - SAC 2006, Montreal, Canada, August 17-18, 2006 Revised Selected Papers*, volume 4356 of *LNCS*, pages 114–133. Springer, 2006.

[15] Nick Howgrave-Graham and Nigel P. Smart. Lattice Attacks on Digital Signature Schemes. *Des. Codes Cryptogr.*, 23(3):283–290, 2001.

[16] Jan Jancar, Vladimir Sedlacek, Petr Svenda, and Marek Sys. Minerva: The Curse of ECDSA Nonces (Systematic Analysis of Lattice Attacks on Noisy Leakage of Bit-Length of ECDSA Nonces). *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):281–308, 2020.

[17] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, 2001.

[18] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.

[19] Langer. ICR HH 250-75. https://www.langer-emv.de/en/product/near-field-microprobes-icr-hh-h-field/26/icr-hh250-75-near-field-microprobe-0-5-mhz-to-2-ghz/105, 2019. [online; accessed 1-June-2021].

[20] Langer. ICR HH 500-6. https://www.langer-emv.de/en/product/near-field-microprobes-icr-hh-h-field/26/icr-hh500-6-near-field-microprobe-2-mhz-to-6-ghz/108, 2019. [online; accessed 1-June-2021].

[21] Chae Hoon Lim and Pil Joong Lee. More Flexible Exponentiation with Precomputation. In Yvo G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, pages 95–107, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[22] S Mangard, ME Oswald, and T Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007. Other identifier: 0387308571.

[23] Gabrielle De Micheli, Rémi Piau, and Cécile Pierrot. A Tale of Three Signatures: Practical Attack of ECDSA with wNAF. In Abderrahmane Nitaj and Amr M. Youssef, editors, *Progress in Cryptology - AFRICACRYPT 2020, Cairo, Egypt, July 20-22, 2020, Proceedings*, volume 12174 of *LNCS*, pages 361–381. Springer, 2020.

[24] Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, and Nadia Heninger. TPM-FAIL: TPM meets Timing and Lattice Attacks. In *USENIX Security 20*, Boston, MA, August 2020. USENIX Association.

[25] MOUSER. NXP A700x datasheet, secure authentication microcontroller. https://www.mouser.fr/datasheet/2/302/a700x_fam_sds-1187735.pdf. [online; accessed 1-June-2021].

[26] Elke De Mulder, Michael Hutter, Mark E. Marson, and Peter Pearson. Using Bleichenbacher's Solution to the Hidden Number Problem to Attack Nonce Leaks in 384-bit ECDSA: extended version. *J. Cryptogr. Eng.*, 4(1):33–45, 2014.

[27] NS02 Q. Nguyen and Igor E. Shparlinski. The Insecurity of the Digital Signature Algorithm with Partially Known Nonces. *J. Cryptol.*, 15(3):151–176, 2002.

[28] Phong Q. Nguyen and Igor E. Shparlinski. The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces. *Des. Codes Cryptogr.*, 30(2):201–217, 2003.

[29] NIST. FIPS 186-2, Digital Signature Standard (DSS). https://csrc.nist.gov/csrc/media/publications/fips/186/2/archive/2000-01-27/documents/fips186-2.pdf, 2001. [online; accessed 1-June-2021].

[30] NXP. NXP LPC11U2x datasheet, 32-bit ARM Cortex-M0 microcontroller. https://www.nxp.com/docs/en/data-sheet/LPC11U2X.pdf. [online; accessed 1-June-2021].

[31] NXP. NXP SmartMX family brochure. https://www.nxp.com/docs/en/brochure/75017515.pdf. [online; accessed 1-June-2021].

[32] Oracle. JavaCard Connected Platform Specifications 3.0.1. https://www.oracle.com/java/technologies/javacard/platform-

specification-3-0-1.html. [online; accessed 1-June-2021].

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[34] Pico Technology. PicoScope 6000 Series datasheet. https://www.picotech.com/download/datasheets/PicoScope6000CDSeriesDataSheet.pdf, 2019. [online; accessed 1-June-2021].

[35] Thomas Roche, Laurent Imbert, and Victor Lomné. Side-channel attacks on blinded scalar multiplications revisited. In Sonia Belaïd and Tim Güneysu, editors, *CARDIS 2019, Prague, Czech Republic, November 11-13, 2019, Revised Selected Papers*, volume 11833 of *LNCS*, pages 95–108. Springer, 2019.

[36] Keegan Ryan. Return of the Hidden Number Problem.: A Widespread and Novel Key Extraction Attack on ECDSA and DSA. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(1):146–168, Nov. 2018.

[37] Thorlabs. Manual 3-axes Stage PT3/M. https://www.thorlabs.com/thorproduct.cfm?partnumber=PT3/M#ad-image-0, 2019. [online; accessed 1-June-2021].

[38] Samuel Weiser, David Schrammel, Lukas Bodner, and Raphael Spreitzer. Big Numbers - Big Troubles: Systematically Analyzing Nonce Leakage in (EC)DSA Implementations. In *USENIX Security 20*, pages 1767–1784. USENIX Association, August 2020.

[39] B. L. Welch. The Generalization of 'Student's' Problem when Several Different Population Variances are Involved. *Biometrika*, 34(1/2):28–35, 1947.

[40] Carolyn Whitnall and Elisabeth Oswald. A critical analysis of ISO 17825 ('testing methods for the mitigation of non-invasive attack classes against cryptographic modules'). In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, volume 11923 of *LNCS*, pages 256–284. Springer, 2019.

A Details of the Lattice Construction

Let us jump back to the modular equations given in Section 5.2 that involves the nonces k_i, the secret key d and the public data h_i and (s_i, r_i):

$$k_i = A_i d - B_i \pmod{q},$$

where $A_i = s_i^{-1} r_i$ and $B_i = -s_i^{-1} h_i$ for each ECDSA signature $(i = 1, \ldots, N)$.

We first remove the secret key d from the equations. This removes one unknown and one equation and results in a lattice of smaller dimension (thus improving the efficiency of the lattice reduction). By subtracting the equation for $i = 1$ to the others we obtain $N-1$ equations of the form

$$k_i = A_i k_1 - B_i \pmod{q},$$

where $A_i = s_1 r_1^{-1} s_i^{-1} r_i$ and $B_i = r_1^{-1} s_i^{-1} r_i h(m_1) - s_1^{-1} h(m_i)$ for $i = 2, \ldots, N$.

Now, writing the nonce k_i as in equation 1, with its known part $\hat{k}_i$ and its unknown parts $\sum_{j=1}^{\ell_i} u_{i,j} 2^{\lambda_{i,j}}$ we obtain the following $N-1$ congruences:

$$u_{i,1} \equiv \sum_{j=2}^{\ell_i} \sigma_{i,j} u_{i,j} + \sum_{j=1}^{\ell_1} \tau_{j,i} u_{1,j} + \gamma_i \pmod{q},$$

where $\tau_{j,i} = A_i 2^{\lambda_{1,j} + \lambda_{i,1}}$, $\sigma_{i,j} = -2^{\lambda_{i,j} - \lambda_{i,1}}$ and $\gamma_i = -2^{-\lambda_{i,1}} (\hat{k}_i - A_i \hat{k}_1 + B_i)$.

The unknown blocks $u_{i,j}$ represent sequences of unknown bits of the nonces, we then know an upper bound $U_{i,j} = 2^{\mu_{i,j}}$ for each of them (where $\mu_{i,j}$ is the size of the sequence of unknown bits). As explained in section 5.2, we can optimize our attack by shifting the interval of each $u_{i,j}$ from $[0, U_{i,j}]$ to $[-U_{i,j}/2; U_{i,j}/2]$. To this end, we set $u'_{i,j} = u_{i,j} - U_{i,j}/2$ and we report this change of variable to the previous equations, to get:

$$u'_{i,1} \equiv \sum_{j=2}^{\ell_i} \sigma_{i,j} u'_{i,j} + \sum_{j=1}^{\ell_1} \tau_{j,i} u'_{1,j} + \gamma'_i \pmod{q},$$

where $\gamma'_{i,j} = \sum_{j=2}^{\ell_i} \sigma_{i,j} \frac{U_{i,j}}{2} + \sum_{j=1}^{\ell_1} \tau_{j,i} \frac{U_{1,j}}{2} + \gamma_i - \frac{U_{i,1}}{2}$ for $i = 2, \ldots, N$.

Finally, using the embedding technique described in [18] which is known to be more efficient, we build the lattice $\mathcal{L}$ given by the following base

$$M_{\mathcal{L}} = \begin{pmatrix} q & 0 & 0 & 0 & & & & \\ 0 & \ddots & \vdots & \vdots & & & & \\ 0 & & q & 0 & & 0 & & \\ \tau_2 & \ldots & \tau_t & 1 & & & & \\ \sigma_2 & & 0 & & \ddots & & & \\ & \ddots & \vdots & & & \ddots & & \\ & & \sigma_t & & 0 & & \ddots & \\ \gamma'_2 & \ldots & \gamma'_t & & & & & 1 \end{pmatrix} \times D$$

The coefficients τ_i represent the column vector $(\tau_{j,i})_{j=1}^{\ell_1}$, the coefficients σ_i represent the column vector $(\sigma_{i,j})_{j=1}^{\ell_i}$ and

D represent the diagonal matrix defined by:

$$D = diag(J_{2,1}, \ldots, J_{t,1}, J_{1,1}, \ldots, J_{1,\ell_1}, J_{2,2}, \ldots, J_{2,\ell_2}, \ldots, J_{t,2}, \ldots, J_{t,\ell_t}, J/2),$$

where $J = 2^{\lceil \log_2 q \rceil}$ and $J_{i,j} = J/U_{i,j} \in \mathbb{Z}$.

By solving SVP in $\mathcal{L}$, we hope to find the following short vector:

$$v = (u'_{2,1}J_{2,1}, \ldots, u'_{t,1}J_{t,1}, u'_{1,1}J_{1,1}, \ldots, u'_{1,\ell_1}J_{1,\ell_1}, u'_{2,2}J_{2,2}, \ldots, u'_{2,\ell_2}J_{2,\ell_2}, \ldots, u'_{t,2}J_{t,2}, \ldots, u'_{t,\ell_t}J_{t,\ell_t}, J/2),$$

and from v retrieve the secret key d. The smaller the norm of v, the more chance we have to find it using a lattice reduction algorithm. Shifting the interval where the $u_{i,j}$'s live allows to search for a vector v whose squared norm is bounded by $\sum_{\substack{1 \le i \le t, \\ 1 \le j \le \ell_i}} (J/2)^2$. Without this re-centering optimization, the squared norm of the vector v would have been bounded by $\sum_{\substack{1 \le i \le t, \\ 1 \le j \le l_i}} J^2$ which is 4 times bigger.

Figure 14 shows the impact of this re-centering optimization in the *Titan* case (*i.e.* a single block of 5 known bits randomly located in the upper half part of the nonces). All our experiments were done using the BKZ reduction algorithm with a blocksize 25.

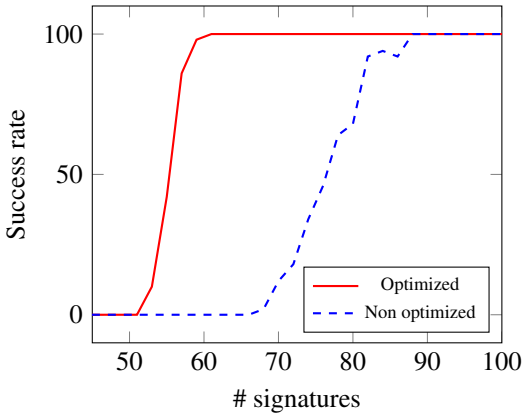

Figure 14: Comparison of the success rate of the optimized and non-optimized attack on the ECDSA signature scheme with the P-256 curve

B Attack Success Rate with BKZ

Our initial attack targeted 80 ECDSA signatures and used LLL for the lattice reduction since early simulations showed that with 80 signatures we could expect up to 50% success rate[16]. It is however well known that BKZ can perform better than LLL. We then conducted further experiments to evaluate how BKZ could improve the data complexity of our attack. Figures 15 provides the success rates for BKZ with various medium blocksizes, these results clearly outperform the ones with LLL since with blocksize 35 one obtains 100% success rate with less than 60 signatures.

In Figure 16, the success-rates relate to similar experiments but where the number of known bits is reduced to 4 (instead of 5). With a blocksize of 35, about 75 signatures are sufficient to reach 100% success-rate. Using these results in the *Titan* attack would drastically reduce the number of ECDSA observations.

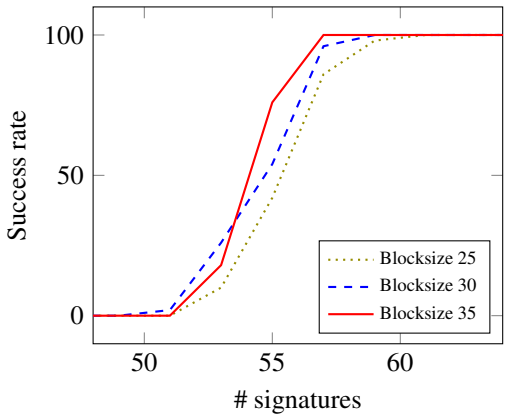

Figure 15: Success rates of the optimized attack using BKZ, in the *Titan* case with 5-bit known block.

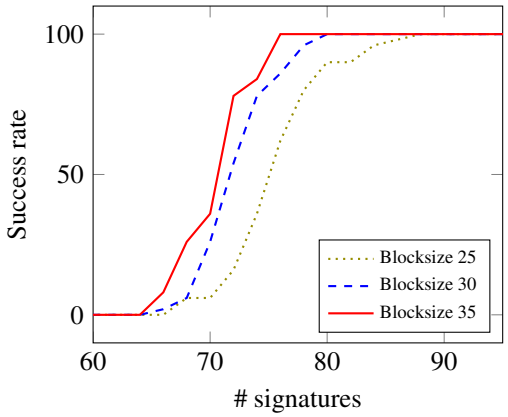

Figure 16: Success rates of the optimized attack using BKZ, in the *Titan* case with 4-bit known block.

[16] in the *Titan* case, meaning when the attacker knows 5 consecutive bits located in the upper-half of each nonce.