# A Side Journey to Titan

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

Thomas Roche[1] & Victor Lomne[1] & Camille Mutschler[1,2] & Laurent Imbert[2]

[1]NinjaLab, Montpellier, France

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


April 11, 2022
Leuven, Belgium

**COSADE 2022**

# Agenda


Context and Preliminaries
- The Google Titan Security Key
- Rhea
- Side-Channel Setup

A Side-Channel Vulnerability in the ECDSA Algorithm
- Reverse-Engineering of ECDSA Signature Algorithm
- Sensitive Leakage in the Scalar Multiplication

A Key-Recovery Attack
- Lattice-based Attacks on ECDSA
- Key Recovery Attack on Rhea
- Key Recovery Attack on Titan

Conclusions

# Study Motivation

<u>CHES 2018 conference in Amsterdam, Netherlands</u>

- Keynote from Elie Bursztein (*Google anti-abuse research team leader*):
  1. *Leveraging Deep-Learning to Perform SCA Attacks against AES Implementations*
  2. Promotion about the new flagship Google security product
     $\rightarrow$ ***Google Titan Security Key***
  3. Elie offered some samples to the attendees


## Trusted hardware

Titan Security Keys are designed to make the critical cryptographic operations performed by the security key strongly resistant to compromise during the entire device lifecycle, from manufacturing through actual use.

The firmware performing the cryptographic operations has been engineered by Google with security in mind. This firmware is sealed permanently into a secure element hardware chip at production time in the chip production factory. The secure element hardware chip that we use is designed to resist physical attacks aimed at extracting firmware and secret key material.

These permanently-sealed secure element hardware chips are then delivered to the manufacturing line which makes the physical security key device. Thus, the trust in Titan Security Key is anchored in the sealed chip as opposed to any other later step which takes place during device manufacturing.

# Product Description

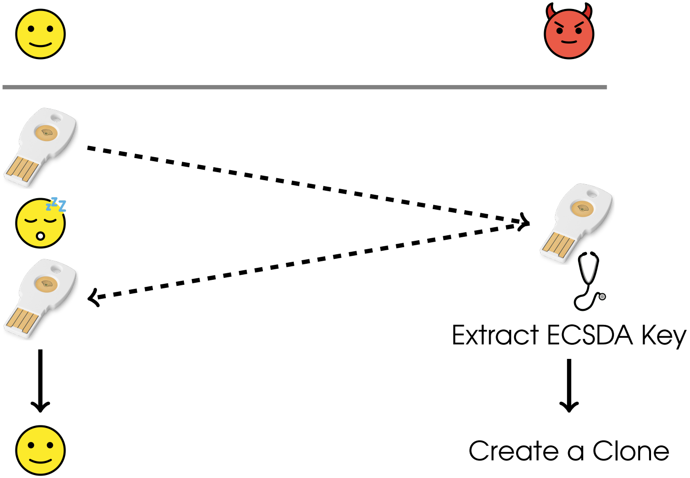
- **Google Titan Security Key**: hardware FIDO U2F token
- **Hardware token** to be used as **two-factor authentication (2FA)** for:
  - Your Google account
  - All apps supporting FIDO U2F protocol (Facebook, Dropbox, GitHub, . . .)
- 3 versions:
  - Micro-USB, NFC and BLE interfaces
  - USB type A and NFC interfaces
  - USB type C interface

# FIDO U2F Protocol in a Nutshell

- **FIDO U2F**: open standard for two-factor authentication
  - Hosted by FIDO alliance
  - Historically developed by Google, Yubico and NXP
- For every application supporting FIDO U2F:
  - Basic authentication based on login & password
    $\rightarrow$ Login & password can easily be stolen (e.g. phishing attack)
  - FIDO U2F enforces user to present FIDO U2F token for each authentication
- FIDO U2F based on **public key cryptography**:
  $\rightarrow$ ECDSA signature with `elliptic curve NIST P256`
- FIDO U2F protocol works in two steps:
  - *Registration* $\rightarrow$ ECDSA key pair generation
  - *Authentication* $\rightarrow$ ECDSA signature

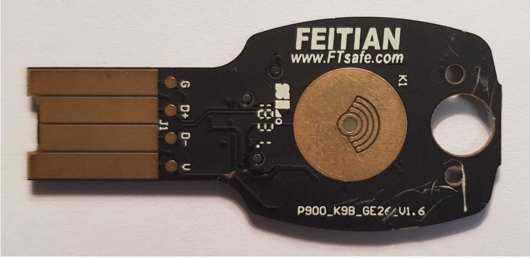
# Side-Channel Attack Scenario

- Initial hyp.: *adversary already stolen victim's account login & password*

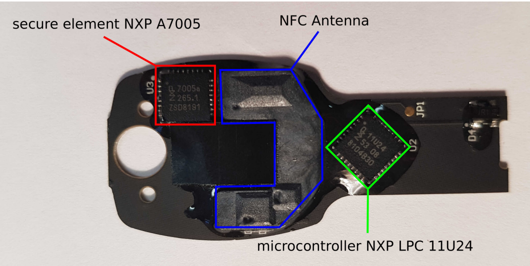


# Google Titan Security Key Teardown

- Recto: HW manufacturer is **Feitian**




- Verso: secure element is **NXP A7005a**




Other FIDO U2F products based on **NXP A700X** chip:

- Yubico Yubikey Neo
- Feitian K9, K13, K21, K40

# FIDO U2F Limitation for Cryptanalysis Attacks

- No way in FIDO U2F protocol to extract an ECDSA private key
  $\rightarrow$ Impossible to transfer user credentials from one FIDO U2F token to another
- Blackbox SCA in unknwown key setup $\rightarrow$ really hard !
- Looking for samples similar to **NXP A700X**
  $\rightarrow$ but with known key setup
- **NXP A700X** datasheet analysis:
  - OS: JCOP 2.4.2 R0.9 or R1 (`JavaCard v3.0.1 / GlobalPlatform v2.1.1`)
  - Technological node: 140$\mu$m
  - CPU: `Secure_MX51`
  - 3-DES, AES, PKC (`FameXE`) co-processors
  - NXP crypto. lib.: RSA up to 2048 bits / ECC up to 320 bits

# Similarities with other NXP Products

- **NXP A700X** characs really similar to several NXP JavaCard smartcards (JCOP)
- Those similar to **NXP A700X** are based on **NXP P5x** chips:


  - NXP J3D081_M59_DF and variants
  - NXP J3A081 and variants
  - NXP J2E081_M64 and variants
  - NXP J3D145_M59 and variants
  - NXP J3D081_M59 and variants
  - NXP J3E145_M64 and variants
  - NXP J3E081_M64_DF and variants
- **NXP P5x / SmartMX**: first generation of NXP secure elements
  → Common Criteria and EMVCo certified (Last CC certif. 2015)
- Several NXP JavaCard smartcards (JCOP) can be purchased on the web

# Rhea

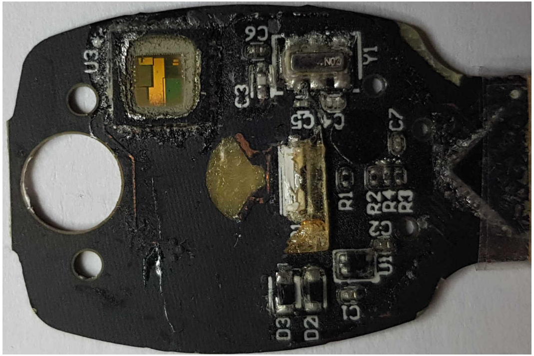
- We chose the **NXP J3D081** Javacard smartcard:
  $\rightarrow$ it has the closest characteristics to those of **NXP A700X**

We called it **Rhea**
in ref. to $2^{nd}$ largest moon of Saturn after **Titan**

- We developed a custom JavaCard applet allowing to:
  - Load a chosen ECDSA private key
  - Perform ECDSA signatures
  - Perform ECDSA signature verifications

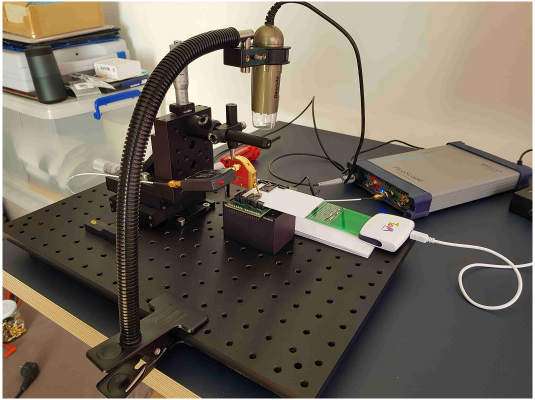
# Titan / Rhea Package Openings

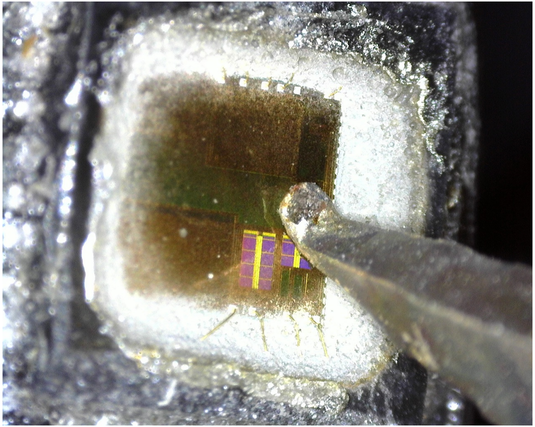
- **Titan's NXP A700X**:
  - wet chemical opening
    *aluminium tape + fuming nitric acid*
  - **Google Titan Security Key** still alive !
- **Rhea**:
  - mecanical opening
    *scalpel + acetone*
  - **Rhea** still alive !




# EM Side-Channel Acquisition Setup (about 10k€)

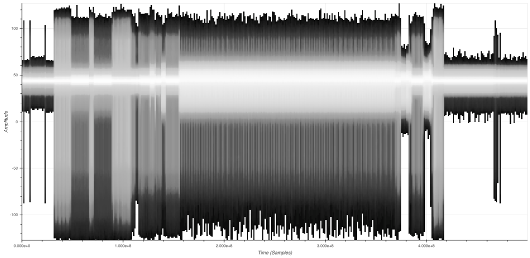
- **EM sensor:** Langer ICR HH 500-6 (diam. 500$\mu$m, freq. BW 2MHz to 6GHz)
- **Manual micro-manipulator:** Thorlabs PT3/M 3 axes (X-Y-Z)
- **Oscilloscope:** PicoScope 6404D, freq. BW 500MHz, SR 5GSa/s



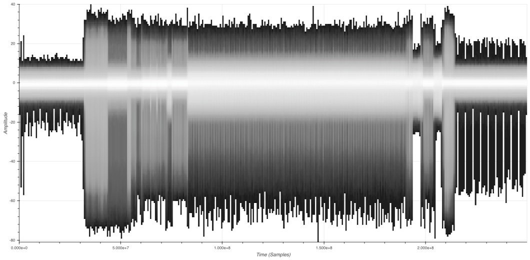


# EM activity of ECDSA signature on Titan / Rhea

- **Titan**




- **Rhea**




- ECDSA signature EM activities on **Titan** and **Rhea** look very similar !

# Agenda

Context and Preliminaries
The Google Titan Security Key
Rhea
Side-Channel Setup

A Side-Channel Vulnerability in the ECDSA Algorithm
Reverse-Engineering of ECDSA Signature Algorithm
Sensitive Leakage in the Scalar Multiplication

A Key-Recovery Attack
Lattice-based Attacks on ECDSA
Key Recovery Attack on Rhea
Key Recovery Attack on Titan

Conclusions

# ECDSA Signature Scheme

- Elliptic curve base point is $G_{(x,y)}$, elliptic curve order is $q$
- Inputs:
  - Long term **secret key d**
  - Input message to sign $h = H(m)$

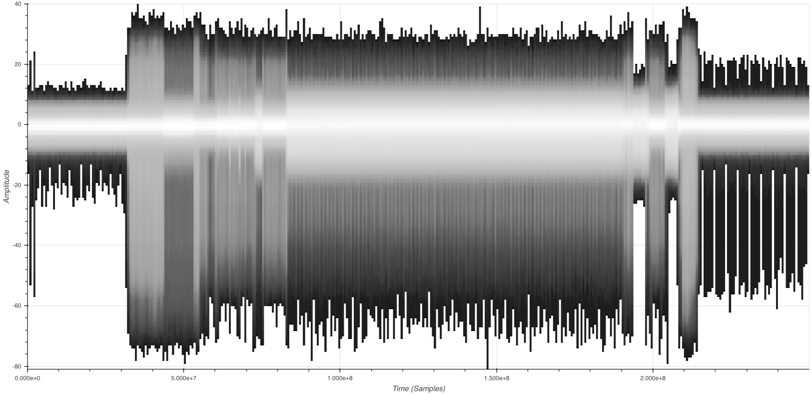
1. randomly generate a **nonce k** in $\mathbb{Z}/q\mathbb{Z}$
2. compute scalar multiplication $Q_{(x,y)} = [\mathbf{k}]G_{(x,y)}$
3. denote by $r$ the $x$-coordinate of $Q$: $r = Q_x$
4. compute $s = \mathbf{k}^{-1}(h + r\mathbf{d}) \bmod q$
5. return $(r, s)$

# ECDSA Signature Scheme

- Elliptic curve base point is $G_{(x,y)}$, elliptic curve order is $q$
- Inputs:
  - Long term **secret key d**
  - Input message to sign $h = H(m)$

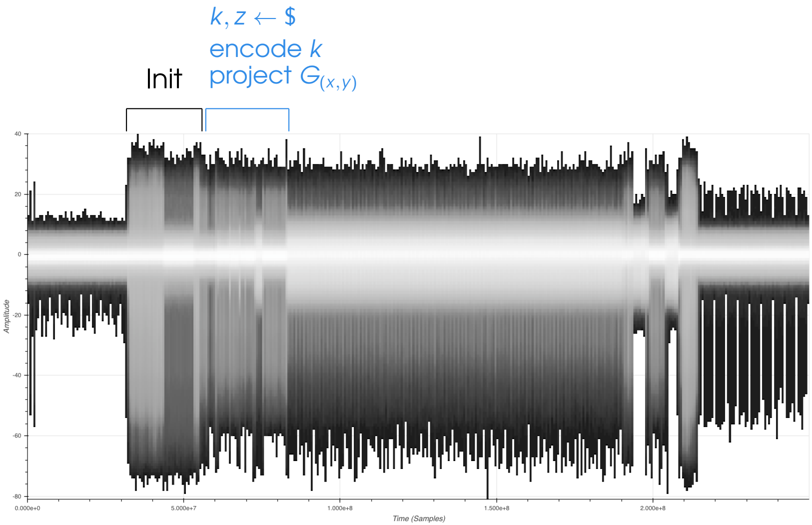
1. randomly generate a **nonce k** in $\mathbb{Z}/q\mathbb{Z}$
2. random projection $G_{(x,y)} \rightarrow G_{(x,y,z)}$
3. compute scalar multiplication $Q_{(x,y,z)} = [\mathbf{k}]G_{(x,y,z)}$
4. inv projection $Q_{(x,y,z)} \rightarrow Q_{(x,y)}$
5. denote by $r$ the $x$-coordinate of $Q$: $r = Q_x$
6. compute $s = \mathbf{k}^{-1}(h + r\mathbf{d}) \bmod q$
7. return $(r, s)$

# Rhea – ECDSA Signature Command – EM Radiations



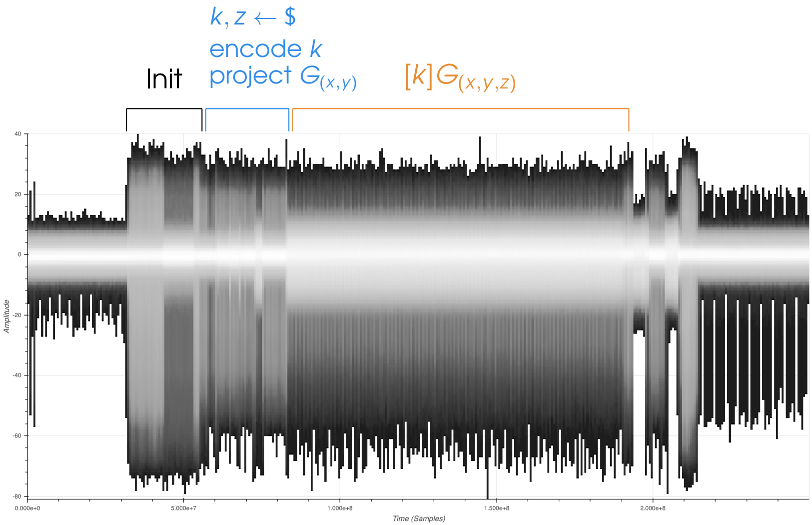

E = P256, SHA-256

# Rhea – ECDSA Signature Command – EM Radiations



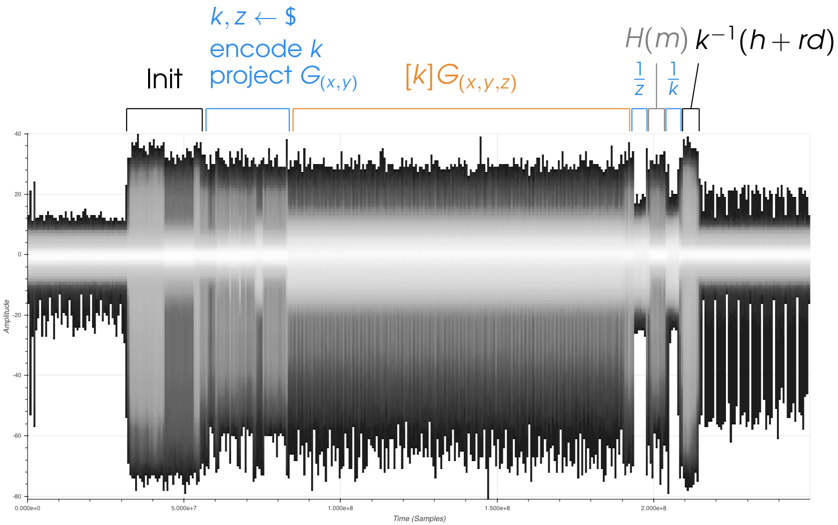

E = P256, SHA-256

# Rhea – ECDSA Signature Command – EM Radiations



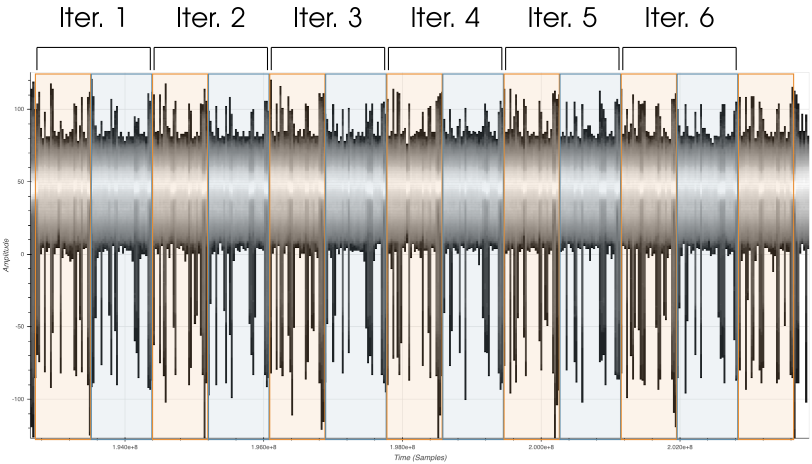

E = P256, SHA-256

# Rhea – ECDSA Signature Command – EM Radiations



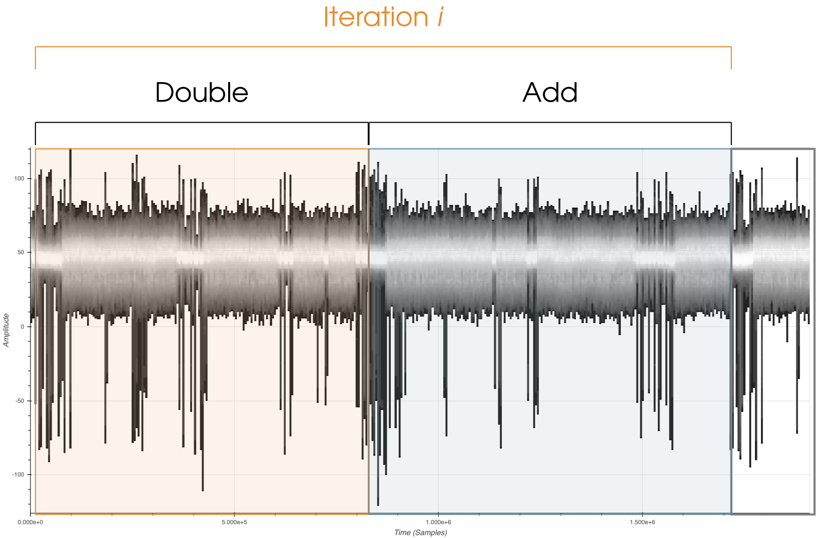

E = P256, SHA-256

# Rhea – ECDSA Signature – Scalar Multiplication

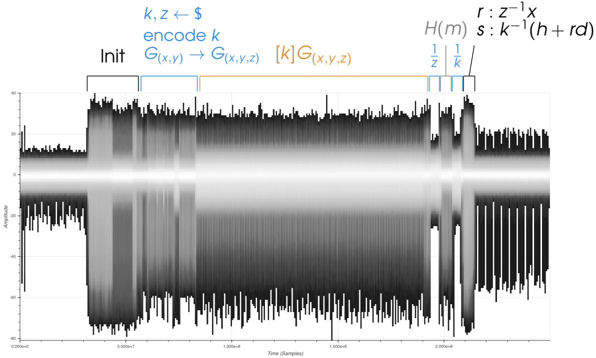



- Regular Implementation (Double & Add Always)
- Exactly 128 iterations

# Rhea – ECDSA Signature – Scalar Mult. Single Iteration



# Rhea – ECDSA Signature – Summary

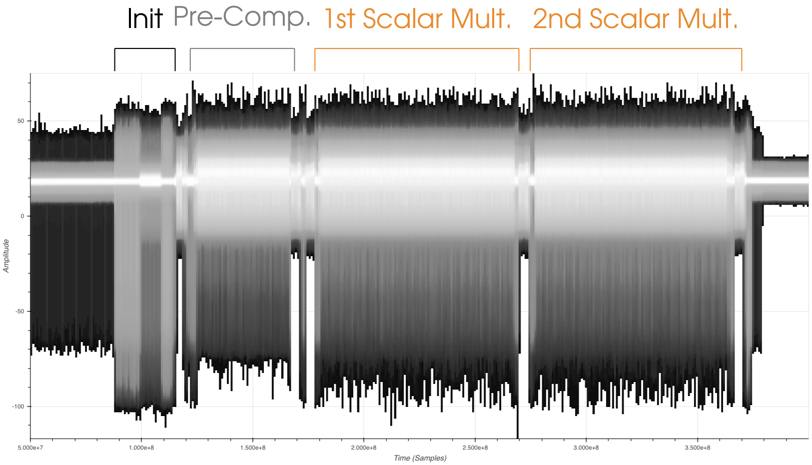



E = P256, SHA-256

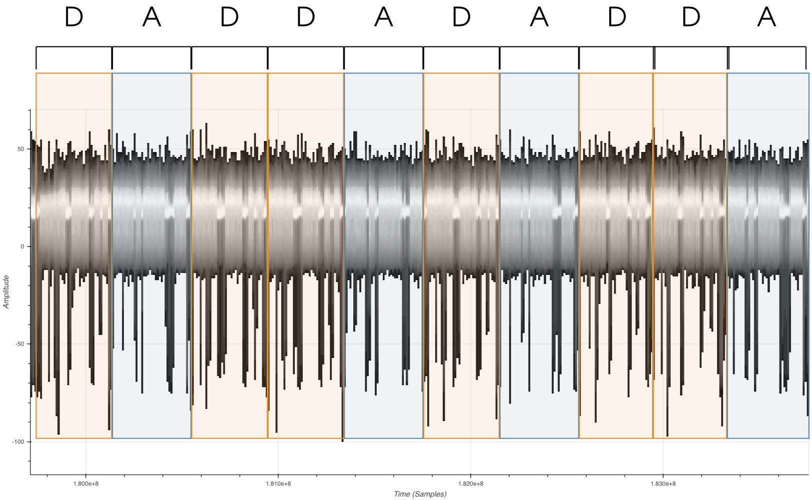
Scalar Multiplication $[k]G$

- **Constant time** algorithm
  → ***Double-and-Add-Always***
- 128 iterations for a 256-bit nonce $k$
  → ***2 bits of*** $k$ ***by iteration***
- Randomized point on projective coordinates

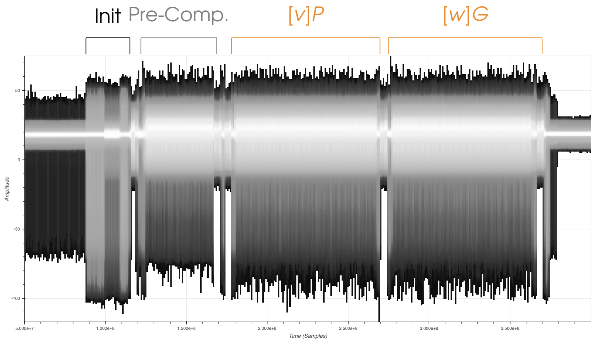
# Rhea – ECDSA Verification Command – EM Radiations



# Rhea – ECDSA Verification – 1st Scalar Mult.



# Rhea – ECDSA Verification – Summary



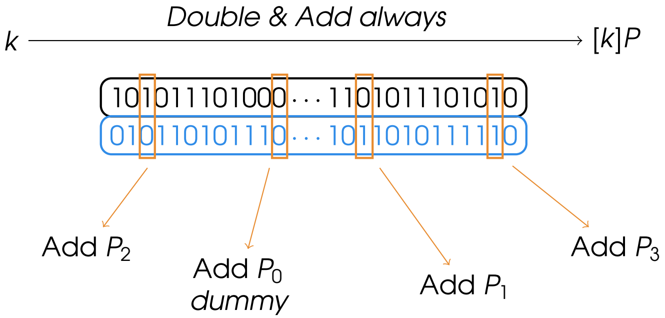

Scalar Multiplications $[v]P$ and $[w]G$

- Scalar mult. are **not constant time** $\rightarrow$ **simple *Double-and-Add***
- Expensive pre-comp. before $[v]P$
- Scalar mult. ***reverse engineering*** $\rightarrow$ **left-to-right *comb method* of width 2**
- Scalars are not blinded.

# Scalar Mult. w. Left-To-Right Comb method of width 2

Precomp: $\mathbf{P_0}$, $\mathbf{P_1} = P$, $\mathbf{P_2} = [2^{128}]P$, $\mathbf{P_3} = P_2 + P_1$

$k =$ 101011101000 ⋯ 11010111101010 | 0101101011110 ⋯ 10110101111110

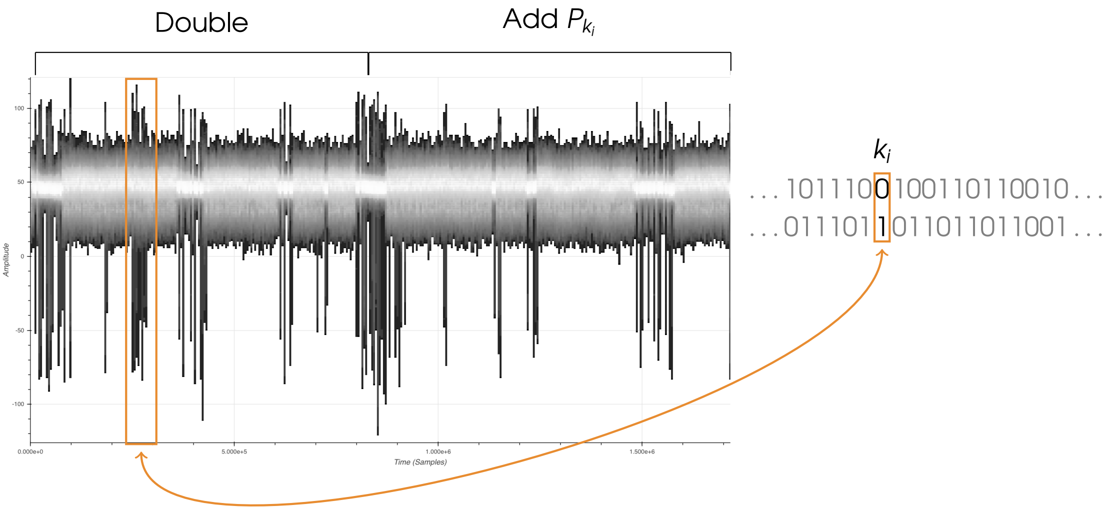


# Exhibiting a Sensitive SCA Leakage

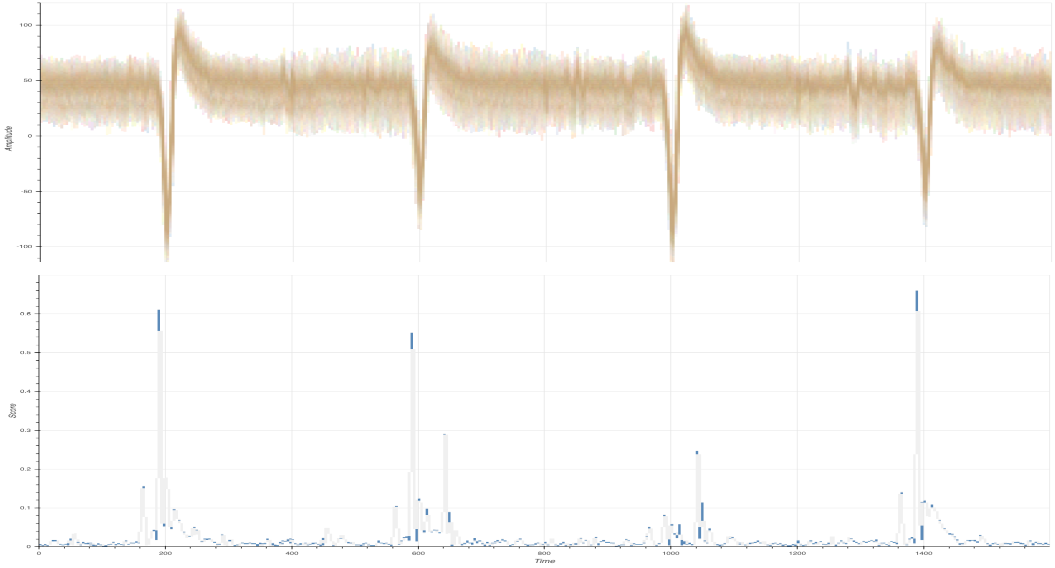
- We can now associate:
  - a **nonce bits pair** to
  - an **EM sub-trace** corresponding to a scalar multiplication **iteration**

- To exhibit a sensitive SCA leakage linked to nonce bits pairs:
  1. Consider $N$ ECDSA signatures EM traces of 128 iterations each
  2. Split each EM trace in sub-traces corresp. to scalar mult. iterations
     $\rightarrow$ We get $128 \times N$ sub-traces
  3. Compute a SCA statistical test (SNR or T-Test) between:
     - the nonce bits pairs &
     - the EM sub-traces
  4. If **significant peak(s)** appear in SNR or T-Test trace
     $\rightarrow$ correlation between nonce bits pairs & EM sub-traces

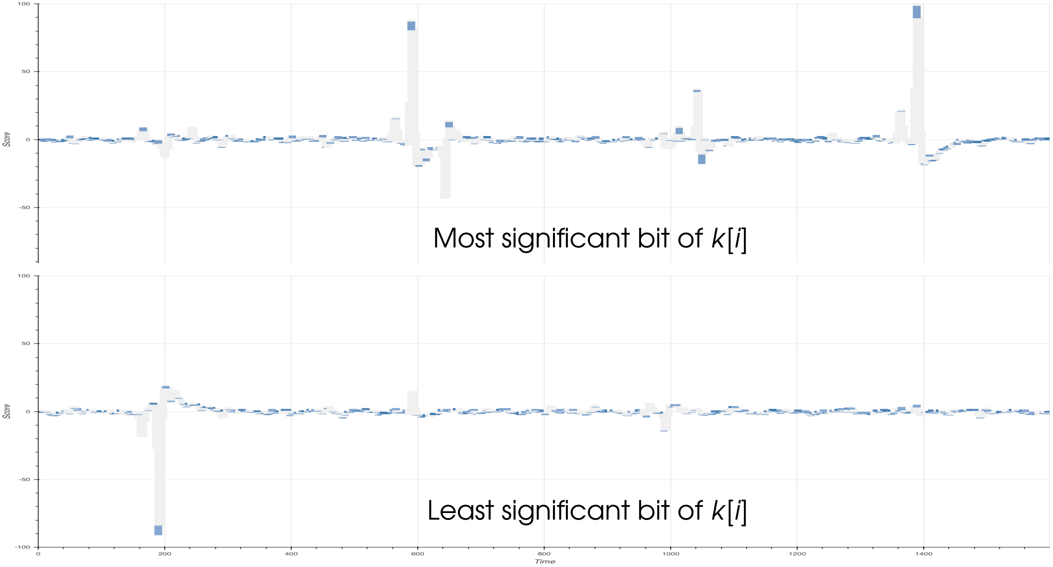
# Rhea – ECDSA Sign. Scalar Mult. Single Iter. – Leakage Area



# Rhea – SCA Leakage – Signal-to-Noise Ratio (SNR) over $k[i]$

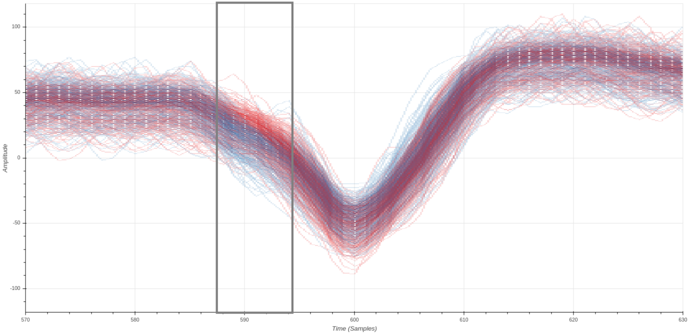


# Rhea – SCA Leakage – T-Test over MSB / LSB of $k[i]$



# Rhea – SCA Leakage – Illustration

## 1000 Superposed Iterations – Zoom in Leakage Area



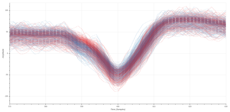

$k_i = \frac{0}{x}$

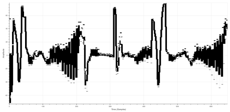
$k_i = \frac{1}{x}$

# Exploiting the SCA Leakage

- We exhibited a SCA leakage linked nonce bits pairs
- But SCA statistical tests require thousands of sub-traces to work
- Yet in ECDSA signature scheme:
  - **New nonce** is randomly picked up for every **new signature**
  - Need to design side-channel process matching w. **high proba.** in **one shot**:
    Single EM sub-trace $\rightarrow$ nonce bit pair value
- Supervised SCA tools for single trace matching:
  - Template Attacks
  - Deep Learning
- Unsupervised SCA tool for single trace matching:
  - **Clustering**
  - Non supervised Deep Learning

# Single Trace Matching Process





99%

0 X

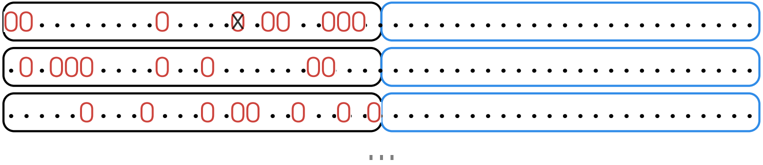
X X

PoI extraction
Signal Processing

Unsupervised Clustering

256-bit Nonces

Identificat. of bits at 0:
27.5 bits at 0 by nonce
in average



# Agenda


Context and Preliminaries
The Google Titan Security Key
Rhea
Side-Channel Setup

A Side-Channel Vulnerability in the ECDSA Algorithm
Reverse-Engineering of ECDSA Signature Algorithm
Sensitive Leakage in the Scalar Multiplication

A Key-Recovery Attack
Lattice-based Attacks on ECDSA
Key Recovery Attack on Rhea
Key Recovery Attack on Titan

Conclusions

# (Extended) Hidden Number Problem

- Recovering an **ECDSA secret key** given some partial knowledge on **nonces** can be expressed as a **(Extended) Hidden Number Problem** (**HNP** / **EHNP**)
- **HNP** and **EHNP** can be defined as games with an oracle
- The oracle reveals $x$ and $f_m(\alpha x)$ for several random values of $x$
  The player should find the hidden value $\alpha$
- **HNP – Hidden Number Problem**:
  $f_m$ discloses the $m$ most significant bits of $\alpha x$

  1101001010 $\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$

- **EHNP – Extended Hidden Number Problem**:
  $f_m$ discloses $m$ bits of $\alpha x$, not necessarily consecutive

  $\cdots 1 \cdots 01 \cdots 0 \cdots 101 \cdots 0 \cdots 10 \cdots$

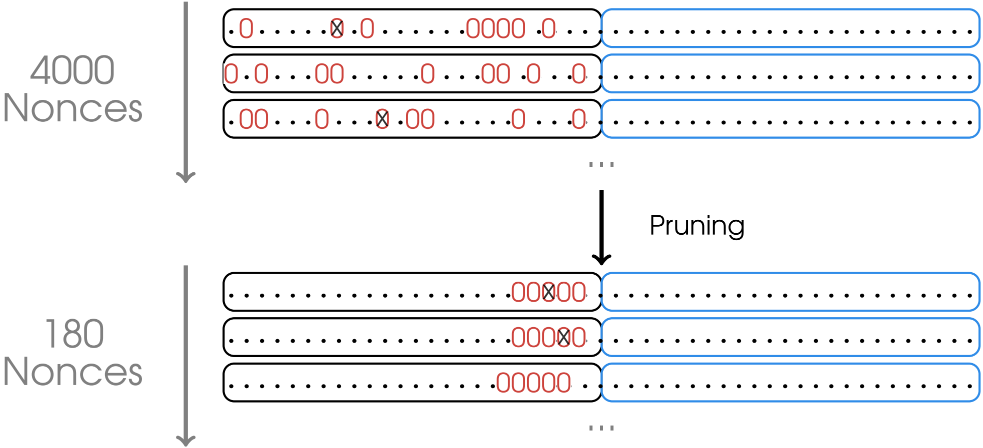
# Solving (E)HNP

- **(E)HNP** can be reduced to instances of lattice-based problems (SVP, CVP) that may be solved using **lattice reduction** techniques (**LLL**, **BKZ**)
- **# oracle queries** and **# known bits** dictate:
  - the size of the lattice
  - the probability of success
- In practice **EHNP** can be solved when the **m bits** revealed by the oracle form **blocks** of sufficiently **many consecutive bits**

$$\cdots\cdots 0110 \cdots\cdots\cdots\cdots 10110 \cdots\cdots\cdots\cdots\cdots 110 \cdots$$

- We ran simulations and deduced that in our case (on `P256`):
  $\rightarrow$ We need **80 signatures** with at least **5 consecutive known nonce bits**

# Rhea – Nonces Selection



# Rhea – Brute-Forcing the Key

- LLL reduction (for 80 signatures) takes about 100s
- 5 errors among 180 available signatures

$\hookrightarrow$ Brute-force attack on random subsets

**Final Attack**

- Acquisition of 4000 traces: $\sim 4h$
- Trace Processing: $\sim 4h$
- Brute-force attack: $\sim 20min$

# Touchdown on Titan

- Use Rhea parameters for PoI extraction
- Pruning: from 6000 signatures to 156
- 7 errors among 156 available signatures

$\hookrightarrow$ Brute-force attack on random subsets

**Final Attack**

- Acquisition of 6000 traces: $\sim 6h$
- Trace Processing: $\sim 6h$
- Brute-force attack: $\sim 30min$

# Agenda

Context and Preliminaries
The Google Titan Security Key
Rhea
Side-Channel Setup

A Side-Channel Vulnerability in the ECDSA Algorithm
Reverse-Engineering of ECDSA Signature Algorithm
Sensitive Leakage in the Scalar Multiplication

A Key-Recovery Attack
Lattice-based Attacks on ECDSA
Key Recovery Attack on Rhea
Key Recovery Attack on Titan

Conclusions

# Impact on Google Titan Security Key

- Proposed attack allows to physically extract an ECDSA private key linked to application account secured by FIDO U2F
- But high attack requirements:
  - Adversary already stolen victim's account login & password
  - Physical access to **Google Titan Security Key** during 10 hours
  - Adversary has access to:
    - Chemical lab or mean to thin IC package
    - Expensive SCA setup
    - Custom softwares
- **So still safer to use *Google Titan Security Key* than nothing !**
  *or a software FIDO U2F app.*

# List of Impacted Products

- FIDO U2F tokens:
  - Google Titan Security Key
  - Yubikey Neo (old product discontinuited)
  - Feitian K9, K13, K21, K40
- NXP JavaCard platforms (JCOP) based on P5x chips:
  - NXP J3D081_M59_DF and variants
  - NXP J3A081 and variants
  - NXP J2E081_M64 and variants
  - NXP J3D145_M59 and variants
  - NXP J3D081_M59 and variants
  - NXP J3E145_M64 and variants
  - NXP J3E081_M64_DF and variants
- NXP Product Security Incident Response Team (PSIRT) confirmed that:
  all ***NXP P5x and A7x products using ECC crypto. lib. up to v2.9*** are vulnerable

# Attack Mitigations

1. Hardening the NXP P5x / A7x cryptographic library:
    1.1 Blinding of the scalar
    1.2 Re-randomizing table lookup of precomputed points in comb implementation at each new access

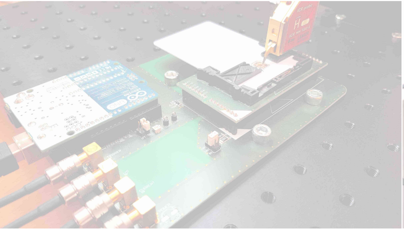
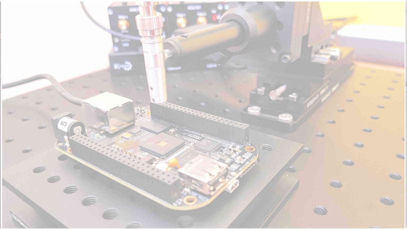
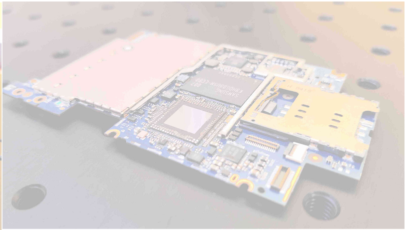
2. Use FIDO U2F counter to detect clones:
    - U2F counter *may* be used for detecting cloned U2F devices
    - Only limit validity of attack
    - Dependent on FIDO U2F server implementation

# Project's Timeline

- September $11^{th}$, 2018
  - Elie Bursztein's keynote talk at CHES2018
  - Obtaining of first **Google Titan Security Key** samples
- June $29^{th}$, 2020
  - Full attack validated on **Rhea**
- July $2^{nd}$, 2020
  - Full attack validated on **Google Titan Security Key**
- October $1^{st}$, 2020
  - Email sent to Google, Feitian, NXP, ANSSI and BSI with:
    1. Short technical description of our work
    2. Summary of our **coordinated responsible disclosure** plan (3 months)
- January, 2021
  - Publi. of technical report on NinjaLab website and on IACR eprint
  - We assigned CVE-2021-3011

# NinjaLab

*Improve the Security of your Cryptographic Implementation*

https://ninjalab.io
contact@ninjalab.io

NinjaLab
161 rue Ada
bâtiment 4 - CC477
34095 MONTPELLIER CEDEX 5
FRANCE