RADBOUD UNIVERSITEIT NIJMEGEN

Trezor One side-channel analysis setup

BACHELOR THESIS COMPUTING SCIENCE

Author:
Ellen Gunnarsdóttir

First Supervisor:
Lejla Batina

Second reviewer:
Łukasz Chmielewski

June 2020

Abstract
Hardware wallets were designed to keep cryptocurrencies safe in the hands of their owners. However, not all hardware wallets seem to be designed with the threat of these wallets in mind. They remain appealing targets to criminals because of the great reward one might come across when these wallets are "broken" (E.g. at current date, 25-06-2020, 1 Bitcoin equals about 8222 euros). In order to counter these criminals and prevent those of ill intent from stealing cryptocurrencies, it is extremely important to find vulnerabilities of these hardware wallets before them. Then the vulnerabilities can be patched before they are exploited. Furthermore, we might prevent these same mistakes from being made again in the future. One of these wallets is Trezor One from SatoshiLabs. It is not only appealing because of the vulnerable micro-processor they use, but also because of its low price and open source repository. In this thesis we show how a researcher might go about the first steps of performing a side channel analysis on Trezor One. We first lay the groundwork for understanding of these devices and the cryptography used by Bitcoin. Then we establish communication with Trezor One and describe how to prepare for a side channel attack.

Contents

1 Introduction

With cryptocurrencies in growing popularity over the years it is increasingly important to ensure the safety of ones coins. In order to do so, the private key associated with the wallet of the owner must be kept secret. This private key must never be intercepted, for it is with this key that the owner of any cryptocurrency verifies its ownership. To ensure the safety of the private key, hardware wallets have been developed to store the key offline and carry out any computations that involve it. This way, the key never leaves the hardware wallet.

One might think that the only way to exploit these wallets would be to find errors and vulnerabilities in the software used to communicate with the wallet, or in the firmware used by the hardware wallet itself. However, the computations performed on the micro-processors of these devices make them susceptible to all kinds of leakage such as power consumption leakage. This makes the wallets vulnerable to side-channel power analysis attacks. To keep these devices safe, it is important to constantly challenge their security, and find vulnerabilities before those of malicious intent.

Contributions Ledger is a company that is dedicated to provide secure hardware wallets and actively researches vulnerabilities of existing hardware wallets. E.g. The Ledger Donjon team has successfully recovered a scalar using side-channel analysis on the Trezor One hardware wallet from SatoshiLabs.

However, there seems to be little research on how to successfully communicate with a device such as Trezor One, and how to bridge the gap between the setup and the analysis itself. For this purpose, Ledger Donjon uses Lascar[1]. Lascar is a tool developed by Ledger Donjon that communicates with the Trezor One device (and other similar devices), stores the read traces, and processes them for the attack.

In this thesis we look into how to make a setup for power analysis on Trezor One with the intent of extracting the private key from the target device without the use of a tool such as Lascar. We aim to present the necessary requirements for such a setup, its practicality, and future improvements.

Structure First, we lay the necessary groundwork in the preliminaries, followed by related work. Then, Section 4 gives a detailed description of how to build custom binaries for Trezor One, and Section 5 verifies the success of these binaries. Section 6 describes our side channel attack setup and our pattern analysis of the signature generation. In Section 7 we look into how this setup can be used to extract the private key, followed by a brief discussion on the mitigation of side channel attacks.

[1]https://github.com/Ledger-Donjon/lascar

2 Preliminaries

2.1 Classical cryptography

As far as records go back, there has been need for secure and private communication. Traditionally, people would use some sort of encryption to achieve confidentiality. Encryption schemes change the original message using a shared secret key e such that the message is illegible to someone that does not have access or knowledge of the encryption key. Cryptosystems using the same key for encryption and decryption are called *symmetric* [2]. The problem with this approach is that there is no given way to safely distribute and manage the keys. This proves to be quite a challenge.[1]

In *Asymmetric* cryptosystems there is no need for a shared secret. Asymmetric cryptosystems rely on seemingly one-way-functions to ensure their security (e.g. the difficulty of factoring integers into primes, and the discrete logarithm problem). Using these functions, two keys are generated for every communicating party: A private key and public key. Thus, deriving the private key from the public key should be infeasible. In asymmetric cryptosystems the public key of the receiver is used to encrypt the message. Then the only person able to decrypt the message is the one in possession of the matching private key.[2]

A common analogy used to make this topic more understandable is thinking of the private and public key as a physical key and lock. In order to exchange a message, the receiver send his own lock (public key) to the sender. Then the sender can use this lock to secure the message inside a box, and send the box to the receiver. Finally, the receiver uses his own key (private key) to unlock the box. This way, the receivers key never leaves him. Thus there is no way for an attacker copy they key, or steal it. An attacker would have to find a way around the lock to intercept the message.

In most cases, asymmetric cryptosystems are slower because they are computationally more expensive, and consume more power than symmetric systems. Hence, asymmetric systems are often used to securely exchange an encryption key, that can then be used to carry on the conversation using a symmetric system.

2.2 Mathematical preliminaries

The set of integers, denoted as $\mathbb{Z}$, is a set comprised of zero, the positive whole numbers, and their additive inverses. Thus, $\mathbb{Z} = \{0, 1, 2, 3, ...\} \cup \{-1, -2, -3, ...\}$. The set of integers, $\mathbb{Z}$, is a subset of $\mathbb{Q}$, thus in turn a subset of the real numbers $\mathbb{R}$.

- $\mathbb{Z}^*$ denotes the set of integers excluding the negative integers and zero, $\mathbb{Z}^* = \{1, 2, 3, ...\}$.

[2]One can also have a separate key for decryption d in symmetric systems, but then the key d is derived from e. Making the knowledge of e sufficient for decryption

- Let p be a prime number. $\mathbb{Z}_p^*$ denotes a cyclic group of order $p-1$, $\mathbb{Z}_p^* = \{1, 2, 3, ..., p-1\}$.
- An integer r from $\mathbb{Z}_p$ is a primitive root modulo a prime p if every nonzero element of $\mathbb{Z}_p$ is a power of r[3].

2.2.1 Discrete logarithm problem

Let g be a primitive root mod p. For any integer $I \in \mathbb{Z}_p^*$, there is an exponent $e \in \mathbb{Z}_p^*$ such that

$$I \equiv g^e \; mod \; p$$

Here, e is the *discrete logarithm* of I to the base g. With I, g, and p given, the calculation of e ($e = log_g I$) is considered to be difficult. There is no efficient algorithm known for solving this problem[1].

2.2.2 Finite fields

Section 12.4, definition 3 in [3], describes a field as follows:

> Let $(S, +, \cdot)$ be an algebraic system with two binary operations. $< S, +, \cdot >$ is called a field if:
>
> 1. $(S, +)$ is an abelian group
> 2. $(S - \{0\}, \cdot)$ is an abelian group where 0 is the additive identity.
> 3. The operation $\cdot$ is distributative over the operation $+$.

A finite field is a field where the set S contains a finite number of elements.

2.2.3 Elliptic curves

The graph of an equation of the form

$$y^2 = x^3 + ax + b$$

is an elliptic curve. The variables defined in this equation can be elements of a field such as the real numbers $\mathbb{R}$, the complex numbers $\mathbb{C}$, or the rational numbers $\mathbb{Q}$. When it comes to cryptography, these variablies usually come from the finite field $\mathbb{Z}_p^*$.[2]

2.3 Elliptic Curve Digital Signature Algorithm (ECDSA)

Digital signatures make use of asymmetric cryptography to sign electronic documents. They provide authentication to all parties and integrity of messages sent. In some cases they can also provide non-repudiation. There are various schemes, one of which is the Digital Signature Algorithm (DSA).

DSA was specified by the Federal Information Processing Standard (FIPS) where they call it the Digital Signature Standard(DSS). The security provided by DSA is based on the Discrete logarithm problem (DLP) in prime-order subgroups of $\mathbb{Z}_p^*$ [4]. The Elliptic Curve Digital Signature Algorithm (ECDSA) is yet another digital signature scheme, and a variation on DSA. Elliptic curves are often used in cryptorgraphic situations because they offer great increases in speed without compromising on security. They achieve this by using fewer bits, and thus require smaller chips. Furthermore, they use less energy[2].

For ECDSA, one replaces the subgroup $\mathbb{Z}_p^*$ with the group of points on an elliptic curve E over a finite field. Hence, making the basis of the security for this algorithm the elliptic curve discreet logarithm problem (ECDLP). ECDLP appears to be harder to solve than DLP, making the strength-per-key-bit greater in elliptic curve systems[4].

2.3.1 ECDSA signature generation

Given elliptic curve E, base point G, and multiplicative order of G as n. One can generate an ECDSA signature with the following steps:

1. Select a cryptographically secure random integer k and compute:
   $(x_1, y_1) = k$ x G (Where x is scalar multiplication)
   Here, (x_1, y_1) represents a point on the curve.

2. $r = x_1 \bmod n$

3. With message m. Choose an appropriate cryprographic hashing function H and compute:
   $e = H(m)$

4. With L_n as the bit length of the group order n. Establish z as the L_n left most bit of e

5. With private key d. Compute:
   $s = k^{-1}(z + rd)$ (mod n)

The resulting signature is the tuple (r, s)[4].

2.3.2 Scalar Multiplication

In the ECDSA signature generation, we have to perform scalar multiplication. There are various ways to implement this operation and the method chosen will often rely on the type of hardware at hand. E.g. when there is only a small amount of memory available, one might use the *Width-w NAF method*[5]. Numerous other approaches are defined in [6], such as the Montgomery ladder algorithm.

The scalar multiplication performed in the signature generation of ECDSA is an interesting target for side channel attacks. That is because by reading power consumption leakage one can deduce the value of the aforementioned k. Then, with the knowledge of k, the value of the private key (d) can be computed.

Computing d:
With k given, we know that: $s = k^{-1}(z + rd) \pmod{n}$

1. Multiply both sides with k
   $sk = z + rd \pmod{n}$
2. Move z
   $sk - z = rd \pmod{n}$
3. Divide by r
   $d = \frac{sk-z}{r}$

2.4 Side Channel Attacks

Side channel attacks are carried out by acquiring information on the target through the implementation of the system. For example by monitoring timing of execution[7], power consumption leakage, or EM leakage[8][9]. When finding targets for side channel attacks, one can thus look into many aspects of the system. An attacker can try to exploit the hardware itself, access to the system, or software vulnerabilities that surface because the hardware type or system access is not being considered.

Side channel attacks are most often made with some gain in mind. Therefore, cryptographic operations are often targeted. Reading power consumption of a device during cryptographic operations and interpreting the information gained, is a technique called Simple Power Analysis (SPA)[10]. SPAs are often grouped into two categories[11]:

- Vertical: Where multiple execution curves (power readings) from the target device are analysed and compared to each other.
- Horizontal: Where a single execution curve from the target device will suffice.

2.4.1 Online Template Attack (OTA)

Template attacks are a type of side channel attack. When carrying out a template attack, the attacker first gathers traces (e.g. power consumption readings) from a device identical to the target device. These traces are called templates. Then the attacker compares the templates to traces caught from the target device.

The template attack technique described in [6] is referred to as an *Online template attack*. With the use of this technique, one can recover a complete scalar from one power trace of a scalar multiplication that uses this scalar. This template attack is not a typical template attack since no template building preprocessing phase is needed. The templates are created after acquiring the target trace. In order to carry out an online template attack, the attacker needs only limited control of the device that is used to generate the templates. With the most important assumption being the ability to choose the input point to a scalar multiplication. When carrying out this attack, the attacker will build templates using the same input point as the one used in the target trace. Then, for every scalar bit, the attacker builds one template and compares it to the target trace. If the part of the traces that correspond to the manipulated bit match, then the scalar bit was true. If not then the bit needs to be flipped. Looking at every bit separately, the attacker can deduce the whole scalar. Thus, with the use of these templates there is only need for a single template trace per key bit.

2.5 Trezor One

Trezor One is a hardware wallet used to store cryptocurrency. This hardware wallet supports the use of Bitcoin, one of the most well known cryptocurrencies. The system used to safely make transactions for Bitcoin is an asymmetric cryptosystem that uses Elliptic curve cryptography. During a signing process, the private key stored on the wallet will be used for the scalar multiplication. The power leakage throughout this procedure indicates the power consumption of the device. This can be read and used to extract the private key by performing e.g. an *Online template attack*[6].

3 Related work

This field of work is actively researched and the progress made is rapid. During my work on this thesis multiple articles have been released on research with the same goal in mind. That is, reveal and fix security vulnerabilities using side-channel analysis. The most notable article would be the *Side-Channel assessment of Open Source Hardware Wallets* [12] published by the Ledger Donjon team. In this article, Ledger Donjon describes three side-channel attacks carried out on Trezor One. With one of these attacking the PIN authentication mechanism (which lead to a code patch that circumvents this attack), and two other side-channel attacks that they used to recover a scalar. These latter attacks were not seen as threats to SatoshiLabs because the attacker would have to get past the PIN authentication to carry them out. Hence, the scalar attacks have not been patched.

Next to the PIN, and scalar recovery, Ledger Donjon and Kraken Security Labs have launched two successful attacks [13][14] where they extract the seed stored on the Trezor One devices using hardware attacks. This seed can be used to recover a wallet, thus giving anyone in possession of it complete access to the funds on the wallet. Kraken security labs blog post explains how they successfully extracted seeds from Trezor One and Trezor Model T with only 15 min of physical access to the device. They did this with high voltage glitching which lowers the read protection level of the processor from RDP Level 2 to RDP Level 1. Then with further voltage glitching they were able to access RDP Level 0 functions that should not be available at RDP Level 1. This allowed them to dump flash memory, and simply brute-force the 1-9 digit PIN protection. This attack can not be patched by a firmware update and is possible because of the type of processor used on Trezor One and Trezor model T. Kraken security lab thus demonstrated that the TM32-family of Cortex-M3/Cortex-M4 microcontrollers are not suited for the storage of sensitive data, even if it is encrypted.

The blog post on the seed extraction done by Ledger Donjon was less specific in its approach. However, the necessary materials needed to launch this attack cost around $100, and the seed extraction takes only about 5 minutes. According to Ledger Donjon, this attack can not be patched by a firmware update. A complete hardware redesign is needed in order to fix the vulnerability that they encountered on Trezor One.

Ledger Donjon states that there is only one mitigation: The use of a strong passphrase of about 37 random characters.

For a full list of previous issues addressed by SatoshiLabs, one can check out the security page on their website[3].

4 Customized binaries

In order to perform a side channel template attack, we need to be able to customize the firmware. That is because we want to be able to set the point entering the scalar multiplication, and carry out the signature generation at our own accord. In order to successfully carry out the power analysis that we describe here, the following prerequisites are assumed:

1. The target device has no PIN authentication (Or the attacker has a way around the PIN).

[3]https://trezor.io/security/

2. The attacker has physical access to the device long enough to trace a signature generation. Assuming the attacker is prepared, 10 minutes should be sufficient.

Furthermore, due to the vast variety of cryptosystems and protocols out there. We will be focusing on the cryptocurrency Bitcoin.

SatoshiLabs has an open source git repository of their firmware[4] which corresponds to the firmware acquired on their website, and thus will (most likely) correspond to the firmware used by the target device. With this repository you can make builds for Trezor devices, and flash them on the wallet. Thus, this repository will be the basis for our custom firmware that we can use to build templates for a side channel attack.

In order to work with the repository, understanding must be built on its structure and inner workings. To aid in this, it is beneficiary to look into the history of this repository. Initially, there were two repositories. One for Trezor One, and another for model T. These two repositories were later joined to form a monoreposiory that builds binaries for both models. The main directory of the monorepository includes these previously separated repositories in the directories 'core' and 'legacy' for the model T, and Trezor One respectively. These directories work on their own and one can build a binary for each model using only the directory corresponding to their model. However, one will see that some of the folders included in these directories also stand in the main directory. Further inspection revieals that these folders are no duplicates but symbolic links to folders found in the monodirectory. Thus if one wants to change something in these folders, they should be aware of this link. Hence, we see that the main directory includes shared resources of Trezor One, and model T in files such as 'crypto' which includes code for all cryptographic operations. Next to shared resources, the main directory includes a single build script that builds binaries for both models with the most recent version of the monorepository.

The first step of the setup for a side channel attack on Trezor One is to clone this repository using *git*. Then, with *Docker* installed, run the script *build-docker.sh* to check if everything functions properly.

4.1 trezorctl

With working firmware on the Trezor One device, we need to be able to communicate with the device and carry out instructions. For this purpose, SatoshiLabs has provided its users with the python library *trezorctl*. You can use this library to communicate with Trezor One through terminal commands. This library achieves this with the use of protocol buffers (protobuf), which serializes the given commands and sends them through to the connected Trezor One device.

[4]`https://github.com/trezor/trezor-firmware`

This library can be used to upload customized firmware to the device, and generate signatures for given messages.

4.2 Custom Firmware

For carrying out a side channel attack we need to be able to customize the firmware. However, the firmware is built in a way that protects it from someone accidentally changing their firmware. Furthermore, the build script is configured to always pull the most recent version of the monorepository from online. This is not what we want, thus there are several changes that need to be made to *build-docker.sh* and *trezor-firmware/legacy/makefile* (See Appendix for altered files) in order to successfully build customized firmware from the local repository.

4.2.1 *build-docker.sh*

When trying to build custom firmware for the first time, you will come across an error. That is because the build script is set to detect custom firmware and warn users of the change. In order to allow custom firmware, 'MEMORY_PROTECT' needs to be set to zero. Furthermore, the build script will build two binaries for both models. As the attacker will be making many builds, it is advisable to remove the code corresponding to the build of the model that is not being used. This will speed up the build process significantly. Additionally, one might want to remove the loop which allows for a normal build, followed by a bitcoin-only build, and choose only one of the two.

Next, the docker image will need acess to the local repository to build the binaries. This can be done with the following line:

```
-v PATH:/trezor-firmware \
```

Where PATH is the path to the local repository.

Lastly, within the docker script, all lines starting with *git* need to be replaced with corresponding local actions (See Appendix for resulting script, and list of changes).

4.2.2 *trezor-firmware/legacy/makefile*

The changes made here are minor. The last line updates all submodules of the git repository to the most recent version, this could overwrite any changes that you made. Therefore, we simply remove this line.

4.2.3 Upload firmware

After successfully building the custom firmware. This firmware needs to uploaded to Trezor One. The custom made binaries built in the build script *build-docker.sh* can be found in the folder `trezor-firmware/build/legacy/firmware/` (unless this path was altered in the build script). Then, the following commandline option can be used to upload customized firmware to the device:

```
trezortctl firmware-update -f firmware.bin
```

(Make sure the device is in bootloader mode[5]).

4.2.4 Initialize device

After uploading firmware onto Trezor One, the device needs to be initialized. That can be done at `https://trezor.io/start/`. With the wallet initialized you will be able to access the BIP32 path of your device (It should look similar to the following path: m/44'/0'/0'/0/0). This path is needed alongside the master seed to derive the private and public key of a device.

5 Verification

We carried out two tests that verify our ability to modify the code and execute it on the Trezor One device. Furthermore, we looked into the feasibility of reading the power leakage through the plastic casing and analysed the patterns observed in the readings.

5.1 Changing the default backup screen

After initializing the device, the device will tell you that it needs backup. This can be observed in Figure 1. We altered the text displayed here by modifying the file `trezor-firmware/legacy/firmware/layout2.c`. In this file we changed the string "NEEDS BACKUP!" in line 294, to "B". This change can be seen in Figure 2.

This alteration demonstrates the success of building and uploading custom firmware onto the Trezor One device. However, for carrying out a side channel attack, we are interested in changing the code responsible for cryprographic operations. Therefore, we conduced a second test.

5.2 trezorctl sign-message

It is important that we have some way of executing the signing procedure and modifying it on our own accord. Therefore, we were interested in changing the code in the file `trezor-firmware/crypto/ecdsa.c`[6]. However, this proved a challenge since importing any additional libraries (e.g. a library that provides some sort of output text) made the bootloader to big to fit in the device. In order to check that the command 'sign-message' from the trezorctl commands does indeed make use of

[5]`https://wiki.trezor.io/User_manual-Updating_the_Trezor_device_firmware__T1`

[6]The legacy firmware crypto folder has an established symbolic link from to here

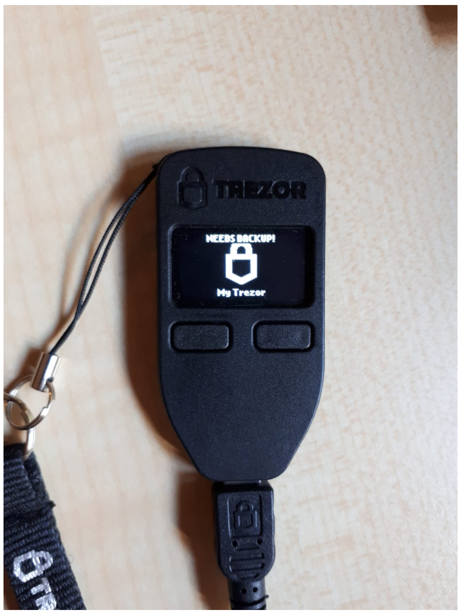

Figure 1: Backup screen - unaltered

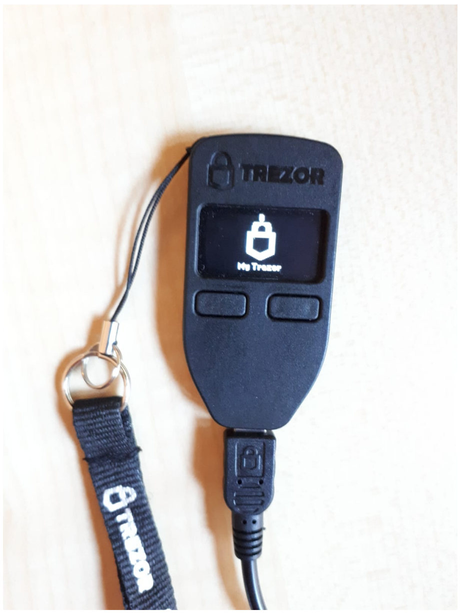

Figure 2: Backup screen - altered

the signing procedure, we used of one of the already included library called 'stdlib.h'. This library includes the function 'abort()' which causes an abnormal termination of the program when called. We found out that including it within the function 'ecdsa_sign_digest()' would halt the program and render all other operation on the device inoperable until the device was disconnected and the corresponding USB port was set free. This shows us that 'sign-message' does indeed arrive at that point in the file. Meaning that we are able to reach this point using 'sign-message' which in turn gives us the power to execute any existing or customized code hereafter.

5.3 Initial readings

An image included in [12] shows us the location of the processor within the device. Thus, we were interested to see if it was possible to acquire the readings without opening up the device. That is, gather the readings through the plastic.

We found that the positioning of the probe was crucial and that we obtained better readings from the backside of the device. Several re-locations of the probe might be needed to attain proper readings. However, it remains to be seen whether the acquired traces are sufficient for the power analysis. If they turn out to be insufficient then the approach described in [12] can be taken.

In Figures 3 and 4 one can see the patterns we observed when carrying out the command:

```
trezorctl sign-message -n "m/49'/0'/0'/0/0" "hi"
```

These traces indicate that we were successful in reading through the plastic.

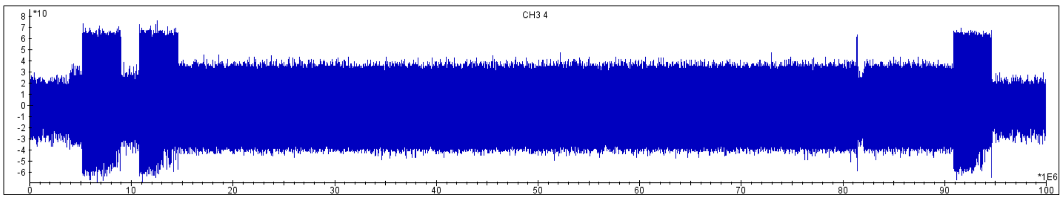

Figure 3: Readings through plastic when running tezorctl sign-message

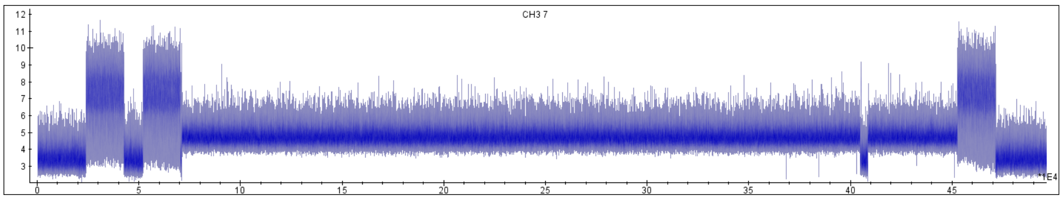

Figure 4: Processed readings through plastic when running tezorctl sign-message

6 Analysing the observed patterns

For our second round of readings we used the oscilloscope Waverunner 9804M-MS from Teledyne lecroy, in combination with the probe RF-U 5-2 from LANGER EMV-TECHNIK, and the low noise amplifier HD24248 from Riscure. Figure 5 depicts how these components were connected. We then configured a trigger on the high amplitude of the signal on the Oscilloscope. This trigger signals when the Oscilloscope should start to record its readings. With the probe positioned against the backside of Trezor One, the signals read from the probe went through the amplifier and on to the Oscilloscope. The oscilloscope then in turn sent its readings through to the computer where we ran the program Inspector from Riscure[7].

We expected similar results as the initial readings, but came to find that the last spike was missing, and that the spikes appeared less prominent.

The missing spike can be explained by the used of a looping script during the initial readings. This script repeated the signing command in every loop followed by a short sleep. With the capture window too big we interpreted the pattern incorrectly, and thought that this was part of the signing process. When in fact, this is the beginning of a new signing process.

The reason behind the spikes being less prominent is not as easy to identify. This might be the result of several factors:

[7]https://www.riscure.com/news/inspector-4-12-released/

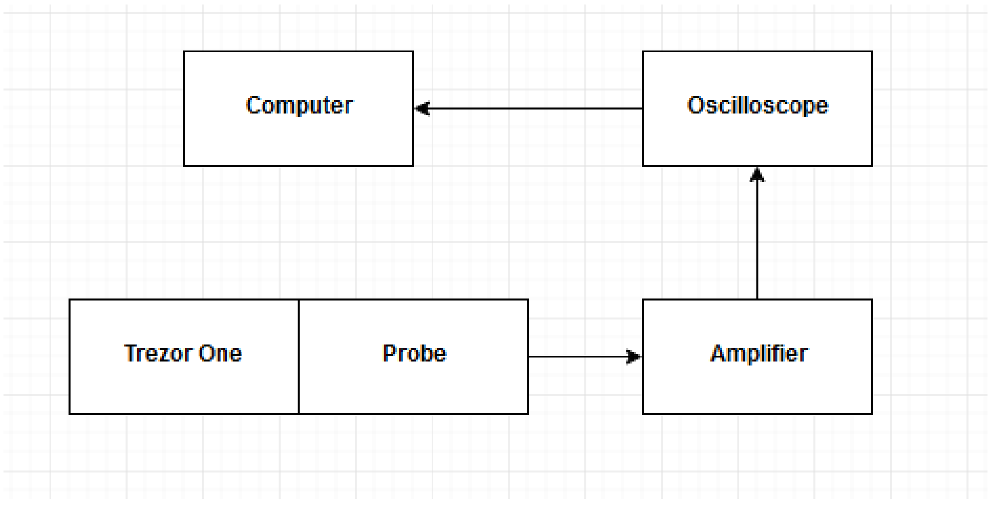

Figure 5: Connections

- Positioning of the probe
- Firmware update [8]
- The use of a different probe

For this round we were interested to locate where the scalar multiplication takes place in the traces as this is a requirement for many side channel attacks, such as OTA.

We first gathered new readings of the baseline, acquired by running the original code using the aforementioned command. The command was carried out before the readings started. This prompts the reader to either confirm or cancel the generation of the signature. After this we verified that we read the lower frequency traces seen at the begin of Figure 6. This lower frequency indicates some sort of waiting state on Trezor One. With these readings established, we confirmed the signature generation by pressing a button on the device. This would then be caught by a trigger that we configured beforehand on the oscilloscope (see Figure 6).

Secondly, we carried out the signature generation with the firmware configured to halt during the process as described in Section 5.2. This resulted in the trace seen in Figure 7.

In this trace we observe neither of the two peaks, indicating that the process was halted somewhere during or before the first peak in the baseline.

[8]We did not yet have the altered build script at the time of the initial readings

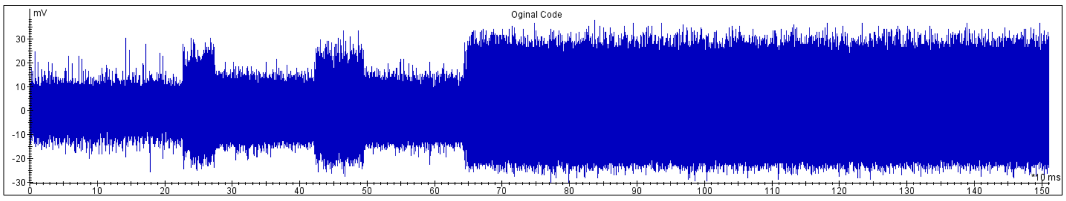

Figure 6: Original code - baseline for second readings

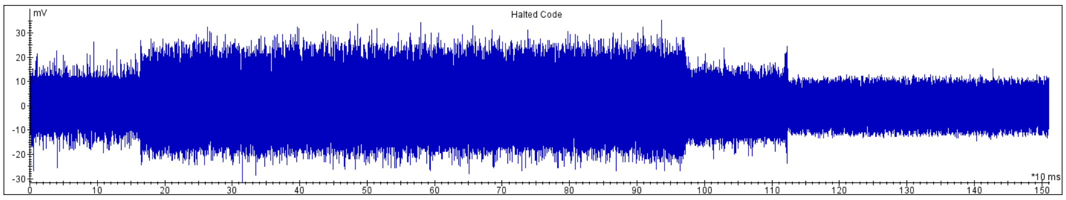

Figure 7: Halted code, with the use of abort()

Next, with a clean state we removed the call to 'scalar_multiply' within 'ecdsa_sign_digest()', which results in a signing process without the scalar multiplication. The trace from this operation can be observed in Figure 8.

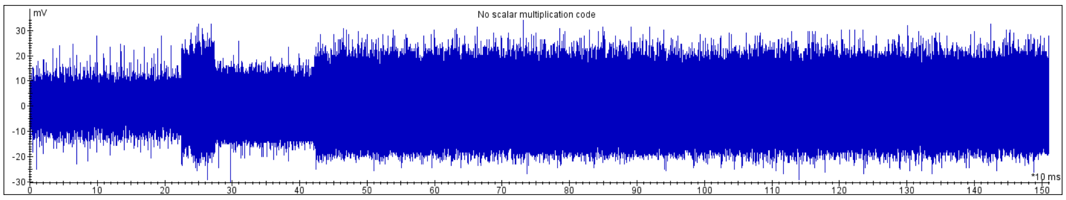

Figure 8: No scalar multiplication

Here, we see that one of the initial peaks is no longer traced. Indicating that the scalar multiplication can be observed in one of the two peaks. To confirm this, we restored the call to 'scalar_multiply' and performed it twice, see Figure 9.

This last trace shows us that the second peak observed in the baseline corresponds to the scalar multiplication. One can observe these traces zoomed in on the second peak (the scalar multiplication), in Figures 10 and 11. From these images, one can assume that the scalar multiplication takes place in one of the blocks presented in Figure 10. Finding the block that represents the scalar multiplication could be accomplished by comparing two traces captured from the same device where the only variance is the scalar itself.

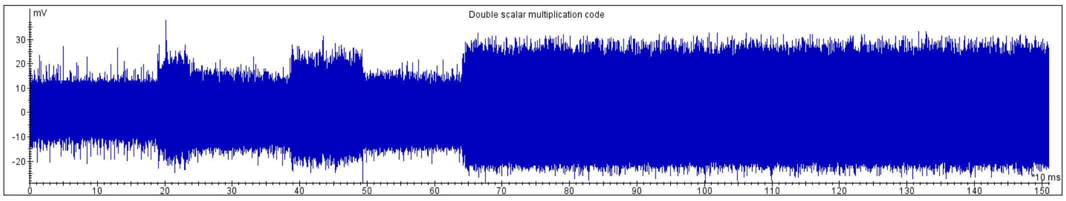

Figure 9: Double scalar multiplication

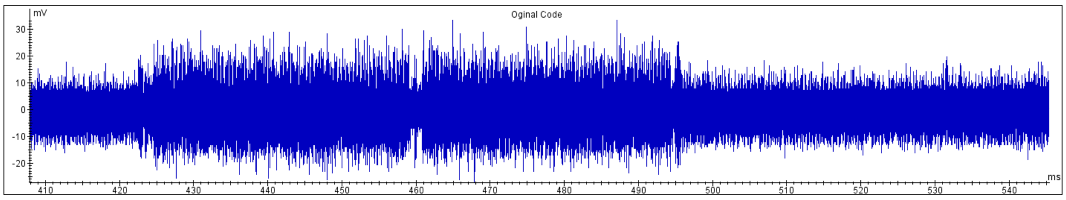

Figure 10: Original code - zoom in on scalar multiplication

7 Carrying out the analysis

The setup described in the previous section is most likely sufficient for carrying out a side channel attack such as OTA. That is because we have the ability to change the point entering the scalar multiplication, and we do get characterized readings through the plastic. However, there are several improvements that could be made to this setup.

7.1 Improvements

7.1.1 Installing a Trigger

It is advisable to set up some sort of trigger for the oscilloscope within the customized code. If the oscilloscope starts reading long before the targeted code is executed then it will have read a lot of noise that needs to be cut out. If your oscilloscope has sufficient memory, then this might not be a problem. However, someone making use of oscilloscopes with limited memory will greatly benefit from a trigger. Furthermore, with a trigger configured on the high amplitude of the signal, the oscilloscope will sometimes miss the signature generation because of variance in the leakage.

With a trigger included in the code, the analyst can also be sure of what the traces read by the oscilloscope corresponds to within the code. See related work [12] for an example of how a trigger can be installed.

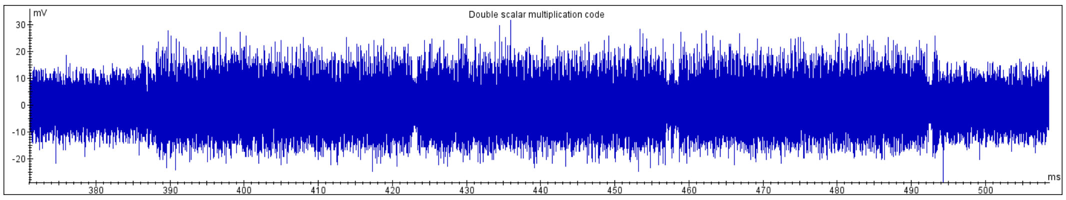

Figure 11: Double scalar multiplication zoom

7.1.2 Automatic confirmation of signature generation

With the current setup, one needs to confirm the generation of a signature with a button click on the device. It would be advisable to look for this button click within the code and make it automatic to speed up the process of template building. Furthermore, if the device is not well secured during the tracing, then the button click might cause disturbance in the readings. That is because the device might move slightly due to the force of the button click.

7.1.3 Bash script

During the initial and second readings, we used a script that carried out the desired trezorctl command followed by a sleep within a loop. Automating the process of gathering the template traces and matching in a script would simplify and speed up the analysis.

7.2 Extracting the private key

With a functioning setup, one would first have to gather the target trace. As described earlier, the attacker would need physical access to the device long enough to carry out a signature generation, or about 10 minutes. This should be enough time to connect the Trezor device, secure it next to the probe, run the predefined signature command, and re-position the probe for further readings if the initial readings seem insufficient. When doing this, the attacker needs to make note of the parameters used for the signature generation, so that he can match the point going into the scalar multiplication on the templates.

Section 2.3.2 describes how a private key can be computed with the knowledge of the scalar. However, the targeted function 'point_multiply' in ecdsa.c has already been the subject of a side-channel attack [15] and has since then been patched to protect against side-channel attack. Nonetheless, the Ledger Donjon team has demonstrated that the countermeasure taken does not protect Trezor One from their attack [12]. In this article they show how they retrieve a scalar from only one trace of the execution of 'point_multiply'.

8 Mitigation

When it comes to the design of a device that handles sensitive data it is extremely important to keep in mind the use of the device, and the access to the device. There are many actions that can be taken to make the device less appealing, and harder to break. Which in turn decreases the chance of a breach from someone of ill intent. In order to mitigate side channel attacks, the designers of devices handling sensitive data should keep up to date with cryptographic standards, use appropriate hardware, and consider how appealing the device is to attackers.

Keeping up to date with the cryptographic standards for the cryptography used on devices such as Trezor One is crucial for its success. Any vulnerabilities found in the methods used on the device need to be patched promptly, because as soon as they are found they can be exploited and pose a threat to every user. Users of these devices should be alerted immediately of the threat, and the patched firmware should be made available as soon as possible.

As seen in Section 3 the hardware itself also needs to be suited for sensitive data. Every component handling this sensitive data needs to be capable of handling it without being vulnerable to some sort of leakage. This can e.g. be achieved with the use of components that are designed to handle sensitive data, or by shielding the components from probes. Additionally, a device can be configured in such a way that prying the device open will trigger a mechanism that wipes or corrupts sensitive data.

Lastly, the appeal of the data stored on the device, and access to the device needs to be kept in mind during design. For example, hardware wallets are extremely appealing targets and thus will be targeted by attackers. This fact should motivate the designers to protect the sensitive data on these devices from early on in the design phase. This can be contrasted by e.g. a smart toothbrush. The data kept on a smart toothbrush is less appealing to the average person, or group, and thus there is less incentive for attackers to extract this data.

9 Conclusions

We have demonstrated that the Trezor One device from SatoshiLabs can be successfully setup for a side channel attack. This is made possible by their open source repository, vulnerable micro-processor, and accessibility. Custom binaries can be built to suit the needs of any type of attack, and uploaded to the device. This is further aided by the trezorctl library which offers great variety in communication with Trezor One. Furthermore, we were able to gather traces with the use of an oscilloscope through the plastic of the device, and locate the scalar multiplication within the traces. The setup described in this thesis can be further improved by the installment of a trigger within the code, automated button presses, and a bash script which carries out all commands needed for the analysis.

From reported attacks on Trezor One, we can see that the wallet is relatively secure if the attacker does not have access to the device. This comes from proper implementation of the asymmetric cryptography used by these devices. Alas, Trezor One seems to lack in the security provided by the use of appropriate hardware and physical protection of components on the board. One could also contemplate the disclosure of the code as open source, because this makes the device more vulnerable to attacks. On the other hand, open source code gives the company credibility. With the code open source, users are able to verify that the data is properly handled, and users can build trust for the company. Like with so many other aspects in life, this is a balancing act and there is no right or wrong when it comes to open source.

Future Work This setup can be used to carry out a side channel attack. The next step would be to e.g. attempt to extract the scalar. This could be done by analyzing the function that carries out the scalar multiplication. This function has been patched to protect against side channel attacks. However, there are bound to be overlooked aspects or mistakes when it comes to such young technology that is a constant subject to change.

References

[1] Johannes A. Buchmann. *Introduction to cryptography*. Springer, 2004.

[2] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography*. Chapman Hall/CRC, 2003.

[3] Kenneth H. Rosen and Kamala Krithivasan. *Discrete Mathematics and Its Applications*. McGraw-Hill Education, 2013.

[4] Don Johnson, Alfred Menezes, and Scott Vanstone. "The Elliptic Curve Digital Signature Algorithm (ECDSA)". In: *International Journal of Information Security* 1.1 (Aug. 2001), pp. 36–63. ISSN: 1615-5262. DOI: 10.1007/s102070100002. URL: https://doi.org/10.1007/s102070100002.

[5] Katsuyuki Okeya and Tsuyoshi Takagi. "The Width-w NAF Method Provides Small Memory and Fast Elliptic Scalar Multiplications Secure against Side Channel Attacks". In: *Topics in Cryptology — CT-RSA 2003*. Ed. by Marc Joye. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 328–343.

[6] Lejla Batina et al. "Online template attacks". In: *Journal of Cryptographic Engineering* (Aug. 2017). ISSN: 2190-8516. DOI: 10.1007/s13389-017-0171-8. URL: https://doi.org/10.1007/s13389-017-0171-8.

[7] Paul C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: *Advances in Cryptology — CRYPTO '96*. Ed. by Neal Koblitz. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 104–113. ISBN: 978-3-540-68697-2.

[8] Jean-Jacques Quisquater and David Samyde. "ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards". In: *Smart Card Programming and Security*. Ed. by Isabelle Attali and Thomas Jensen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 200–210. ISBN: 978-3-540-45418-2.

[9] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. "Electromagnetic Analysis: Concrete Results". In: *Cryptographic Hardware and Embedded Systems — CHES 2001*. Ed. by Çetin K. Koç, David Naccache, and Christof Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 251–261. ISBN: 978-3-540-44709-2.

[10] Paul Kocher, Joshua Jaffe, and Benjamin Jun. "Differential Power Analysis". In: *Advances in Cryptology — CRYPTO' 99*. Ed. by Michael Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397.

[11] Christophe Clavier et al. "Horizontal Correlation Analysis on Exponentiation". In: *Information and Communications Security*. Ed. by Miguel Soriano, Sihan Qing, and Javier López. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 46–61.

[12] Manuel San Pedro, Victor Servant, and Charles Guillemet. “Side-Channel assessment of Open Source Hardware Wallets”. In: *SSTIC* (2019). URL: `https://www.sstic.org/2019/presentation/side_channel_assessment_hardware_wallets/`.

[13] KRAKENFX. *Kraken IDentifies Critical Flaw in Trezor Hardware Wallets.* 2020. URL: `https://blog.kraken.com/post/3662/kraken-identifies-critical-flaw-in-trezor-hardware-wallets/`.

[14] Karim Abdellatif, Charles Guillemet, and Olivier Hériveaux. *Unfixable Seed Extraction on Trezor - A practical and reliable attack.* 2019. URL: `https://donjon.ledger.com/Unfixable-Key-Extraction-Attack-on-Trezor/`.

[15] Jochen Hoenicke. *Extracting the Private Key from a TREZOR.* 2018. URL: `https://jochen-hoenicke.de/crypto/trezor-power-analysis/`.

10 Appendix

10.1 build-docker.sh

10.1.1 List

1. Set 'MEMORY_PROTECT' to zero.
2. (Optional) Remove code that builds the Trezor core binary file. We do not need the .bin for the other device, thus we can remove it and the script will run faster.
3. Give the docker image access to the local repository:

   ```
   -v PATH:/trezor-firmware \
   ```

   Where PATH is the path to the local repository.
4. Remove all lines starting with *git*, and replace their actions with corresponding local actions (See appendix for resulting script)

10.1.2 Altered file

```
#!/usr/bin/env bash
set -e

if [ "$1" = "--gcc_source" ]; then
  TOOLCHAIN_FLAVOR=src
  shift
else
  TOOLCHAIN_FLAVOR=linux
fi

IMAGE=trezor-firmware-build.$TOOLCHAIN_FLAVOR

TAG=${1:-master}
REPOSITORY=${2:-local}
PRODUCTION=${PRODUCTION:-1}
MEMORY_PROTECT=0

if [ "$REPOSITORY" = "local" ]; then
  REPOSITORY=file:///local/
else
  REPOSITORY=https://github.com/$REPOSITORY/trezor-firmware.git
fi
```

```
docker build -t "$IMAGE" --build-arg TOOLCHAIN_FLAVOR=$TOOLCHAIN_FLAVOR ci/

USER=$(ls -lnd . | awk '{ print $3 }')
GROUP=$(ls -lnd . | awk '{ print $4 }')

mkdir -p $(pwd)/build/core $(pwd)/build/legacy
mkdir -p $(pwd)/build/core-bitcoinonly $(pwd)/build/legacy-bitcoinonly

# build legacy

for BITCOIN_ONLY in 0 1; do

  DIRSUFFIX=${BITCOIN_ONLY/1/-bitcoinonly}
  DIRSUFFIX=${DIRSUFFIX/0/}

  docker run -it \
    -v $(pwd):/local \
    -v $(pwd)/build/legacy"${DIRSUFFIX}":/build:z \
        -v /home/ellen/trezor-firmware:/trezor-firmware \
    --env BITCOIN_ONLY="$BITCOIN_ONLY" \
    --env MEMORY_PROTECT="$MEMORY_PROTECT" \
    --user="$USER:$GROUP" \
    "$IMAGE" \
    /bin/sh -c "\
          cp -r /trezor-firmware /tmp/trezor-firmware && \
      cd /tmp && \
      cd trezor-firmware/legacy && \
      ln -s /build build &&
      pipenv install && \
      pipenv run script/cibuild && \
      mkdir -p build/firmware && \
      cp firmware/trezor.bin build/firmware/firmware.bin && \
      cp firmware/trezor.elf build/firmware/firmware.elf"

done
```

10.2 trezor-firmware/legacy/makefile

```
ifneq ($(EMULATOR),1)
OBJS += startup.o
endif
```

```
OBJS += buttons.o
OBJS += common.o
OBJS += flash.o
OBJS += layout.o
OBJS += oled.o
OBJS += rng.o

ifneq ($(EMULATOR),1)
OBJS += setup.o
endif

OBJS += util.o
OBJS += memory.o
OBJS += supervise.o

ifneq ($(EMULATOR),1)
OBJS += timer.o
endif

OBJS += usb_standard.o
OBJS += usb21_standard.o
OBJS += webusb.o
OBJS += winusb.o

OBJS += gen/bitmaps.o
OBJS += gen/fonts.o

libtrezor.a: $(OBJS)

include Makefile.include

libtrezor.a:
	@printf "  AR      $@\n"
	$(Q)$(AR) rcs $@ $^

.PHONY: vendor

vendor:
```