

BACHELOR THESIS
COMPUTER SCIENCE



RADBOUD UNIVERSITY

Power Analysis of the Ledger NanoS

Author:

Wouter de Boer
s4626265

First supervisor/assessor:

Prof. L. Batina
lejla.batina@ru.nl

Second supervisor/assessor:

drs. L. Papachristodoulou
l.papachristodoulou@science.ru.nl

June 27, 2018

vires in numeris

Abstract

With the rising popularity of cryptocurrencies there is a concurrent increase in the need for safely storing them. The security of these assets stands or falls with the safekeeping of the private keys. In this thesis the focus lays on deriving the private key by only observing the Ledger Nano S hardware wallet. Recovering the private key is almost the same as access to the assets stored on the wallet. In this case, cryptocurrencies and the Fiat money they represent.

To begin with, the thesis will provide information about several fundamental pillars on which cryptocurrencies are based. To find a suitable attack strategy a considerable amount of common and effective analysis methods will be discussed with regards to hardware wallets. At the end the practical analysis and results are presented. A full attack could not be performed but some interesting observations were made that could help others in their research on hardware wallets, or in specific the Ledger Nano S.

Contents

1	Introduction	3
2	Preliminaries	5
2.1	Elliptic curves	5
2.2	Cryptocurrencies	7
2.3	Wallets	8
3	Side-channel Attacks	10
3.1	Introduction	10
3.2	Simple Power Analysis	12
3.2.1	Visual Inspections of Traces	13
3.3	Template Attacks	13
3.3.1	Template Building Phase	14
3.3.2	Template Matching Phase	14
3.4	Differential Analysis	15
4	Practical Analysis	16
4.1	Technical Analysis	16
4.2	Application	18
4.3	Setup	21
4.4	Data collection	23
4.5	Data processing	25
4.6	Results and Observations	28
5	Related Work	32
6	Conclusions	34
6.1	Future Work	34
A	Appendix	39
A.1	Installation and Loading without screen	39
A.2	Final Setup Specifications	40
A.3	Shellscripts	41
A.3.1	make_env.sh	41

A.3.2	install.sh	41
A.3.3	SetEnvironment.sh	41
A.3.4	reset.sh	41
A.4	Main.C	42
A.5	Capture.py	42
A.6	MakeFile	43

Chapter 1

Introduction

The Ledger Nano S is a dedicated device specialized in the cryptographic operations used by cryptocurrencies. A device that is designed specifically for this purpose is called a hardware wallet. Everybody who owns money obviously values its security highly and the hardware wallets, on which it is stored, should increase this security. However the main concern remains, are my funds really safe?

In this thesis a specific hardware wallet, earlier introduced as Nano S, will be tested for possible security vulnerabilities. Although several attack strategies could be researched, this thesis will focus on observing the hardware wallet for possible useful leakage. With the use of side-channel attacks this leakage could be retrieved. Side-channel attacks are used when the cryptographic algorithm itself is secure but a secure implementation isn't necessarily guaranteed. The hardware wallets make use of cryptographic algorithms for its core functionalities. These algorithms can be monitored and previously hidden information could be derived through power consumption or electromagnetic emanations. The two subparagraphs below will explain more about the cryptographic algorithms and the core functionalities in short. More about these will follow in the preliminaries.

The cryptographic algorithms used in the Nano S are based on Elliptic Curves. These special type of equations are used to create a trapdoor one-way function, which is easy to compute in one direction but hard to revert without special information. The public key is used in the easy way, the private key is used to revert and can be seen as the special information. However the private key will only revert in combination with the public key, hence the term keypair.

The core functionalities of the hardware wallet are safekeeping the private keys, generating public keys (addresses on which you can receive coins) and

signing transactions. The signature that is made for a particular transaction proves, mathematically, that you are the rightful owner and therefore can send the cryptocurrency with this transaction. The algorithm that the Nano S uses to create these signatures is called Elliptic Curve Digital Signature Algorithm. It requires the hexadecimal value of a transaction and the private/public keypair as an input. This means that compromising the private key can result in falsifying signatures. All the assets on the corresponding address are now at risk since the attackers can now prove, mathematically, that the cryptocurrency is theirs to spend.

To prevent this from happening it is important that research is conducted in this field. The paper published by Gkaniatsou et al. [3] is one of the few papers looking into the implementation details of hardware wallets. However the low level attacks are more often researched and as shown in a publishing by Hoenicke [13] could successfully be performed. Each paper looks at the hardware wallet with a specific attack strategy in mind. For example the attack by Hoenicke requires the hardware wallet to be stolen and will then be tested for it's security. While a more intrusive attack by Rashid [23] requires not only access to the Nano S after the funds have been stored, but also during the installation process. This thesis will opt for a less intrusive attack strategy.

An attack scenario would require just the measurements of an ordinary transaction performed by the owner. To match the measurements several templates have to be derived in advance. The successful matching of the templates to the measurements can result in the discovery of the private key. This attack would require to passively measure the hardware wallet only once which is far less invasive.

Chapter 2

Preliminaries

The rising popularity of cryptocurrency hasn't gone unnoticed, it even reached the US Senate, "Virtual currencies, perhaps most notably Bitcoin, have captured the imagination of some, struck fear among others, and confused the heck out of the rest of us.", as Thomas Carper said. In the preliminaries chapter an attempt is made to take away the confusion it is causing and let you decide whether it has captured your imagination or it has struck fear.

2.1 Elliptic curves

Before we can try to understand cryptocurrencies and the mechanisms it is build on, the fundamental math behind it should be made clear. One of the mechanisms used is cryptography, it is the study of secure communication and the implementations that come with it. The safe communication requires a shared secret with which the encryption/decryption takes place. When the sender and the receiver have a shared secret it is called symmetric cryptography. Another field of cryptography is asymmetric cryptography. In this type the secret is not shared but only kept by the receiver, the private key. The sender knows a public key of the receiver which can be used to encrypt a message. The fundamentals on which asymmetric cryptography relies is called the trapdoor function, as seen in the introduction. Several implementations of the trapdoor function were presented such as prime factoring[25] or the discrete logarithm problem[12].

Early journals about the use of Elliptic curves in cryptography date back from the eighties in which Neil Koblitz [16] and Victor Miller [19] describe it's improvements to the standard cryptography back then. The computational hardness of the reverse function in previous implementations are subexponential, the implementations with elliptic curves are exponential. This means that the reverse function is harder to compute without the private key, and a possible attack takes longer. This can be viewed like an

increase in security. As an additional argument they present that a computational improvement could be achieved when encrypting compared to e.g. prime factoring. Besides that the length of the keys are also smaller than with it's predecessors. Hence the recommended key length by NIST is 256-383, compared to the 3072-bit keys for RSA[5]. This in combination with the efficient computations is why cryptocurrencies use Elliptic Curves as their standard of cryptography.

An elliptic curve is an algebraic curve defined by an equation of the form, $y^2 = x^3 + ax + b$. The Elliptic curve domain parameters are specified in the form of a six tuple. $T = (p, a, b, G, n, h)$.

- p, an integer specifying the finite field
- a and b, used to specify the equation
- point G, the base point
- n, the order of G
- h, the cofactor.

Signatures

The particular algorithm that uses this elliptic curve is called Elliptic Curve Digital Signature Algorithm (ECDSA). It is used to craft a signature over a specific message. This signature can be verified by using the public key and can only be crafted with the private key. This means that the message is indeed unchanged and written by the owner of the private key. In short the public key is a point on the curve. This point was calculated by adding the base point to itself a number of times. This addition is called a scalar multiplication. The number of multiplications performed represent the private key. The most common way to implement this is with the double and add algorithm. When calculating this point the binary representation of the private key is used. After every iteration, a point doubling is performed. For each bit a point addition is only done if this bit is 1. If it is a 0 it will be skipped. The calculation of the public key is easy but knowing how many times the base point was multiplied is hard. The signature consists of three parts (m, R, s):

- m, the message which is usually a hash of the actual message.
- R, a point on the curve denoted by (x,y). Calculated by multiplying base point G with a random integer k.
- s, calculated by the following equation $s \equiv k^{-1}(m + ax)$ where the private key equals a.

EC in cryptocurrency

Secp256k1 refers to the parameters of the Elliptic curve used in Bitcoin and Ethereum, and is defined in the Standards for Efficient Cryptography (SEC)[11]. The parameters are defined as follows:

- $p = \text{FFFFFFFF} \dots \text{FFFFFFFFE FFFFFFFC2F} = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$
- $y^2 = x^3 + 7$ because $a = 0$ and $b = 7$
- $G = 02\ 79\text{BE}667\text{E}\ \text{F9DCBBAC}\ 55\text{A06295}\ \text{CE870B07}\ 029\text{BFCDB}\ 2\text{DCE28D9}\ 59\text{F2815B}\ 16\text{F81798}$
- $n = \text{FFFFFFFF} \dots \text{FFFFFFFFE BAAEDCE6}\ \text{AF48A03B}\ \text{BFD25E8C}\ \text{D0364141}$
- $h = 01$

The secp256k1 was especially constructed to prevent randomness in order to increase efficiency in the computation. In addition to that the constants are picked in a predictable way, this reduces the possibility of a backdoor into the curve. [1]

2.2 Cryptocurrencies

It all started out as a Peer-to-Peer Electronic Cash System, introduced by Satoshi Nakamoto. The concept is pretty simple, you have a large ledger, a collection of financial accounts, on which you keep track of payments. The analogy between this concept and digital cash is hence easy to explain. However the age old problem with a ledger is so-called double-spending. Double-spending is pretty much what it implies, an account spends the same cash twice. To prevent double-spending the electronic cash system has to determine which transactions are valid and which are not. Previously this was always done by a centralized entity e.g. a bank. The ingenious idea arose to let a decentralized network keep consensus about the ledger and Satoshi actually proved that this was possible. The consensus is achieved using several techniques. The peers of the network all keep a copy of the ledger, the peers can be so-called miners but are not obligated to be. Miners verify if a transaction is valid according to their copy of the ledger. A lot of transactions together will be put into a block. This block will be added to the blockchain, an immutable data structure, once the miner has proven he did the required work. This proof consists of finding the SHA256 hash of this block and nonce. Changing this nonce obviously yields different results. The nonce is used to create a change in the difficulty. Computing

one SHA256 hash doesn't require a lot of computational power but the hash has to have a particular amount of leading zero's. The miners now need to choose the right nonce that leads to the amount of leading zero's required. Since the work the miner put into "mining" that block costs a lot of computational power it can be seen as a legit block. The legitimacy of a block can however be verified easily by the other peers once the nonce has been published. This nifty mechanism works because of the high computational power required and the high amount of peers. So why are peers mining? The miner with the correct hash gets access to the so-called coinbase transaction. This transaction yields new coins as an incentive for the miner. This is the only way to create new valid coins and this amongst others makes it a viable currency.

2.3 Wallets

Virtual currencies can indeed capture your imagination or struck fear but no matter your point of view we can't ignore that it is a growing phenomenon. As with every growing technology new market segments are opening. Companies like Ledger are actively trying to capture their share of the hardware wallet market. The Ledger Company has sold over a million hardware wallets since its establishment in 2014. It was founded by a team of eight experts with various backgrounds in e.g. embedded security. They are famous for their Ledger NanoS and their Ledger Blue. In addition to these two hardware wallets they also provide several blockchain applications. In their aim to secure cryptocurrencies they do not rely on security through obscurity but rather openness. Organizing several capture the flag events through their Bounty Program helps them to increase their security even more. This is viewed as good way to approach security and more companies are starting to operate like it.

In order to understand what hardware wallets actually are we need to take a look at what wallets are in general. The necessity of wallets originates from the way valid transactions are made. A transaction is valid once the owner of the sending address can prove that the assets on that address are actually his. This can be proven by signing the transaction with his credentials. The credentials exist of a private key and a public key generated to fit the requirements the elliptic curve upholds. The wallet stores all the keypairs a user might have, generates new pairs and checks the addresses to determine your balance.

As seen before the private key is your access to your assets. The main priority of a wallet is to safely generate and store the private keys. There are several implementations depending on the platform on which the wallet operates. The host is one of the most important aspects when taking security

into account. From least secure to most secure, there are three generic types of wallets:

- **Web wallets**

Implemented on the internet, a trusted third party handles all the functionality of a wallet. They are often easy to setup and use. The user has no control over the main priorities of the wallet and can thus be seen as a less secure implementation. The necessity of a TTP is in conflict with the self-governing and decentralized character of cryptocurrencies.

- **Software wallets**

These can be installed on your device and if implemented correctly are the best combination of convenience and safety. A correct implementation is hard to achieve in the ever changing crypto space. Internet access can be a liability in some software wallets. That's a reason why they come in two types.

- **Cold Storage Wallet**

A computer without internet access is used to sign the transactions offline. They are then transferred to a computer with internet access to publish onto the network for the miners to verify.

- **Hot Wallet**

Whereas the cold storage wallet can't monitor for outputs and broadcast transactions, the hot wallet has all the features of a full service wallet.

- **Hardware wallets** By using a dedicated embedded device an additional layer of security is created. This makes hardware attacks nearly impossible. The offline, "secure" environment provides the generation and storage of the keypairs. This, in combination with the signing of transactions, is all done on a microprocessor. The hardware wallet has to be connected to a computer in order to publish transactions. The specifically designed microprocessor has a significant increase in security compared to an everyday processor found in a computer.

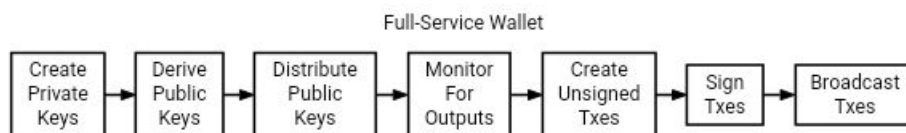


Figure 2.1: Tasks of a Wallet

Chapter 3

Side-channel Attacks

This chapter delves into all theoretical details that could be used in the search for an exploit.

3.1 Introduction

When thinking about attacks on cryptography the specific algorithm used is often the first thing under question. The field of research that looks into the mathematical analysis of these cryptographic algorithms is often called mathematical cryptanalysis.

The cryptographic primitives are being researched with access to a black-box. This entails that only the input and output are known i.e. the plaintext and ciphertext.

When doing a theoretical analysis you are either trying to break the algorithm, or try to find an algorithm to break the primitives. It is thus used to verify a certain level of security claimed by the inventor. The security claim can be seen as broken when an attack can be constructed that requires less resources and/or beats the probability that has been claimed.

A security claim consists of a series of parameters and a limit on expected security of a cryptographic algorithm. These claims are defined differently for symmetric cryptography and asymmetric cryptography.

In symmetric cryptography the claim consists of the probability an attacker has in successfully breaking the scheme defined with the following parameters:

- N , amount of computation
- M , amount of input/output computed with the secret key
- ϵ the upper bound, represents a function of M and N .

In asymmetric cryptography the claim is represented by the computational hardness assumption of the trapdoor function and the length of the keys. The legitimacy of an algorithm is therefore often proven mathematically and by the amount of analytic effort invested in it. The longer the algorithm has lasted without its security claim broken, the more secure and thrust worthy it will become.

The difference between theoretical analysis and practical analysis emerges from the real world implementation of these cryptographic schemes in chips and/or software. The blackbox no longer upholds in these implementations because of all types of leaked information. Practical analysis exploits the implementation details to manipulate and alter the output to reveal previously hidden information. The two general ways to abuse the leakage are fault-injection attacks[4] and side-channel[2] attacks. When intentionally inducing faults onto the implementation an attacker might invoke error messages or incorrect outputs. The results may leak information about e.g. the keys. Details about the implementation can thus be derived and security measures could be bypassed. In the case of a hardware wallet, the security measures in the secure microprocessor preventing the extraction of private keys could hypothetically be bypassed.

The faults can be initiated by different attacking mechanisms. The more common used attacks make use of out-of-range circumstances e.g. increasing the power usage or clock frequency and can be classified as *Injection with contact*[14].

With a low budget these can often be very successful. However if your budget is big enough highly expensive ion-beam attacks can be performed to break even the most sophisticated implementations. Although the heavy-ion radiation or electromagnetic fields are hard to control precisely. These types of attacks can be classified as *Injection without contact*.

The fault injection attacks require a lot of time and careful adjustments of the hardware in the target device. With hardware wallets in mind this would not be a likely attack vector for this thesis. The signing and publishing of transactions can only be done by the owner, which would require physical access to the device. An alternation will then soon be noticed.

With the initial assumptions being, “only observing the hardware wallet”, side-channel attacks are a promising attack strategy. Unknowingly a lot of implementations leak information about the key or secret data.

Measurements of e.g. execution time or electromagnetic emanations in combination with knowledge of the underlying system can be used to derive certain aspects of the device. The knowledge required makes the attacks highly specialized.

Execution time relates to timing attacks which abuse the fact that different

operations have a different computation time for each input. The execution time can therefore be linked to a specific input, leaking valuable information. Electromagnetic attacks abuse the fact that every device emits electromagnetic radiation. The wires and logic gates all carry current which in turn creates a magnetic field when performing computations. These electromagnetic waves can be captured and analyzed to reveal previously hidden information about the device. Several other side channels could be abused e.g. cache memory or acoustics but in general every measurement is a different attack vector which can be looked into.

The underlying system in this case is the secure micro-controller which seems like a vulnerable target for power analysis and in particular Simple Power Analysis or Simple Electromagnetic Analysis.

3.2 Simple Power Analysis

The framework in which the attack takes place as stated in the introduction lends itself for the capturing of measurements when using the hardware wallet for a transaction. The microprocessor performs operations such as key generation and ECDSA in which private keys are used. These conditions meet the characteristics as described by Kocher et al.[17] “Simple Power Analysis (SPA) is a technique that involves directly interpreting power consumption measurements collected during cryptographic operations.”

At the start it will be necessary to capture multiple traces* of the same operation over time to get a clear view of how and where the operations take place. This is called a multiple-shot SPA attack. Once the underlying architecture is clear the attack will transfer in a single-shot SPA attack and would require a minimal amount of traces of the victims wallet.

The algorithm that is performed during the signing of transactions is ECDSA. As seen in the preliminaries the private key is used when calculating the public key and s . Calculated by the equation, $s \equiv k^{-1}(m+ax)$ where the private key equals a . This scalar multiplication is an operation worth looking into since it can reveal a lot of information about ECDSA and in turn the private keys.

* A trace is a set of samples of a specific operation.

3.2.1 Visual Inspections of Traces

The specific assembly instructions performed during a scalar multiplication such as add, xor or move have a particular power consumption. Their uniqueness comes from, among others, the differences in bytes or components used. When capturing and processing the traces a clear distinction could for example be seen between the microprocessor being idle or under load. Another distinction could be visible between the instructions stated earlier. However the signal has to be quite strong and reliable. When attempting to look for information with visual inspection the spikes or drops in power usage are of importance. For example in figure 3.1, the signal clearly spikes in the red area. These parts of the trace can then be inspected further in the search for hidden information. In addition to the spikes and drops, repetitive pattern are also interesting. They can indicate a repeating pattern in the implementation of an operation.

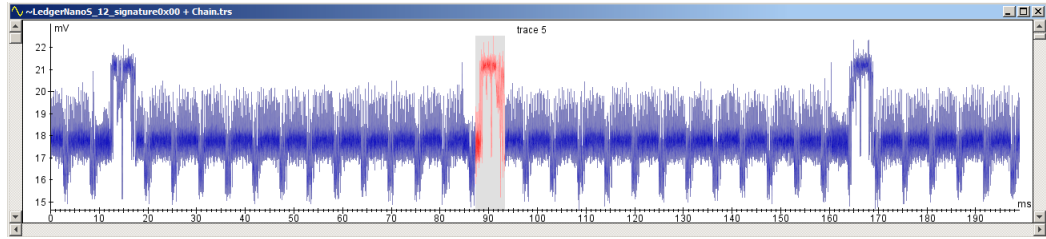


Figure 3.1: Spike in voltage

3.3 Template Attacks

Since it is not always possible to notice the disparity in traces, templates are used to characterize inputs according to a multivariate distribution[18]. This distribution consists of a mean vector and a covariance matrix. The mean vector has a mean for every variable in the measurements. The covariance matrix represents how the variables co relate to each other and how far the variable deviates from the average value, the mean. The covariance matrix grows quadratically because for every new variable the covariance between it and every other value has to be determined. To reduce the size it is important to look only at the points of interest. The parts of the trace noticed with visual inspection can thus be used to reduce the size. The generation of the distribution and application are often performed on different data sets. Hence template attacks are split into two phases.

3.3.1 Template Building Phase

In the building phase the mean vector and covariance matrix are initialized. As stated earlier the amount of traces is of great importance. The more traces captured and used, the better the templates will be. When trying to create a template for every possible key with an advised keysize between 256-383 it can take several thousand traces. In the case of ECDSA and especially scalar multiplications it is more interesting to look at the differences in the double and add algorithm. This reduces the amount of traces needed and thus the complexity of the covariance matrix. In theory the template would look like a combination of the multivariate distribution and the scalar. The building of the template for this particular example would include:

- Find every trace in which the operation is performed and store the points of interest.
- For every point of interest compute the average power consumption. This will result in the mean vector.
- Compute the variance of the points.
- Compute the covariance of every pair of points.
- Combine the last two and place the variance on the diagonal starting at the top left and the covariance between points i,j on position i,j of the matrix.

After following these steps the result will yield two matrices, the mean vector and covariance matrix. Since we have two particular operations defined (addition with 0 and 1) we need to repeat this process once more. After all the calculations two templates are formed and ready for matching.

3.3.2 Template Matching Phase

The previously created templates can now be matched with a random trace. The trace received should be of a transaction performed by the owner. When the scalar multiplication is performed in the signature algorithm we can try to classify it as an addition with either a 0 or a 1. However a single trace is not enough to perform a reliable template attack.

The traces have to be processed first:

- Find all the points of interest in the traces and put them in a vector.
- Calculate the PDF using the covariance matrix for both the templates. The Probability density function can be interpreted as:
“Looking at trace i , how likely is it that 0 is the correct one?”

- When comparing both probabilities, the template with the highest correlation might indicate the correct one.

After combining all these comparisons a guess for the scalar used in the multiplication can be made and thus a guess for the private key.

3.4 Differential Analysis

When the differences between traces are still not very clear after processing, differential analysis could be used to compare a lot of traces. While SPA often looks at differences in power consumed by a single operation, differential analysis looks at the tiny differences in power caused by different data over time. These tiny differences are often not noticeable with visual inspection because of noise or outliers. Therefore a lot of traces are needed to do a statistical analysis on specific parts of an algorithm. Kocher et al.[17] talked about the applications of differential analysis in their paper from 1999. They state that when analyzing the traces one could reverse-engineer particular operation details or even break whole cryptographic implementations. If all other side-channel attacks fail on the hardware wallet this might be a useful option.

Chapter 4

Practical Analysis

In the previous sections the foundation for the practical analysis has been laid. In general the remaining research conducted can be split into four parts.

- Analyzing the details of the Ledger NanoS and craft a custom exploitable program.
- Creating the setup and learn to work with the Picoscope software to take measurements.
- Getting familiar with the Inspector software and process the collected data.
- Performing an attack on the Ledger NanoS using the newly gathered information.

The following part of the thesis has been structured to show the results of the above four parts.

4.1 Technical Analysis

This small section at the start will contain the analysis of the hardware and software. Some of these will be applicable to every device and are an important part of the design choices while others will be more specific to the Nano S. But first some general information about the NanoS is presented.

The specific hardware wallet that has been observed and researched is the Nano S manufactured by Ledger. The Nano S is a Bitcoin, Ethereum and Altcoin hardware wallet. It is based on robust safety features for storing cryptographic assets and securing digital payments. Using USB it can connect to any computer and use a Google Chrome application to update the

firmware, load new programs or remove them. The OLED display in combination with the two side buttons are used to navigate through the menu and to double-check and confirm each transaction with a single tap on its side buttons. [10]

The underlying hardware and software is important because it not only tells something about potential vulnerabilities but also about how to develop programs. Since we are trying to make a program for the Nano S that can be exploited it is important to know the architecture.

In figure 4.1 can be seen that the Nano S has two microcontrollers, the STM32F042K [21] and the ST31H320 [20]. The STM32 handles all the input and output, it is connected to the screen, buttons and USB. It notifies the ST31 when new input is received. The communication with the "Secure Element" (ST31) is handled by the protocol called SEPROXYHAL. Through Events, sent by the STM32, the ST31 is called to perform cryptographic operations. It can respond with commands and status updates.

The Nano S uses a custom operating system called the Blockchain Open Ledger Operating System, or BOLOS in short. The firmware version used in this thesis is 1.3.1, although it has been updated to 1.4.2 at the time of writing. The applications are loaded on the ST31 using the BOLOS loader after which they can call the BOLOS kernel using syscalls. These syscalls can be used to perform the cryptographic operations.

It is important to denote that only the ST31 stores and uses the cryptographic secrets used by applications. Through their clear distinction between the two microcontrollers Ledger created an interesting proxy between these secrets and the user (and attackers). [9]

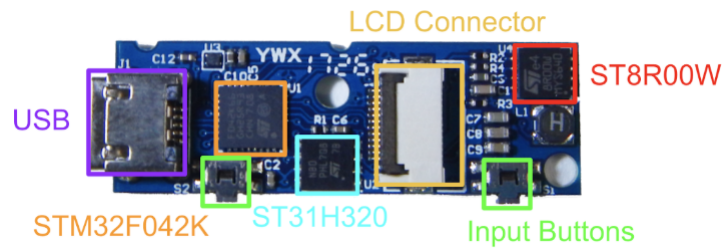


Figure 4.1: Inside the Nano S

4.2 Application

The Ledger NanoS, as stated earlier, uses the BOLOS platform which is build on top of a secure element. The applications ran on the BOLOS platform are compiled and linked before loaded onto the device. Hence to build custom applications for the NanoS a couple of things have to be setup correctly. Since we are working with two microcontrollers we are going to need a toolchain. This toolchain includes compilers and a linker. The MakeFile as provided by Ledger requires you to specify the path to the compilers. The compilers used are:

- GNU Arm Embedded Toolchain 5
- LLVM Clang 4.0.0

These pre-built compilers have to be put in the correct directories according to the names specified in the MakeFile, see A.6. After everything has been downloaded and configured this all has to be setup inside a virtual environment. To make sure the BOLOS platform can find the compilers, the environmental variable `BOLOS_ENV` has to be set to the `/bolos-devenv/` directory in which the compilers are located. The other necessary part is the software development kit (SDK) which provides the development libraries for ST31 applications. Again an environmental variable has to be set. The `BOLOS_SDK` has to be linked to the `/nanos-secure-sdk/` folder. The building, loading and communication is performed in the virtual environment initiated by the following command:

```
$ virtualenv virtualenv_ledger
```

After the initialization the right files will be copied into the environment. (See Appendix A.3.1 for details) To activate the virtual environment the

```
$ source virtualenv_ledger/bin/activate
```

command is invoked. When all the files are in place the installation phase can be started. In this phase the required python library's are being installed using pip. The ledgerblue library didn't install properly for this particular combination of NanoS, firmware and SDK. This has been bypassed by copying and installing a different version. The last phase is to start the compiling with the command:

```
$ make load
```

The makefile will remove the old application before installing a fresh version. The python library to connect to the NanoS is used like this:

```
python -m ledgerblue.loadApp $(APPLoadParams)
```

This whole process was automated with a couple of shell scripts to improve the speed and reliability in which changes could be processed and tested. In

addition to the speeding up of the installation process I also made a cleanup script (See A.3.4) which returns everything to the old state. It is pretty much what the filename says, but it is important to denote that the necessary files have to be copied back to the main folder. After which the virtual environment can be deleted and a new test can be run.

As a reference the official Sample Applications of Ledger were used and modified. In particular the SampleSign app. It asks for an input, verification and signs the input. The function in which the signing happens is:

```
io_seproxyhal_touch_approve(const bagl_element_t *e)
```

The touch approve part in the function name indicates the final button press on the device before the signature will be calculated. If the code is injected before the actual algorithm and it's return statement, the secure element can be triggered in running the code. The signing algorithm used, as claimed earlier, is ECDSA called by:

```
tx = cx_ecdsa_sign ((void*) &N_privateKey ,  
                   CX_RND_RFC6979 | CX_LAST,  
                   CX_SHA256, result , sizeof(result) ,  
                   G_io_apdu_buffer , 0);
```

This gave the opportunity to run scalar multiplications before the actual signatures. As seen in A.4 a chosen scalar can be used in combination with the following function:

```
cx_ecfp_scalar_mult(CX_CURVE_SECP256K1, point , scalar , 32);
```

It is now possible to have the device perform scalar multiplications on command.

In order to get the best measurements some kind of sleep or trigger function would be necessary to implement around the scalar multiplications. The sleep function would show idle-like behaviour before and after the injected code which would clearly indicate in which part of the power trace the points of interest are located. A trigger would give the best results as it can trigger the data collector to start and stop. This would immediately give a trace with points of interest.

Not all possible implementations of this functionality were attempted and the ones tried failed, besides the fact that some were too intrusive to attempt and did not represent the initial assumptions.

The use of NOP instructions looked like a promising implementation, since this uses the standard C library's. They are included in the SDK and are ready to be used without changing too much. The code below was put around the scalar multiplications. The hope was to see long pauses in between the different scalar multiplications ran. Unfortunately the NOP instructions were not noticeable on the traces.

```

for(i = 0; i < COUNTER; i++){
    asm volatile ("nop" ::) ;
}

```

Other possible implementations that could be tried are:

- The `sleep()` function from the `unistd.h` library, which would require a way to include the library into the SDK. It is used to make a program wait for the short interval specified in seconds. The function `nanosleep()` could also be used which offers, as the name suggests, an interval in nanoseconds.
- A software trigger that signals the capturing device to start before the scalar multiplication and stop after. Signaling a device from the NanoS could be challenging since it has a limited and protected communication functionality.
- A hardware or external trigger that would only capture data once the power is below or above a certain threshold. However this would require soldering of a cable on one of the capacitors. This might result in burning the capacitor if not done precisely or without specialized equipment.

4.3 Setup

With the application build and loaded onto the device the microprocessor is triggered to perform the scalar multiplications. The next step is to capture measurements. Several devices and programs were used in the full setup in figure 4.2:

- Picoscope PC Oscilloscope version 3207B with a memory of 512 MS, bandwidth of 250MHz and a resolution of 8 bits at 1GS/s.
- A Near Field Probe from Langer EMV-Technik, the RF U 5-2.
- Picoscope 6 Software for the 3000 series.
- Several python programs. (More on these in section 4.4)

The initial attempt was to capture traces without removing any casing, to get useful results the top lid was taken off eventually.(See figure 4.3) However the signal still wasn't very strong and clear and the setup could use some improvements.



Figure 4.2: Full Setup

After testing with several probes, the RF U 5-2 didn't prove to yield the best results, instead they were captured with the RF B 3-2. In addition to changing the probe, the PA 303 amplifier was added to boost the signal. With the earlier setup a difference in power could be seen. This was likely caused by pressing the buttons. Unintentionally the probe might have moved a little bit although it was pretty sturdy. After some trial and error to find the location on the chip with the strongest signal, the circuit board was fixed in place in a Stanley fixture. The choice to remove the circuit board from the casing was made to really eliminate possible measuring errors. The new setup should yield no difference in power unless it was measured correctly and it indicates some implementation details. The improved setup can be seen in figure 4.4.

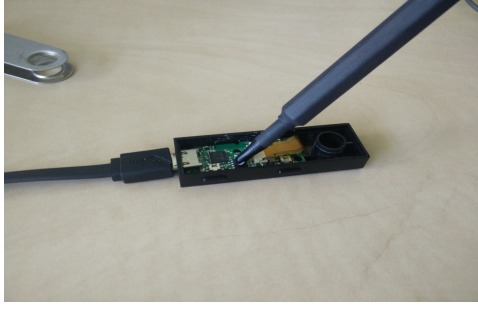


Figure 4.3: First setup



Figure 4.4: The improved setup

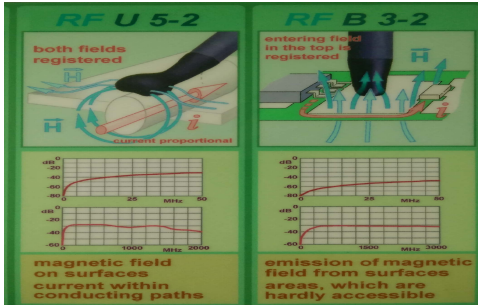


Figure 4.5: The two probes

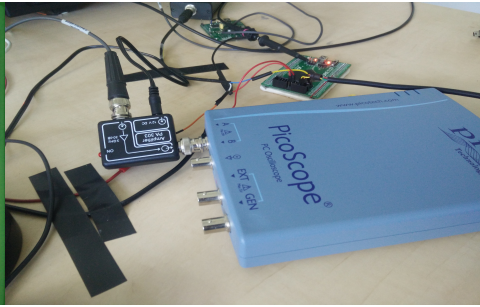


Figure 4.6: Amplifier and Picoscope

With the removal of the casing, the screen has also been disconnected to remove any static noise in an attempt to get an even clearer signal. The navigation through the menu and installation process could easily be written down. Since the Ledger has only two buttons the commands "Left", "Right" and "Both" (Left and Right at the same time) are used. An example of the unlock combination of the PIN 5556 would be BBBRBB, where the last "Both" is to confirm the PIN. Once the ledger is unlocked you can navigate through the applications and open them with "Both". In the case of the custom application it was even easier since it sent status information back to the terminal. See Appendix A.1 for more details.

Edit After firmware version 1.4.2 the start numbers on the PIN-code have been randomized. It will now be impossible to write the button combination down and unlock it without the screen attached. This also increases the level of security since the PIN-code can't be retrieved anymore by using acoustics or simply remembering the order in which the keys are pressed. However capturing traces can still be done while the owner is using the device, e.g. when signing a transaction.

4.4 Data collection

At the start of the experiments the data collection wasn't very optimal. The picoscope which was used to capture the data didn't have the functionality required, thought wrongly. With the code below the first measurements were taken since presumably the only form of output was to the console.

```
for ($i=1; $i -le 750; $i++){picoscope.com /a Measurements.CSV?  
>> C:\path\to\directory\measurements_full.csv}
```

The resulting CSV file was then processed and visualized using the python library Numpy and Excel. This makeshift solution was to check whether the actual changes to the application could be noticed. In figure 4.7 the difference between the application and the idle state could clearly be noticed, even under the bad circumstances in which the measurements were taken. This lead to believe that when improving both the setup and the way of collecting measurements some interesting previously hidden information might be revealed.

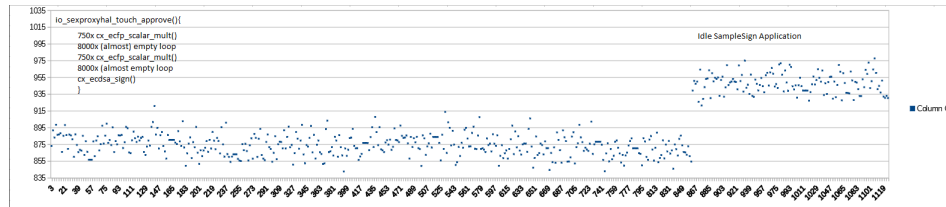


Figure 4.7: Piped output to .csv

As stated in section 3.6 the setup needed some improvements. The same could be said for the data collection. Although the earlier collection showed results it was soon proven that this was, as expected, not very precise. In order to process the data correctly it has to be split into traces. Each trace has to have enough samples with the aim of getting the most accurate results. When capturing these samples it would be wise to already store them in the required traces. This would help in the data processing part later on.

A library that could help with this called pico-python came to the attention. This library was written by O'Flynn and Harfouche [6]. It has support for the PS3000A Series which includes the 3207B that is used in the setup. It provides a PS3000 class and the dll files which can be used for the communications with the instrument. This is significantly more precise than the console output used earlier. The python program that uses this library was developed earlier by a researcher of the Radboud University. After receiving the permission to use it for this thesis, it was modified for the specific purpose. In general the program captures a traceset with a specified number of traces. On average the amount of traces captured were around a thousand.

The traceset is written to a file with the .trs extension, following its particular format. In the processing phase the file can conveniently be opened by the Inspector software.

The program first creates all the necessary headers to fit the format a .trs file upholds. It then writes the header to the file and starts the data collection. For every trace it will capture the necessary samples, collect them into a trace and writes the trace to the file. There are three main variables in the program which can be changed(See A.5 for all settings):

- The sampling rate, which has a maximum of 1GS/s limited by the Picoscope. One GS equals 10^9 samples. To get the highest precision the maximum sampling rate is used. In the code this parameter has been set to $1e-9$ since it requires $1/\text{sampling rate}$.
- The trace duration, which is set to $199000e-6$. This was the maximum before a memory error occurred. It estimates to roughly 199 million samples a trace.
- The voltage range, which is set to $500e-3$ which equals 500 millivolts. The maximum voltage measured shouldn't exceed this threshold.

Changing these variables yield differences in the traces, after some testing the specified numbers worked for this particular purpose. However the settings are not claimed to be optimal.

After configuring, the python program can be executed from the command line and the data collection will start. In the meantime the NanoS should be executing the SampleSign application for a large number of scalar multiplications. It is important to note that in the current setup the capture program and the NanoS application are ran separately and controlled by different computers. To prevent capturing traces of an idle application a sufficient amount of multiplications have to be performed. A rough estimate for a thousand traces is around two hundred thousand multiplications. After measuring the time it took I came to an estimate of fifteen multiplications a second, in total this will result in around three or four hours of capturing time.* This should be enough time for the capture program to finish.

*Make sure the two computers don't go in sleep mode after a while

4.5 Data processing

After capturing a significant amount of traces the data processing can be done with the Inspector software. Inspector is a service provided by Riscure[24]. It is commonly used for Side Channel Analysis and can also be used for the capture of traces. With the current setup the latter was unfortunately not possible but could however improve the quality of the traces.

The general impact that was sought after with processing is the reduction of noise or outliers. After which the expected results are a clear pattern that indicates where the scalar multiplications are performed. In order to achieve this several filters can be put on the traces. After testing with a couple of them and some advice from the people over at Riscure a chain of filters was setup. The chain consists of two, the absolute and the windowed resample filter. In figure 4.8 the settings for the chain can be seen. The "First:" and "Number:" indicate at what sample or trace the chain should start and how many it should process onwards.

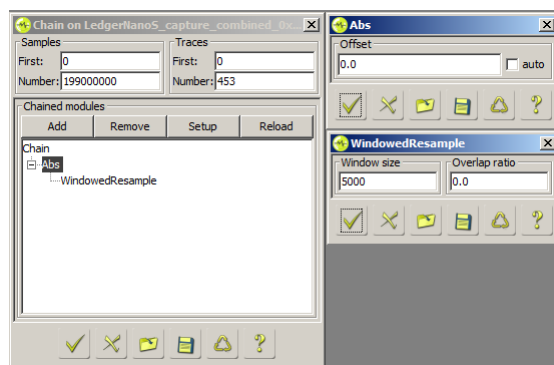


Figure 4.8: Settings of the chain

In this case every sample and trace could be of value and that's why the full trace set was put through the chain of filters to hopefully achieve the desired results. In figure 4.9 the raw trace can be seen but there is no distinct pattern (yet).

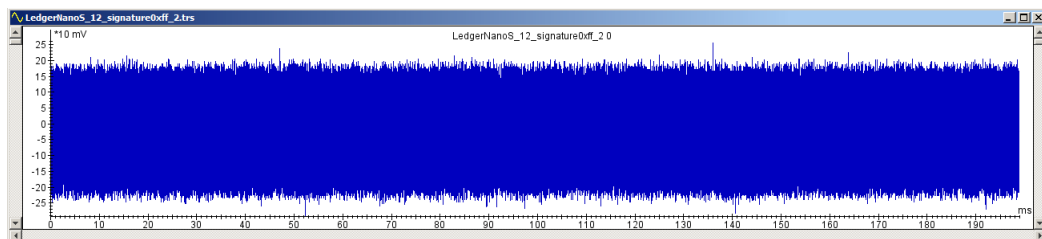


Figure 4.9: Unprocessed trace

The two filters are explained shortly below, for more information see the Inspector Documentation [24].

1. The "abs" filter took the absolute value of the samples relative to the offset. It is mainly used to rectify an AC signal, which basically means that you get rid of the negative voltages. The offset can be calculated automatically by Inspector but in this case the offset was set to 0.

The mathematical notation for this operation is:

For trace set S : $Abs_{offset}(S) = \{Abs_{offset}(s) | s \in S\}$.

For trace s : $Abs_{offset}(s_i) = |s_i - offset|$

After applying the filter the result looked like figure 4.10.

2. With a thousand full-size traces the computation time for statistical analysis can increase quickly. The "windowed resample" filter can be used for compression and reduce the 300GB trace set to a workable format. Compression will therefore result in an increase in performance later on. The filter requires two inputs:

- (a) The window size, an integer that indicates how many of the samples have to be compressed into one.
- (b) The overlap ratio, indicates what percentage of the samples should overlap.

In addition to compression the windowed resample also provides a better Signal-to-noise-ratio. The noise around the points of interest is reduced by averaging several samples that carry the same signal.

After applying all the filters a repeating pattern can be seen in figure 4.11, which wasn't visible in figure 4.9 before.

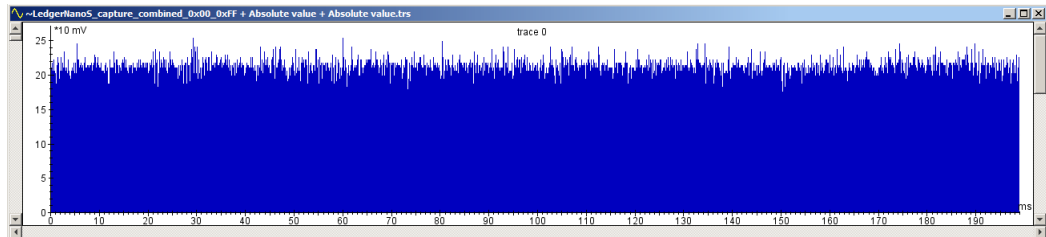


Figure 4.10: Abs filter applied

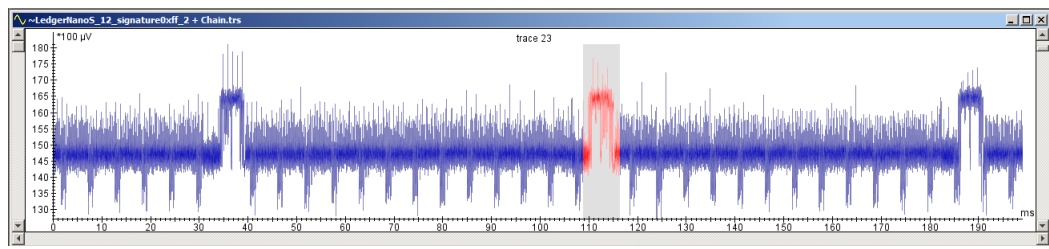


Figure 4.11: Chain of filters applied

4.6 Results and Observations

After the data processing a pattern emerged as seen in figure 4.11. This spike is interesting because an idle state should consume less power than an actual operation. This led to the belief that this might have been the scalar multiplication. When zooming in on the spike as in figure 4.12, we can see several drops and spikes in this pattern. This could not be processed with just visual inspection. After performing an Inspector feature called Pattern

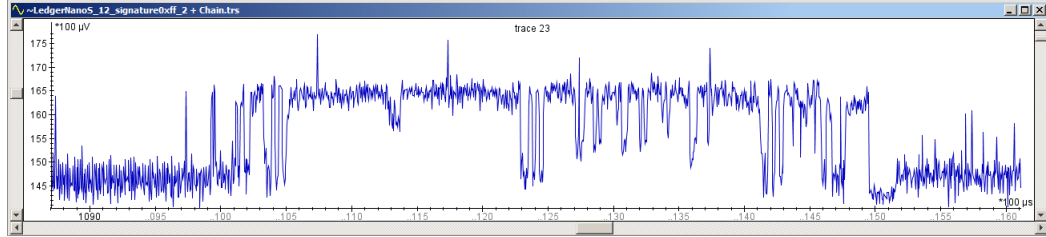


Figure 4.12: Assumed Zoomed Scalar Multiplication

Matching, as seen in figure 4.13 a correlation between the peaks of around 90 percent was shown. When comparing this spike in a trace of a 0x00 scalar and a 0xFF scalar there was still a clear correlation. After consulting with

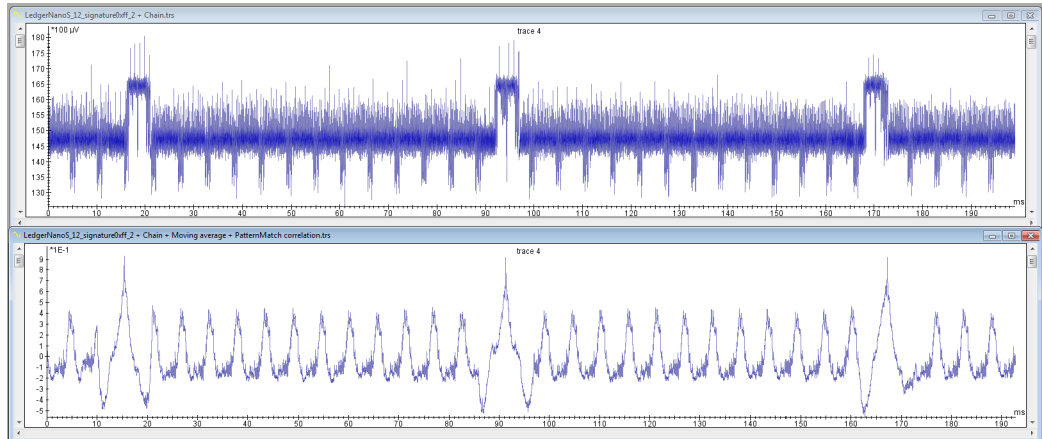


Figure 4.13: Pattern Matching

the people from Riscure it was concluded that this was too short to be the scalar multiplication, in addition to the lack of differences between the two scalars. As seen in figure 4.14 the only real difference between the blue and orange trace is the signal strength. The other possibility was that the power actually drops when performing the scalar multiplication and as a counter-measure the idle state consumes more power. This would mean that the long pattern in between the spikes in figure 4.13 is the scalar multiplication.

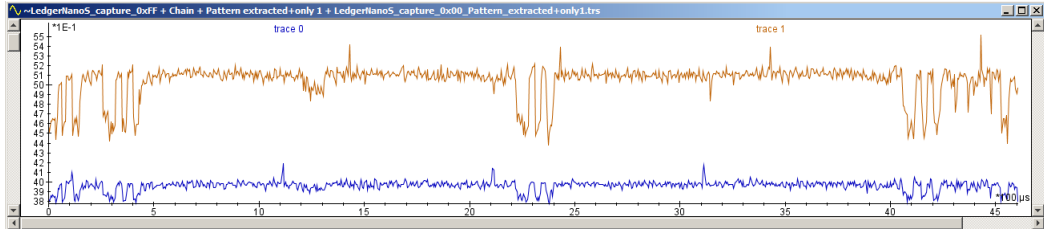


Figure 4.14: Comparison of 0xFF and 0x00

When overlapping the traces of the idle state(blue) with traces of the scalar multiplications(orange) in figure 4.15 a drop in power could indeed be seen. With this strong assumption some sort of confirmation was sought after.

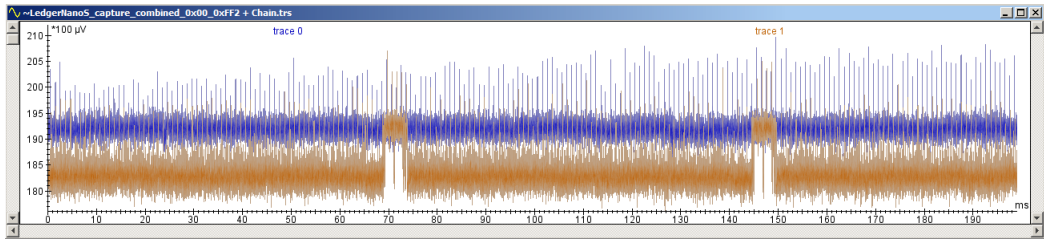


Figure 4.15: Drop in Power

This confirmation came from the amount of scalar multiplication performed in one second, measured earlier in section 4.4. In figures 4.16 and 4.17 can be seen that the time measured between two spikes is nearly constant at a time around 70ms (selected:14143(70.72ms)).

In combination with the measured time of 15 multiplications a second this would imply that this is the actual multiplication. The total time is 1 second or 1000 milliseconds, $1000\text{ms}/15$ equals around 67 ms.

It can also indicate that the implementation is a constant time scalar multiplication function. The most common way to implement the scalar multiplications function is the double and add method. As explained earlier the add will only be executed in the case of a 1. These implementations of scalar multiplication thus have side channel leakage since the scalar with 0's should have a shorter execution time than the one with 1's. Ways to prevent this are e.g. to always double and add. In the case of a 1, nothing changes. In the case of a 0, just compute the addition but don't store it. These types of scalar multiplications therefore have a constant execution time. This observation can tell something about the implementation details of the scalar multiplication function used in the Nano S.

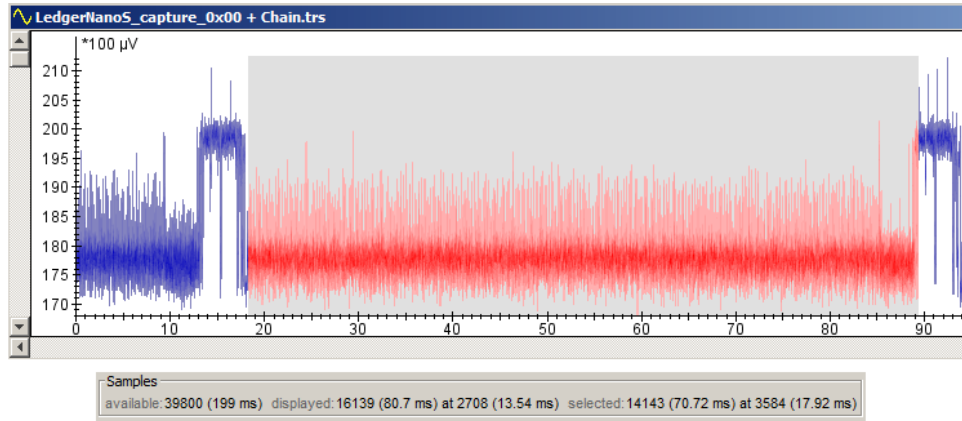


Figure 4.16: Time of a presumed scalar multiplication 0x00

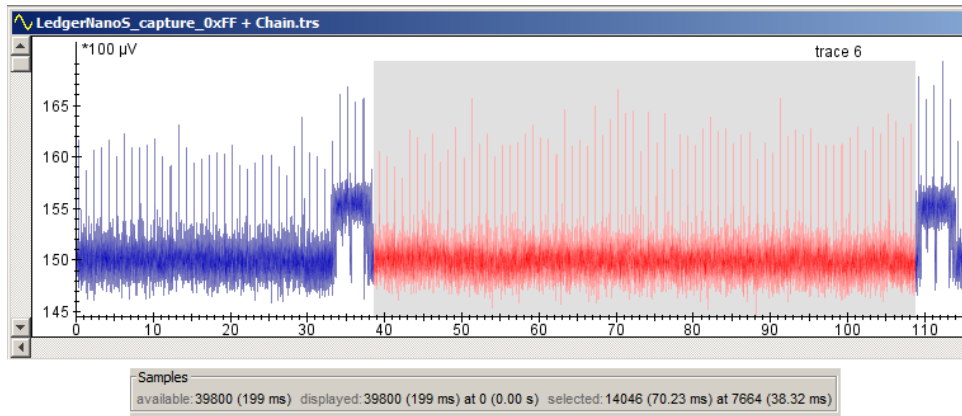


Figure 4.17: Time of a presumed scalar multiplication 0xFF

Another observation that could be made is that there still is a difference in power between the two scalars, as seen in figure 4.18. Either the improvements to the setup didn't help or this could be interesting to research further. In addition to that it also shows a repeating pattern of 12 to 13 drops in between the idle spikes.

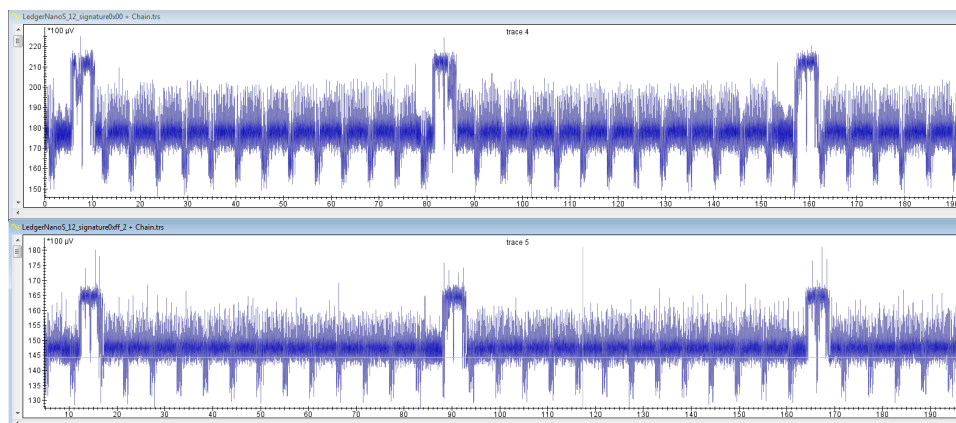


Figure 4.18: 0x00 and 0xFF aligned

Chapter 5

Related Work

As touched on earlier in the introduction there is quite a lot of active research in this particular field. The fundamentals that appear in every paper are often the more general approaches such as power analysis [17] or electromagnetic analysis[2]. These methods have proven time and time again that they can be useful, especially when attacking public key cryptographic systems[22][26].

Through the bugbounty program of Ledger several possible attacks have been reported. In the newest updates to the Ledger Nano S these flaws, such as a Man-in-the-middle-attack[8][7], have been adjusted. This attack exploited the address display on the host machine. It could change the Chrome Application to always return the attackers address instead of the one provided by the hardware wallet. However this attack should not be very likely considering the hardware wallet also displays the address. When seeing this mismatch the user should be alarmed. It can become a valid attack when it is combined with the fact that in early versions the addresses were not fully displayed on the wallets screen. The attacker can now generate an address which looks like the one provided by the hardware wallet and the user will not be alarmed that easily. Fortunately after a firmware update the addresses are now displayed fully.

Other interesting attack strategies involve the PIN protection since recovering the PIN basically gives full access to the hardware wallet. On the Nano S the right combination of button presses would indicate a unique PIN. The entry of a number from the PIN would always start with a 5. Press Right to go up and Press Left to go down, confirm with both buttons pressed. On firmware version 1.4.2 they randomized the start of an entry.

A presentation at DEF CON 25 revealed another attack on the PIN protection. Datko et al. [15] demonstrated that the microprocessor STM32F205 could be glitched. They made use of a combination of Vcc and clock glitching attacks which lead to bypassing the pin.

These high-level attacks focus on the communication between the hardware wallet and the user or host machine. Low-level attacks such as the one performed by Salim [23] focus more on the implementation details of the hardware wallet itself. It mainly looks at the architecture built around the ST31 "Secure Element". He states that while the secure element might be safe from tampering the other microprocessor often is not. Another interesting publication briefly mentioned already is the one by Hoenicke [13]. He performed a successful SPA attack on one of Ledger's competitors, the Trezor hardware wallet. The specific part of the implementation that is used to generate the public keys was visible on the power traces. As we know the public key is computed by performing the `scalar_multiplication` function on the private key and the base point. The traces eventually revealed the private key.

Chapter 6

Conclusions

At the start of the practical analysis several goals were set and the majority of them were met.

To recap, all the work prior to the actual attack on the Ledger Nano S were successful to some extent. The analysis at the start was pretty straight forward and showed some interesting points of entry. The application that has been built for the Nano S works and could be used for the attack in mind. To capture the traces needed for the attack the final setup has been built which seems far more reliable than earlier versions. After completing the measurement phase the Inspector software could be effectively used in the processing of these traces. After the chain of filters some interesting patterns emerged. However the final attack wasn't accomplished unfortunately. It would require more time and effort in combination with additional analysis.

6.1 Future Work

In future research the implementation of the application can be improved to add some kind of software trigger or sleep functionality. This gives a clearer indication of where the actual scalar multiplications are done, when they start and when they stop. The measurements can then be performed more precisely since the trigger can start the measuring and if we know when it stops it is also possible to stop the measuring. This can be automated with the Inspector Software and ran multiple times until the wanted results are achieved. There are two options to proceed now:

- Try to use differential analysis on the new traces. The traces should include the part where the scalar multiplications are happening, as observed earlier in section 4.6.
- The template building phase could be initiated with the traces and a template for the 0's and 1's could be derived. The template matching phase has to be performed on the measurements of the Bitcoin

application, which also uses ECDSA with the secp256k1 parameters.

If these are successful a private key can be derived. This private key gives access to the funds corresponding to that transaction. In order to get all the funds the master key has to be derived. Which is a challenge on it's own.

Bibliography

- [1] Bitcoin wiki. URL: <https://en.bitcoin.it/wiki/Secp256k1>.
- [2] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The em side—channel(s). In Burton S. Kaliski and Christof Koç, çetin K. and Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 29–45, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
- [3] Myrto Arapinis Andriana Gkaniatsou and Aggelos Kiayias. *Low-Level Attacks in Bitcoin Wallets*. Springer International Publishing, 2017.
- [4] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proceedings of the IEEE*, 100(11):3056–3076, Nov 2012. doi:10.1109/JPROC.2012.2188769.
- [5] Elaine B. Barker, William C. Barker, William E. Burr, W. Timothy Polk, and Miles E. Smid. Sp 800-57. recommendation for key management, part 1: General (revised). Technical report, Gaithersburg, MD, United States, 2007.
- [6] Mark Harfouche Colin O’Flynn. Pico-python, 2013. URL: <https://github.com/colinoflynn/pico-python>.
- [7] Ledger Company. *Ledger Receive Address Attack*. URL: <https://www.docdroid.net/Jug5LX3/ledger-receive-address-attack.pdf>.
- [8] Ledger Company. *Man in the Middle Attack – Am I at risk?* URL: <https://www.ledger.fr/2018/02/05/man-middle-attack-risk/>.
- [9] Ledger Company. *Ledger Documentation Hub*. 2016 - 2017. URL: https://ledger.readthedocs.io/en/latest/bolos/hardware_architecture.html.
- [10] Ledger Company. *Product Information*. 2018. URL: <https://www.ledgerwallet.com/products/ledger-nano-s>.

- [11] Certicom Corp. *SEC 2: Recommended Elliptic Curve Domain Parameters*. 2010. URL: <http://www.secg.org/sec2-v2.pdf>.
- [12] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc. URL: <http://dl.acm.org/citation.cfm?id=19478.19480>.
- [13] Jochen Hoenicke. *Extracting the Private Key from a TREZOR*. 2015. URL: <https://jochen-hoenicke.de/trezor-power-analysis/>.
- [14] Mei-Chen Hsueh, T. K. Tsai, and R. K. Iyer. Fault injection techniques and tools. *Computer*, 30(4):75–82, Apr 1997. doi:10.1109/2.585157.
- [15] Kirill Belyayev, Josh Datko, Chris Quartier. *Glitches cause stitches!* URL: <https://media.defcon.org/DEF%20CON%2025/DEF%20CON%2025%20presentations/DEFCON-25-Datko-and-Quartier-Breaking-Bitcoin-Hardware-Wallets.pdf>.
- [16] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987. URL: <http://www.jstor.org/stable/2007884>.
- [17] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 388–397, 1999. URL: https://doi.org/10.1007/3-540-48405-1_25, doi:10.1007/3-540-48405-1_25.
- [18] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security) Chapter 5*. Springer-Verlag, Berlin, Heidelberg, 2007.
- [19] Victor Miller. Use of elliptic curves in cryptography., 01 1985.
- [20] STMicroelectronics NV. St31h320. URL: http://www.st.com/content/ccc/resource/technical/document/data_brief/ac/ed/36/cb/1c/04/43/a8/DM00240763.pdf/files/DM00240763.pdf/jcr:content/translations/en.DM00240763.pdf.
- [21] STMicroelectronics NV. Stm32f042. URL: <http://www.st.com/content/ccc/resource/technical/document/datasheet/52/ad/d0/80/e6/be/40/ad/DM00105814.pdf/files/DM00105814.pdf/jcr:content/translations/en.DM00105814.pdf>.

- [22] Elisabeth Oswald. Enhancing simple power-analysis attacks on elliptic curve cryptosystems. In Burton S. Kaliski and Christof Koç, çetin K. and Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 82–97, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
- [23] Saleem Rashid. *Breaking the Ledger Security Model*. 2018. URL: <https://saleemrashid.com/2018/03/20/breaking-ledger-security-model/>.
- [24] Riscure. Inspector software. URL: <https://www.riscure.com/security-tools/inspector-sca/>.
- [25] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978. URL: <http://doi.acm.org/10.1145/359340.359342>, doi:10.1145/359340.359342.
- [26] Elena Trichina and Antonio Bellezza. Implementation of elliptic curve cryptography with built-in counter measures against side channel attacks. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 98–113, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

Appendix A

Appendix

A.1 Installation and Loading without screen

To install and run the application without the screen connected the following buttons have to be pressed in combination with the right commands on the computer connected:

- "BBBRBB" → Unlock NanoS
- source make_env.sh
- "R" → Remove Application
- wait 5-10 seconds
- "R" → Install application
- wait for "application full hash ..." message
- "R" → Confirm
- "BBBRBB" → Loading from User Key
- "RRB" → Navigate and open SampleSign application
- "R" → Confirm non-genuine application
- python demo.py
- Type input to sign and press enter twice
- "R" → Verify Message
- "R" → Start Signing

After starting the signing, the scalar multiplications in front will also start. This is the time to start capturing traces. Some error messages found:

- If the console outputs "Dongle not Found", the device is not unlocked properly.
- If the installation doesn't succeed after "Application Full Hash", retry installation.
- If the wrong application is selected the console will output "Exception: Invalid status 6e00(Unknown Reason)".

A.2 Final Setup Specifications

- Picoscope PC Oscilloscope version 3207B with a memory of 512 MS, bandwidth of 250MHz and a resolution of 8 bits at 1GS/s.
- A Near Field Probe from Langer EMV-Technik, the RF B 3-2.
- Picoscope 6 Software for the 3000 series.
- PA 303 amplifier
- capture.py (https://github.com/wouter-db/Ledger_NanoS)
- SampleSign application (https://github.com/wouter-db/Ledger_NanoS)
- Samsung Series 5 535U3C running Ubuntu 17.10 (<https://www.notebookcheck.net/Review-Samsung-Series-5-535U3C-Notebook.83159.0.html>)
- Ledger NanoS firmware version 1.3.1 (newest firmware 1.4.2)
- Inspector 4.12
- Transcend StoreJet 35T3 HDD extern 8TB 3.5 inch USB 3.0

A.3 Shellscripts

A.3.1 make_env.sh

```
#!/bin/bash
clear
virtualenv virtualenv_ledger
mv ./nanos-secure-sdk ./virtualenv_ledger
mv ./bolos_env ./virtualenv_ledger
mv ./SetEnvironment.sh ./virtualenv_ledger
mv ./src ./virtualenv_ledger
mv ./demo.py ./virtualenv_ledger
mv ./Makefile ./virtualenv_ledger
mv ./install.sh ./virtualenv_ledger
source virtualenv_ledger/bin/activate
cd virtualenv_ledger
source install.sh
source install.sh
```

A.3.2 install.sh

```
#!/bin/bash
pip install ledgerblue
pip install secp256k1
pip install -I ECPy==0.8.1
pip install python-u2flib-host
source SetEnvironment.sh
cd ..
cp -R ./ledgerblue ./virtualenv_ledger/lib/python2.7/site-packages/
cd virtualenv_ledger
pip install ledgerblue
make load
```

A.3.3 SetEnvironment.sh

```
#!/bin/bash
clear
export Ledger_NanoS_path=$(pwd)
export BOLOS_SDK="$Ledger_NanoS_path/nanos-secure-sdk"
export BOLOS_ENV="$Ledger_NanoS_path/bolos_env"
env | grep "BOLOS"
```

A.3.4 reset.sh

```
#!/bin/bash
mv ./virtualenv_ledger/nanos-secure-sdk .
mv ./virtualenv_ledger/bolos_env .
mv ./virtualenv_ledger/SetEnvironment.sh .
mv ./virtualenv_ledger/src .
mv ./virtualenv_ledger/demo.py .
mv ./virtualenv_ledger/Makefile .
mv ./virtualenv_ledger/install.sh .
rm -r virtualenv_ledger
```


A.4 Main.C

```
unsigned char point[sizeof(C_SECP256K1_G)];
unsigned char scalar[32];
for (int i = 0; i < 15; i++) {
    memcpy(point, C_SECP256K1_G, sizeof(C_SECP256K1_G));
    for (int j = 0; j < 32; j++)
        scalar[j] = 0x00;
    scalar[31] = 2;
    ty = cx_ecfp_scalar_mult(CX_CURVE_SECP256K1, point, scalar,
        32);
}
```

A.5 Capture.py

```
# tracing context
tc = {}
tc["collection_title"] = "LedgerNanoS_capture_0x00"
tc["collection_description"] = "LedgerNanoS_capture_0x00, _
    capture_date_2018-05-22"
tc["collection_input_range"] = range(0,16)
tc["traces_per_traceset"] = 100
tc["sample_interval"] = 1e-9 # 1 / sampling rate: 1GS/s -> 1e-9
tc["trace_duration"] = 199000e-6 # Works out to 60000 samples /
    trace.
tc["capture_channel"] = "A"
tc["voltage_range"] = 500e-3
tc["coupling"] = "DC"
tc["trigger_voltage"] = 0.0
# How many traces do you need per rapid block call?
# Keep memory constraints of the device in mind.
tc["traces_per_rapid_block_call"] = 1
tc["scope_serial"] = "CO130/005"

# Do we want to truncate the end of the trace?
# Since the trigger happens at the end,
# about 35000 samples are superfluous.
# Truncation makes trace size more manageable.
tc["truncate"] = 0
```

A.6 MakeFile

```
ifeq ($(BOLOS_SDK),)
$(error BOLOS_SDK is not set)
endif
include $(BOLOS_SDK)/Makefile.defines

# Main app configuration
APPNAME = "Sample Sign"
APPVERSION = 1.0.0
APPLoad.PARAMS = --appFlags 0x00 $(COMMONLoad.PARAMS)

# Build configuration
APP.SOURCE_PATH += src
SDK.SOURCE_PATH += lib_stusb lib_stusb_impl

DEFINES += APPVERSION="\$(APPVERSION)\\"

DEFINES += OS_IO_SEPROXYHAL IO_SEPROXYHAL_BUFFER_SIZE_B=128
DEFINES += HAVE_BAGL HAVE_SPRINTF
DEFINES += PRINTF\(...\)=

DEFINES += HAVE_IO_USB HAVE_L4_USBLIB IO_USB_MAX_ENDPOINTS=7
          IO_HID_EP_LENGTH=64 HAVE_USB_APDU

# Compiler, assembler, and linker
CLANGPATH := $(BOLOS_ENV)/clang-arm-fropi/bin/
GCCPATH := $(BOLOS_ENV)/gcc-arm-none-eabi-5.3-2016q1/bin/

CC := $(CLANGPATH)clang
CFLAGS += -O3 -Os

AS := $(GCCPATH)arm-none-eabi-gcc
AFLAGS +=

LD := $(GCCPATH)arm-none-eabi-gcc
LDFLAGS += -O3 -Os
LDLIBS += -lm -lgcc -lc

# Main rules
all: default
load: all
      python -m ledgerblue.loadApp $(APPLoad.PARAMS)
delete:
      python -m ledgerblue.deleteApp $(COMMONDelete.PARAMS)
# Import generic rules from the SDK
include $(BOLOS_SDK)/Makefile.rule
```