Radboud University

MASTER'S THESIS COMPUTING SCIENCE

IN CYBER SECURITY

Online Template Attack On ECDSA:
Extracting Keys Via The Other Side

By
N.W.A. ROELOFS

Supervisor
Prof. dr. L. BATINA

Second Assessor
Prof. dr. J.J.C. Daemen

Daily Supervisor
N. SAMWEL MSc

Radboud University Nijmegen
Institute for Computing and Information Sciences
Digital Security

August 18, 2019

Abstract

The last couple of decades side-channel analysis has proved to be an adequate method to extract confidential material such as cryptographic keys in unforeseen ways. This thesis extends that notion by compromising the secret ECDSA key used while signing a message, by applying the concept of an online template attack via its verification counterpart. Via the attack it is possible for an adversary to reconstruct a secret scalar bit by bit where only one power trace is needed of the target, and two templates per bit to recover. Once the scalar is known, the secret key can be computed trivially. The attack is feasible because there is an operation when signing which depends on a bit value of the secret scalar, a squaring in the case of our investigated Montgomery ladder as scalar multiplication method. From the verification side, this operation can be imitated by an adversary with different inputs, which allows the creation and thereafter matching of templates. In order to prevent this attack from happening, dissimilarities between the signing and verification part of ECDSA should be created. That is, somehow we should make it nearly impossible for an attacker to build meaningful templates. An effective method to achieve this goal is to make use of randomized projective coordinates while signing. In this case it becomes nearly impossible for an attacker to calculate intermediate values of the Montgomery ladder which are required to create templates. As a consequence, the discussed attack becomes unfeasible.

Acknowledgments

First of all I would like to thank my supervisor Lejla Batina for setting the scope of my thesis, providing me directions and helping me to finalize the project. On top of that, I would like to thank her for providing me the opportunity to participate in the Summer school on real-world crypto and privacy in Šibenik, Croatia. Additionally I want to thank my second assessor Joan Daemen for introducing the ideas that have been implemented in this thesis.

Furthermore, a big shout-out to Niels Samwel for helping me with practical implementation issues during my thesis, acting as a sparring partner and being nice company in the side-channel lab these months!

Finally, I would like to thank my employer for giving me the opportunity to invest in my academic schooling. I hope it will prove to be of added value for the organization.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

AES	Advanced Encryption Standard
CWLC	ChipWhisperer-Lite Classic
DPA	Differential Power Analysis
DRAM	Dynamic Random Access Memory
DSA	Digital Signature Algorithm
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
EdDSA	Edwards-curve Digital Signature Algorithm
EM	ElectroMagnetic
FFT	Fast Fourier Transform
HMAC	Hash-based Message Authentication Code
LSB	Least Significant Bit
MSB	Most Significant Bit
NAF	Non-Adjacent Form
NIST	National Institute of Standards and Technology
NSA	National Security Agency
OTA	Online Template Attack

SCA	Side-Channel Attack
SPA	Simple Power Analysis
TEMPEST	Not an abbreviation, refers to EM emission
VM	Virtual Machine

Chapter 1

Introduction

Cryptology is the research field which investigates the methods on how to achieve and compromise the confidentiality, integrity and authenticity of information. Confidentiality is all about preventing the disclosure of information to unwanted parties, typically achieved by encrypting a piece of information with a secret key using symmetric encryption such as the Advanced Encryption Standard (AES) [10]. The term symmetric refers to the aspect that both communicating parties use the same key to encrypt and decrypt information.

The term integrity refers to the concept that it should be impossible to modify a message as an outsider in a conversation without any party noticing, which can be accomplished by sending a hash of the message along side the message itself. When the the receiving party computes its own hash over the message and it matches the hash received from the other party, it is highly likely that the message has not been modified. In today's practice, integrity is often implemented with a Hash-based Message Authentication Code (HMAC) [22].

Finally, the principle of authenticity entails proving that a message indeed originated from a certain party, which is commonly done via some signing principle based on asymmetric cryptography. The latter one means that different keys are used for both signing and verifying a message, often referred to as a private and public key. Two very common examples are RSA [32] and algorithms based on Elliptic Curve Cryptography (ECC) such as the Elliptic Curve Digital Signature Algorithm (ECDSA) [16] and the Edwards-curve Digital Signature Algorithm (EdDSA) [6].

1.1 Overview of Side Channel Attacks

As the title of this document already suggests, the goal of this master thesis is to investigate the security of an ECDSA implementation against a certain type of attack in order to extract secret signing keys. Once an adversary gets possession of a key, it can impersonate the original holder of that key, which as a result breaks the authenticity goal described before.

The method applied in this thesis to extract secret information is called a Side Channel Attack (SCA). With this approach we do not attack a cryptographic protocol from a formal mathematical perspective but instead target its practical implementation of it on some device. That is, we try to extract something secret via something unforeseen, a side channel. This sounds rather vague but can be made more clear by giving an overview of some, not all, different types of side channels and introduce the most important taxonomy, as has been down below.

Thanks to Kocher in 1996, the concept of SCAs went mainstream. He introduced a paper [21] describing by simply measuring the execution time of a signing algorithm he could somehow extract secret key material. To get a more concrete idea of the concept of a SCA, we simplify the invention of Kocher and apply it on Algorithm 1.

Algorithm 1: Naive PIN verification.

```
   Input  : $pin[4]$
   Output: $pin\_accepted$
1  $pin\_accepted \leftarrow 0$
2  if $pin[0] == 4$ then
3      if $pin[1] == 2$ then
4          if $pin[2] == 4$ then
5              if $pin[3] == 2$ then
6                  $pin\_accepted \leftarrow 1$
7              end
8          end
9      end
10 end
11 return $pin\_accepted$
```

The algorithm itself is implemented correctly in a sense that the variable *pin_accepted* contains the value 1 only when the PIN 4242 is provided. However, the algorithm is not time constant. Namely, when the first digit of the PIN supplied is correct, the execution of the algorithm will take longer because the second if statement will be evaluated. If an incorrect value was supplied, this would not happen. An adversary can exploit this weakness to extract digit by digit of the PIN when measuring the execution time. On average, this attack method will take $5^4 = 625$ tries to retrieve the PIN which is only a fraction of the $10000/2 = 5000$ tries a brute force approach would take. In Section 2.3.1 we give another example which implements the timing attack principle of Kocher but then in a more cryptographic setting.

In 1999 Kocher provided another breakthrough in the field of SCA: extracting secret material via power analysis [20]. He showed that it was possible to extract key material via simply inspecting a power consumption trace and as an alternative by applying basic statistics on multiple power consumption measurements. Both techniques will be more thoroughly introduced in Section 3.1.

Once the power consumption was recognized as a viable side channel, it was also exploited indirectly via the ElectroMagnetic (EM) emission of a device [1]. After all, the amount of emission is proportional to the power consumption used. The general term of a side channel attack exploiting EM emission is also referred to with the term TEMPEST, from origin a code name from the National Security Agency (NSA), and is described further in Subsection 3.2.1.

Chari et al. brought the power consumption analysis to another level by introducing template attacks [8]. It is the best known example of a profiled attack, in contrast to the techniques introduced by Kocher, which are non-profiled attacks. The term profiling refers to creating templates of a device which captures the behavior of a device while executing a certain operation. See Subsection 3.1.3 for more details.

Since we are giving a little overview, we do not want to leave out some significant pointers to other SCAs before we are going to narrow our scope to the topics relevant for this thesis. The out-of-order execution is a type of attack which received a lot of media attention with Spectre [19], Meltdown [25] and later on with RIDL [37] and Fallout [28]. It is important to distinguish between these type of attacks and those related to power consumption introduced earlier in a sense that out-of-order execution attacks focus on reading arbitrary data from memory, which does not have to be related to cryptology at all.

Furthermore, we want to mention the concept of fault attacks as an effective SCA method, for which [7] serves as an example. This type of attack distinguishes itself from most other SCA methods in a sense that it is an active attack. That is, in this case the adversary actively interacts with the system under attack in order to learn something secret, for example by making use of lasers. In contrast, the power consumption techniques mentioned before require an adversary to only passively measure the power consumption. As a consequence, such a method is labeled as a passive attack. Fault attacks however, are not limited to the usage of lasers. Changing the voltage, current or clock speed of the processor are other examples of attack vectors.

Finally, we want to name one more famous active type of side channel attack. It involves attacking Dynamic Random Access Memory (DRAM) with Rowhammer [17] to flip bits or even read secret data in the case of RAMBleed [24]. Also this type of attack does not have to be related to cryptology in any way.

1.1.1 SCAs And ECC

By now we have introduced some very high level concepts of SCAs and provided some pointers to further on in this thesis. Now we are going to mention some practical SCAs which are linked to ECC specifically.

An interesting example of a SCA related to ECC would be the attack described by Samwel et al. on breaking Ed25519 in WolfSSL [33]. However, this actually attacks the hash function used in Ed25519 [6] and not the arithmetic of Elliptic Curve (EC) operations itself. Nevertheless, there are some papers available which target the scalar exponentiation in RSA which can be expanded to the ECC scalar multiplication, such as [38].

Another example which directly does target EC arithmetic would be the paper of Genkin et al. [11]. In there a methodology is described of extracting an ECDSA private key because a scalar multiplication depending on a secret value was not implemented in a time constant way. Interesting to note is that they actually were able to only extract some scalar bits with template techniques. However, by applying a so called lattice attack [13] afterwards, they were able to reconstruct the whole secret scalar. The usage of lattice attacks is rather common when attacking ECC, see for example [4].

In contrast, there is also a rather new attack which primarily targets the scalar multiplication in ECC, that does not make use of lattice attacks, called Online Template Attack (OTA) [3]. The technique is also thoroughly discussed in the PhD thesis of Papachristodoulou [31]. OTA is based on the principle of template attacks but has as a considerable benefit that only a limited amount of power traces need to be taken, ideally one per bit to recover. On top of that, the templates can be generated on the fly, hence the ‘online’ reference, and do not have to be computed beforehand, also referred to as ‘offline’. OTA primarily targets Elliptic Curve Diffie-Hellman (ECDH) because it requires that an attacker has control over the EC point used in the multiplication. Although it can also be applied to the Elliptic Curve Digital Signature Algorithm (ECDSA) under the assumption there is a way to control the EC point used in the scalar multiplication. The OTA principle will be thoroughly discussed in Subsection 3.2.2 and Chapter 5.

1.2 Scope and Outline

As mentioned before, the goal of this master thesis is to investigate the security of an ECDSA implementation against the OTA in order to determine the feasibility of extracting secret signing keys. We will target an 8 bit microcontroller, which is commonly used in resource constrained devices such as a smartcard. Before any attack is actually discussed, we will investigate in Chapter 2 some basics of ECC and ECDSA. Thereafter, we will discuss in detail some scalar multiplication methods because this a critical operation which sometimes leaves a window for a SCA to extract a secret key.

In Chapter 3 we will examine different kind of SCA techniques in relation to the power side channel. We provide some background techniques after which we end up describing TEMPEST and OTA.

The reason we discuss TEMPEST is because the original goal of this thesis was not to apply the OTA but to use the EM side channel to extract an ECDSA secret key from a smartphone. However, after several months into the thesis our attack strategy proved inadequate and therefore the topic was dropped and the switch was made to OTA. Nevertheless, our methods and findings of this approach are written down in Chapter 4.

Hereafter, in Chapter 5, we will finally introduce the practicalities and further details of the OTA. It is good to realize that what is described in this chapter is different from the standard OTA technique and as a consequence this thesis is not a simple exercise of something already achieved before. The novelty of this thesis involves extracting the secret key via the ECDSA verification algorithm, a public function which does not use secret values, instead of the signing counterpart. Additionally, not a separate device is used for building the templates. Further details we will leave for Chapter 5.

In the end of the thesis we will discuss some countermeasures to prevent the attack discussed from happening in the future. Finally, in the last chapter we give an answer to the following research question:

‘How feasible is an OTA on a 8 bit target in order to extract the signing key via the ECDSA verification algorithm?’

Chapter 2

Cryptographic Background

This part is meant to serve as a cryptographic background to understand what is discussed further on this thesis. First we will give an introduction into relevant parameters when implementing ECC, without giving much attention to the mathematical details. Thereafter we introduce the signature scheme under attack after which we zoom in more deeply on how to perform a scalar multiplication when using ECs. We finish this chapter by giving an optimization trick to combine two single scalar multiplications to gain an optimization in speed due to some small pre-computation step.

2.1 Notes On Elliptic Curve Cryptography

Elliptic curves where originally introduced independently in [18] and [27] as a means to perform asymmetric cryptography. Its security is based on the assumption that the Elliptic Curve Discrete Logarithm Problem (ECDLP) is hard to solve. That is, having Equation (2.1), where the points P and Q are known to an attacker, the only way for him to find a is via brute force approaches.

$$Q = aP \ \ (mod\, p) \tag{2.1}$$

Remember that there is an addition law for elliptic curves, see Equation (2.2) for the formulas in the affine coordinate system. The value of λ in the equation differs for an addition $R = P + Q$ with $\lambda = \frac{y_q - y_p}{x_q - x_p}$ and a doubling $R = 2P$ with $\lambda = \frac{3x_p^2 + a}{2y_p}$.

$$x_r = \lambda^2 - x_p - x_q \qquad y_r = \lambda(x_p - x_r) - y_p \tag{2.2}$$

A naive implementation of the multiplication given in Equation (2.1) can be done as a repeating loop of additions. If a would be a big number, say 256 bit long, it would roughly take 2^{256} addition operations to execute the multiplication which is more expensive than brute forcing the ECDLP problem. Luckily there are some methods known which can reduce this multiplication to a far more feasible computing time. Some well known approaches are discussed in Section 2.3.1 and 2.3.2.

We will not discuss the concept of elliptic curves any further. For those interested we refer to the references at the beginning of this paragraph. The only thing we will specify for now are the six domain parameters which are needed van implementing ECC for some application:

- p: An EC is defined over a finite field p, written as $\mathbb{F}_p$. Typically p is a large number, for example 256 bits long.
- a & b: The standard notation for an EC in (short) Weierstrass form is $y^2 = x^3 + ax + b$. Changing the values of a and b changes the form of the curve.
- B: The base point which serves as a generator within its subgroup.
- l: The order of B is defined as the smallest value l where $lB = \mathbb{O}$, the point at infinity. Typically l needs to be prime and as big as possible. After all, the bigger the order, the bigger the amount of points B can generate on the curve which makes brute force attacks less feasible.
- h: If B cannot generate all the points on the curve chosen, there are several subgroups present. The cofactor h is defined as $\#E(\mathbb{F}_p)$ (the order of the curve) divided by l. Normally the cofactor should be kept as low as possible to prevent certain types of cryptographic attacks from happening. Often it is kept equal to 1. One of the most well known exception would be Curve25519, which has an cofactor of 8 [5].

Typically, users do not come up with values for these domain parameters by themselves. Instead they use a tuple with parameters provided by for example the National Institute of Standards and Technology (NIST).

2.1.1 Alternative Coordinate Systems

The downside of doing addition and/or doubling operation in affine coordinates is that it is requires modular inversions in the underlying $\mathbb{F}_p$, which is a costly operation in terms of speed and therefore preferably avoided. A common method to achieve this goal is to represent an affine point $P = (x, y)$ in a different coordinate system, two common ones are:

- Standard projective coordinates: an affine point (x, y) is represented as (X, Y, Z) where $x = X/Z$,$y = Y/Z$ and $Z \neq 0$.
- Jacobian coordinates: an affine point (x, y) is represented as (X, Y, Z) where $x = X/Z^2$, $y = Y/Z^3$ and $Z \neq 0$.

2.2 ECDSA

After introducing ECs we can now focus on the signing algorithm under attack in this thesis. Almost twenty years ago Johnson et al. came up with the EC variant of the Digital Signature Algorithm (DSA) [16]. It is one of the most used signature schemes next to RSA [32] and Ed25519 [6]. It is for example used in the Bitcoin protocol.

When implementing ECDSA one of the first steps to be taken is specifying the domain parameters as discussed in Section 2.1. Thereafter the setup continues by the signer who needs to select a random integer in the domain $[1, l - 1]$ which serves as the private key d. From d he needs to compute its corresponding public key $D = dB$. Now the setup is finished and the actual signing operation can take place. See Algorithm 2 for the signing procedure, Algorithm 3 for the verification and Table 2.2 for the corresponding symbol clarification. The algorithms are self explaining. However, we want to emphasize on three points.

First of all, with both algorithms the message must be hashed, for example with SHA-256. However, not the whole hash is being used: only the first significant bits are taken and converted into an integer. To be precise, the amount of bits to convert are equal to the bit length of l. Secondly, we want to emphasize that every time a new signature is created, either for a new message or another iteration of the *while* loop a new random k must be used. If an attacker in some way learns the value of k, via side-channel analysis or because of reusage of k, then the secret signing key d can be retrieved via Equation 2.3. After all, all variables except k are publicly

Symbol	Meaning
d	Private key (secret)
k	Random fresh nonce (secret)
l	Order of base point
(r, s)	Signature pair
D	Public key ($D = dB$)
B	Base point
$H(...)$	Hash of input
M	Message

Table 2.1: Symbol clarification for ECDSA related algorithms.

available. Finally, we stress the importance of implementing the parts of Algorithm 2 that depend on secret data happens in a time constant fashion. Otherwise a window is created for a SCA for a timing attack.

$$d = \frac{s \cdot k - z}{r} \tag{2.3}$$

Algorithm 2: ECDSA signature generation.

```
Input  : $d, l, B, M$
Output: $(r, s)$
1  $z \leftarrow H(M)$      //convert first log2(l) bits of hash to integer
2  while $true$ do
3      $(x, y) \leftarrow kB$          // new random k generated every time
4      $r \leftarrow x \ mod \ l$
5      if $r == 0$ then
6          continue              // new k needed
7      end
8      $s \leftarrow k^{-1}(z + rd)$
9      if $s == 0$ then
10         continue              // new k needed
11     end
12     break
13 end
14 return $(r, s)$
```

Algorithm 3: ECDSA signature verification.

```
   Input  : $l, r, s, B, D, M$
   Output: $signature\_accepted$
1  $signature\_accepted \leftarrow 0$
2  $z \leftarrow H(M)$      //convert first log2(l) bits of hash to integer
3  $w \leftarrow s^{-1}\ mod\ l$
4  $u_1 \leftarrow zw\ mod\ l$
5  $u_2 \leftarrow rw\ mod\ l$
6  $(x, y) \leftarrow u_1B + u_2D$
7  if $x == r\ mod\ l$ then
8      $signature\_accepted \leftarrow 1$
9  end
10 return $signature\_accepted$
```

2.3 Scalar Multiplication Algorithms

2.3.1 Double And Add

As noted in Section 2.1, some tricks are needed so that elliptic curve scalar multiplication is implemented in a time sensible way. One of the most well known methods to achieve this goal is the 'left-to-right double-and-add' algorithm which is given in Algorithm 4 . Indeed, there is also an 'right-to-left double-and-add' algorithm, which is very similar and has the same benefits and drawbacks as Algorithm 4 which are discussed below.

Algorithm 4: Double and add algorithm.

```
  Input  : $(k_{n-1}, k_{n-2}, ..., k_0), P$
  Output: $Q = kP$
1 $Q \leftarrow P$
2 for $i \leftarrow n - 2$ down to $0$ do
3     $Q \leftarrow 2Q$
4     if $k_i == 1$ then
5         $Q \leftarrow Q + P$
6     end
7 end
8 return $Q$
```

Basically, starting from the second Most Significant Bit (MSB) of the scalar k, it doubles the intermediate value Q and adds the EC point P to it when the bit of the scalar equals 1. This process repeats itself for all bits of the scalar.

To give an impression of the gain in speed, a small example. When computing $200 \cdot P$ this algorithm would require 7 doubling and 2 addition operations in total, while computing it via repeated addition only would take 200 addition operations. So one can now imagine when not an 8 bit value is chosen, but a 255 bit one which is rather standard when using ECDSA.

However, this algorithm is highly insecure to use in cryptographic applications when the scalar has to be kept private. Namely, one of the first rules of safely implementing some cryptographic primitive is to not branch on sensitive material. An execution of a round in Algorithm 4 will take a longer period of time when a bit is equal to 1 than when it would be equal to 0 because of the execution of the addition operation. As a consequence, a side-channel is created because the code does not run in constant time, its execution time depends on the bit values of the scalar.

2.3.1.1 NAF Notation

The double and add algorithm is still a viable option when the multiplication is not depending on secret information. An example of such a situation would be line 6 in Algorithm 3 where two separate scalar multiplications occur after which their results are combined by addition. However, to gain an optimization in speed when doing the multiplication the scalar k is typically converted to a representation in Non-Adjacent Form (NAF). In this form an integer can be represented by a combination of $(-1, 0, 1)$ digits where every two non-zero digits are separated by atleast one zero digit. For example, the number 7 in binary form would be (0,1,1,1), that is $(0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 1 \cdot 1)$ while in NAF form it would be (1,0,0,-1), that is $(1 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 - 1 \cdot 1)$. The benefit of this representation is that on average the amount of non-zero digits can be reduced to 1/3, where it would be on average 1/2 in binary representation. As a consequence a reduced amount of addition operations have to be executed. After all, line 4 of Algorithm 4 shows that an addition only happens when a scalar bit is not equal to 0.

Algorithm 5 gives the overview of how a number can be converted to its NAF representation after which Algorithm 6 show the adjusted double and add algorithm. Both are taken from the book 'Guide to Elliptic Curve Cryptography' [12].

Algorithm 5: Compute the NAF of an integer.

```
   Input  : $k$
   Output: $NAF(k)$
1  $i \leftarrow 0$
2  while $k \geq 1$ do
3      if $k$ $mod$ $2 == 0$ then
4          $k_i \leftarrow 2 - (k \text{ mod } 4)$
5          $k \leftarrow k - k_i$
6      end
7      else
8          $k_i \leftarrow 0$
9      end
10     $k \leftarrow k/2$
11     $i \leftarrow i + 1$
12 end
13 return $(k_{n-1}, k_{n-2}, ..., k_0)$
```

Algorithm 6: Scalar multiplication with NAF.

```
   Input  : $k, P$
   Output: $Q = kP$
1  $NAF(K) = \sum_{i=0}^{n-1} k_i 2^i$        // n = amount of bits k
2  $Q \leftarrow \infty$
3  for $i \leftarrow l - 1$ down to $0$ do
4      $Q \leftarrow 2Q$
5      if $k_i == 1$ then
6          $Q \leftarrow Q + P$
7      else if $k_i == -1$ then
8          $Q \leftarrow Q - P$
9      end
10 end
11 return $Q$
```

2.3.2 Montgomery Ladder

An alternative to the double and add algorithm which is both relatively fast and runs in constant time is the Montgomery ladder. Originally in 1987 it was meant as an algorithm to speed up factorization using ECs [29]. After the rise of SCAs in the late nineties it became an effective method to guarantee a time constant execution of code. On top of that, it provides protection against some other types of SCAs such as Simple Power Analysis (SPA), see Algorithm 7 for the overview.

Algorithm 7: Montgomery ladder.

```
  Input  : $(k_{n-1}, k_{n-2}, ..., k_0), P$
  Output: $Q = kP$
1 $R_0 \leftarrow P$
2 $R_1 \leftarrow 2P$
3 for $i \leftarrow n-2$ down to $0$ do
4     $b \leftarrow 1 - k_i$
5     $R_b \leftarrow R_0 + R_1$
6     $R_{k_i} \leftarrow 2R_{k_i}$
7 end
8 return $R_0$
```

The code in Algorithm 7 has a very regular structure: every operation gets executed in every iteration of the loop because there are no branches present. The code nevertheless differentiates between scalar bits by placing the result of the addition in R_0 and the doubles in R_1 or the other way around, depending on the value of the bit. Because of the lack of branching it can therefore be stated that the code runs in constant time. This, however, does not mean it is resistant to all side-channel attacks, it is simply not vulnerable to the timing side-channel.

Nevertheless, Algorithm 7 only gives a high-level overview of how a Montgomery ladder is typically implemented into computer code. A lot of research has been done the last couple of decades in optimizing the Montgomery ladder for speed and/or memory usage and improving its resistance against side-channel analysis. One such a technique is described in the appendix of [14] in Algorithm 4, which is used for this thesis. Later on in Chapter 5 we will discuss a few aspects of this implementation in more detail because we will show that it is vulnerable to an OTA.

2.4 Shamir's Trick

As noted before, the double and add algorithm can be a valid method to perform scalar multiplication. It can for example be called two times to perform the two scalar multiplications of line 6 in Algorithm 3 while verifying an ECDSA signature. The disadvantage however is that in such an implementation the two scalar multiplications happen independently. Shamir suggested a method, based on work by Straus [34], to combine both multiplications to gain an optimization in speed which received the nickname 'Shamir's trick', see Algorithm 8. The big speedup is gain by the precomputation of $P + Q$ which is added when both scalar bits equal 1. Note that no addition takes place when both bits are 0, just like with the standard double and add algorithm.

Algorithm 8: Shamir's trick.

```
   Input  : $(a_{n-1}, a_{n-2}, ..., a_0), (b_{n-1}, b_{n-2}, ..., b_0), P, Q$
   Output: $R = aP + bQ$
1  $R \leftarrow 0$
2  $S \leftarrow P + Q$
3  for $i \leftarrow n - 1$ down to $0$ do
4      $R \leftarrow 2R$
5      if $a_i == 1$ AND $b_i == 0$ then
6          $R \leftarrow R + P$
7      else if $a_i == 0$ AND $b_i == 1$ then
8          $R \leftarrow R + Q$
9      else if $a_i == 1$ AND $b_i == 1$ then
10         $R \leftarrow R + S$
11     end
12 end
13 return $R$
```

Chapter 3

Power Side-Channel Analysis

In the first chapter of this thesis we gave an intuition of what a SCA is. The goal of this chapter is to introduce some principles of SCA when the power consumption of a device is used as the side-channel for extracting for example key bits. The first paragraph provides a general background which serves as a basis for the second paragraph, whose principles are actually applied in this thesis.

3.1 Background Techniques

3.1.1 Simple Power Analysis

Whenever a processor does some computation, it consumes power. The amount of consumption is depending on the operation performed and often on the input used for that operation. The idea of Simple Power Analysis (SPA) [20] is to measure the power consumption of a device while it is performing some operations of interest, in our case scalar multiplication. When analyzing the power consumption we expect to see somehow 'leakage' of the targeted variable, say a scalar or a key. This may sound rather abstract, but can quite easily be visualized in a figure.

Namely, a standard example of SPA is attacking the square and multiply algorithm of RSA used for exponentiation. It is an algorithm which is on an abstract level very similar to Algorithm 4, but one should replace double by squaring operations and additions by multiplications. A power consumption

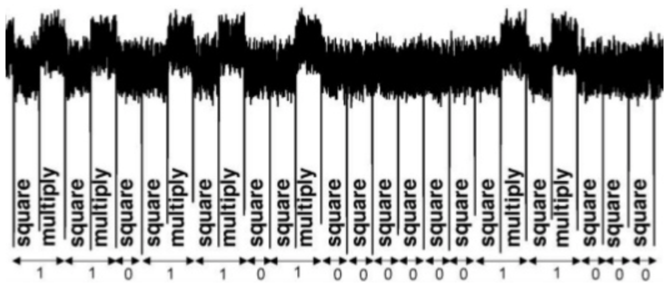


Figure 3.1: Schematic overview of SPA on RSA's square and multiply algorithm [30]. The X-axis represents the time and the Y-axis the related power consumption.

of the square and multiply algorithm could look like 3.1 when no countermeasures are taken.

In Figure 3.1 it is quite straightforward to differentiate between a squaring and a multiplication because the latter one leads to a higher power consumption of the processor. Since a multiplication is only being done when the exponentiation bit equals 1, we know the value of the bit in a certain round of the algorithm. Similarly, a lack of a multiplication indications that the bit equals 0. Finally, because every operation takes around the same amount of time, it is possible to state where a round of the square and multiply algorithm starts and finishes. Since we now know how to differentiate between a bit and where a single round of the algorithm takes place, the whole scalar can be reconstructed bit by bit.

Depending on the quality of the trace taken, additional measurements are needed to fully reconstruct a scalar. Traces of the same operation can simply be combined by averaging them all.

The success of a SPA attack mainly depends on whether the execution path of a program depends on the data being processed, as is the case in Algorithm 4. If such a dependency is not present, SPA will probably not work but that does not mean the program is safe for a SCA, there are more advanced techniques possible for an adversary.

3.1.2 Differential Power Analysis

One such more advanced SCA technique is called Differential Power Analysis (DPA) [20] which makes use of statistics to exploit even small biases in the power consumption to recover a secret value. In comparison to SPA, DPA typically requires more traces to be taken with the same secret value, while the input message to the encryption/signing scheme should differ but must be known. Additionally, DPA does not analyze the power consumption among the time axis, like SPA does, but instead analyzes how the power consumption depends on processed data on fixed moments in time. The structure of DPA is quite generic and is therefore explained below based on [26].

As a first step, the attacker must choose a part of the cryptographic algorithm under attack which uses some data d, often either the plaintext or the ciphertext, and is depending on the key k to retrieve.

Thereafter, power consumption measurements are taken of the device while executing the relevant part of the algorithm. It is important that every single trace has the same amount of samples and that they are aligned. As a result of this step we are left with a matrix $M = D \times T$ where D stands for the total amount of input values d and T for the amount of samples in a single trace. Once again, the values of d should differ but the used k needs to remain the same.

The third step consists of computing hypothetical intermediate values for the function part selected in step one for every possible value of k for every input chosen at the previous step. As a result this step gives a a matrix $v = D \times K$. As of now, one column matches they key used while taking the power consumption measurements in step two. The goal from here on is to find which column that is.

Now that we know for every input d out of D the hypothetical intermediate value for every key, a model can be used to translate this hypothetical intermediate value into a hypothetical power consumption value. The model used to calculate this value highly depends on the platform under attack. For a software implementation typically a model is used which counts the number of non-zero bits, which is called the Hamming weight. Its counterpart is the Hamming distance model, which is used for hardware implementations of some cryptographic primitive. Instead, this model counts the total number of bits flipped between the input and the hypothetical intermediate value. The result of of this step is once again a matrix with the dimension $D \times K$ which we call H.

As a final step, we compare every column h_i of H to every column m_j of M. So, the hypothetical power consumption of every value of k is compared with the recorded traces at every position. Once again, the attacker has several options for how to make this comparison. One very common method is to calculate the Pearson correlation coefficient which is defined in Equation 3.1

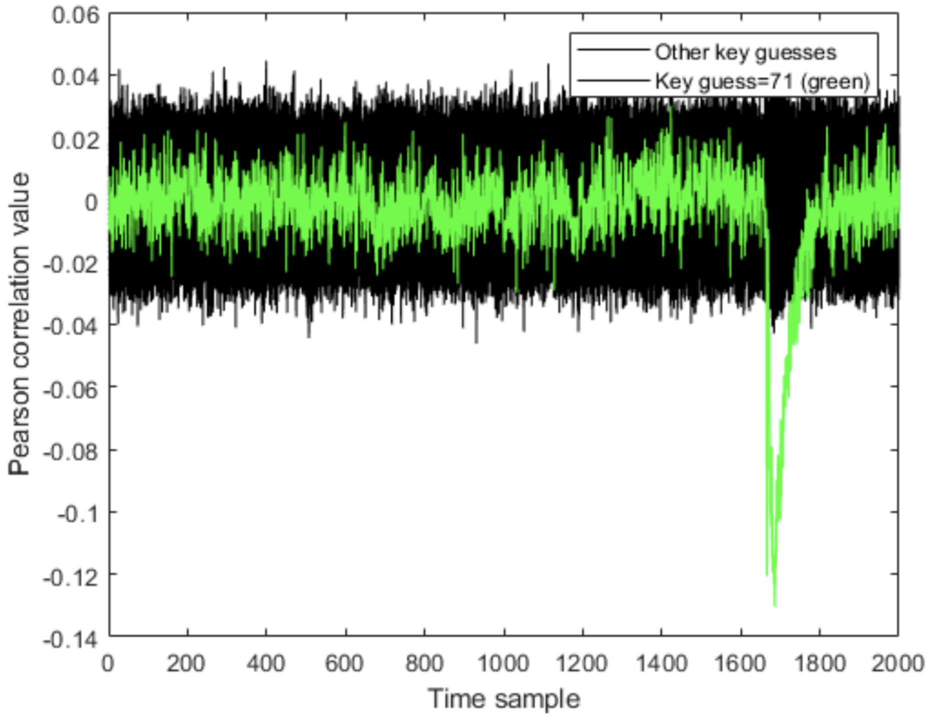


Figure 3.2: Example result of DPA attack.

together with its application to fill the matrix R for every combination of h_i and m_i. In this case, the correlation indicates the Hamming weight leakage of operations which depend on the key while a cryptographic algorithm is executing.

$$r_{i,j} = \frac{cov(h_i, m_i)}{\sigma_{h_i}\sigma_{m_i}} = \frac{\sum_{d=1}^{D}(h_{d,i} - \bar{h_i}) \cdot (m_{d,j} - \bar{m_j})}{\sqrt{\sum_{d=1}^{D}\left(h_{d,i} - \bar{h_i}\right)^2 \cdot \left(m_{d,j} - \bar{m_j}\right)^2}} \tag{3.1}$$

Almost all values in R will have a value relatively close to zero, which indicates that there is no linear, when the Pearson coefficient is used, relationship between h_i and m_i. The column h_i which contains the highest value for the correlation coefficient, is most likely representing the key used when taking the power consumption measurements in step two. Usually the correct key candidate can also be visualized by plotting $r_{i,j}$ against the time axis, as demonstrated in Figure 3.2.

Figure 3.2 comes from a homework assignment where a DPA attack had to be implemented in order to recover a key byte used in an implementation of AES. The black background in the figure represents the correlation of all wrong 255 key possibilities and the green shows the correct candidate. So, visually it can also be very easy to decide on the the correct key value after applying DPA because there is only one candidate whose correlation coefficient stands out.

3.1.3 Template Attack

Although the two techniques above can be very effective attack strategies, they cannot be applied to the signing algorithm of ECDSA in order to secret scalar k, as discussed in Section 2.2. Namely, as further discussed later on in this thesis, the ECDSA implementation under attack is (presumed to be) constant time and does not branch based on a bit of k. So, SPA is not an option. Furthermore, since the value for k changes for every single signing operation, DPA is also not possible. After all, that technique requires at least hundreds, possibly thousands, of traces with the same scalar in order to extract its value.

However, Chari et al. introduced in 2004 the principle of a template attack [8]. The attack utilizes the idea that it might be hard to attack a device directly, which is required with SPA and DPA, so an indirect approach is taken.

Concretely this means that a person should start with acquiring an identical device as it wants to attack. Thereafter he chooses some model which describes the power consumption to build up a profile, a template. In the most simple model, called the 'reduced model', this entails doing a certain operation N times while a secret bit equals zero and another N times while it equals one. For both cases the mean is taken of all the power consumption traces, which results in two averaged power traces. After building the two templates, a power trace can be taken of the actual device under attack. Now it becomes the question to which built template this power trace taken is more similar. In the reduced model this question is answered by simply subtracting the built templates separately from the power consumption trace taken. The result closest to zero is suspected to represent the correct value for the key bit. From here on this process can be repeated bit by bit to recover the whole secret scalar.

The success degree of this attack highly depends on choosing a model which captures the leakage adequately. Additionally, enough measurements should be taken to build accurate templates. Furthermore, it is essential that the device used for building templates is identical to the one actual under attack in terms of power consumption.

Note that this attack does not necessarily needs to recover bit by bit. It is also plausible to work with for example bytes at a time. However, this has as a downside that more different templates need to be built simultaneously which requires more storage space.

3.2 Applied Techniques

3.2.1 TEMPEST

In some scenarios it might be unfeasible to measure the power consumption directly. As an alternative, an electromagnetic probe can be used to capture the EM transmitted by the device. Exploiting this side-channel is also referred to as TEMPEST. By computing the Fast Fourier Transform (FFT) of the power trace the signal is translated to the frequency domain which makes it possible to analyze energy densities. After establishing around which frequencies a relative big amount of energy is being transmitted, which seems to be depending on the operation of interest, signal processing techniques can be applied to remove noise from the signal. Some standard pointers to look for interesting emissions are the clock frequency of the device and its harmonics. If one is successful in separating noise from the information of interest, numerous (statistical) approaches, such as applying correlation, can be taken to extract the secret value.

However, the main bottleneck of a TEMPEST attack is to actually separate the noise from the signal. The problem is that every device leaks differently and as a consequence no predefined methods exist. There are only some best practices available in the research world that worked in a certain scenario on some specific device. Furthermore, industry is aware of the danger of TEMPEST and is continuously trying to reduce the radiation transmitted by electrical components. Nevertheless, successful attack do exist, of which some are mentioned in Chapter 4.

As a side note, it should be mentioned that the main use case of TEMPEST attacks originally was not to extract secret key material but instead about reconstructing the content on someone's computer monitor. Allegedly it has been a rather common technique used in the cold war. Some papers who go into detail about this technique are [36] and [23].

3.2.2 Online Template Attack

As discussed in Section 3.1.3, template attacks require to build profiles of the targeted device before conducting an actual attack. Therefore it is sometimes also referred to as a profiled SCA, in contrast to a non-profiled attack like SPA and DPA. In reality however, it can be a very cumbersome activity to generate these templates because often hundred or even thousands of measurements are needed per bit to recover.

This struggle of template generation is almost avoided completely by applying the Online Template Attack (OTA) principle [3]. As the name suggests, it is based on the idea of template attacks but distinguishes itself by interleaving the process of template generation, template matching and the amount of power traces needed. Namely, an adversary is only required to take one target trace and additionally at most two additional traces per bit to recover.

All EC scalar multiplications methods perform on an abstract level a combination of double and add operations where the scalar used determines exactly how that combination looks like. In an oversimplified case, a power trace of the doubling on line 3 of Algorithm 4 will differ based on whether the input of Q equals $2P$ or $3P$. That is, in the algorithm k_{n-1} is equal to 1 and after the first iteration of the loop in Algorithm 4 Q will be equal to $2P$ if $k_{n-2} = 0$ and $3P$ if $k_{n-2} = 1$. So, by creating two templates for this doubling operation, one hypothesis for $2P$ and one for $3P$, and applying correlation techniques between the target and the templates the correct key bit can be estimated. In [3] it is written that the correlation differences between hypothesizes are rather constant and equal in value, therefore it suffices to just build one template. By applying a certain threshold the hypothesis can be accepted or rejected.

Once a key bit has been estimated the next MSB can be attacked by creating new templates for the doubling operation on line 3 of Algorithm 4. In our example, assuming that $k_{n-2} = 0$, the templates generated are $4P$ and $5P$ for k_{n-3}. From here on the process repeats itself. Once again, when making these templates, it is not relevant to get a power trace of the whole scalar multiplication. In fact, only the doubling operation in the second round of the for loop matters. Furthermore, if we want to find the value of k_{n-4}, assuming we already know the three MSBs, we need to take the target trace part of the doubling operation in the iteration of processing k_{n-5}, so the fourth round of the loop.

One of the most significant advantages of OTA is that it only requires one target trace, to the contrary of DPA. As a consequence, algorithms whose security is partially depending on the usage of nonces, such as ECDSA and ECDH, come within the scope of a SCA via power consumption. Furthermore, in order for the attack to succeed, only very limited access to the target device is needed. The only requirement is that an attacker is able to collect a power trace. The templates can be generated on an identical but different device, just as is the case with a standard template attack.

Since we already discussed that Algorithm 4 in itself is vulnerable to a timing SCA, it is unnecessary to apply OTA to it. Nevertheless, it is the easiest example to get a feeling for the attack principle. In Chapter 5 we will move on an actual attack which is based on OTA where we will provide an in depth explanation of all the details involved to conduct such an attack.

Chapter 4

Key Extraction Via TEMPEST

Like mentioned in the introduction, the original goal of the thesis was to extract a signing key used in the ECDSA scheme of the latest version of OpenSSL: 1.1.1b via TEMPEST. Sadly, our setup did not prove to be adequate, therefore the topic was dropped. The main issue was that the leakage caused by the signing operation was probably too low and therefore could not be distinguished from noise. As a consequence, no further analysis was possible. Nevertheless we will describe below the the setup, and the 'results' retrieved from it. In the end we reason why the attack did not prove to be successful.

4.1 Setup

When conducting a SCA on an ECC implementation the scalar multiplication is often the point of interest to learn something about the secret key. In case of ECDSA this would refer to the multiplication between the random k and base point B in line 3 of Algorithm 2. When it is somehow possible to retrieve the value for k, the signing key can be retrieved via Equation 2.3.

In the recent past some papers appeared which exactly targeted the multiplication mentioned via the EM side-channel, such as [11] from 2016. This paper targeted the EM radiation emitted in the lower area of the spectrum, say < 125 kHz, because they showed that the implementation of ECDSA was not fully time constant.

Knowing that the timing vulnerability in OpenSSL would be fixed now, the idea was to look in the higher regions of the EM spectrum near the clock frequency of the mobile phone used. This second attack vector is commonly used to more closely learn something about the data being processed directly, not via a timing side-channel. Such an attack is more expensive and troublesome to perform but should be feasible nevertheless because it has been done by people from Georgia Tech [2] in 2018. However, they did not target ECC but instead OpenSSL's time constant blinded RSA implementation.

Then it is now time to provide details about the setup of the attack. The phone targeted was a Sony Ericsson X10a smartphone from 2010 running Android version 2.3.3 in airplane mode. Using Android NDK version 10e we were able to build OpenSSL-1.1.1b with its default configuration except with the 'no-threads' flag added simply because Android API level 9 does not support it. However, we did make some small tweaks in the original OpenSSL code to simplify the SCA at first:

- The nonce k from Algorithm 2 was made constant.
- Coordinate blinding was turned off.
- Print statements for debugging purposes were added.

Thereafter a dummy program was written where the ECDSA signing was performed on a static SHA-256 hash. This happened in a loop with a sleep call after every iteration to identify more easily the start and ending of an operation when analyzing the EM spectrum. The curve used was prime192v3 which is one of the many curves implemented in OpenSSL. However, the curve used should have no effect on the side-channel analysis because the scalar multiplication method used remains unchanged: Izu's and Takagi's implementation of the Montgomery ladder [15].

The leakage from the phone was captured by the RF-R 400-1 probe which can measure signals in the range 30MHz-3GHz within 10 centimeter of the targeted component. It was connected to an PA 303 amplifier which increases signal strength by 30 dB in the range DC-3GHz. Both components are from the company Langer. The output of the amplifier was fed to the USRP X310 transceiver via an UBX-160 daughter board from Ettus. As a final step was the output of the transceiver fed into a computer via an gigabit Ethernet cable. On the computer the program Baudline [1] was running, which allows for quick visual inspection of the EM spectrum while collecting a trace. Further post processing of a trace happened via Matlab [2].

To get a feeling for the total setup, see Figure 4.1 and Table 4.1 for the

[1]http://www.baudline.com
[2]https://nl.mathworks.com

Number	Meaning
1	Sony Ericsson X10a
2	Langer RF-R 400-1 probe
3	Langer PA 303 amplifier
4	Ettus USRP X310 transceiver
5	Baudline
6	USB cable

Table 4.1: Number clarification for TEMPEST setup Figure 4.1.

number clarification. The only component not mentioned in the figure is the USB cable. It was used to connect to phone with a secondary computer to trigger the dummy program via SSH to start performing signing operations and inspect the print messages generated by the phone.

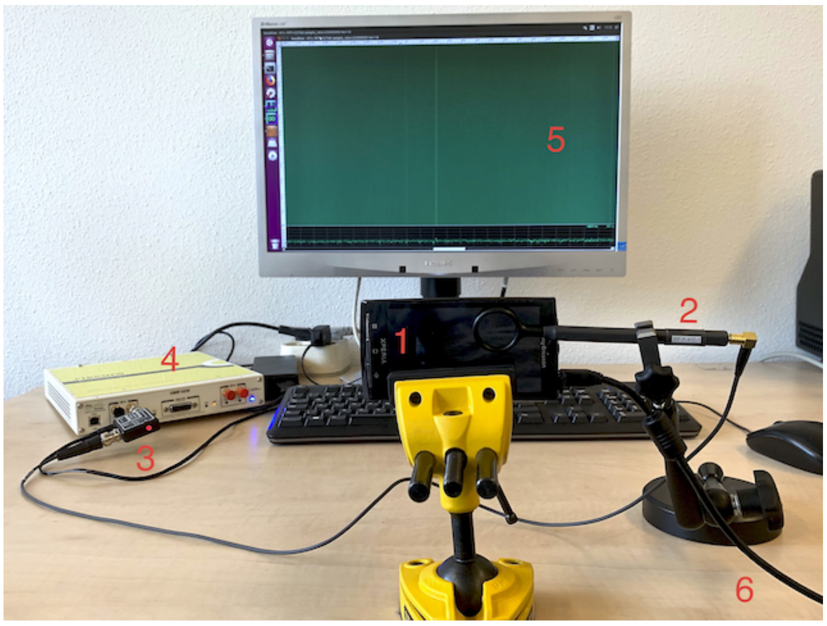


Figure 4.1: Overview of the TEMPEST attack setup.

4.2 Finding The Leakage

The first step of the attack is to identify some leakage of the phone in the EM spectrum related to the signing operation performed. We made the phone run at a constant clock frequency of 384MHz, so this was the first frequency to inspect. Some leakage was identified but we first looked whether the leakage was higher somewhere else. After inspecting a bandwith of in total 100MHz around the first 3 harmonics of the clock frequency it indeed appeared that the area around 384MHz was indeed the most interesting. So, we increased the sampling rate to a maximum with Baudline with 25Msamples/second, which allows for a bandwidth of 12.5 MHz, and exported a trace containing one signature generation for further analysis in Matlab. An example of the leakage identified in Baudline can be found in Appendix A Figure A.1.

With Baudline a user needs to specify a center frequency. For Figure A.1 this was put on 388.8MHz, which explains the offset of 200kHz in the figure. Indeed, the bright line in the figure represents the clock frequency of the phone. Note that the signing process starts at 525 ms and finishes at 125 ms in the past, so the total time taken is 400 ms. What looks promising in this figure that one can clearly differentiate between the program running and a sleep.

The scalar multiplication happens via the Montgomery ladder [29], which happens in a loop per bit of the scalar at a time. The values of the bits might be unknown but the instructions applied on it are not. Actually, they are identical for every loop. Therefore the next step of the SCA is to identify repeating patterns in the trace with Matlab by applying signal processing techniques.

There is no standard method to reduce noise in the trace and make the repeating pattern clearly visible. It is a process of trial and error. As already announced at the start of this chapter, this is were the research crashed. We were not able in any way to discover a repetitive pattern in the trace. It has no added value to now show dozens of plots of signal which look like noise, so instead below an enumeration is given from the techniques applied. Note that also combinations of the techniques below were applied.

Applied methods:

- Demodulation techniques:
  - Frequency
  - Amplitude
- Different filters:
  - Bandpass
  - Highpass
  - Lowpass
  - Notch

- Singular Spectrum Analysis (SSA), section 3.2 of [11]

Since we did not get the expected results, we decided to take more traces with different setups. The variables that we played with were:

- Change clock frequency phone.
- Look at harmonics of the clock frequency.
- Use different probes.
- Use different phones.
- Use the LeCroy waveRunner 610Zi oscilloscope instead of the Ettus transceiver for higher sampling rate up to 1 GH/s which allows a bandwidth up to 500 MHz.
- Create C program which only performs the scalar multiplication.

To our dissatisfaction, we still did not get any results. As a final step we modified the C program again in a way that all print statements were removed to reduce any further unnecessary EM emancipation from the phone. The resulting spectrogram with the same parameters used as Figure A.1 can be found in Figure A.2 in the same appendix.

Comparing both figures from Appendix A show some significant differences. Namely, with Figure A.2:

- The signal strength of the clock frequency (at a offset of 194 kHz) is less intense.
- The duration of one signing operation shortened with a factor of $\pm$ 40.
- No clear leakage visible in the bandwidth around the center frequency.

Therefore we draw the conclusion that the printf statements had a significant impact on the EM radiation leaked. So, new traces were taken again, signal processing techniques were applied, but to no further avail. Therefore, the research regarding extracting the signing key used for ECDSA via TEMPEST was ceased.

4.3 Discussion Of Results

We still do think that the concept of the attack is feasible. The main problem here is how to separate the noise from the leakage of the cryptographic operation. All common signal processing techniques were applied, without success. So, to that extend the problem might lay in capturing a trace itself.

This would mean to use a different probe or a different transceiver. The transceiver and the oscilloscope used were already worth thousands of euros, so changing something here to gain a better signal/noise ratio would be very costly. On the other hand, the paper [2] used a special own-made probe which they do not make publicly available. Thus, perhaps this special probe can capture the leakage we want. Furthermore, the paper referenced targeted different phones which maybe leaked more significantly information than the phones we used.

Chapter 5

Key Extraction Via OTA

We now reached the point in this thesis where we will discuss the conducted OTA attack in detail. To kick off, at first we give the scenario off the attack and introduce the board on which the attack has been applied. Thereafter we give a step by step overview of how to successfully achieve bit extraction of the secret scalar used in line 3 of Algorithm 2.

5.1 Scenario

Originally OTA was introduced with the idea to get one target trace from the device under attack and generate the templates on a different identical device controlled by the attacker. This means that both devices use the same ECDSA signing algorithm in order to built up the secret scalar bit by bit.

However such approaches, no matter the actual scalar multiplication algorithm used, have already been attacked successfully by Papachristodoulou, as described in her PhD thesis [31]. Although she did not target ECDSA specifically, breaking the scalar multiplication algorithm effectively breaks ECDSA.

For now, we will take the concept of OTA but tweak it a little. We now assume, as is common in practice, there is only one device available, which does both the signature generation as well as the verification of a message. The attacker gets to record one trace of a signature generation and afterwards only has access to the verification function of which he can also take power consumption measurements. Since the the verification algorithm, as given in Algorithm 3, does not use any secret material it, this doesn't seem as an unsafe thing to do. After all, how can a device leak information about something it is not actually using?

To provide some further details, the ECDSA implementation under attack uses the Montgomery ladder from Algorithm 7 for scalar multiplication when signing in standard projective coordinates. Its implementation comes from [14], which has some memory and speed optimization and is meant for low-resource and embedded devices. For the verification part, we mainly care about line 6 of Algorithm 3, as we will show later. We will use an implementation using Shamir's trick from Section 2.4 and additionally one applying the sum of two NAF multiplications from Subsection 2.3.1.1. However, as we will also show later, it doesn't really matter which scalar multiplication method is chosen for the verification. Both methods do their computations in Jacobian coordinates. Note that these details mentioned are common choices in real-world implementations.

As a final set of remarks we want to state that we use the EC 'sec256r1'. Note that this actually is quite irrelevant because the attack is not based on a certain curve but instead on general EC arithmetic. However, for completion sake, we want to mention it. Furthermore, at this point in the thesis we assume that no further countermeasures are taken against SCA except for the fact that the code used for signing is time constant. Finally, we emphasize that this attack does not modify the code base for ECDSA in any way, which further enhances the complexity of the attack.

5.2 Platform

The platform used for the OTA attack is the ChipWhisperer-Lite Classic (CWLC) board, which is displayed in Figure 5.1.

The CWLC is from origin meant as a training board to support people in learning the basics of side-channel analysis. As a consequence, it provides some easy to use interfaces to conduct power consumption analysis. Actually, the CWLC consists of two separate boards which can also physically be split by breaking the the break-away connections. The left part in Figure 5.1 serves as the main board which serves as a middle man between a computer connected via Micro-USB and the Xmega 128D4 8-bit processor from Atmel, which is the actual target running at 32 MHz. It is a processor that also can be found in real-world applications.

The standard way to communicate with the board is via a specially prepared

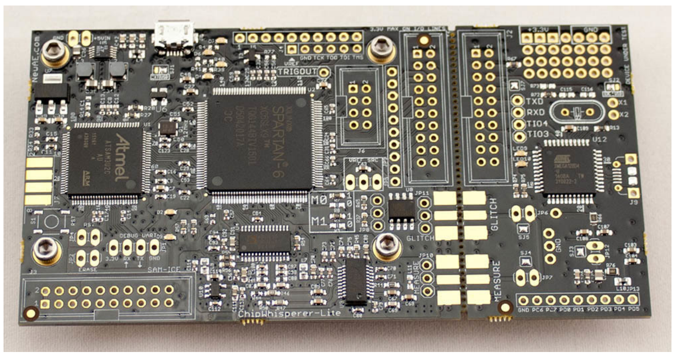


Figure 5.1: The ChipWhisperer-Lite (CW1173) board [35].

Virtual Machine[1] (VM), which is based on Ubuntu 18.04, running in VirtualBox[2]. Using this setup allows simple editing, compiling and running of code on the Xmega target via a Jupyter notebook[3].

5.3 Spotting The Attack Vector

Because we want to extract the secret scalar used in the signature generation part and extract it via the verification algorithm, we need to find some similarity between the algorithms. On top of that, the resembling part in the verification needs to be controllable by the adversary.

In Appendix B the detailed Montgomery ladder step can be found, based on [14]. This algorithm is on a high level the implementation of lines 5 and 6 of Algorithm 7. Using the algorithm in the appendix in a loop to process every scalar bit effectively results in the full Montgomery ladder, if also the conditional swap principle is applied. Observe that this ladder only works with the x- and z-coordinate of EC points, the y-coordinate is recovered separately once the complete scalar has been processed.

Before going into further details, we want to mention again that the basic principle of OTA works with generating the templates $2P$ and $3P$ and somehow compare those power traces with the target trace to retrieve the second MSB. This approach will not work here however, because the coordinate systems differ in signing and verification. In the former algorithm the base point is provided in affine coordinates, which is internally converted to standard

[1]https://github.com/newaetech/chipwhisperer
[2]https://www.virtualbox.org
[3]https://jupyter.org

projective coordinates. Equivalently, in the verification algorithm EC points are converted internally from affine to Jacobian. So if we want to apply the principle of OTA by needing only one target trace and two templates per bit, we need to find an alternative attack strategy.

When looking at the input parameters of ECDSA, given in Algorithm 2, we realize that an adversary has little to no influence on the intermediate computed values and the final signature. Namely, we consider the private key, a scalar, to be secret. After all, the whole goal is to somehow find its value. On top of that, the scenario does not allow to put in a random private key. Furthermore, the curve parameters are also predefined and cannot be changed. So, this leaves us only with the message to sign as controllable input to gather one power trace. Even though this might sound as a way in, it is actually completely useless because the message and its derived hash do not influence the scalar multiplication in any way. In other words: trying to achieve something by manipulating the ECDSA input parameters is a dead end.

As mentioned before, for the verification of a signature either Shamir's trick or two separate NAF multiplications are common techniques to use. For now we will focus the NAF method.

The verification function has more input parameters than the signing one that can be influenced by the attacker, as Algorithm 3 shows. An adversary can choose whatever it deems necessary for the signature (r, s), the message M to compute the hash of and the public key D. By manipulating any of these values we directly change the computations made on line 6 of Algorithm 3.

Hypothetically seen, if somehow we can 'remove' the scalar multiplication u_2D from that line we are left with a scalar multiplied with the base point B, just as was the case for the signature generation part in Algorithm 2. On top of that, in the verification algorithm the scalar u_1 to multiply B with is controlled by the adversary by manipulating either M or s. In the case that the scalar multiplication method here was also a Montgomery ladder this would provide us our attack window: create two templates with the ladder step, one with swap and one without, and do a correlation analysis of both with the ladder step of the target trace. Thereafter the same process is repeated for the other scalar bits with every ladder step. Effectively this attack structure becomes then quite similar to using the signature algorithm to generate templates. As we mentioned before however, the scalar multiplication method used in signing and verifying differ, which makes this strategy invalid.

In order to find a valid attack vector we need to dig deeper into the arithmetic performed when doing scalar multiplication in the verification. The NAF method itself is still a combination of double and add operations but the underlying field arithmetic consists of additions, subtractions, multiplications and squarings. See Algorithm 9 for the first few statements of calculating the double of an EC point in Jacobian coordinates.

Algorithm 9: First few statements of calculating a double of an EC point in Jacobian coordinates.

```
  Input  : $P$
  Output: $Q = 2 \cdot P$
1 $T_1 \leftarrow {P_x}^2$                 // x-coord of EC point
2 $T_2 \leftarrow {P_z}^2$
3 $T_2 \leftarrow {T_2}^2$
4 ....
```

Now we arrive at the point which provides a window for a SCA. Observe that line 1 shows a squaring of a x-coordinate. On top of that, line 17 of Algorithm 10 in Appendix B shows also a squaring, of a x-coordinate! In fact, it is a squaring whose input depends on the secret scalar we want to extract!

To clarify in Algorithm 10, the value of X_2 is swapped with X_1 if the scalar bit equals zero. By computing the two possible values of X_2 as input for this squaring, swap or no swap, and providing both cases as input for the doubling operation in Jacobian coordinates we have two templates we can compare with the target trace. So, we can extract the scalar one bit at a time by calculating the correlation between the power trace of a single square from the target trace with the power trace of a single square from both templates.

5.4 Preparing The Input

Even though we now have identified our attack window, there is still the question whether it can be exploited. There are two main problems to overcome: compute the two possible input values for squaring X_2 in the Montgomery ladder and secondly, feed them into the Jacobian doubling operation which contains the squaring.

The first problem can be solved by implementing our own Montgomery ladder step function in SageMath[4], a program which provides convenient functions to perform finite field arithmetic.

The second problem is more complex to solve. The computed X_2 values represents a x-coordinate of an EC point. Inspecting line 6 of Algorithm 3 shows that 2 EC points are used: the base point and the public key. The first one cannot be modified, but the latter one is controllable by the adversary. However, he cannot input every value he wants to execute this attack. Namely, as part of the ECDSA verification algorithm, typically before any verification occurs, the algorithm checks whether the public key is legitimate. That is, it checks whether the point lies on the curve, in our case 'secp256r1'. The problem that now occurs is that not every computed X_2 necessarily is a legitimate point on that curve. That is, if filling in X_2 for x in the short Weierstrass equation $y^2 = x^3 + ax + b$, see Section 2.1, does not resolve in an integer value for y, X_2 and its corresponding y-coordinate do not lie on the curve.

Papachristodoulou [31] describes that this problem can be circumvented by applying bit flipping on the Least Significant Bit (LSB) of X_2. After flipping the bit, the new y' coordinate is computed from x' and again is checked whether the alternative coordinate (x', y') lies on the curve. If so, we can use this alternative EC point as the public key. Otherwise we reverse the flipped bit and apply the same process on the second LSB. This process continues until a legitimate EC point has been found. Based on experiments of Papachristodoulou and ourselves, a maximum of five tries are needed to find a EC point that lies on the curve. For notation purposes, we refer to the valid x-coordinate found in this step as x in the rest of this chapter.

The reasoning behind flipping the LSB is that the power trace will still be as much as possible the same as the target trace, which starts its scalar multiplication algorithm from the second MSB. It might be that the correlation between a template and a target trace becomes somewhat lower when a bit is flipped, in comparison when it was not flipped, but it will still be higher than the the other wrong template.

Now we know what to input for the public key variable, we continue with manipulating the scalar u_2 from Algorithm 3. When looking at Algorithm 6, we want that the doubling of line 4 takes as input our computed x. This can be achieved by making sure that the MSB of u_2 equals one, the other 255 bits are irrelevant. This can be achieved by some simple brute forcing with the signature values r and s in the multiplication of line 5 of Algorithm 3

[4]http://www.sagemath.org

and should take a couple of tries at the most because you have 50% chance per attempt. Once achieved, the desired doubling occurs in the beginning of the second loop of the algorithm. Taking a power trace of the squaring inside this double operation gives us the wanted template. So, the message to hash is completely irrelevant because we do not make use of the scalar multiplication u_1B.

This brings us to the only difference in setup when Shamir's trick is used instead of the NAF method. Because Shamir's trick interleaves the two scalar multiplications, we need to make sure that the MSB of u_1 is zero so that in the first round of Algorithm 8 only our chosen 'public key' gets added to the intermediate result R. This in turn, serves again as a input for the Jacobian doubling and the squaring operation we are interested in. The value of u_1 can be brute forced in a similar way as u_2, but then with the message to hash instead of the partial signature r.

As a final remark, it is good to realize that the signature verification results will be completely bogus. This is the result of extracting intermediate values of one scalar multiplication and provide it as input to a different one. Which on top of that, also make use of different coordinate systems. But the takeaway here is that it does not matter at all that the results are bogus. After all, we are only interested in the power traces of single squarings, not in successfully verifying a signature.

5.5 Measuring

When the power consumption has to be measured, the CWLC needs to be broken in two parts. It allows to connect the Xmega side of the board to an oscilloscope, in our case the Waverunner 8404M-MS from Teledyne LeCroy running at a sample rate of 25Msamples/second.

The oscilloscope itself is connected to another computer where the processing of the power consumption signals happens. The tool used for analyzing the signal captured by the oscilloscope is Inspector[5], which we will discuss later in more detail. For now, see Figure 5.2 for the total measurement setup and Table 5.5 for the corresponding number clarification.

A nice feature of the CWLC is that it supports triggering, which makes it relatively easy to localize relevant parts of the power consumption during the

[5]https://www.riscure.com/security-tools/inspector-sca/

Number	Meaning
1	CWLC Xmega target
2	CWLC main board
3	Micro-USB connector
4	Measuring cable to oscilloscope
5	Wiring for triggering to oscilloscope
6	Computer running Jupyter notebook
7	Waverunner 8404M-MS oscilloscope
8	Processing power trace with Inspector

Table 5.1: Number clarification for OTA setup Figure 5.2.

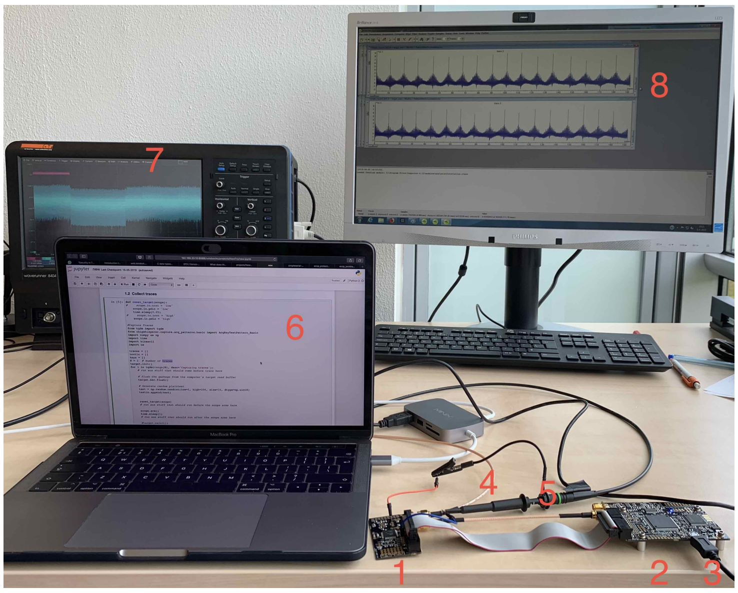


Figure 5.2: Overview of the OTA attack setup.

execution of some code. In our case we put a trigger around the squaring in the Jacobian double operation. Placing a trigger requires a small modification in the code base, which therefore might look to be in direct violence with the scenario described earlier. However, we do not think this is the case because the platform under attack runs programs in a constant way and there is no scheduler present that can interrupt its execution. So, the squaring would always happen at a constant offset of the start of a certain measurement and finishes after a constant amount of time. Therefore, the triggering only serves as a method to make the processing of the power trace a bit more convenient. The attack can also be executed without this triggering mechanism.

On a different note, performing a scalar multiplication on an 8-bit processor with a 256-bit EC is a costly process. It takes around 60 seconds, while a standard home computer does it in a fraction of that time, in the order of milliseconds. Additionally, the oscilloscope can capture up to a maximum of 128M samples, which means it can measure ± 5 seconds of the scalar multiplication with a sampling rate of 25Msamples/second. After doing some test measurements, we realized that one round of the Montgomery ladder takes 192 milliseconds and that the squaring under investigation occurs at an offset of 111.5 milliseconds and takes 10.8 milliseconds. As a consequence, our setup allows us to process ± 20 scalar bits used in the Montgomery ladder. After all, ECDSA uses a new scalar for every signature created and therefore it is useless to get a new power trace in order to extract 'the next' 20 bits. Using an oscilloscope which can store more samples would solve the problem. On top of that, an adversary might try to lower the sampling rate, however this has not been tested.

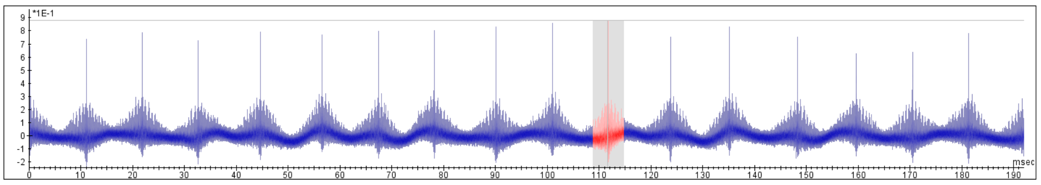
5.6 Extracting Bits

After gathering the target trace and both templates for bit two, remember that bit one is set to the value one by default, the analysis with Inspector can commence. The idea is to perform correlation analysis, with Equation 3.1 from Subsection 3.1.2. Inspector automates this process by calculating the Pearson coefficient between the template and every offset of the target trace.

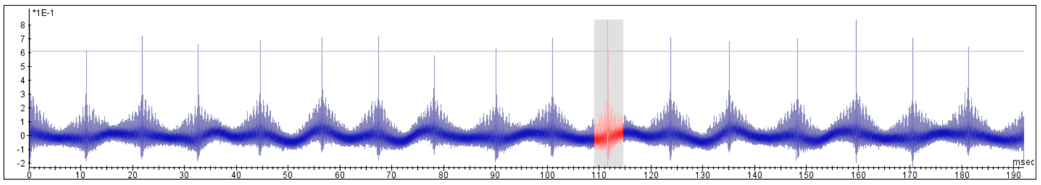
To reduce the computation time of the correlation calculation we applied the principle of window resampling with a window size of 20 with an overlap factor of 0.15, which leads to the reduction of the amount of samples with a factor 17. This means that the average is taken of every 20 sample points and gives 1 new sample, where the last 3 samples of a window are reused in the next one. These parameters used were found empirically and are a balance between computation time and distinctiveness between the templates. Namely, providing a too large window leads to similar templates and therefore the correlation analysis becomes rather meaningless.

Applying the concept of window resampling and correlation results in the end in Figure 5.3 where (a) is for the template with bit zero and (b) with bit one.

In both figures of Figure 5.3 the first complete ladderstep of the target trace



(a)



(b)

Figure 5.3: Correlation results second MSB templates. (a) represents bit 0 and (b) bit 1. The areas marked red represent the correlation peaks of both templates with the square of x in the target trace.

is displayed, correlated with the different templates. In the 192 milliseconds seventeen peaks, note the one at 0 seconds, can be observed. Such a peak indicates correlation between the template square operation and whatever happened in the target trace at that point in time. Closer inspection of the figure and Algorithm 10 from Appendix B indicates that every peak resembles either a multiplication or square operation and our targeted squaring should be the eleventh peak. And indeed, the eleventh peak occurs at an offset of 111.5 milliseconds.

The only thing that is now left to do is comparing the correlation coefficients of both templates at the offset of 111.5 milliseconds. In the case of Figure 5.3, (a) has a coefficient of 0.88 and (b) 0.61. Therefore, the bit represented in (a) is more likely to be correct and we assume that the second bit of the scalar is 0. And indeed, when gathering the target trace, the most significant byte of the secret scalar had a value of 0x97, which is in binary: 10010111. So, our analysis is correct.

From this point onwards, everything described above can be repeated to retrieve the next MSBs. However, first two new possible values for x need to be computed for the next Montgomery ladder step, taking into account the value of the previous found bit. Thereafter the templates need to be collected after which the correlation calculations can take place with Inspector. To clarify, for the n^{th} bit we should not take the correlation peak at 111.5 milliseconds but the one at $(n-2)192 + 111.5$ milliseconds.

We repeated this procedure with eleven bits using the Shamir Trick method and afterwards with an additional five bits using the NAF method for the double scalar multiplication in the ECDSA verification method. We extracted for every bit the correct value without any error, once we established the appropriate parameters for the window resampling, as mentioned above.

While executing these routines, we noticed that the values of the correlation are not very constant. For example, it happened more than once that a correlation value of 0.83 was the 'winning' candidate, while in another round this value was of the 'losing' candidate. Therefore we do not feel comfortable in this setup to only create one template and compare it against some threshold value as the original OTA paper suggests.

Nevertheless, even if somehow the wrong bit was selected as the correct one, this does not have to be problem. Namely, some simple testing showed that the correlation values of the next scalar bit templates would both be significantly lower if the previous bit was guessed wrong. As a consequence, these lower correlation values serve as an indication that something is not correct.

5.7 Countermeasures

In order to prevent this attack from happening, we should make it impossible for an adversary to learn anything about the secret scalar k. In order to do so, we need to create somehow a dissimilarity between the scalar multiplication while signing and those while verifying. The methods we discuss below are based on the influential paper [9] from Coron.

The first method is called scalar randomization, see Equation 5.1.

$$kP \leftarrow k'P - rqP \tag{5.1}$$

In Equation 5.1 k' represents the randomized scalar equal to $k + r\#E(\mathbb{F}_p)$ where r is a random scalar and $\#E(\mathbb{F}_p)$ represents the order of the curve. Because $r\#E(\mathbb{F}_p)$ is equal to the neutral element of the curve, effectively P is still multiplied with k after the modular reduction.

The problem with this counter measure is that it it only works when an attacker needs to capture more than one target trace in order to reconstruct k, which is not the case with OTA. So, now the scalar simply becomes bigger in size and therefore more bits need to be recovered. So the only complication in the attack is that an oscilloscope with a bigger memory is required because the whole of k' needs to be reconstructed with one target trace. Once achieved, k' can be taken modulo the order of the curve, which leaves us once again with k.

A more effective countermeasure is the concept of randomized projective coordinates, which is based on the idea that the projective coordinates of a point are not unique, see Equation 5.2 where λ is a random scalar in $\mathbb{F}_p$.

$$(X, Y, Z) \leftarrow (\lambda X, \lambda Y, \lambda Z) \tag{5.2}$$

In case of our attack, this principle should be applied to the base point used in the scalar multiplication while signing. By multiplying the coordinates λ amount of times, which is unknown to the attacker, it becomes nearly impossible for him to predict the intermediate inputs x for the squaring operation in the Montgomery ladder used in our attack. And if the adversary can no longer create templates based on these possible input values, our attack becomes unfeasible.

One of the main advantages of this countermeasure is that it is a very cheap option to implement in terms of execution time. It only takes three finite field multiplications, where as a full Montgomery ladder already takes thousands of such operations. So, there is very little overhead.

5.8 Implications

In the attack described we used the implementation of [14] for the Montgomery ladder. However, that does not mean that this is the only variant affected. Namely, every implementation where on the signing part some secret value dependent operation occurs, for which the input value can be computed by an adversary, is affected. That is, as long as that similar, in our case the squaring, operation also occurs on the verifying part where the adversary has control over the input value.

Even though this new applicability sounds like a rather simple and powerful attack, its reach is somewhat limited. Although it provides a new attack vector, the countermeasures required are the same as for the standard OTA attack. Furthermore, in prominent cryptographic libraries, such as OpenSSL, the concept of randomized projective coordinates is enabled by default. On the other hand, it would be wishful thinking to assume that every cryptographic implementation around the world currently implements this countermeasure, so there should be opportunities left to apply this attack in a real world setting.

5.9 Future research

Below we want to give some pointers which might serve as pointers to extend the research conducted in this thesis.

As a first idea it might applicable to apply the attack vector discussed in this chapter on a device and its corresponding vulnerable implementation which is actually implemented 'in the wild'. It would be the most convenient to attack 8 bit platforms because of their limited processing power. This makes it easier and cheaper to conduct a SCA because the value of the sample rate for the oscilloscope can be limited.

A second research could focus on trying to expand the attack to different architectures and try to identify whether the approach is still feasible. Every architecture has its own unique characteristics and it might be that it becomes more difficult to capture the leakage.

As a final proposal we suggest to take a broader scope to see which implementations are actually affected by this attack. This would include making an enumeration of the cryptographic libraries employed in practice and inspect its code base. This research can for example also target implementations using NAF representations in the scalar multiplication while signing. This would lead to the interesting case that an adversary needs to create three templates to match with the actual target trace, instead of the regular two ones.

Chapter 6

Conclusion

In the beginning of this thesis we asked ourselves the question whether an OTA attack could be performed on an 8 bit platform in order to extract the secret signing key when targeting the ECDSA verification algorithm. In Chapter 5 we demonstrated that this is indeed feasible, under certain assumptions.

We showed that extracting a secret signing key does not happen directly, but via reconstructing the secret scalar used in the scalar multiplication with the base point during the signature generation. To make the attack as realistic as possible, we attacked an implementation of the Montgomery ladder, which provides protection against standard SCA techniques.

One of the big constraints of extracting the secret scalar via the 'verification side' is that somehow there must be some kind of similarity in the underlying field arithmetic between both the signing and verifying algorithms. That is, there must be some operation on the signing part, which is depending on the secret scalar, which can be imitated by an adversary via the verification algorithm by providing a certain input. In our case, the relevant operation was a squaring in the Montgomery ladder whose input was depending on a secret scalar bit.

Once the relevant operations have been identified and the precomputations for the input of the squaring have been made, the actual power consumption templates can be made. As a final step, some relatively simple postprocessing steps allow for reconstruction of the secret scalar bit by bit.

However, to be fair, there is a relatively simple countermeasure to prevent the OTA from happening, which is making use of randomized projective coordinates. Because prominent cryptographic libraries such as OpenSSL already have this countermeasure enabled by default, the impact of the attack is limited. Nevertheless, this thesis shows an interesting proof of concept by extracting a secret signing key via the verification part of an algorithm, which has not been done before with OTA on ECDSA.

References

[1] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM side-channel(s). In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 29–45. Springer, 2002.

[2] M. Alam, H. A. Khan, M. Dey, N. Sinha, R. Callan, A. Zajic, and M. Prvulovic. One&Done: A Single-Decryption EM-Based Attack on OpenSSL's Constant-Time Blinded RSA. In *27th USENIX Security Symposium Security 18*, pages 585–602, 2018.

[3] L. Batina, Ł. Chmielewski, L. Papachristodoulou, P. Schwabe, and M. Tunstall. Online template attacks. In *EUROCRYPT*. Springer, 2014.

[4] N. Benger, J. Van de Pol, N. P. Smart, and Y. Yarom. "ooh Aah... Just a Little Bit": A small amount of side channel can go a long way. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 75–92. Springer, 2014.

[5] D. J. Bernstein. Curve25519: new Diffie-Hellman speed records. In *International Workshop on Public Key Cryptography*, pages 207–228. Springer, 2006.

[6] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.

[7] R. B. Carpi, S. Picek, L. Batina, F. Menarini, D. Jakobovic, and M. Golub. Glitch it if you can: parameter search strategies for successful fault injection. In *International Conference on Smart Card Research and Advanced Applications*, pages 236–252. Springer, 2013.

[8] S. Chari, J. R. Rao, and P. Rohatgi. Template attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 13–28. Springer, 2002.

[9] J.-S. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 292–302. Springer, 1999.

[10] J. Daemen and V. Rijmen. AES proposal: Rijndael, 1999.

[11] D. Genkin, L. Pachmanov, I. Pipman, E. Tromer, and Y. Yarom. ECDSA key extraction from mobile devices via nonintrusive physical side channels. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1626–1638. ACM, 2016.

[12] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, Berlin, Heidelberg, 2003.

[13] N. A. Howgrave-Graham and N. P. Smart. Lattice attacks on digital signature schemes. *Designs, Codes and Cryptography*, 23(3):283–290, 2001.

[14] M. Hutter, M. Joye, and Y. Sierra. Memory-constrained implementations of elliptic curve cryptography in co-Z coordinate representation. In *AFRICACRYPT*. Springer, 2011.

[15] T. Izu and T. Takagi. A fast parallel elliptic curve multiplication resistant against side channel attacks. In *International Workshop on Public Key Cryptography*, pages 280–296. Springer, 2002.

[16] D. Johnson, A. Menezes, and S. Vanstone. The elliptic curve digital signature algorithm (ECDSA). *International journal of information security*, 1(1):36–63, 2001.

[17] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *ACM SIGARCH Computer Architecture News*, pages 361–372. IEEE Press, 2014.

[18] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[19] P. Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre Attacks: Exploiting Speculative Execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[20] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Annual International Cryptology Conference*, pages 388–397. Springer, 1999.

[21] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Annual International Cryptology Conference*, pages 104–113. Springer, 1996.

[22] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication, 1997.

[23] M. G. Kuhn. *Compromising emanations: eavesdropping risks of computer displays*. PhD thesis, University of Cambridge, 2002.

[24] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom. RAMBleed: Reading Bits in Memory Without Accessing Them. In *41st IEEE Symposium on Security and Privacy (S&P)*, 2020.

[25] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg. Meltdown: Reading Kernel Memory from User Space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[26] S. Mangard, E. Oswald, and T. Popp. *Power analysis attacks: Revealing the secrets of smart cards*. Springer Publishing Company, Incorporated, 2010.

[27] V. S. Miller. Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques*, pages 417–426. Springer, 1985.

[28] M. Minkin, D. Moghimi, M. Lipp, M. Schwarz, J. Van Bulck, D. Genkin, D. Gruss, F. Piessens, B. Sunar, and Y. Yarom. Fallout: Reading Kernel Writes From User Space. *arXiv preprint arXiv:1905.12701*, 2019.

[29] P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.

[30] D. Oswald. ID and IP Theft with Side-Channel Attacks, Presentation, slide 24, Ruhr-Universität Bochum, 2014.

[31] L. Papachristodoulou. *Masking Curves: Side-Channel Attacks on Elliptic Curve Cryptography and Countermeasures*. PhD thesis, Radboud University Nijmegen, 2019.

[32] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[33] N. Samwel, L. Batina, G. Bertoni, J. Daemen, and R. Susella. Breaking Ed25519 in WolfSSL. In *Cryptographers' Track at the RSA Conference*, pages 1–20. Springer, 2018.

[34] E. G. Straus. Addition chains of vectors (problem 5125). *American Mathematical Monthly*, 70(806-808):16, 1964.

[35] N. Technology. Chipwhisperer-Lite (CW1173) Basic Board. `http://store.newae.com/chipwhisperer-lite-cw1173-basic-board/`.

[36] W. Van Eck. Electromagnetic radiation from video display units: An eavesdropping risk? *Computers & Security*, 4(4):269–286, 1985.

[37] S. van Schaik, A. Milburn, S. Österlund, P. Frigo, G. Maisuradze, K. Razavi, H. Bos, and C. Giuffrida. RIDL: Rogue In-Flight Data Load. *S&P (May 2019)*, 2019.

[38] C. D. Walter. Sliding windows succumbs to Big Mac attack. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 286–299. Springer, 2001.

Appendices

Appendix A

Baudline Spectrum Analysis Phone

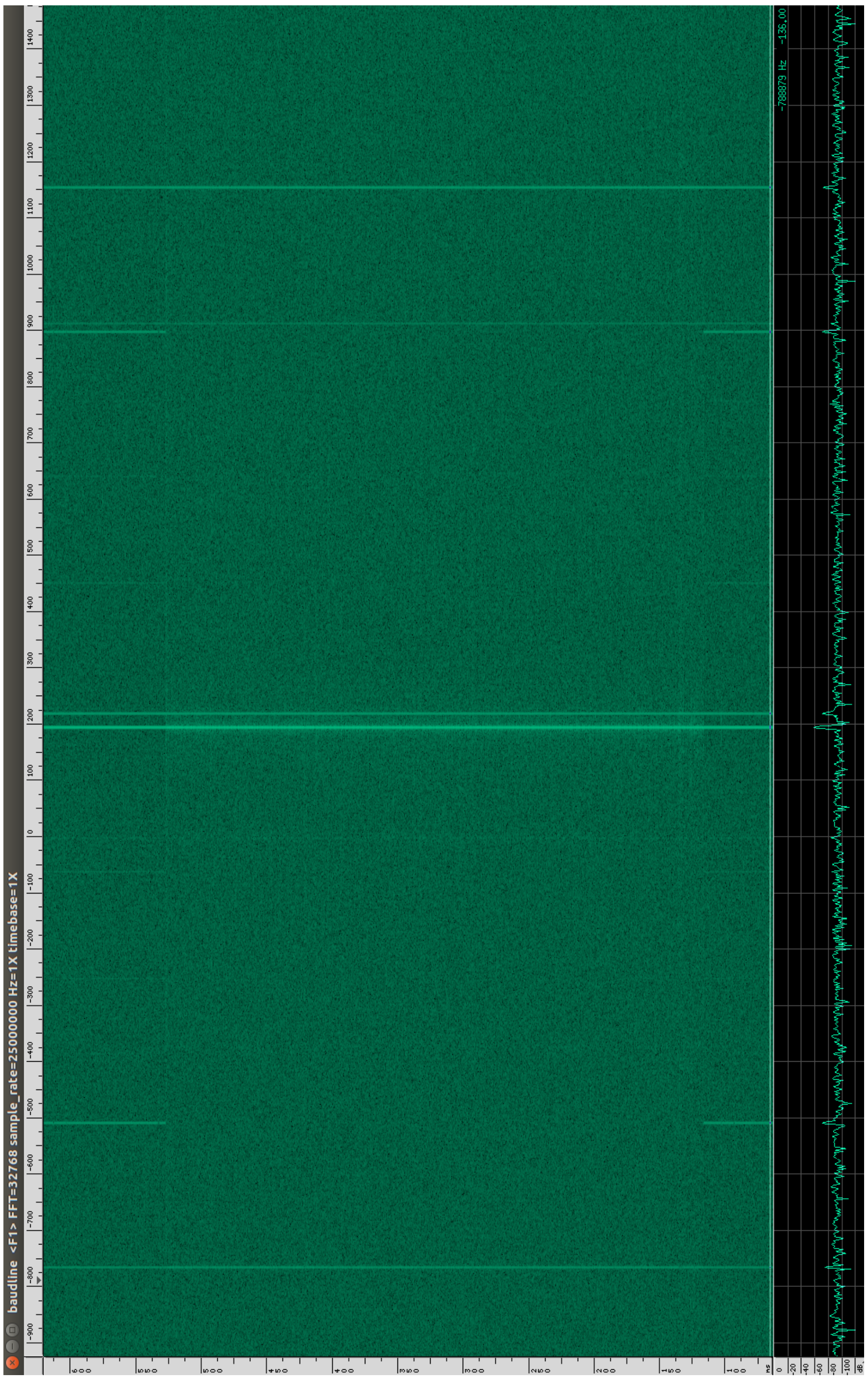

baudline <F1> FFT=32768 sample_rate=25000000 Hz=1X timebase=1X
-788879 Hz -136.00

Figure A.1: Visualization of leakage phone during signing program.

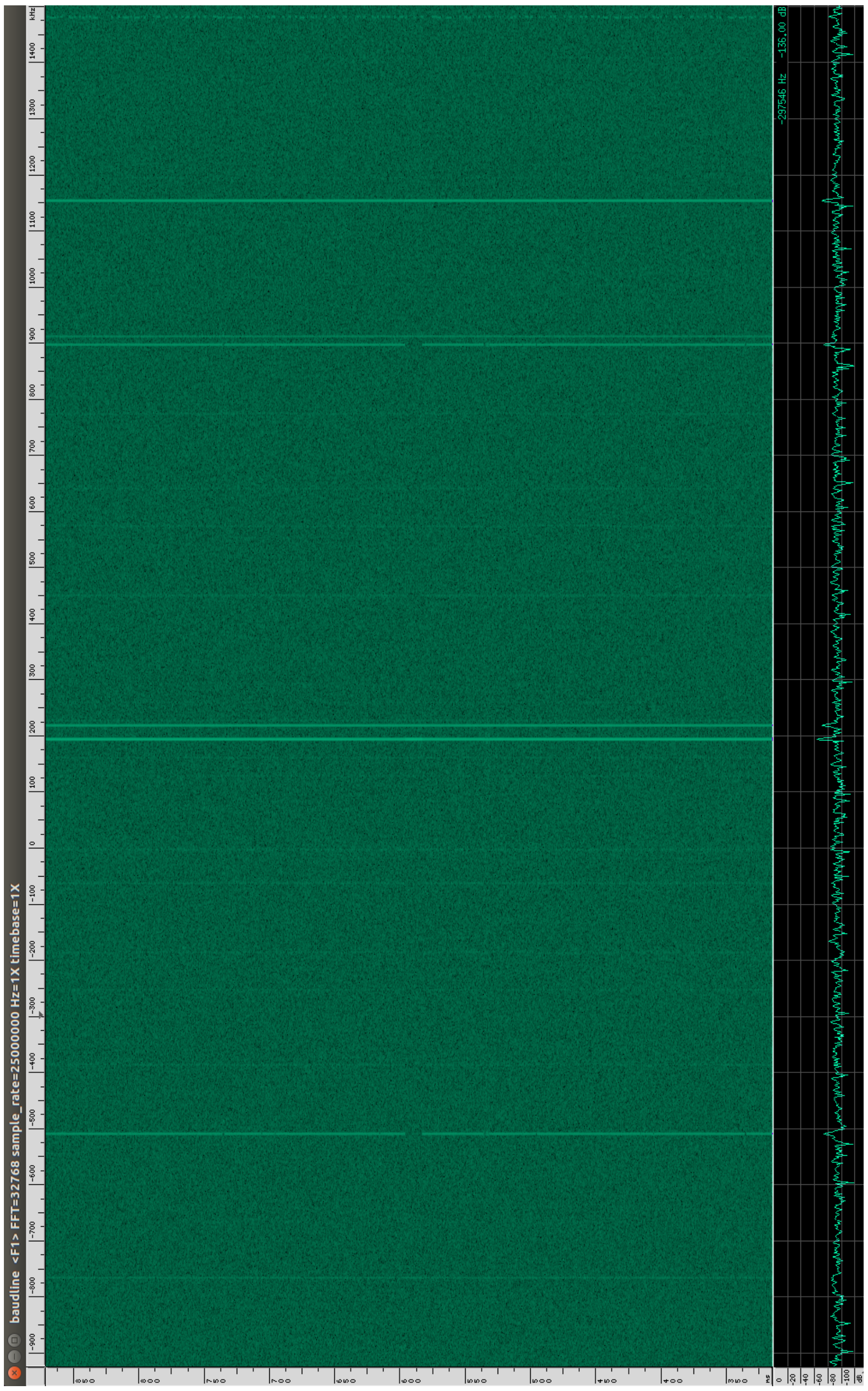

baudline <F1> FFT=32768 sample_rate=25000000 Hz=1X timebase=1X
-297546 Hz -136.00 dB

Figure A.2: Visualization of leakage phone during signing program without printf statements.

Appendix B

Montgommery Ladder Step

Algorithm 10: One Montgommery ladder step in standard projective coordinates [14]. Note that R_0 and R_1 from Algorithm 7 resemble with X_1 and X_2 in Algorithm 10.

```
Input  : $X_1, X_2, Z, x_D, a, 4b$
Output: $(X_1, X_2, Z)$
1  $R_1 \leftarrow X_1 \cdot X_2$
2  $R_3 \leftarrow Z^2$
3  $R_4 \leftarrow Z \cdot R3$
4  $R_2 \leftarrow a \cdot R_3$                  // a is curve parameter
5  $R_1 \leftarrow R_1 + R_2$
6  $X_1 \leftarrow X_1 + X_2$
7  $R_3 \leftarrow X_1 \cdot R_1$
8  $X_1 \leftarrow X_1 - X_2$
9  $X_1 \leftarrow X_1 - X_2$
10 $R_1 \leftarrow 4b \cdot R_4$                 // b is curve parameter
11 $R_4 \leftarrow {X_1}^2$
12 $X_1 \leftarrow R_4 \cdot Z$
13 $R_3 \leftarrow R_3 + R_3$
14 $R_3 \leftarrow R_3 + R_1$
15 $Z \leftarrow X_2 \cdot R_4$
16 $R_4 \leftarrow R_1 \cdot X_2$
17 $R_1 \leftarrow {X_2}^2$
18 $R_2 \leftarrow R_1 + R_2$
19 $R_1 \leftarrow R_1 + R_1$
20 $X_2 \leftarrow x_D \cdot X_1$                 // $x_d$ is x-coord EC point
21 $R_3 \leftarrow R_3 - X_2$
22 $X_2 \leftarrow R_1 \cdot R_2$
23 $X_2 \leftarrow X_2 + X_2$
24 $R_2 \leftarrow R_2 - R_1$
25 $R_1 \leftarrow R_4 + R_4$
26 $R_4 \leftarrow X_2 + R_4$
27 $X_2 \leftarrow {R_2}^2$
28 $R_1 \leftarrow X_2 - R_1$
29 $X_2 \leftarrow R_1 \cdot Z$
30 $Z \leftarrow X_1 \cdot R_4$
31 $X_1 \leftarrow R_3 \cdot R_4$
32 return $(X_1, X_2, Z)$
```