MASTER

A template attack on elliptic curves using classification methods

Yeilbek, E.

Award date:
2015

Technische Universiteit Eindhoven

Master Thesis

A Template Attack on Elliptic Curves using Classification methods

Author:
Elif Özgen

Supervisors:
Lejla Batina
Berry Schoenmakers

A thesis submitted in fulfillment of the requirements for the degree of Master of Science (MSc)

in the

Coding and Cryptology Group
Mathematics and Computer Science, Technische Universiteit Eindhoven

November 2015

Technische Universiteit Eindhoven

Abstract

Mathematics and Computer Science
Information Security Technology (Kerckhoffs Institute for Computer Security)

Master Degree

A Template Attack on Elliptic Curves using Classification methods

by Elif Özgen

Side-channel analysis (SCA) is a cryptanalytic technique that uses information leaked from a device when it performs a cryptographic operation in order to retrieve the private key. For instance, timing differences, power consumption or electromagnetic emanation are well-known leakage types of a device. In this project, we aim to perform a side-channel attack based on power consumption leakage of a device running an elliptic curve cryptographic implementation, particularly we focus on point multiplication using a Double-and-Add Always algorithm. We decided to attack a Double-and-Add Always algorithm implementation since that algorithm is specifically designed to be resistant against Differential Power Analysis (DPA) by having a constant time format. Some of the classical SCA techniques use correlation methods (e.g. Pearson correlation) as side-channel distinguishers during the key hypothesis test. In our project we plan to apply classification algorithms as side-channel distinguishers and in this way we should be able to retrieve the key efficiently. By classification we name the method of classifying a data set according to similarities. In our project we will apply a template attack, in a similar approach as in Online Template Attacks (OTA) [1], with classification algorithms as a side-channel distinguisher. To the best of our knowledge, our analysis constitutes the first application of using classification techniques of k-Nearest Neighbour, Naïve Bayes and Support Vector Machines with template attack on the Double-and-Add Always algorithm for elliptic curves. According to the results, we propose relevant countermeasures.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

It is well known that cryptographic devices like smart card readers leak some physical information, for instance it could be the amount of power they consume in a specific time period.

Therefore, a set of attacks is proposed in order to take advantage of these physical leakages to obtain sensitive information. That sensitive information is usually the cryptographic private key. As those attacks are using "side-channel" information, they are called side-channel attacks.

As an important aspect of these attack we can say that they work efficiently even though the algorithm under attack has been shown to be mathematically very secure. That is because, most of the time these leakages occur due to the way the algorithm is implemented. Therefore, the leakage has correlation with the algorithm that is being used. That is the reason that Side-channel Analysis (SCA) are successful since we know that the measured leakage like power consumption depends on the instructions (operations) and the data itself (operands).

For instance, one of the first shown side-channel attacks was applied to a DES algorithm running device using the power consumption of that device by Kocher et al. in [7]. Moreover, there are many examples of those attacks that are applied to AES, RSA and Elliptic Curve Cryptography (ECC).

In our thesis, we focus on ECC. As we mention in Chapter 2, ECC plays an important role in cryptography, because of the hardness of solving the Discrete Logarithm Problem on elliptic curves. Therefore, when we apply a side-channel attack on ECC, our aim is to find the secret key using the side-channel information (the power consumption) from the scalar multiplication algorithm.

We elaborate a specific type of side-channel attack called template attacks, in Chapter 3. The template attack that we apply on a ECC scalar multiplication algorithm called Double-and-Add Always, makes use of the classification methods. These classification methods are introduced in Chapter 4.

In this chapter, we give more details on side-channel attacks on ECC and give related work. In the following we explain our contribution.

Chapter 2 introduces ECC in details in order to explain the algorithm that we are attacking. In Chapter 3 we elaborate on side-channel attacks and especially the template attacks. In Chapter 4 we introduce the classification algorithms that we used for our template attack. Chapter 5 explains and analyzes our attack. Lastly, Chapter 6 gives our conclusion and future work.

1.2 Side-channel Attack on Elliptic Curve Cryptography

In this section, we give a brief introduction on Side-channel Attacks on Elliptic Curve Cryptography.

Side-channel Attacks is a cryptanalytic technique of using leaked side-channel (physical) information of a device while running a cryptographic operation and via that leaked information retrieving the private key.

From the very first papers of SCA, this attack has been applied to many different cryptosystems such as AES as shown in [8, 9], DES in [7, 10], RSA in [11] and ECC [12–15] moreover [16] presents an attack for both DES and AES.

Briefly, SCA on ECC aims at retrieving the scalar value (the private key) via using the leaked information (in our case it is the power consumption) during the scalar multiplication operation.

The scalar is denoted as k and assuming that we have two points Q and P from an elliptic curve E, satisfying $Q = k \times P$. The public information Q and P are known while k is kept secret. Depending on the implementation and the case of known or unknown input, while performing scalar multiplication, different types of SCA can be applied to retrieve k.

To accomplish this task, SCA (particularly Differential Power Analysis, is introduced in Chapter 3) often consists of several certain steps according to [17], such as;

1. Collect measurements (traces)
2. Use statistics on measurements for noise elimination and alignment
3. Select a model for power consumption

4. Select a side-channel distinguisher as in [16, 18, 19]
5. Test the hypothesis
6. Find the correct key

Meanwhile, as the attacks improve, many countermeasures are introduced as well. The best known three countermeasures according to [14] are randomization of the private exponent, blinding the point P (where $Q = k \times P$) and randomized projective coordinates (these type of coordinates are discussed in Section 2.2.1).

We are aiming to apply a template attack on Double-and-Add Always algorithm, where we have templates and use classification methods as side-channel distinguisher. And as we focus on chosen sub-parts of target point's power trace, we try to classify it with our template power traces in order to extract the information of the current bit of the key.

Template attacks are discussed in Chapter 3. Briefly, this type of side-channel attacks consists of two steps called template building and template matching. In the first phase, template building, the attacker is assumed to have an identical device to the device under the attack and is able to execute instructions with chosen input data. Each of these instructions' leakages are recorded and we call each of the measurements *templates*.

The attacker tries to *characterize* the device under attack with those templates. The next step is the template matching phase. In that phase, the attack uses the actual device under attack. This time only one trace (called target trace) is collected and the attacker tries to capture the private key of this device via matching the target trace with one of the template traces.

The details of the attack can be found in Chapter 5. In the Chapter 2, we provide information on Elliptic Curve Cryptography. The Chapter 3 introduces and elaborated Side-Channel Attacks ob Chapter 4 introduces Classification and Clustering algorithms.

In Chapter 5, we demonstrate our experiments and explain how we implemented SCA in practice. Our approach and the tools that we used to run the attack is described in that chapter as well.

Chapter 6 concludes our overall work and mentions future work.

1.3 Related Work

In [20], a template attack is applied to ECDSA algorithm in a 32-bit setting. Moreover, it is concluded that, in order to make the implementations secure against template attacks, we need to use countermeasures against DPA.

The work described in ota proposes a template attack using the *online templates* that requires only one power trace per key-bit plus the target trace (e.g. for a 256-bit exponent we need to have 257 power traces) that is captured during the scalar multiplication operation run on the device under the attack. The name *online templates* is used for the templates because they are generated after the target trace is acquired. Moreover, it is elaborated that there is no need for the preprocessing phase on template building.

The distance metric is determined by the classification algorithms and depending on that chosen metric we can decide on data closeness. There is another technique similar to classification, called clustering. This method clusters the data, but in this case we do not have the labels. Labels refer to the name of the classes. That difference makes this method an unsupervised learning. Classification is a supervised learning as it has labels for the data sets.

As an unsupervised learning technique, clustering in SCA is used in many proposed studies, such as [21] and [19].

In [19], the steps of DPA are followed and as a side-channel distinguisher, cluster analysis is used. That is done via choosing clustering instead of for instance taking the difference of means after partitioning. The candidate for the current bit of the key that gives the best clustering result is chosen as the correct bit of the secret key. Moreover, the quality of the clustering is to be measured by the defined clustering criterion functions,

There is a similar work to ours which uses an unsupervised learning method called clustering in SCA is [22] that is applied to elliptic curve scalar multiplication. In this work, we have a single-execution (horizontal) setting. The aim is to find similar segments of the execution via clustering, and then recover the corresponding key bits. That paper uses the electromagnetic emanations and location-based leakages and shows that the combined measurements from different locations on FPGA (a type of programmable circuit even after being manufactured) gives a better clustering result. The k-means clustering algorithm with Euclidean distance metric is used. The k-means clustering algorithm is introduced in Section 4.3.1.

Another close approach to ours is [23]. In[23], template attacks are taken into account and data reduction techniques, namely mean class variance, the Spearman correlation, and principal component analysis, are tested for the template building (profiling) phase in combination with Linear Discriminant Analysis. Linear Discriminant Analysis is used as a distinguisher for the classification step (template matching) in which they are attacking to AES 128-bit.

1.4 Our Contribution

For this thesis, we performed a template attack on the Double-and-Add Always algorithm using the classification algorithms called k-Nearest Neighbour, Naïve Bayes and Support Vector Machines.

To the best of our knowledge, our analysis constitutes the first application of using classification techniques of k-Nearest Neighbour, Naïve Bayes and Support Vector Machines with template attack on the Double-and-Add Always algorithm for elliptic curves.

Our attack can also be applied to Montgomery Ladder as that algorithm has a similar characteristics with Double-and-Add Always algorithm.

Chapter 2

Elliptic Curves

In this chapter, we introduce Elliptic Curve Cryptography as this is the cryptographic setting that we are using and the Double-and-Add Always algorithm as this is the algorithm that we are applying a template attack.

2.1 Elliptic Curve Cryptography

Public key cryptosystems are known to provide security based on hard number theoretical problems. For instance, RSA uses the integer factorization problem in order to achieve a strong security level.

However, the problem with most public key cryptosystems was that they usually require long key length and therefore cause computation overhead which is not very efficient. In the study [2], it is shown that ECC provides the same security level with RSA with much shorter key lengths. That result also indicates that this particular PKI (Public Key Infrastructure) can be used for IoT (Internet of Things) devices which require light weight implementations.

Elliptic Curve Cryptography is using elliptic curves over a finite field as cryptographic primitive.

Elliptic curves form the geometrical structure which can be defined by different forms and therefore by different formulae. Some of these equations are introduced below.

The ECC forms a group structure. For example, because of the closeness property of the binary operation, we have the same feature in a group therefore when we do an addition of two elements we acquire another element that is from the same group. As elliptic curves are also group, then when we make addition of two points of an elliptic curve, we get another point that is in that curve as well.

We have a special element called "unit element" (also called "point at infinity"). This is the point from an elliptic curve, such that when we add another point P to this element then we get P itself back. Thus the unit element corresponds to the identity element of group structure.

Moreover, there is a number theoretical problem called Elliptic Curve Discrete Logarithm Problem (ECDLP). Let P and Q be two points on an elliptic curve where $Q = k \times P$, where k is an integer. Solving ECDLP means finding that scalar k which satisfies $Q = k \times P$. Therefore, scalar multiplication has an important role in ECC.

2.1.1 Weierstrass Curve

For an elliptic curve E over a finite field F_q we have the following curve equation, it is called Weierstrass form, in affine form as in follows:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.1}$$

where a_i's are coefficients.

Figure 2.1 represents sample elliptic curve in the short Weierstrass form from Equation 2.3.

2.2 Doubling and Adding on Short Weierstrass form Curves in Affine Coordinates

Let us use the same notation in [2, Chapter 9], the group operation defined is addition (+). Now, we have two cases; namely point addition and point doubling.

Let $P = (x_1, y_1)$, $Q = (x_2, y_2)$, $R = (x_3, y_3)$;then

$$P + Q = R \tag{2.2}$$

Here, the coordinates of the sum is not calculated as in classical analytical geometry but with specific-curve type dependent formulas.

As in [2], we can use the following point addition formulae for an elliptic curve defined over finite field F_p where p is prime and in short Weierstrass form:

$$E : y^2 = x^3 + ax + b; \quad a, b \in F_p \tag{2.3}$$

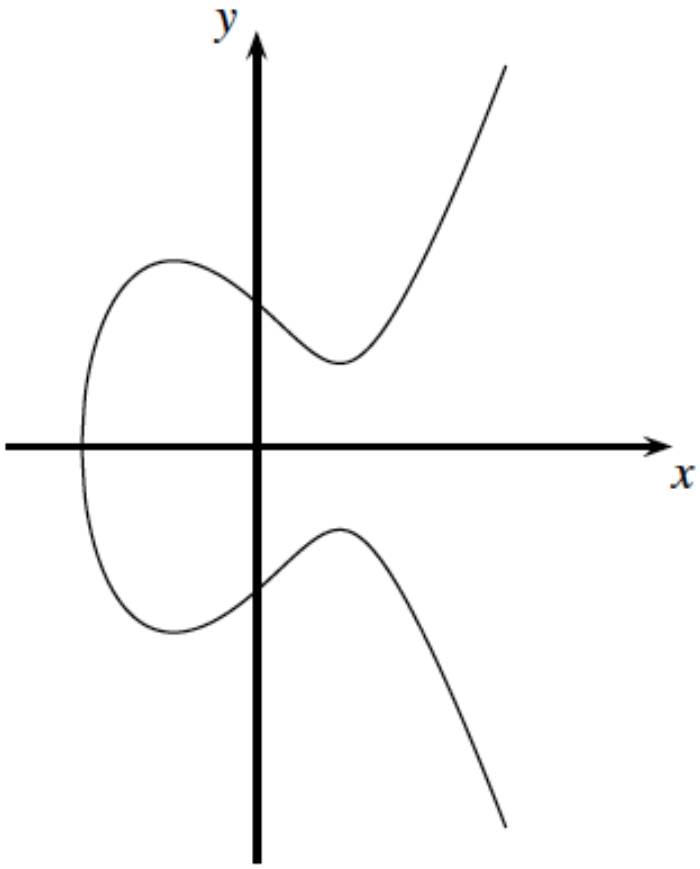

FIGURE 2.1: A sample elliptic curve ($y^2 = x^3 - 3x + 3$) with from [2, Chapter 9]

If $P \neq Q$ (Addition):

$$s = \frac{y_2 - y_1}{x_2 - x_1} \mod p \tag{2.4}$$

If $P = Q$ (Doubling):

$$s = \frac{3x_1^2 + a}{2y_1} \mod p \tag{2.5}$$

The sum has the coordinates as;

$$x_3 = s^2 - x_1 - x_2 \mod p \tag{2.6}$$

and

$$y_3 = s * (x_1 - x_3) - y_1 \mod p \tag{2.7}$$

In this representation, we see that, for addition there are $t(A + A) = I + 2M + S$ that is 1 inversion, 2 Multiplications and 1 squaring operation are performed while for doubling

$t(2A) = I + 2M + 2S$ that is 1 inversion, 2 Multiplications and 2 squarings operation [1] are performed [3].

As we see in Section 2.2.1 and in Section 2.2.2, we do not have any inversion operations on these representations, which makes the implementation of the ECC design more efficient. However, in affine coordinates we do have inversion operations. We can conclude that, according to the number of multiplication, inversion and squaring operations, the efficiency of Jacobian representation is greater than or equal to the efficiency of Projective representation is greater than or equal to the efficiency of Affine representation. Here efficiency is in terms of time.

We can refer to that conclusion from Figure 2.2 where from [3] also demonstrates their method of using a new and mixed coordinate system is the most efficient one. In this figure we can also verify that inversion is the most time consuming operation.

In Figure 2.2, under the field *operations* we can see the time required by each type of operations. Under the "Elliptic Curve operations", $t(\mathcal{A} + \mathcal{A})$ refers to addition in affine coordinates, $t(\mathcal{J}^c + \mathcal{J}^c)$ refers to addition in Chudnovsky Jacobian coordinates system which is a different type of Jacobian coordinates that are designed to make the addition relatively faster moreover $\mathcal{J}^m$ and "mixed coordinates" in the figure is the modified Jacobian coordinate system that is proposed by [3].

2.2.1 Projective Coordinates

Projective coordinates are acquired via switching from the affine form (x, y) to (X, Y, Z) where $x = X/Z$ and $y = Y/Z$, giving the following elliptic curve equation

$$E_P : Y^2Z = X^3 + aXZ^2 + bZ^3 \tag{2.8}$$

[3]. The benefit of using projective representation is that this form of representation does not require inversion operation (as it can be seen from the addition and doubling formulas) instead it uses more multiplications [2, Chapter 9]. The reason of eliminating inversion operations is that, these operations require the most time complexity compared to multiplication and squaring operations. Figure 2.2 is explaining this situation.

As we have different form of representation for coordinates than affine coordinates, the addition and doubling changes as well. We use the same notation and illustration as Cohen et al.[3] for explaining these operations.

For $P = (X_1, Y_1, Z_1)$, $Q = (X_2, Y_2, Z_2)$ and $P + Q = R = (X_3, Y_3, Z_3)$.

Addition $(P \neq \pm Q)$ for that representation is as follows:

[1] One of M's in both addition and doubling comes from not to have zero divison

	160 bit key	192 bit key	224 bit key
field operations (μsec)			
160/192/224 bit addition	0.59	0.64	0.71
160/192/224 bit multiplication	6.50	8.93	12.00
160/192/224 bit squaring	5.35	7.22	9.01
reduction (320/384/448 $\rightarrow$ 160/192/224 bit)	2.37	2.77	2.62
160/192/224 bit inverse	166	213	261
elliptic curve operations (msec)			
addition ($t(\mathcal{A}+\mathcal{A})$)	0.203	0.257	0.314
addition ($t(\mathcal{J}^c+\mathcal{J}^c)$)	0.130	0.171	0.215
addition ($t(\mathcal{J}+\mathcal{J})$)	0.144	0.191	0.239
doubling ($t(2\mathcal{J}^m)$)	0.079	0.103	0.127
doubling ($t(2\mathcal{J})$)	0.094	0.122	0.148
elliptic curve exponentiation (msec)			
mixed coordinates (case 1)	16.17	24.93	35.73
mixed coordinates (case 2)	16.66	25.54	37.53
single coordinate (Jacobian coordinate)	18.66	28.79	41.86
single coordinate (projective coordinate)	20.33	30.17	44.79

FIGURE 2.2: Times for Elliptic Curve operations (UltraSPARC) from [3]

$X_3 = vA$,

$Y_3 = u(v^2X_1Z_2 - A) - v^3Y_1Z_2$,

$Z_3 = v^3Z_1Z_2$

with $u = Y_2Z_1 - Y_1Z_2$,

$v = X_2Z_1 - X_1Z_2$,

$A = u^2Z_1Z_2 - v^3 - 2v^2X_1Z_2$.

Doubling ($R = 2P$) for that representation is as follows:

$X_3 = 2hs$,

$Y_3 = w(4B - h) - 8Y_1^2s^2$,

$Z_3 = 8s^3$

with $w = aZ_1^2 + 3X_1^2$,

$s = Y_1Z_1$, $B = X_1Y_1s$ and $h = w^2 - 8B$.

In Section 2.2 we see that, in affine coordinates system for addition there are $t(\mathcal{A}+\mathcal{A}) = I+2M+S$ that is 1 inversion, 2 multiplications and 1 squaring operation are performed while for doubling $t(2\mathcal{A}) = I + 2M + 2S$ that is 1 inversion, 2 multiplications and 2 squaring operations are performed (t function refers to computation time).

However, in projective representation, for addition there are $t(\mathcal{P} + \mathcal{P}) = 12M + 2S$ that is 12 multiplications and 2 squaring operations are performed while for doubling $t(2\mathcal{P}) = 7M + 5S$ that is 7 multiplications and 5 squaring operations are performed. That is why, as indicated in [2, Chapter 9], projective representation costs less than affine representation since inverse operation is more expensive than addition, multiplication and squaring operations. This difference can be observed from Figure 2.2.

2.2.2 Jacobian Representation

We again use the same notation and illustration as Cohen et al.[3] for explaining these operations.

Jacobian coordinates are similar to projective, however in this case we are switching from the affine form of (x, y) to (X, Y, Z) where $x = X/Z^2$ and $y = Y/Z^3$ that is giving the following elliptic curve equation.

$$E_J : Y^2 = X^3 + aXZ^4 + bZ^6 \tag{2.9}$$

For $P = (X_1, Y_1, Z_1)$, $Q = (X_2, Y_2, Z_2)$ and $P + Q = R = (X_3, Y_3, Z_3)$.

Addition ($P \neq \pm Q$) for that representation is as following:

$X_3 = -H^3 - 2U_1H^2 + r^2$,

$Y_3 = -S_1H^3 + r(U_1H^2 - X_3)$,

$Z_3 = Z_1Z_2H$

with $U_1 = X_1Z_2^2$,

$U_2 = X_2Z_1^2$, $S_1 = Y_1Z_2^3$,

$S_2 = Y_2Z_1^3$, $H = U_2 - U_1$ and $r = S_2 - S_1$.

Doubling ($R = 2P$) for that representation is as following:

$X_3 = T$,

$Y_3 = -8Y_1^4 + M(S - T)$,

$Z_3 = 2Y_1Z_1$

with $S = 4X_1Y_1^2$,

$M = 3X_1^2 + aZ_1^4$ and $T = -2S + M^2$.

We can see that in the Jacobian representation for addition there are $t(\mathcal{J}+\mathcal{J}) = 12M + 4S$ that is 12 multiplications and 4 squaring operations are performed while for doubling $t(2\mathcal{J}) = 4M + 6S$ that is 4 multiplications and 6 squaring operations are performed. That performance is more efficient than both projective and affine representations in terms of time. Figure 2.2 from [3] illustrates that conclusion as well.

2.3 Elliptic Curve Cryptography Binary Scalar Multiplication Algorithms

Our project aims to attack the Double-and-Add Always algorithm. Moreover, as another similar binary scalar multiplication algorithm, the Montgomery Ladder can also be exploited by our attack.

An elliptic curve scalar multiplication is the method of computing $k \times P$ where P is a point from the given elliptic curve and k is a l-bit scalar.

Binary scalar multiplication algorithms use the binary representation of the scalar k that is

$$k = \sum_{i=1}^{l-1} k_i 2^i \tag{2.10}$$

And the scalar multiplication is :

$$k \times P = \sum_{i=1}^{l-1} k_i \times (2^i \times P) \tag{2.11}$$

That scalar multiplication can be performed via Double-and-Add multiplication algorithm as in Algorithm 1.

It is explained in [24] that, since power consumption of addition and doubling operations are different, due to different short Weierstrass formulae as showed in Section 2.2, one can distinguish that difference in power consumption traces and therefore retrieve the key. The key retrieval is easy because there is the "if" clause which makes the entire algorithm process dependent on the current bit value of the key.

Algorithm 1 Double-and-Add Scalar Multiplication Algorithm [24]

Input: **P**, l-bit scalar $k = (1, k_{l-2}, ..., k_0)_2$
Output: $\mathbf{Q} = k \times \mathbf{P}$

1: $R_0 \leftarrow P$
2: **for** $j \leftarrow l-2$ **down to** 0 **do**
3: $\quad R_0 \leftarrow 2R_0$
4: $\quad$ if$(k_j = 1)$ then $R_0 \leftarrow R_0 + P$
5: **end for**
6: **return** R_0

2.3.1 Double-and-Add Always Algorithm

That algorithm is created as it uses one of the countermeasures proposed by [14] against SPA that is inserting dummy instructions to regular double-and-add algorithm, hence there is always an addition after a doubling operation as [24] demonstrates. That is because, in Double-and-Add algorithm 1, when the current bit of the key is 0, there is only a doubling operation performed however in algorithm 2 case there is addition as well even though that addition does not have an importance on the computation of the actual value. This technique makes this algorithm an algorithm with data-independent execution path. In Algorithm 2, Double-and-Add Always is illustrated.

Algorithm 2 Double-and-Add Always [24]

Input: **P**, l-bit scalar $k = (1, k_{l-2}, ..., k_0)_2$
Output: $\mathbf{Q} = k \times \mathbf{P}$

1: $R_0 \leftarrow P$
2: **for** $j \leftarrow l-2$ **down to** 0 **do**
3: $\quad R_0 \leftarrow 2R_0$
4: $\quad R_1 \leftarrow R_0 + P$
5: $\quad b \leftarrow k_j$
6: $\quad R_0 \leftarrow R_b$
7: **end for**
8: **return** R_0

In our project we are attacking to this algorithm as it is explained in Chapter 5.

2.3.2 Montgomery Ladder

Montgomery Ladder is another scalar multiplication algorithm. Our attack can also be applied to that algorithm since it has the same structure as Double-and-Add Always. We give further information on how we can use our technique to attack that algorithm on Chapter 6.

Algorithm 3 Montgomery Ladder [24]

Input: $\mathbf{P}$, l-bit scalar $k = (1, k_{l-2}, ..., k_0)_2$
Output: $\mathbf{Q} = k \times \mathbf{P}$

1: $R_0 \leftarrow P$
2: $R_1 \leftarrow 2P$
3: **for** $j \leftarrow l-2$ **down to** 0 **do**
4: $\quad b \leftarrow k_j$
5: $\quad R_{1-b} \leftarrow R_0 + R_1$
6: $\quad R_b \leftarrow 2R_b$
7: **end for**
8: **return** R_0

Chapter 3

Side-Channel Attacks

In this chapter, we elaborate on side-channel attacks as well as template attacks since it is the type of attack that we apply. Moreover, we analyze the structure of power traces which gives us more insight about analyzing side-channel leakages.

Our work focuses on using the classification methods that we introduce in Chapter 4 within template attack.

3.1 Side-Channel Analysis

In a cryptosystem as we intend to have strong cryptographic primitives that satisfy the target security level, there are two points of view that we focus on in terms of cryptanalysis. The first point of view would be classical cryptanalysis that deals with mathematical aspect of primitives, which can be considered as a theoretical/mathematical approach.

On the other hand, there is another view that deals with physical information which is leaking from the system. The last approach can be considered as a practical/concrete approach as it tries to attack the system using those acquired physical information.

SCA can utilize many different sorts of physical information, for instance timing [11] or power consumption [7].

SCA represents the practical aspects of cryptanalysis techniques. The very first paper of an example of SCA is taking the timing measurements as a side-channel leakage in order to attack and retrieve the key, by Kocher [11].

The main idea behind timing attacks is to retrieve the private key via using the information of elapsed time during the key-dependent operations that is using that private key. As it was indicated by Kocher [11], operations consume amount of time depending

on the data they are processing and in our case that data would be the private key and other input data such as the plaintext [11].

Power consumption is considered another physical leakage that may cause key retrieval. In 1999, Kocher et al. proposed in [7] the usage of power consumption traces (power traces) of a device during running a cryptographic operation.

Similar to timing attacks, power consumption also depend on the data as well as the operation that is being processed. For instance Hamming weight model is used most of the time, which focuses on switches between 0 and 1.

For instance, there will be a high power consumption while switching from 0 to 1 and a low power consumption while the value stays the same or switching from 1 to 0. Furthermore, besides a power model we need a side-channel distinguisher that helps to find out the similarities or differences between collected power measurements. We refer to those measurements as traces in SCA terminology. Thus, trace is a power consumption data that is measured from the target device.

That kind of analysis on power traces of a device, later in sections 3.3 and 3.4, is elaborated.

Generally for SCA, as we are using physically leaked information, there are probabilities that those leakages (traces) have electrical noise or misalignment (can be static or elastic). These problems might occur due to the nature of gathering physical information or intentionally injected countermeasures by the implementer.

Moreover, we have two different settings in SCA; namely horizontal and vertical. Horizontal setting in SCA refers to make analysis/attack via only one target power trace in different times [25, 26]. On the other hand, vertical setting takes more than one power traces for same time and processes the analyze on them.

The work in this thesis is using the power analysis technique, particularly a variant of template attack, therefore we focused on that particular type of SCA on ECC platform.

Moreover, we introduce a different side-channel distinguisher to be applied to the measurements collected from a device running cryptographic operations which is the classification algorithms.

3.2 Characteristics of Power Traces

In order to observe the SCA works on power traces, we introduce the characteristics of power traces which also helps us to define they way our attack works.

In [27, Chapter 4], more details on that topic is represented, and in this section (including the subsections 3.2.1 and 3.2.2) we follow the same explanations, definitions and notations from this work.

For power traces, there are two important cases for an attacker to make observations, such as operation-dependent data and data-dependent (operand-dependent) data. The power, that is consumed, depends on the operation that the device is running and the data that the device is processing.

There are several components of traces namely; P_{op}, P_{data}, $P_{el.noise}$ and P_{const}.

- P_{op}: Operation-dependent part of the trace
- P_{data}: Data-dependent part of the trace
- $P_{el.noise}$: Electrical noise (In case of keeping the data and the operation fixed, the total power consumption is different for every run due to that component.)
- P_{const}: Constant leakage that every device has in different amounts

As a result we have;

$$P_{total} = P_{op} + P_{data} + P_{el.noise} + P_{const}$$

where P_{total} stands for the entire power consumption sum for the chosen part (point) of the trace. As these values are chosen points of the traces, they form a function of the time structure, moreover that the electrical noise function has a normal distribution form (also called Gaussian Distribution) therefore power traces are also in normal distribution form in that case as [27, Chapter 4] indicates.

3.2.1 Density Function, Mean and Variance

The expressions, in this section, are the primitives that define the characteristics of a trace mathematically. Therefore, to know the structure of these expressions helps us to analyze the traces.

Normal distribution is defined by a density function f which is dependent on two parameters called mean value $-\infty < \mu < \infty$ and standard deviation $\sigma > 0$.

The mean value μ (also called expected value) and standard deviation σ are defined as following. The parameter $Var(X)$ that we used below is the square of the standard deviation σ.

$$\mu = E(X) \tag{3.1}$$

$$\sigma^2 = Var(X) = E((X - E(x))^2) \tag{3.2}$$

According to this notation, a normally distributed variable X with mean μ and standard deviation σ is denoted as $X \sim \mathcal{N}(\mu, \sigma)$. Moreover, if we have $\mu = 0$ and $\sigma = 1$ then we have standard normal distribution.

The role of mean value in a normal distribution is that, it is the parameter that expresses the most possible output out of an experiment as [27, Chapter 4] indicates.

Furthermore, the following equation is the density function.

$$f = \frac{1}{\sqrt{2 \cdot \pi} \cdot \sigma} \cdot \exp(-\frac{1}{2} \cdot (\frac{x - \mu}{\sigma}^2)) \tag{3.3}$$

3.2.2 Multivariate-Gaussian Model

We focus on Multivariate Gaussian Model in this subsection. That is because we use this expression in order to define template attacks in Section 3.5.

That model is important for us to introduce and analyze the Template attacks better in Section 3.5. It is indicated in [27, Chapter 4] that we need to model a power trace $\mathbf{t}$ in form of *multivariate normal distribution* when we take the correlation between points of that power trace into account. That is because we have the power trace as normally distributed due to electronic noise's normal distribution form. In that model, the correlation between points of the power trace are not taken into account. It is also indicated that, multivariate normal distribution can be described by $\mathbf{C}$ that is a covariance matrix and $\mathbf{m}$ that is a mean vector. In that model we have the density function defined in as Equation 3.4 where $'$ stands for taking the transpose of the matrix.

Let us denote the mean vector depending on d_i, k_j as $\mathbf{m}^{(i,j)}$ and the covariance matrix depending on d_i, k_j as $\mathbf{C}^{(i,j)}$

$$f(x) = \frac{1}{\sqrt{(2 \cdot \pi)^n \cdot \det(\mathbf{C}^{(i,j)})}} \cdot \exp(-\frac{1}{2} \cdot (\mathbf{x} - \mathbf{m}^{(i,j)})'(\mathbf{C}^{(i,j)})^{-1} \cdot (\mathbf{x} - \mathbf{m}^{(i,j)})) \tag{3.4}$$

In Equation 3.4, $\mathbf{C}$ is the matrix of covariance values that is computed by *Cov* function of two points, namely it consists of covariance values $c_{ij} = Cov(X_i, X_j)$. i and j here are the time indexes thus X_i is the point at time i. And $\mathbf{m}$ is the vector of mean values $m_i = E(X_i)$ for each point of the power trace.

[27, Chapter 4] defines covariance as follows:

$$Cov(X,Y) = E((X - E(X)) \cdot (Y - E(Y))) = E(XY) - E(X)E(Y) \tag{3.5}$$

3.3 Simple Power Analysis

Simple power analysis (SPA) is a variant of SCA that is aiming to retrieve the key with only one or very few power consumption traces. SPA was firstly proposed by Kocher et al. in [7].

As we said that SPA works with only one or very few traces, [27, Chapter 5] specifies these two types of SPA as *single-shot* SPA and *miltiple-shot* SPA. In single-shot SPA, we apply the attack single-shot SPA with only one power trace while in multiple-shot SPA we use multiple traces.

[27, Chapter 5] also distinguishes these two types of SPA techniques with a benefit namely, in case of multiple-shot SPA we have multiple traces therefore we can take their mean in order to reduce the amount of noise.

3.4 Differential Power Analysis

Differential power analysis (DPA), firstly introduced by [7], is another type of SCA. That attack requires usage of statistical tools such as correlation measurements. As the name "differential" shows, we focus on relations within the acquired traces. That relation refers to behavioral similarities or differences.

In [27, Chapter 6] it is indicated that one of the benefits of applying DPA is that we do not need a detailed knowledge on the device under the attack. It is mentioned that having the knowledge of cryptographic algorithm that is been used by the device under the attack is usually sufficient when we perform DPA. Another pointed out benefit is that, DPA can still retrieve the key successfully even in case of high level of noise. Moreover, [27, Chapter 6] points several differences between DPA and SPA. One of these differences is that in DPA the data dependent power consumption is focused while in SPA patterns, which is basically about the shape that the trace has, in the trace is focused.

3.5 A powerful SCA: Template Attacks

As our attack is a template attack-variant, we need to introduce and elaborate these kinds of attacks. In this section, we use guidance and the same notation of Mangard et al. work that is [27, Chapter 5].

We start by giving a brief description of template attacks and continue on introducing the two phases of that attack namely; template building and template matching phases.

As we also follow the template building phase, in the template matching phase we use a different method to be introduced in Chapter 4 and showed in Chapter 5. That method is using the classification algorithms instead of computing probability density function in the template matching phase.

3.5.1 A brief description of Template Attacks

We see that we can express a power trace in multivariate distribution form in Section 3.2.2. Mangard et al. [27, Chapter 5] uses that model and notates the pair $(\mathbf{m}, \mathbf{C})$ as *template* which corresponds to the mean vector and the covariance matrix.

As we mentioned before, we follow the guidance and the same notation of work by Mangard et al. [27, Chapter 5] in this entire section.

In template attack we have two phases called template building and template matching phases. In the first phase, we are aiming to characterize the device that we are attacking via running the device with different data d_i and keys k_j and log the power consumption signals/traces.

After all the pairs of (d_i, k_j) are collected, we compute the mean vector and covariance matrix. Ultimately, there will be a template for each of these pairs such that (d_i, k_j) : $\hbar_{d_i,k_j} = (\mathbf{m}, \mathbf{C})_{d_i,k_j}$.

Since we have collected our templates, now we can enter to the second phase; Template Building. In that step, we basically use our templates and one power trace (we can call that trace as *target trace* as well) to compute the probability density function.

One power trace (*target trace*) is denoted as $\mathbf{t}$ and the template is denoted as $\hbar_{d_i,k_j} = (\mathbf{m}, \mathbf{C})_{d_i,k_j}$ and the probability density function is computed as in Equation 3.6 where T refers to the number of points of the power trace.

$$p(\mathbf{t}; (\mathbf{m}, \mathbf{C})_{d_i,k_j}) = \frac{\exp(-\frac{1}{2} \cdot (\mathbf{t} - \mathbf{m}^{(i,j)})' \cdot (\mathbf{C}^{(i,j)})^{-1} \cdot (\mathbf{t} - \mathbf{m}^{(i,j)}))}{\sqrt{(2 \cdot \pi)^T \cdot \det(\mathbf{C}^{(i,j)})}} \tag{3.6}$$

After probability density function (Equation 3.6) is computed for each template, we seek for the highest probability. The highest probability indicates the correct template as well as the correct key.

3.5.2 Template Building

As we mentioned in the brief description on how template attacks work in Section 3.5.1, we are collecting a number of template pairs with different data d_i and keys k_j. We do this in order to characterize the device under the attack.

[27, Chapter 5] it is described that we can build those templates in different ways depending on how much information we know about the device that we are attacking. Moreover, we need to find the *interesting points* that we build our templates from, because these are the points that are holding the most information.

Some of these different ways of building templates are given as the following; templates for pairs of data and key, templates for intermediate values, and templates with power models.

The points that are correlated to the (d_i, k_j) pairs are the interesting points.

Templates for intermediate values is the strategy of building templates from a function $f(d_i, k_j)$. Thus, the interesting points that we use to build the templates from are the ones that are correlated to that function.

Templates with power models is another strategy according to our guidance. That strategy is suitable when we choose a specific cryptographic environment that is using a specific power model like Hamming weight.

According to that, when we say correlated to, we mean that correlated to the instructions that are involving the chosen (d_i, k_j) pairs or function $f(d_i, k_j)$ depending on the strategy that we select.

In our work, we perform that phase by generating a classification model from Chapter 4. That model is generated via template traces which has the aim of characterizing the device under attack.

3.5.3 Template Matching

According to procedure of our template attack, we comply that phase by directly using classification model from Chapter 4 that we generated from template building phase with the template traces.

Moreover, in classical template attacks we can use the following mathematical expression to describe and to have efficiency in computations in this phase. We explain this phase according to [27, Chapter 5] description.

In template matching phase, we are trying to assess the Function 3.6 for a given trace. Then, as we mentioned, the highest probability leads to the correct key.

When one may not want to process the exponentiation operation from Function 3.6, so it is advised to apply logarithm to that function as following.

$$\ln p(\mathbf{t}; (\mathbf{m}^{(i,j)}, \mathbf{C}^{(i,j)})) = -\frac{1}{2}(\ln(2\cdot\pi)^{N_{IP}} \cdot \det(\mathbf{C}^{(i,j)})) + (\mathbf{t}-\mathbf{m}^{(i,j)})' \cdot (\mathbf{C}^{(i,j)})^{-1} \cdot (\mathbf{t}-\mathbf{m}^{(i,j)})) \tag{3.7}$$

Moreover, after we apply logarithm, we need to pick the template that is maximizing the logarithm to retrieve the correct key.

Another computational concern that is expressed is that, in case we want to avoid taking the inverse of the covariance matrix $\mathbf{C}$ from Function 3.7, we can take the covariance matrix as the identity matrix (it is the matrix consisting of the identity element, in our case 1s) which is an approximation method.

In that case, correlation between points of the trace is not considered, hence we call these type of templates,the ones that are only are consisting of the mean vector, as *reduced template*.

If we assign the covariance matrix $\mathbf{C}$ to identity matrix, then we have the Function 3.8. N_{IP} is the number of interesting points.

$$p(\mathbf{t}; \mathbf{m}^{(i,j)}) = \frac{1}{\sqrt{(2\cdot\pi)^{N_{IP}}}} \cdot \exp(-\frac{1}{2}(\mathbf{t}-\mathbf{m}^{(i,j)})' \cdot (\mathbf{t}-\mathbf{m}^{(i,j)})) \tag{3.8}$$

Furthermore, if we again avoid the exponentiation operation, we can use the following function.

$$\ln p(\mathbf{t}; \mathbf{m}^{(i,j)}) = -\frac{1}{2}(\ln(2\cdot\pi)^{N_{IP}} + (\mathbf{t}-\mathbf{m}^{(i,j)})' \cdot (\mathbf{t}-\mathbf{m}^{(i,j)})) \tag{3.9}$$

According to our reference again, [27, Chapter 5], we choose the template with probability giving the lowest absolute value to find the correct key. Moreover the type of templates that we use are the reduced templates from the Function 3.8 and Function 3.9. According to [27, Chapter 5], literature call these methods, that are using reduced templates,*least-square* test.

Chapter 4

Classification (and Clustering) Methods

In this chapter, we introduce three classification methods that we used within our template attack as described in Section 3.5.

As classification is a machine learning discipline, we start by briefly introducing machine learning and then we introduce our chosen classification methods. We picked these methods since these are the three of the most utilized ones in machine learning, especially a lot of research is being done on one of our classification methods, that is SVMs.

We also briefly talk about the clustering concept next to the classification in machine learning in order to stress the way the classification works. In the end, we will indicate the importance of feature selection in classification experiments.

4.1 Machine Learning

In the discipline of machine learning, classification and clustering algorithms play an important role. In order to explain their role more elaborately, we use the example given in [5].

That example from the book by Alpaydın [5], assumes that we wish to decide whether an email is spam or legitimate.

What we do in that case is that, we give many samples of spam emails and legitimate emails to the computer (a computer that is running a machine learning procedure) and try to make the computer produce an algorithm to detect the new inputs as spam or legitimate. Here, we are actually making the machine, the computer, learn from the previous data and make predictions for the new data.

In the next subsections, we use examples from [5] to introduce the classification methods that we used.

4.2 Classification

Another example from [5] assumes that a bank is willing to make predictions when a new application to a credit is made by a customer. Here, a credit is the loan that a customer makes and then to be payed by installments back to the bank.

The bank needs to make sure that this customer will be able to pay the loan back. Therefore, the bank needs to make prediction whether that customer would be able to pay back and the bank could profit. For that reason, the bank calculates the risk level of the customer when he/she applies for a credit.

In order to calculate the risk level, the bank uses information previous credit applications. These previous credit applications consist of customers' financial information such as income, occupation, if the loan was paid and other relevant data.

From that past records, a machine learning algorithm can be extracted and decide whether a new credit application, given the application owner's data, has low-risk or high-risk. In that scenario, we are actually facing a classification application, where a bank needs to classify a customer into two classes namely low-risk and high-risk. That classification is made by the input (the new customer's data) and by the algorithm extracted from the previous data (trained data).

As it can be observed from this example, we have a set of data in the beginning. Then we train this set of data. Training is the process of finding associations or patterns between the data and their label.

For instance, in the example the data is the customer's financial data and the label is low or high risk depending on whether the loan was paid back. After whether that pattern was found between data and the label, we complete the training process and have a rule/algorithm extracted out of it. Here, label stands for the name of the class that the data belongs to.

This rule can be the one from [5], that is given as follows, it is also called "discriminant":

IF income $> \theta_1$ AND savings $> \theta_2$ THEN low-risk ELSE high-risk

Now, we are to classify a new data without known label into one of the defined classes. As it can be seen in here, we are using previous data that we call "feature", when we try to find a pattern. In this example, income and savings are the features.

In conclusion, the classification is the task of assigning objects into classes based on chosen features. As it is indicated in [28], those classes are "mutually exclusive and

exhaustive". It means that the classification of the objects must be done to only one class thus a classification of an object to more than one class should not occur.

There are many classification algorithms that train the recorded data and label the new data in different ways. In our paper, we use three of the most commonly known classification methods and we used Matlab for using these classification methods.

The algorithms that we use are:

1. Naïve Bayes Classification
2. k-Nearest Neighbour Classification
3. Support vector machines (SVMs)

4.2.1 Naïve Bayes Classification

In Naïve Bayes Classification method, we use probability concepts rather than rules/algorithms to make the classifications most likely. To explain that classification algorithm (classifier), we make use of example from [28].

In the given example by Bramer, we are willing to classify a train into one of the classes *on time*, *late*, *very late*, and *cancelled* to find out that this particular train will have most likely one of these events.

We define these events and their probabilities respectively; we make sure that each of these probabilities are between 0 and 1 while the sum of all probabilities are 1 in order to satisfy "mutually exclusive and exhaustive" property.

Besides the labels (class names), we have the features for the classification. Those features are day, season, wind and rain. The *training data set* consists of many samples called *instances*. Two of the instances from the training dataset of [28] can be observed from Table 4.1.

day	season	wind	rain	class
weekday	spring	normal	none	on time
saturday	winter	normal	none	late

TABLE 4.1: Two samples from the Training Data Set

In [28], the definition of Naïve Bayes is made as follows:

We have classes as $c_1, c_2, ..., c_k$ (mutually exclusive and exhaustive). Their prior probabilities are denoted as $P(c_1), P(c_2), ..., P(c_k)$. There are n features $a_1, a_2, ..., a_n$ with values $v_1, v_2, ..., v_n$ respectively. For each class $c_1, c_2, ..., c_k$ we calculate the posterior probability as:

$P(c_i) \times P(a_1 = v_1 \text{ and } a_2 = v_2 ... \text{ and } a_n = v_n|c_i)$

for c_i, $i \in 1, 2, ..., k$.

If we assume that the attributes are independent from each other, that expression is equal to

$P(c_i) \times P(a_1 = v_1|c_i) \times P(a_2 = v_2|c_i) \times ... \times P(a_n = v_n|c_i)$

As we compute this value for each class c_i, and we pick the class index that is giving the maximum value.

4.2.2 k-Nearest Neighbour Classification

k-Nearest Neighbour Classification (k-NN) is a classification method based on using the closest instances to the unlabeled data to be classified. Basically, according to [28], it consists of two steps and forms the following structure;

1. The number of k closest instances (from the training set) to the sample is chosen
2. The majority label (class) for the chosen closest instances will be class for the unlabeled data

The distance metric plays an important role as we determine the closest instance according to chosen distance metric. In k-NN, we can utilize Euclidean distance or Manhattan Distance for example.

Manhattan distance between two points A with coordinates (x_A, y_A) and B with coordinates (x_B, y_B) is defined as $|x_A - x_B| + |y_A - y_B|$.

Figure 4.1 from [4] illustrates k-NN as the value k changes. In that example we have two classes namely A and B. ? is the unlabeled data that we apply k-NN to classify it.

In the leftmost situation from Figure 4.1, $k = 1$ we look for the closest labeled data point. Since A is the closest one to ?, we classify ? as A.

For the situation that has been shown in the middle from Figure 4.1, we have $k = 2$ case. We see that both A and B are the two closest labeled data points. Thus, we have A, B in our set of closest data point labels for the given k value. In that case, we are not able to classify ? since we have the same number of elements from those different classes.

In the rightmost situation of Figure 4.1, the closest data point labels for the given k value is A, B, A, hence, ? is classified as A this time.

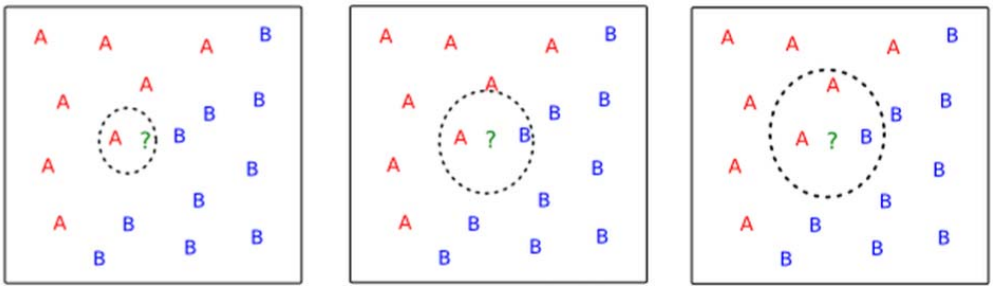

FIGURE 4.1: k-NN classification for $k = 1$, $k = 2$ and $k = 3$ [4]

4.2.3 Support vector machines (SVMs)

The support vector machines (SVMs) is defined as a linear classification method. Linear classification is applied to the data set such that instances of a class are linearly separable from instances of other classes as [5] explains.

To explain what do we mean with linearly separable classes, we can use Figure 4.2 from [5]. As Alpaydın [5] explains, it can be observed that there is a hyperplane H_i for each class C_i where data belongs to C_i locates on the positive side of H_i while other data belonging to other classes stay in the negative side of H_i.

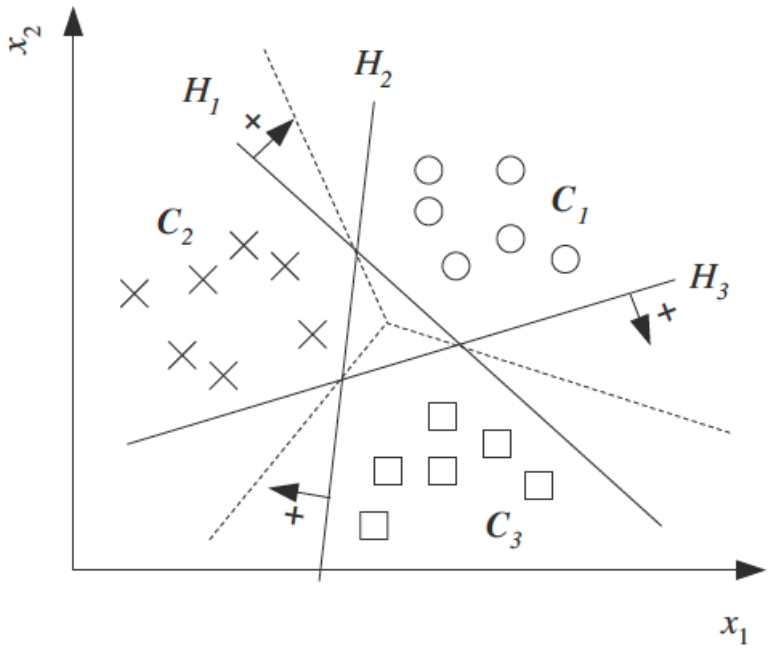

FIGURE 4.2: Sample linearly separable classes [5]

Before we introduce SVMs, we need to take a look at *hyperlanes*.

Definition 4.1. *Hyperplane* For scalar $w_1, w_2, ..., w_n$ with property that not all these scalars equal to 0, we have vectors in $\mathbb{R}^n$, in the form

$$\mathbf{X} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \tag{4.1}$$

where $w_1x_1 + w_2x_2 + ... + w_nx_n = c$.

We call the set $\mathcal{S}$ a hyperplane and denoted as $\mathcal{S} = \{\overrightarrow{x} : \overrightarrow{w} \cdot \overrightarrow{x} = c\}$. In other words, [29] indicates that it is the vector subspace with codimension-1 of a vector space.

According to [6] SVMs technique consists of several aspects.

The first aspect that we talk about is the class separation, which is the main target of SVM classification. We have two classes in the beginning, that is why we have the binary classification term.

Our objective is to find an optimal hyper-plane that is separating the two classes, while leaving the maximum *margin* between two classes' closest points according to [6]. We call these closest points from different classes *support vectors* while the optimal hyperplane locates in the middle of the margin.

In order to express that description better we demonstrate it using Figure 4.3.

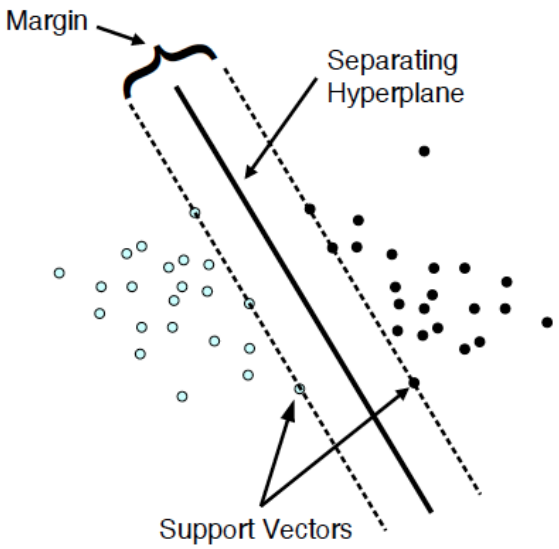

FIGURE 4.3: SVM classification from [6] in linear separable case

Nonlinearity is another property of SVM according to [6], it is a very important one because it says that in case that we are not able to have a separator that is linear, then SVM tries to find another dimensional space where we can find a linear separating hyperplane via projecting the data points into that new dimensional space.

It is also mentioned that, in order to process this projection of the data points into a new dimensional space, a type of method called *kernel techniques* is used.

Moreover, we have another task called problem solution given by [6]. It basically says that the entire function of SVM can be considered as an optimization problem that is quadratic.

That optimization is done to maximize the margin [30]. If we recall the margin, it is the distance between closest points of the classes to the separating hyperplane.

In other words, margin is the distance between the support vectors. Moreover maximizing the margin problem is defined as an optimization problem where that optimization problem is given as in 4.3, [30].

$$u = \vec{w} \cdot \vec{x} - b \tag{4.2}$$

x is the input vector and w is defined as the normal vector to the hyperplane. We have the SVM output in linear setting is as in Eq. (4.2), where the margin is $\frac{1}{||\vec{w}||_2}$ [30].

$$\underset{\vec{w},b}{min}||\vec{w}|| \text{ subject to } y_i(\vec{w} \cdot \vec{x_i} - b) \geq 1, \forall i, \tag{4.3}$$

x_i from 4.3 corresponds to the input training point with index i and y_i (that are either +1 or -1) is the corresponding output of SVM [30].

Furthermore, as SVMs are being used, new features are being added such as multi-class classification. As we mentioned before, SVMs are invented and utilized as a binary classification method in the beginning.

4.3 Clustering

Clustering can be seen as classification without class labels. Here the aim is to keep similar objects together in the same cluster while dissimilar objects are in different clusters, as [28] indicates.

As we have in classification, there are many clustering algorithms as well. We briefly introduce one of the most commonly used clustering algorithms called k-means clustering.

4.3.1 k-means Clustering

In k-means clustering, we first assign a value for k which stands for the total number of clusters that we are willing to have in the end. Then, the algorithm picks k random instances from the data set and sets them as the temporal centroids of the clusters.

The instances are clustered based on their closest centroids, that is the end of first run of the loop. Now a new assignment of centroids is made and we process the same steps in each loop until we reach a point where we do not need to change the centorids (the measurements stay stable).

4.4 Feature Selection for Classifying High Dimensional Data

The features of the instances are given in the examples in previous sections. For instance in Section 4.2 we have the features income and savings for the instances (clients of the bank). They have an important role in classification, as we use them to decide similarities and therefore detect patterns.

Another benefit of working with features is that, when we have good quality of features out of other features, we can run our classification based on these better qualified ones. That gives us twofold advantage namely, we have a better classification in the end because bad quality in features may lead us misclassifications; secondly we get rid of computation overhead if our data set consists of many features.

As we mention in our future work, in case of having large data set for our templates we can use that method to facilitate and qualify the classification within our template attack.

Chapter 5

Our Attack

In this chapter, we demonstrate how we applied our attack both in theory and in practice. We use template attack from Section 3.5 with classification methods we introduce in Chapter 4.

For our analysis, we use traces that were collected with the setup of [31]. The device under attack is an STM34F4 micro-controller with the ARM Cortex-M4 processor with PolarSSL (Version 1.3.7, https://tls.mbed.org/) assembly code and Double-and-Add Always implementation as in Algorithm 2. The template traces were collected with a 54855 Infiniium Agilent oscilloscope and a Langer EMV-TECHNIK RF-U5-2 near field probe. Our attack is performed on the Brainpool curve BP256r1 by [32]. The way we get measurements is that we captured 10^9 sample per second. And we have EM consumption measurements.

We make the same assumptions as [31] such that, the attacker has the information on input P to the device under attack, has the information on the scalar multiplication implementation that is used and is able to choose the input.

Unlikely in classical template attacks from Section 3.5, the attacker does not need to know the keys used in the experiment device since we are attacking bit by bit to the key with similar approach to [1].

Moreover, as [31] indicates, as the attacker can choose the input point our attack also can be applied for the Diffie-Hellman key exchange protocol and if the attacker knows the input to the device under attack then our attack also can be applied to ECDSA.

Furthermore, similarly to [31], we have the input points in affine form and it is converted to Jacobian coordinates (see Section 2.2.2).

Our data set consists of two types of power traces namely, templates and target. We have the template power traces, we refer to those traces as training data in machine

learning terminology as we mention in Section 4.2. While the target trace is the power trace that we gather from the actual device under attack.

For the classification methods we used a computer running a 64-bit Operating System Windows 7 with processor Intel i5-4590 CPU @ 3.30GHz. The software implementations are done in the Matlab (version R2014a) environment.

The main reason that we chose classification algorithms to perform our side-channel analysis is the fact that their structure fit perfectly to the template attack characteristics. That is because, as we have two phases namely template building and template matching, as we introduce in Chapter 3; we first train the data set via creating a model with our classification methods and then we use that method to classify the new unlabeled trace.

We elaborate the attack in the next sections.

5.1 Theory of the Attack

The general steps of our template attack, similarly to [1] , can be described as below:

1. The attacker gives the input Point P to the actual device under attack and gets one power trace for the elliptic curve scalar multiplication run with this input. That received power consumption trace is called the target trace.
2. The attacker gives the input of multiples of point P to the experiment device and gets the respected power trace for the elliptic curve scalar multiplication run with these inputs. We call the power traces that we captured for each of the multiples of P as template trace. In practice, we have the power traces for the $2P$ and $3P$ (160 template trace per each).
3. In the end, the attacker has only one target trace from input point P and 160 template trace from inputs $2P$ and $3P$. Now, the attacker tries to find out the second most significant bit value of the key k, by classifying the first multiplication of the second iteration occurring on target trace using the first multiplication of the first iteration of the template traces.

Figure 5.1 illustrates these steps above.

The inputs P, $2P$ and $3P$ are shown in Figure 5.1. The red rectangles represents the multiplication patterns that we focus. As in [31], we look for the 19th multiplication pattern in the target trace P and the first multiplication pattern in the template traces. That is because, in the Brainpool curves, the first iteration of the Double-and-Add Always algorithm consists of 18 multiplications. When P passes the first iteration (in the first multiplication of the second iteration of the Double-and-Add Always), it becomes

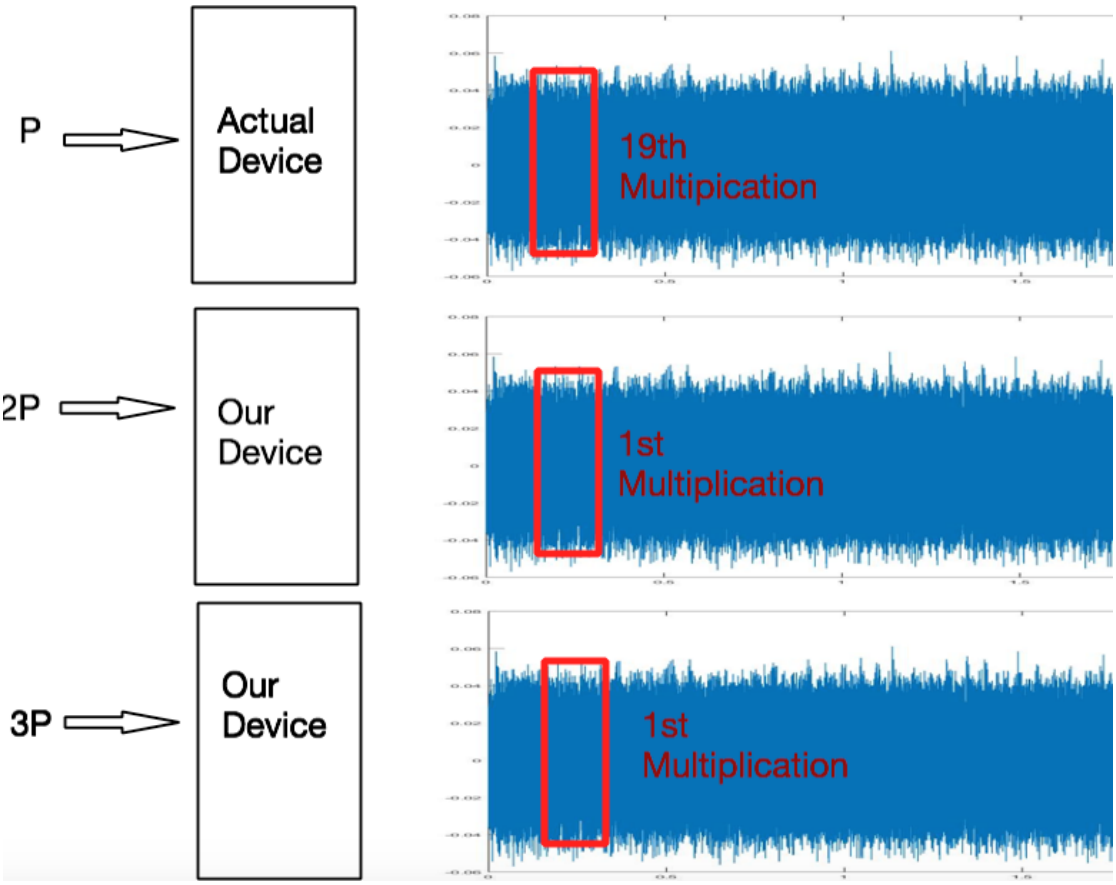

FIGURE 5.1: Sample illustration of our attack. These points and the traces are only for illustration.

either $2P$ or $3P$ at the 19th multiplication (the first multiplication pattern of the second iteration), while the traces with input $2P$ and $3P$ have the same values until the end of the first multiplication iteration.

As we apply a template attack we follow the two phases that we describe in Section 3.5.2 and Section 3.5.3. Our templates consist of $2P$ and $3P$ template traces produced from our experiment that is assumed to be identical to the device under attack (in the theory of our attack we assume to have all multiples (like $2P$, $3P$, $4P$, $5P$) of P in our template traces set, however due to file formats, we were able to apply our attack with $2P$ and $3P$ trace templates) while the target trace is the power consumption when we insert P as input data on the device under attack.

Figure 5.2 illustrates our target trace that we obtain by giving input point P into the device under attack.

The training data (templates) is to be modeled, before the classification, in the *template building* phase because we need to characterize the device under attack. Then we can classify the new trace with that model.

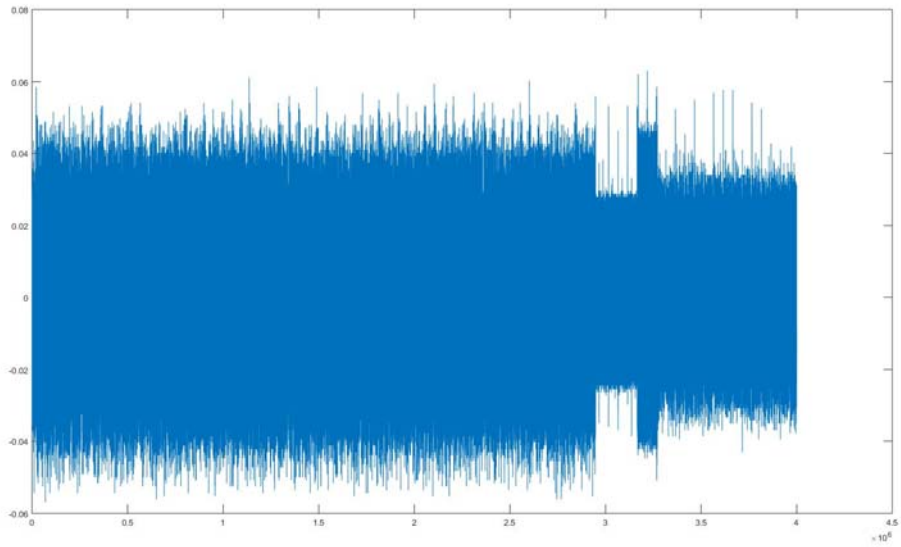

FIGURE 5.2: Our target trace P

That process is completed via chosen classification method specific Matlab functions such as `fitcsvm` for SVMs, `fitcknn` for k-Nearest Neighbour or `fitcnb` for Naïve Bayes Classification. Those functions return a model that is trained via the template traces (training data) which is called as *template building*. That is because, that model reflects the characteristics of the device under attack.

Besides our data set, we have selected mostly used classification methods that we indicated Chapter 4; SVMs, k-Nearest Neighbour and Naïve Bayes Classification. These methods are used for the *template matching* phase.

In the beginning of our attack, we need to split out target trace (we also call it target trace P), that we acquire by giving point P as input to the device under attack, into chunks where Double-and-Add Always algorithm from Section 2.3.1 operates an iteration in each of these chunks. Therefore, we first need to find those chunks. In order to do that, we need to find the multiplication patterns in the template traces as in work [31], where each pattern found is considered as a chunk. [31] suggests to cross-correlate the first found multiplication pattern from the template trace with the entire target trace P. Cross-correlating the multiplication pattern with trace P helps us to investigate the particular parts in this trace where a multiplication occurs because of Double-and-Add Algorithm iterations.

According to [31], PolarSSL provides an integer multiplication as in Algorithm 4.

When Algorithm 4 is running, it forms several patterns. An interesting one for us is the multiplication pattern, shown in Figure 5.3. It indicates the 32-bit multiplication (one for each peak in total 8 peaks, since A and B are 256-bits) during the execution of the algorithm. We can spot this pattern by running our experiment device with

Algorithm 4 PolarSSL Integer Multiplication [31]

Input: A, and $B_7...B_0$ of 256-bits long
Output: $X = A \times B$

1: $X \leftarrow 0$
2: **for** $i \leftarrow 7$ **down to** 0 **do**
3: $\quad (C, X_{i+7}, X_{i+6}, ..., X_i) \leftarrow (X_{i+7}, ..., X_i) + A \times B_i$
4: $\quad j \leftarrow i + 8$
5: $\quad$ **repeat**
6: $\quad\quad (C, X_j) \leftarrow X_j + C$
7: $\quad\quad j \leftarrow j + 1$
8: $\quad$ **until** $C \neq 0$
9: **end for**
10: **return** X

the cryptographic operation activated or not. When the scalar multiplication is not activated, we should not be able to see this 8-peak multiplication pattern. As a result, if we have the first multiplication pattern found on the template trace correctly, then we can cross-correlate it with the target trace P in order to find the multiplication patterns in this trace as well.

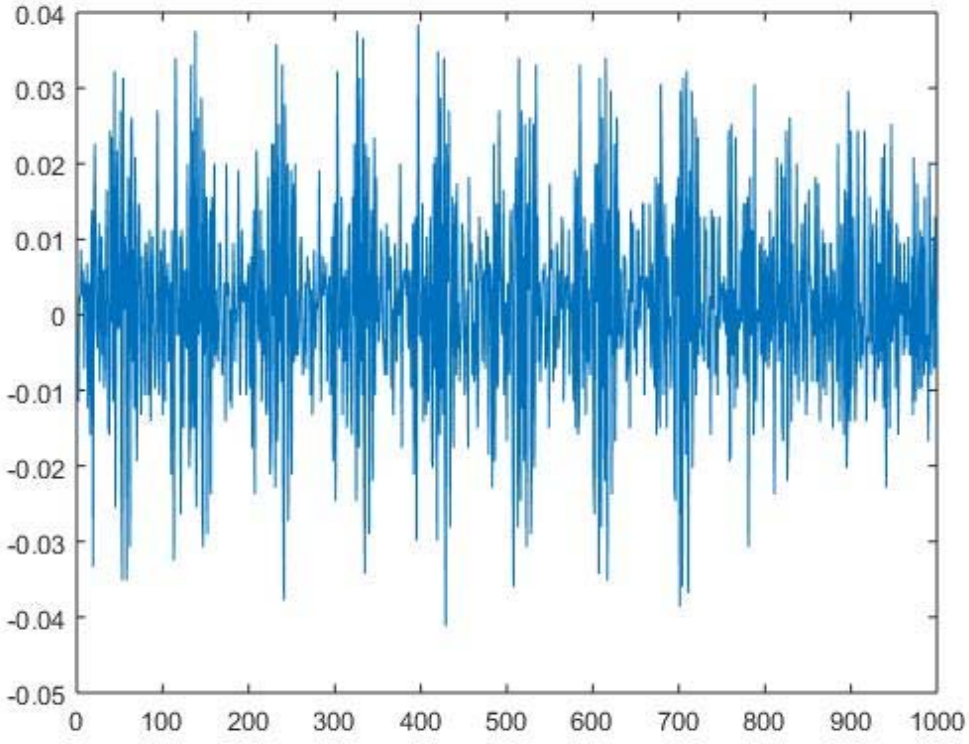

FIGURE 5.3: The first and typical Multiplication Pattern

Now, we have a target trace by giving point P as input to the device under attack as in Figure 5.2 and a multiplication pattern as in Figure 5.3. Before we start to follow steps from Section 3.5, we do the following.

As we mentioned before, we follow the procedure suggested by [31] which is cross-correlating the target power trace with the multiplication pattern in order to find the specific parts of the target trace where the multiplication occurs.

The setup of [31] provides the traces with their interesting parts e.g. the first multiplication pattern of the second iteration for the target trace (the 19th multiplication pattern) and the first multiplication of the first iteration for the templates for computation efficiency.

Therefore, when we cross-correlate the target trace with the multiplication pattern, the first high correlated part of the trace is the 19th multiplication pattern.

Figure 5.4 demonstrates where we have the high-correlated parts between the multiplication pattern and the target trace P. In Figure 5.4 we indicate the coordinates of the 19th multiplication pattern (in figure it is the first one because only the interesting parts are chosen) that we found via counting.

It can be seen in Figure 5.4 the first correlated pattern is the one that we already computed with the x-coordinate starting at $2.236e4$ and 1000 sample long (1 microsecond long), i.e. $2.236e4 + (1 : 1000)$.

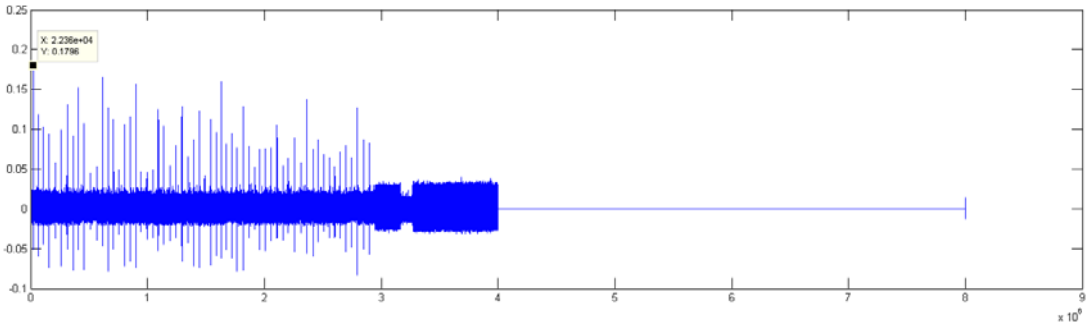

FIGURE 5.4: The correlated patterns and the first pattern (the 19th) is chosen

As we find all high-correlated parts of the target trace P, we select those chunks as these are the ones where the multiplication occurs at the interesting parts of the target trace P for a typical Brainpool curve. We can observe from the Double-and-Add Always algorithm that when we give the input P, and a scalar l- bit long, we then have k number of iterations.

In the first iteration, at the second most significant bit of k, lets denote it as k_{i-2}, with input P we have $2P$ when $k_{i-2} = 0$ or $3P$ when $k_{i-2} = 1$.

As a result we can observe that, in the first multiplication pattern we are at the bit k_{i-2} such that as we give P as input we can either have $2P$ or $3P$ as output.

As we iterate through all multiplication patterns, we have binary choices for each found correlation. Therefore, we can use our templates for those outputs. For the templates that we use for the classification methods, we select the first multiplication pattern of

the first iteration of the template traces always for the second most significant bit of the key.

What we do in our attack is that, since we have templates for $2P$ (160 templates) and $3P$ (160 templates), we used classification methods with those templates and the end of the first multiplication pattern of these templates in order to investigate which point is computed in the end of 19th multiplication of Double-and-Add Always Algorithm.

It is elaborated in the following section that, we check different intervals as well as different number of traces for the beginning of the template traces $2P$ and $3P$ until the first multiplication pattern. And we use the three classification methods as well as we try two different values for $k-$ Nearest Neighbour method.

That described procedure should be applied for each found multiplication pattern in order to retrieve the entire key scalar k. That is, as we move to the next iteration (multiplication pattern) we need to use the related template traces and the end of the current pattern then apply the classification methods.

5.2 Application of the Attack using Matlab

In this section we show how we applied our attack with our pieces of Matlab code.

As we mentioned in Section 5.1, the first multiplication pattern is computed via counting the peaks in the target trace P with respect to multiplication in Algorithm 4 by PolarSSL.

As the traces are cut in their interesting parts for each iteration, its starting coordinate (the 19th multiplication's) is $2.236e4$ for 1000 samples, we use the Matlab function `xcorr` on that pattern and the target trace P. That function cross-correlates the captured multiplication pattern with the entire target trace P.

As a result we can see where multiplication occurs on the target trace for other interesting points e.g. 40th pattern for the next bit of the key, and we use these specific parts for classification.

Our data consists of training data and sample. Training data refers to our templates ($2P$ and $3P$) while the sample is our target trace P.

Thus, we first create our sample data variable to be classified and assign it to the value of the first occurrence of the multiplication pattern (the value can be seen from Figure 5.4).

```
1 %The target point (sample data)%
2 sample = traceP{1,1}(2.236e4+(1:1000));
```

Later, we cross-correlate the multiplication pattern with the templates in order to find the parts where multiplication occurs. The values for the template trace $2P$ can be seen from Figure 5.5 and for $3P$ can be seen from Figure 5.6.

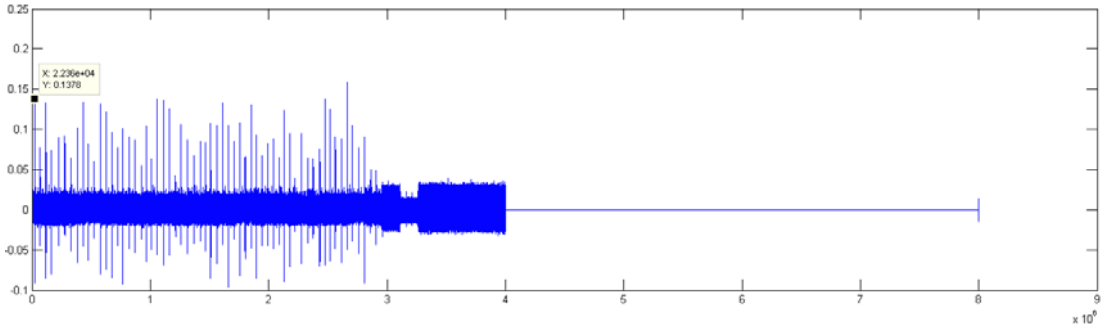

FIGURE 5.5: The correlated patterns and the first pattern is chosen for template trace $2P$

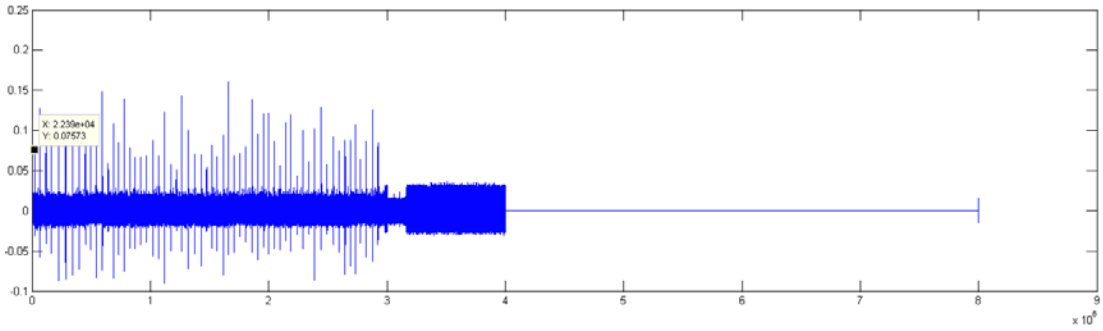

FIGURE 5.6: The correlated patterns and the first pattern is chosen for template trace $3P$

Then we pick these intervals for the templates as follows.

```
%Collecting the training data%
for k = 1 : 10
       for l= 1: 16
           k
           l
           tr2=[tr2; trace2P{k,l}((2.236e+04)+(1:1000))'];
           tr3=[tr3; trace3P{k,l}((2.239e+04)+(1:1000))'];
       end
end
```

In that for-loop, we have 160 template traces for $2P$ and 160 template traces for $3P$. As we mentioned in the beginning of this chapter, we need to consider the first multiplication pattern of the first iteration for the template traces $2P$ and $3P$ while we take the first multiplication pattern of the second iteration, i.e. the 19th multiplication pattern, for the target trace P.

Moreover, we also made our experiments with different number of template traces such as 20, 40, 80, 160, 320 number of traces in total where each of them has the same number of template traces for $2P$ and $3P$.

The template building is characterizing the device under attack via the template traces. In our case, we do that process via training the template traces and retrieve a model out of that process. The training data set is the set of template traces $2P$ and $3P$. This is done as following.

```
%k-Nearest Neighbor Classifier%
    MdlknnX = fitcknn(training,group,'NumNeighbors',4); %k=4
    Mdlk = fitcknn(training,group); %k=1

%Naive Bayes Classifier%
    Mdl = fitNaiveBayes(training,group);

%SVM Classifier%
    SVMModel = fitcsvm(training,group);
```

The variable "group" refers the set of labels of the classes which is "$2P$" and "$3P$". The model creating function depending on the selected classification method takes the training data set (template traces $2P$ and $3P$) and the labels "$2P$" and "$3P$" then reflects the characteristics of the data classification.

Later, we use those models in order to be able to classify an unlabeled data, in our case it is the chosen multiplication pattern. Again, " ' " refers to transpose in that sample illustration.

```
%k-Nearest Neighbor Classifier%
[class_kNN, score_kNN]=predict(knnMdl,sample');
[class_kNNX,score_kNNX]=predict(MdlknnX,sample');

%Naive Bayes Classifier%
class_NB=predict(nbMdl,sample');
score_postNB=posterior(Mdl,sample');

%SVM Classifier%
[class_SVM, score_svm]=predict(SVMModel,sample');
```

That piece of code, uses the Matlab function `predict` which takes the created model and the unlabeled data and returns the class that data belongs to and the *score*.

For SVM classification, score is defined as the distance from the separating hyperplane (denoted as decision boundary as well) while for Naïve Bayes Classification and for k-Nearest Neighbour, score is the posterior probability which Matlab indicates.

For k-Nearest Neighbour functions, `fitcknn(training,group)` trains the training data using $k = 1$ number of neighbours while `fitcknn(training,group,"NumNeighbors",4)` trains the data with $k = 150$ number of neighbours. The default distance metric is Euclidean distance.

Moreover, in the `predict` function used for k-Nearest Neighbour functions follows the same steps from Section 4.2.2. It first looks at the k number of closest neighbours to the given unlabeled data and checks the labels of these closest neighbours. The label that is occurring the most (in other words the highest posterior probability) is given as the class that this new data belongs to.

For Naïve Bayes classification, we used the function `fitNaiveBayes(training,group)`. We need to note that this is the function that is compatible with the Matlab version that we used.

We need to note that, `fitcsvm(training,group)` trains the training data with given labels according to the SVM method in Section 4.2.3. The call of this function with these inputs, solves the defined optimization problem from Section 4.2.3 with a method called Sequential Minimal Optimization (SMO) [30].

5.3 Results and Analysis

As we were able to apply our attack for the second most significant bit of the key scalar k because of having templates for $2P$ and $3P$, we run all three classification algorithms (SVM, knn (for $k = 1$ and $k = 4$) and nb) on the first multiplication pattern.

The key that we use in the experiment has the second most significant bit as 0, therefore the expected class that the classification methods should return is $2P$.

We indicate in previous section that we run our classification according to the intervals that we found via cross-correlating the multiplication pattern with the traces. We also try different number of template traces such as 20, 40, 80, 160, 320 number of traces in total.

Moreover, as we indicated and showed in Figure 4.1 in Section 4.2.2 the k-Nearest Neighbour classification may lead to misclassification depending on the chosen number of neighbours. Therefore, we used that method with two different k values such that; $k = 1$ and $k = 4$. The reason of choosing $k = 1$ is that it is one of the interesting option and the default value in case the number of neighbors is not mentioned on the function `fitcknn`.

Moreover, as we indicate before, we use the score values for the classification methods so that we can see the posterior probability (or the distance from the hyperplane for

# Templates	kNN ($k = 1$)	kNN ($k = 4$)	Naïve Bayes	SVM
320	2 [1, 0]	2 [1, 0]	2 [1, 0]	2 [1.124750e+00, -1.124750e+00]
160	2 [1, 0]	2 [1, 0]	2 [1, 0]	2 [1.084593e+00, -1.084593e+00]
80	2 [1, 0]	2 [1, 0]	2 [1, 0]	2 [1.040720e+00, -1.040720e+00]
40	2 [1, 0]	2 [1, 0]	2 [1, 0]	2 [6.758751e-01, -6.758751e-01]
20	2 [1, 0]	2 [1, 0]	2 [1, 0]	2 [5.546452e-01, -5.546452e-01]

TABLE 5.1: Results from different number of templates

SVM) given to the sample with different training data by the same type of classification method.

The reason that we log the scores is that, score values help us to make comparisons of the classification of different data when we use the same classification method.

Table 5.1 shows the classification results when we pick the 2.236e+04 for target trace P, 2.236e+04 for template trace $2P$ and 2.239e+04 for template trace $3P$ for 1000 sample long. Moreover, output format of this table is "a [b, c]" where a is the classification output (class that the chosen interval of the target trace belongs to); [b, c] is the score value for the given classification method.

Another observation could be that as we have the less number of templates, the score value for the SVM drops which means that the sample becomes closer to the separating hyperplane i.e. we have better performance when we have more number of templates.

In order to check if our code is performing correctly, we also changed the order of creating the arrays e.g. making the class labels $(3, 2)$ instead of $(2, 3)$ and training data consisting of first $3P$ and then $2P$. According to the results, we received exactly the same classification results.

On the other hand, we could use feature extraction in our implementation if we would not have that efficiency, we introduce the feature extraction in Section 4.4.

Our attack in theory and in application uses binary classification since in the end of each iteration the output trace can be assigned to one of two possible classes.

According to the results for each of the classification methods with the function `predict`, we successfully retrieved the bit correctly as experiments returned us the class label '$2P$'.

Chapter 6

Conclusion and Future Work

We designed and applied a template attack on Double-and-Add Always algorithm. In our attack, we made use of three well-known classification methods called Support Vector Machines, k-Nearest Neighbour and Naïve Bayes Classification.

As we apply a template attack, we had two main steps in our attack called template building and template matching. For both of these steps, we used classification techniques such that in template building phase we generated a model with our templates (called training data) in order to make classification while in template matching step we directly apply the classification methods by using the generated model and the unlabeled data.

In order to create a model, we used specific Matlab functions depending on the chosen classification methods, e.g. `fitcsvm` for SVM classification. These functions uses the training data which is our template traces and create a model. That model can be considered as the characterization of the device under attack since it helps to classify the unlabeled data.

After generating the model, the characteristics of the device under attack, we classify the new data which corresponds to template matching phase.

Before apply the phases of template attacks we determine the multiplication patterns in the target phase so that we know where the multiplication occurs.

We applied these three classification methods with our template traces $2P$ and $3P$. In order to see the k-nn classification results depends on the chosen number of neighbours, we also used k-nn with $k = 1$. And we classified the first multiplication pattern in our target trace P with the generated models.

According to the classification results that we acquired from the chosen intervals from cross-correlating, we successfully retrieved the bit value correctly and captured the bit

value as 0 since the classified pattern was labeled as $2P$ by all the classification methods that we used.

Also results show that, as we have more templates we have more success in classification for SVM.

Classification methods using Matlab is convenient for template attacks because the implementation progress consists of two steps namely; model generation and predicting the class of the new data. As we know from Chapter 3, template attacks consists of two phases as well that are respectively correspond to these two steps.

A way to protect a device from our attack is point blinding similar to [31]. The information of input is necessary for our template generation, therefore point blinding and randomization of the projective coordinates can be proposed as countermeasures.

According to explanation by [14], point blinding is the technique where we operate $k \times (P + R)$ for some random point R that we know $k \times R = S$, instead of computing $k \times P = Q$. We can see that, $k \times (P + R) = k \times P + k \times R = Q + S$. Then, we subtract S from $k \times (P + R)$ and we get Q as a result. That countermeasure, makes the attack described in [14] unfeasible since this attack tries to capture k by computing Q_i for all possible P_i where $Q_i = k \times P_i$.

Randomization of the projective coordinates, briefly, uses a random value λ and multiplies the projective coordinates X, Y, Z, from Section 2.2.1, with this value depending on the preference of the implementation e.g. randomization after each specific operation. Therefore, it will be hard for an attacker to track a point since its coordinate values are not kept the same.

As a future work, our attack can be applied to Montgomery Ladder (In Section 2.3.2 we introduce this algorithm) as that algorithm forms a similar structure with Double-and-Add Always algorithm and which also could be applied to the hardware implementation of these algorithms.

Moreover, in case of having more templates and/or high dimensional data, we can make use of feature extraction in order to have more accurate and efficient classification.

Appendix A

Matlab Code

```
1
2  %Addressing the directories of the trace folders is done%
3
4  %Assigning time and trace values is done%
5
6    %Collecting the training data%
7  for k = 1 : 10
8        for l= 1: 16
9            k
10           l
11           tr2=[tr2; trace2P{k,l}((2.236e+04)+(1:1000))'];
12           tr3=[tr3; trace3P{k,l}((2.239e+04)+(1:1000))'];
13       end
14 end
15        training=[tr2;tr3];
16
17 %Collecting the relevant labels (groups)%
18        group=[];
19 for i = 1:160
20     group=[group; 2];
21 end
22 for i = 1:160
23     group=[group; 3];
24 end
25
26 %The target point (sample data)%
27 sample = traceP{1,1}((2.236e+04)+(1:1000));
28
29 %KNN with k=4
30  MdlknnX = fitcknn(training,group,'NumNeighbors',4);
31 [class_kNNX, score_kNNX]=predict(MdlknnX,sample');
32
33 %KNN with k=1
34     Mdlk = fitcknn(training,group);
```

```
35 [class_kNN, score_kNN]=predict(Mdlk,sample');
36
37 %Naive Bayes
38     Mdl = fitNaiveBayes(training,group);
39 class_NB=predict(Mdl,sample');
40 score_postNB=posterior(Mdl,sample');
41
42 %SVM
43     SVMModel = fitcsvm(training,group);
44 [class_SVM,score3]=predict(SVMModel, sample');
```

Bibliography

[1] Lejla Batina, Lukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, and Michael Tunstall. Online template attacks. In *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings*, pages 21–36, 2014. doi: 10.1007/978-3-319-13039-2_2. URL http://dx.doi.org/10.1007/978-3-319-13039-2_2.

[2] C. Paar and J. Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer-Verlag New York Inc, 2010.

[3] Henri Cohen, Atsuko Miyaji, and Takatoshi Ono. Efficient elliptic curve exponentiation using mixed coordinates. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology — ASIACRYPT'98*, volume 1514 of *Lecture Notes in Computer Science*, pages 51–65. Springer Berlin Heidelberg, 1998. ISBN 978-3-540-65109-3. doi: 10.1007/3-540-49649-1_6. URL http://dx.doi.org/10.1007/3-540-49649-1_6.

[4] Classification. http://trevorwhitney.com/data_mining/classification.

[5] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010. ISBN 026201243X, 9780262012430.

[6] David Meyer and Technische Universität Wien. Support vector machines. the interface to libsvm in package e1071, 2001.

[7] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 388–397, London, UK, UK, 1999. Springer-Verlag. ISBN 3-540-66347-9. URL http://dl.acm.org/citation.cfm?id=646764.703989.

[8] Dag Arne Osvik, Adi Shamir, and Eran Tromer. *c*ache *a*ttacks and *c*ountermeasures: *t*he *c*ase of *aes*. In *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*, CT-RSA'06, pages 1–20, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-31033-9, 978-3-540-31033-4. doi: 10.1007/11605805_1. URL http://dx.doi.org/10.1007/11605805_1.

[9] Kai Schramm, Gregor Leander, Patrick Felke, and Christof Paar. A collision-attack on AES: combining side channel- and differential-attack. In *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, pages 163–175, 2004. doi: 10.1007/978-3-540-28632-5_12. URL http://dx.doi.org/10.1007/978-3-540-28632-5_12.

[10] Kai Schramm, Thomas J. Wollinger, and Christof Paar. A New Class of Collision Attacks and Its Application to DES. In Thomas Johansson, editor, *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 206–222. Springer, 2003. ISBN 3-540-20449-0. URL http://dblp.uni-trier.de/db/conf/fse/fse2003.html#SchrammWP03.

[11] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, pages 104–113, London, UK, UK, 1996. Springer-Verlag. ISBN 3-540-61512-1. URL http://dl.acm.org/citation.cfm?id=646761.706156.

[12] Olivier Billet and Marc Joye. The Jacobi Model of an Elliptic Curve and Side-Channel Analysis. In Marc P. C. Fossorier, Tom Høholdt, and Alain Poli, editors, *AAECC*, Lecture Notes in Computer Science, pages 34–42. Springer, 2003. ISBN 3-540-40111-3. URL http://dblp.uni-trier.de/db/conf/aaecc/aaecc2003.html#BilletJ03.

[13] Eric Brier, Marc Joye, and To Erik De Win. Weierstraß elliptic curves and side-channel attacks. In *Public Key Cryptography – PKC 2002, volume 2274 of LNCS*, pages 335–345. Springer–Verlag, 2002.

[14] Jean-Sébastien Coron. Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems. In *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '99, pages 292–302, London, UK, UK, 1999. Springer-Verlag. ISBN 3-540-66646-X. URL http://dl.acm.org/citation.cfm?id=648252.752381.

[15] Pierre-Yvan Liardet and Nigel P. Smart. Preventing SPA/DPA in ECC Systems Using the Jacobi Form. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES*, number Generators in Lecture Notes in Computer Science, pages 391–401. Springer, 2001. ISBN 3-540-42521-7. URL http://dblp.uni-trier.de/db/conf/ches/ches2001.html#LiardetS01.

[16] Lejla Batina, Benedikt Gierlichs, and Kerstin Lemke-Rust. Differential Cluster Analysis. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware*

and Embedded Systems - CHES 2009, volume 5747 of *Lecture Notes in Computer Science*, pages 112–127. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04137-2. doi: 10.1007/978-3-642-04138-9_9. URL http://dx.doi.org/10.1007/978-3-642-04138-9_9.

[17] Lejla Batina, Jip Hogenboom, and Jasper G. J. van Woudenberg. Getting More from PCA: First Results of Using Principal Component Analysis for Extensive Power Analysis. In *Proceedings of the 12th Conference on Topics in Cryptology*, CT-RSA'12, pages 383–397, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-27953-9. doi: 10.1007/978-3-642-27954-6_24. URL http://dx.doi.org/10.1007/978-3-642-27954-6_24.

[18] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004. ISBN 3-540-22666-4. URL http://dblp.uni-trier.de/db/conf/ches/ches2004.html#BrierCO04.

[19] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual Information Analysis - A Generic Side-Channel Distinguisher. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442, Washington DC,US, 2008. Springer-Verlag.

[20] Marcel Medwed and Elisabeth Oswald. Template Attacks on ECDSA. In Kyo-Il Chung, Kiwook Sohn, and Moti Yung, editors, *Information Security Applications*, volume 5379 of *Lecture Notes in Computer Science*, pages 14–27. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-00305-9. doi: 10.1007/978-3-642-00306-6_2. URL http://dx.doi.org/10.1007/978-3-642-00306-6_2.

[21] Colin D. Walter. Sliding Windows Succumbs to Big Mac Attack. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES*, volume 2162 of *Lecture Notes in Computer Science*, pages 286–299. Springer, 2001. ISBN 3-540-42521-7. URL http://dblp.uni-trier.de/db/conf/ches/ches2001.html#Walter01.

[22] Johann Heyszl, Andreas Ibing, Stefan Mangard, Fabrizio De Santis, and Georg Sigl. Clustering Algorithms for Non-profiled Single-Execution Attacks on Exponentiations. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications*, volume 8419 of *Lecture Notes in Computer Science*, pages 79–93. Springer International Publishing, 2013. ISBN 978-3-319-08301-8. doi: 10.1007/978-3-319-08302-5_6. URL http://dx.doi.org/10.1007/978-3-319-08302-5_6.

[23] Peter Karsmakers, Benedikt Gierlichs, Kristiaan Pelckmans, Katrien De Cock, Johan Suykens, Bart Preneel, and Bart De Moor. Side channel attacks on cryptographic devices as a classification problem. Technical report, COSIC technical report, 2009.

[24] Marc Joye. Elliptic curves and side-channel analysis. In *ST Journal of System Research*, volume 4, pages 17–21. ST Journal of System Research, 2003.

[25] Aurélie Bauer, Éliane Jaulmes, Emmanuel Prouff, and Justine Wild. Horizontal and Vertical Side-Channel Attacks against Secure RSA Implementations. In Ed Dawson, editor, *CT-RSA*, volume 7779 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013. ISBN 978-3-642-36094-7. URL http://dblp.uni-trier.de/db/conf/ctrsa/ctrsa2013.html#BauerJPW13.

[26] Aurélie Bauer, Eliane Jaulmes, Emmanuel Prouff, and Justine Wild. Horizontal Collision Correlation Attack on Elliptic Curves. In Tanja Lange, Kristin Lauter, and Petr Lisoněk, editors, *Selected Areas in Cryptography – SAC 2013*, volume 8282 of *Lecture Notes in Computer Science*, pages 553–570. Springer Berlin Heidelberg, 2014. ISBN 978-3-662-43413-0. doi: 10.1007/978-3-662-43414-7_28. URL http://dx.doi.org/10.1007/978-3-662-43414-7_28.

[27] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007. ISBN 0387308571.

[28] Max Bramer. *Principles of Data Mining*. Springer, 2007. ISBN 978-1-84628-765-7.

[29] Eric W. Weisstein. "Hyperplane." From MathWorld–A Wolfram Web Resource. URL http://mathworld.wolfram.com/Hyperplane.html.

[30] John C. Platt. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. Technical report, ADVANCES IN KERNEL METHODS - SUPPORT VECTOR LEARNING, 1998.

[31] Margaux Dugardin, Louiza Papachristodoulou, Zakaria Najm, Lejla Batina, Jean-Luc Danger, Sylvain Guilley, Jean-Christophe Courrege, and Carine Therond. Dismantling real-world ecc with horizontal and vertical template attacks. Cryptology ePrint Archive, Report 2015/1001, 2015. http://eprint.iacr.org/.

[32] M. Lochter and J. Merkle. Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation, 2010.