Side-channel attacks on embedded cryptography libraries

Szanto, J.B.

Award date:
2020

Eindhoven University of Technology

Department of Mathematics and Computer Science

Side-channel attacks on embedded cryptography libraries

Author:
Justin Szanto

Supervisors:
Dr. Boris Škorić
Dr. Georgios Selimis
Dr. Roel Maes

Defence date: April 17, 2020

Abstract

The topic of this thesis is side-channel analysis (SCA) of cryptography on embedded devices. In this thesis we investigated the SCA resistance of two popular embedded cryptography libraries; mbedTLS and MicroECC. We explore the use of TVLA as a testing methodology for side-channel leakage and test several countermeasures for AES. Through practical attacks we show that both mbedTLS and MicroECC do not contain sufficient countermeasures against power-based side-channel attacks. The main contributions of this thesis are a new attack on MicroECC and suggestions for improved side-channel testing methodology based on TVLA.

Acknowledgements

This thesis concludes my journey at Eindhoven University of Technology. A journey, which started off as a first year student pursuing a combined bachelor's in Mathematics and Computer Science, now comes to an end with this master's degree in Information Security Technology. Before we dive in to the full scientific extend of this thesis I would like to thank the people that helped me reach this point.

First and foremost, I would like to thank my supervisors, this result would never have been possible without them. I would like to thank Boris Škorić for his support and guidance during the project. Furthermore, I would like to thank my company supervisors Roel Maes and Georgios Selimis from Intrinsic ID. Thanks for coaching me along the way, the interesting discussions we had and for the many meetings which we had to keep me up to speed. I would like to thank Berry Schoenmakers for taking the time to be on my thesis defense committee.

The next group of people I would like to thank are all my colleagues at Intrinsic ID. Thanks for always being available to help and for providing me with a stimulating and sociable environment wherein I could work on my thesis. I hope our paths will cross again in the future. A special thanks to Pradeep Venkatachalam for occasionally brainstorming with me about with my experiments and for checking in on my progress once in a while.

I would like to thank Tanja Lange and Benne de Weger for making me enthusiastic for cryptography and security during my studies. Their teaching, enthusiasm and invitations for various conferences and symposiums resulted in my choice to pursue a masters in security.

I would also like to express my gratitude to MedApp for employing me for the past 4 years and providing me with valuable work experience alongside my studies. Even though I had to turn down their offer for a graduation internship, I still am glad I could help them in their mission to make the world a healthier place.

Aside from all the serious business, I also would like to thank all the friends I made along the way. Thanks to the wonderful group of people at student sports association All Terrain making my time at the TU/e even more enjoyable and active. Also thanks to GEWIS for the many extracurricular and social activities.

Of course there are many other people who helped me over the course of the past years. Thank you, everyone who I did not mention here for making this thesis possible.

Last, but definitely not least, I would like to thank my parents, my sister and the rest of my family for their continued support during my studies.

Contents

List of Figures

Notation index

The mathematical symbols and abbreviations used throughout this thesis are given below.

Mathematical symbols and functions

$\oplus$	Bitwise exclusive OR.
$+$	Integer addition.
$\cdot$	Integer multiplication.
$\times$	Scalar multiplication on an elliptic curve.
$\|$	byte concatenation.
$\lll$	left bitwise cyclic shift
$\mathrm{ROT8}(x)$	8-bit bitwise cyclic shift.
$\mathrm{HW}(x)$	The hamming weight of a bitstring x.
RK_i	The AES roundkey for round i.
$S(x)$	The value of the AES substitution box applied to x
$k[i]$	The ith bit of an integer k

Abbreviations

SCA	Side-channel analysis
AES	Advanced Encryption Standard.
ECB	Electronic Code Book mode
CTR	Counter mode
TVLA	Test Vector Leakage Assessment
TLS	Transport Layer Security
MSB	Most significant bit
LSB	Least significant bit
ECC	Elliptic-curve cryptography
ECDSA	Elliptic-curve Digital Signature Algorithm
ECDH	Elliptic-curve Diffie Hellman
MS	Mega samples (10^6 samples)
SAD	Sum of Absolute Distances
IV	Initialization Vector

Chapter 1

Introduction

This thesis is the result of the graduation project for the Information Security Technology master at Eindhoven University of Technology (TU/e). The graduation project was carried out at Intrinsic ID in Eindhoven between October 2019 and March 2020.

1.1 Motivation

Embedded devices are becoming increasingly pervasive in today's society. The global market for internet of things (IoT) devices is expected to grow to 248 billion dollars by the end of 2020 [1]. Securing those devices is an ongoing problem. One of the challenges involved in securing these devices is authentication. It is often difficult to securely provision the device with cryptographic keys at the time of manufacturing [2]. To overcome these and other issues, Physical Uncloneable Functions (also known as Physical One-Way Functions) (PUFs) can be used. PUFs were first introduced by Pappu et al. in 2002 [3] as a solution to this problem. Currently various commercial solutions are already available to solve such problems using PUFs. Solutions include Intrinsic ID's software (Broadkey) and hardware (Quiddikey) products which have been shipped to over 170 million devices to date [4]. Other companies supplying PUF technology include Verayo, Pufsecurity and Enthentica.

Even if there is no way in which secret keys can be directly extracted from a device, there still are other threats which could expose these keys to an attacker. On embedded devices one of the threats is power-based side-channel attacks. Power-based side-channel attacks were first found by Kocher et al. in 1999 [5] and since then have been slowly evolving.

In this thesis we research whether publicly available encryption software libraries are secure against power-based side-channel attacks. We analyze the AES implementation in ARM's mbedTLS library [6] and the full MicroECC [7] library. We aim to determine whether we can find and exploit side-channel leakage using relatively low-cost ($< \$1000$) equipment.

One of the methods used for analyzing side-channel leakage is Test Vector Leakage Assessment (TVLA) [8]. Through an extensive literature study we explore the limitations of this method and determine the best way to use it in practice.

Besides studying side-channel attacks and leakage detection, we also focus on countermeasures against side-channel attacks. We evaluate the usefulness and performance of various countermeasures on AES implementations.

We also evaluate various experimental countermeasures provided by Intrinsic ID. These results have been ommited from the public version of this thesis due to confidentiality agreements with the company.

1.2 Our contribution

In this thesis we experimentally show that the Mbed TLS library is not secure against power-based side-channel attacks. We propose modifications to the Mbed TLS library to make it secure against side-channel attacks, namely by using a threshold scheme which splits the computation of AES in to multiple parts. Furthermore, we investigate the side-channel security of MicroECC and introduce a new attack in which the Montgomery ladder implementation can be broken using only one power trace from the victim device.

1.3 Outline

This thesis is divided into six chapters. In chapter 2 the required background information is given. An overview is given of the attacker model, cryptography, side-channel attacks and countermeasures. In chapter 3 we give more details about our experimental setup and explain how it was validated to perform correct measurements. We also show the use of TVLA as a testing method for side-channel leakage. In chapter 4 we present our results on the side-channel evaluation of the AES implementation in Mbed TLS and possible countermeasures to make it resilient against side-channel attacks. In chapter 5 we discuss our attack on MicroECC and the use of TVLA on ECC. We end with chapter 6, which gives our conclusions, a discussion of the results and pointers for future work.

Chapter 2

Background

2.1 Cryptography

In this section we introduce the cryptographic algorithms and protocols used throughout this thesis.

2.1.1 Advanced Encryption Standard

The advanced encryption standard (AES) is a symmetric block cipher initially proposed by John Daemen and Vincent Rijmen in [9]. It was standardized in [10] by the National Institute for Standards and Technology (NIST). The cipher can process 128-bit blocks using keys of 128, 192 or 256 bits. All steps in AES operate on a 16-byte state X, which can be represented by a 4x4 matrix. AES consists of a sequence of 10, 12 or 14 rounds depending on whether the key k is 16, 24 or 32 bytes. The key k is expanded into 16-byte round keys RK_i. To describe the remainder of the cipher we use the following notation:

- $RK_{i,j}$ denotes the j^{th} byte of the round key for round i, where $j \in [0, 15]$.
- X_j denotes the j^{th} byte of the state, where $j \in [0, 15]$.

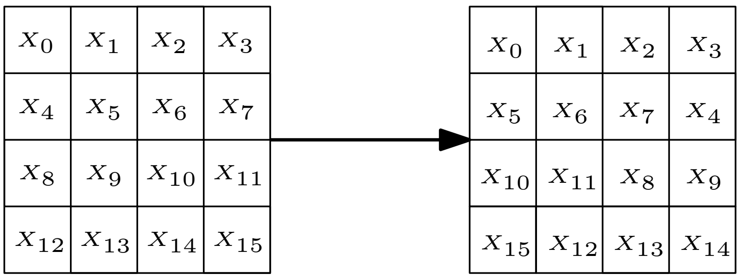
The state X is initialized with the input to the cipher, after which the following four steps are applied to it in each round.

1. **AddRoundKey:** In this step the round key RK_i is XOR'ed with the state X.
2. **SubBytes:** Each byte of the state is individually transformed through using the S-box. The S-box is a non-linear mapping from an 8-bit input to an 8-bit output. Both the input and the output can be interpreted as a polynomial in GF(2) by using the bits as coefficients. The S-box is computed by finding the multiplicative inverse of the input in GF(2^8) and subsequently applying to following affine transformation to it.

$$x \oplus (x \lll 1) \oplus (x \lll 2) \oplus (x \lll 3) \oplus (x \lll 4) \oplus c \qquad (2.1)$$

   In which x is the multiplicative inverse and c is a constant defined by 01100011.

3. **ShiftRows:** In this step the bytes of X are permuted according to the following fixed permutation.


x_0 x_1 x_2 x_3
x_4 x_5 x_6 x_7
x_8 x_9 x_{10} x_{11}
x_{12} x_{13} x_{14} x_{15}
x_0 x_1 x_2 x_3
x_5 x_6 x_7 x_4
x_{10} x_{11} x_8 x_9
x_{15} x_{12} x_{13} x_{14}

4. **MixColumns:** MixColumns is an invertible non-linear transformation performed on the columns of X. The kth column of the output of the MixColumns step can be represented by the following matrix multiplication in GF(2^8).

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} X_{4k} \\ X_{4k+1} \\ X_{4k+2} \\ X_{4k+3} \end{bmatrix}, k \in [0,3] \tag{2.2}$$

In the last AES round the MixColumns step is omited and replaced by an additional AddRoundKey step.

AES forward tables

On 32-bit architectures, AES is often implemented using *forward lookup tables*, sometimes also refered to as *T-tables*. This approach was described in the original AES proposal in [9], however it was never included in the official specification. The main idea is to use a single lookup table which returns the combined output of the SubBytes, ShiftRows and MixColumns steps. In a forward table a single byte input is mapped to 4 bytes of output.

We shall now give a brief explanation of this implementation. At compile time, or if desirable at runtime a set of 4 lookup tables $T_k : \{0,1\}^8 \rightarrow \{0,1\}^{32}$ are generated. The lookup tables are defined by the following equations.

$$T_0[x] = 2 \cdot S(x) \;\|\; S(x) \;\|\; S(x) \;\|\; 3 \cdot S(x), \tag{2.3}$$
$$T_1 = \mathrm{ROT8}(T_0), \quad T_2 = \mathrm{ROT8}(T_1), \quad T_3 = \mathrm{ROT8}(T_2) \tag{2.4}$$

Similar tables are generated for decryption, however for now we shall only focus on encryption. For the actual encryption operation, all data is split in to 32-bit words, each representing four bytes of either plaintext, round-keys or the AES state. We now use the following notation to the define all further operations:

- $\widetilde{RK}_{i,j}$ denotes the j^{th} word of the round key for round i.
- $\tilde{X}_j$ denotes the j^{th} word of the state.

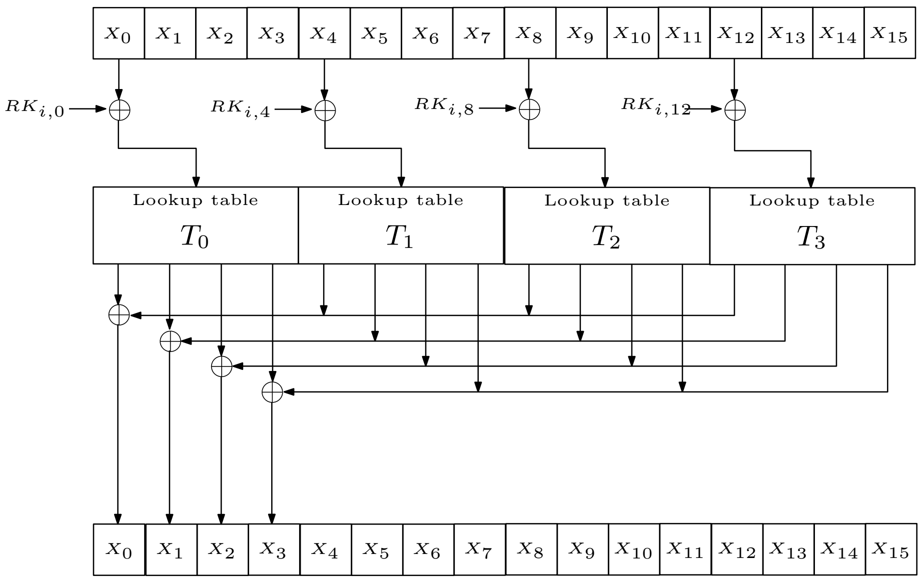

x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
$RK_{i,0}$
$RK_{i,4}$
$RK_{i,8}$
$RK_{i,12}$
Lookup table T_0
Lookup table T_1
Lookup table T_2
Lookup table T_3
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}

Figure 2.1: *Schematic overview of a single encryption round using forward tables.*

A schematic overview of this operation is given in figure 2.1. For some operations we need to retrieve a single byte from a word, this is denoted using the array index operator, i.e. $\tilde{X}_j[k]$ denotes the k_{th} byte from $\tilde{X}_j$. An AES round now only consists of the following two steps.

1. **AddRoundKey:** Equivalent to the traditional implementation, every word of the round key $\widetilde{RK}_i$ is XOR'ed with the state $\tilde{X}$.
2. **Table lookup:** The output of the AddRoundKeys step is used for a table lookup in the previously generated tables. A set of four table lookups is performed to compute one word of the output. The result of the tables lookups is combined using the XOR operation. The kth word of the state is now defined by the following equation.

$$T_0[\tilde{X}_k[0]] \oplus T_1[\tilde{X}_{(k+1) \bmod 4}[1]] \oplus T_2[\tilde{X}_{(k+2) \bmod 4}[2]] \oplus T_3[\tilde{X}_{(k+3) \bmod 4}[3]] \tag{2.5}$$

AES modes of operation

In a block cipher like AES the mode of operation is an algorithm which applies block cipher operations in a certain order to produce ciphertexts. For decryption the algorithm inverts this process. The most basic mode, given in figure 2.2, is Electronic Codebook (ECB) mode. In ECB mode each block of plaintext is encrypted independently of the other blocks. An issue with this mode is the lack of diffusion. If two blocks of plaintext are the same, then their ciphertext also is the same. A more commonly used alternative to ECB mode is Counter (CTR) mode [11]. Counter mode, given in figure 2.3, effectively turns a block cipher in to a stream cipher. In CTR mode a random nonce is generated before the encryption starts. This nonce is combined with a counter which increments after

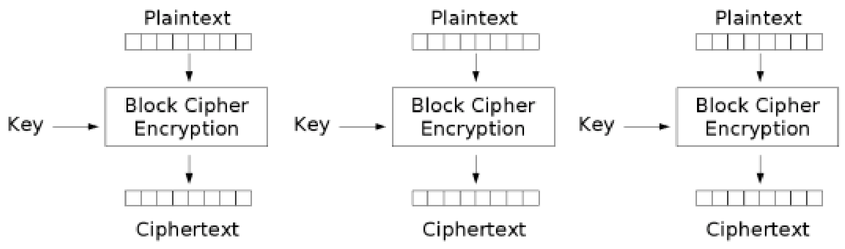


Figure 2.2: *Electronic Codebook (ECB) mode encryption*

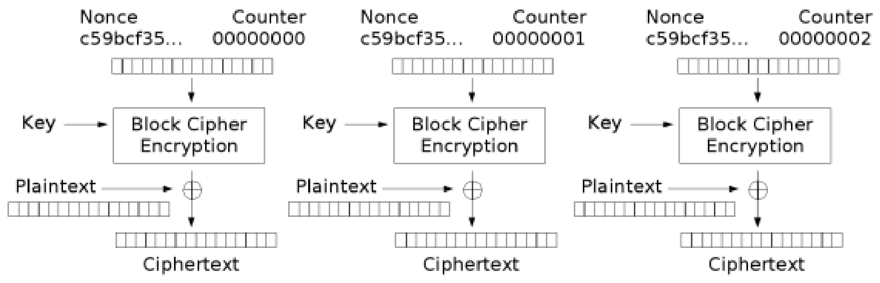


Figure 2.3: *Counter (CTR) mode encryption*

every block which is encrypted. The combination of the nonce and the counter is encrypted, the resul is XOR'ed with the plaintext to obtain the ciphertext. Since the counter increases, every ciphertext is guaranteed to be different. In total five encryption modes are standardized in [11]: ECB, CBC, OFB, CFB, CTR. More modes exist for authentication and authenticated encryption. In this thesis we only use ECB and CTR mode.

2.1.2 Elliptic Curve Cryptography

Elliptic curve cryptography (ECC), first introduced in [12], is public-key cryptography based on elliptic curves over finite fields. An elliptic curve is a set of all points (x, y) with coordinates in a finite field K which satisfy the following equation.

$$y^2 = x^3 + ax + b \qquad a, b \in K \tag{2.6}$$

This equation expresses the curve in *affine coordinates*, other representations also exist and will be covered later on. To perform an addition $R = P + Q$ on an elliptic curve the formulas given in equation 2.7 can be used. For $P \neq Q$, we have $\lambda = \frac{y_Q - y_P}{x_Q - x_P}$. If $P = Q$ then we have $\lambda = \frac{3x_P^2 + a}{2y_P}$, in which a is one of the curve parameters corresponding to the curve as defined in equation 2.6. Division is performed by computing the modular inverse in K.

$$x_R = \lambda^2 - x_P - x_Q, \quad y_R = \lambda(x_P - x_R) - y_P \tag{2.7}$$

Scalar multiplication

In all ECC operations point multiplication is an important step. Scalar multiplication is defined by taking $k \in \mathbb{N} \backslash \{0\}$ and multiplying it with a point $P = (x_P, y_P)$ on an elliptic curve.

Scalar multiplication algorithms

Using the addition formulas from equation 2.7 we can define algorithms to perform multiplications. The naieve approach would be to use a repeating loop of additions to perform multiplication. However, this would be very inefficient for a large scalar. A very basic efficient algorithm for performing scalar multiplication is the double and add algorithm, which is given in algorithm 2.1.

Algorithm 2.1: *Double and add.*

```
input: k, P, n
output: k× P
begin
$R \leftarrow R$
for $i \leftarrow n-2$ down to 0 do
    $R \leftarrow 2R$
    if $k[i]$ = 1 then
        $R \leftarrow R + P$
    end
end
return $R_0$
end
```

Another, more commonly used algorithm is the Montomery ladder [13], which is given in algorithm 2.2.

Algorithm 2.2: *Montgomery ladder.*

```
input: k, P, n
output: k× P
begin
    $R_0 \leftarrow P$
    $R_1 \leftarrow 2P$
    for $i \leftarrow n-2$ down to 0 do
        $R_{1-k[i]} \leftarrow R_0 + R_1$
        $R_{k[i]} \leftarrow 2 \cdot R_{1-k[i]}$
    end
    return $R_0$
end
```

The main advantage of the montgomery ladder, which will become more evident later on, is that it always takes the same amount of time to compute the multiplication. The idea behind the montgomery ladder is that always the same computations are done, irregardless of the value of k. The only thing determined by k is the location in which the result of a computation is stored.

ECDLP

Given $k \times P$, recovering k is considered a hard problem, since division is not defined over elliptic curves. This problem is also known as the Elliptic Curve Discrete Logarithm Problem (ECDLP). The security of a all ECC cryptosystems depends on the attacker's inability to solve the ECDLP and therefore inability to recover k.

Alternative coordinates

Performing additions and doublings in affine coordinates requires modular inversions (in order to compute λ), which is a costly operation in terms of computing resources. To overcome this issue alternative point representations can be used, which remove the need for modular inversions. The following two methods can be used to represent an affine point (x, y) as (X, Y, Z) with $X, Y, Z \in K$.

- **Standard projective coordinates [14]:** $x = \frac{X}{Z}, \qquad y = \frac{Y}{Z}, \qquad Z \neq 0$
- **Jacobian projective coordinates:** $x = \frac{X}{Z^2}, \qquad y = \frac{Y}{Z^3}, \qquad Z \neq 0$

ECDSA

The Elliptic Curve Digital Signature Algorithm (ECDSA) [15] is an elliptic-curve variant of the Digital Signature Algorithm (DSA) [16]. To create a digital signature using ECDSA the following parameters are used.

- A base point G on the curve, which serves as a generator for a subgroup of points on the curve.
- The order of G given by n.
- A randomly chosen private key p.
- A public key $Q = p \times G$.
- The message hash e.

Given theses parameters an ECDSA signature is computed in the following way.

1. Generate a unique and cryptographic secure random value k.
2. Compute the curve point $(x, y) = k \times G$.
3. Compute $r = x \bmod n$. If $r = 0$, then restart from step 1.
4. Compute $s = \frac{e + p \cdot r}{k} \bmod n$. If $s = 0$, then restart from step 1.
5. The signature is now given by the pair (r, s).

If an attacker can recover the value of k, then the private key p can trivially be computed using $p = s \cdot k - e$.

ECDH

Elliptic-curve Diffie–Hellman (ECDH) [17] is a key-agreement protocol used to establish a shared secret between two parties. It is an elliptic-curve variant of the Diffie-Hellman protocol [18]. To describe the ECDH protocol we use the same parameters as defined for ECDSA. Suppose Alice and Bob want to establish a shared secret. Alice has a keypair (p_A, Q_A) and Bob has a keyparir (p_B, Q_B), the protocol then proceeds as follows.

1. Alice and Bob agree on a curve and a generator G to use for the key exchange.
2. Alice and Bob exchange their public keys $Q_A = p_A \times G$ and $Q_B = p_B \times G$.
3. Alice computes $(x, y) = p_A \cdot Q_B$ and Bob computes $(x, y) = p_B \cdot Q_A$.

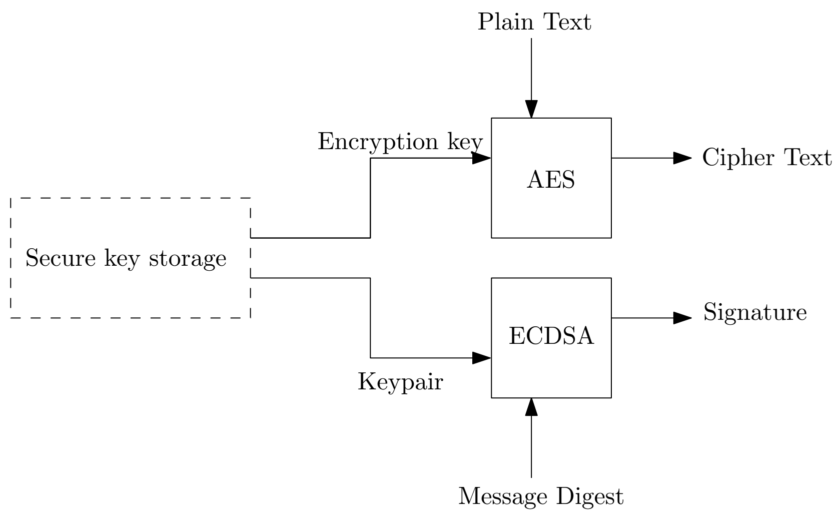


Figure 2.4: *A schematic representation of the device under attack*

4. Both parties now have the same shared secret given by x.

2.2 Attacker model

In the context of power-based side-channel attacks we adopt the standard attacker model for cryptographic hardware, as described in [19]. In this model we distinguish between active/passive attacks and invasive/semi-invasive/non-invasive attacks.

Passive attacks
In this type of attack, the attacker can not influence any of the data processed by the device. The attacker can only observe the behavior of the device. It only is possible to record power traces of the device under normal operation. The attacker can gain knowledge of plaintexts or chiphertexts processed by the device.

Active attacks
In this type of attack, the attacker can influence the data processed by the device. The attacker can provide input to the device within the range of valid inputs for the device.

Invasive attacks
In an invasive attack the attacker can physically modify the device in any way needed for the attack. Generally these attacks involve depackaging the chip and removing all protective layers on the chip. After this is done, either signals are read out directly from the chip (passive attack) or signals are altered to change the behavior of the chip (active attack).

Semi-invasive attacks
In a semi-invasive attack the attacker can modify the device, however the passivation layer on the chip is always preserved, so no direct contact with the surface of the chip is made.

Non-invasive attacks
In a non-invasive attack the attacker can not physically modify the device. Only existing interfaces on the device are utilized for the attack. This means that this attack leaves no traces on the device and generally does not require any expensive equipment to perform.

We assume attacker can perform an active, non-invasive attack on the device. A schematic overview of the device under attack is given in figure 2.4. The device is considered to be a black-box for the attacker; the attacker can not obtain any cryptographic keys from the device. The attacker can also not observe any intermediate outputs of computations done by the device. In addition to this basic model we make the following assumptions:

- The attacker can perform a standard chosen-plaintext attack or a chosen-ciphertext attack on the AES module. The attacker can provide any plaintext or ciphertext as input to the device and observe the device as it processes it in either an encryption or decryption operation.
- The attacker can perform a chosen-plaintext attack on the ECDSA module. The attacker can provide any plaintext as input and observe the device as it processes it during a signing operation. In addition the attacker can observe the resulting signature.
- The attacker can initiate an ECDH key agreement with any public key.
- The attacker can make an unlimited number of queries to the device.
- The software of the device can not be modified by the attacker in any way.
- The attacker can not inject any faults in to the device.

2.3 Power-based side-channel attacks

Our attacker model as described in 2.2 assumes an attacker who is able to capture power consumption traces from the device under test (DUT). Side-channel attacks rely on differences in power consumption depending on the information being processed the DUT. In this section we introduce the attacks which are used in this thesis.

2.3.1 Power models

A power model is a model which models the power consumption of a device based on which data the device is processing. In the following attacks, picking an accurate power model is of great importance for an efficient attack. When choosing a power model we are interested in a value which correlates to the power consumption of the device rather than the actual power consumption of the device. The following three power models are the most commonly used models for side-channel attacks.

Hamming weight
The hamming weight model assumes that a device consumes more power when a 1 is stored in a bit versus when a 0 is stored in a bit. It is most often used to model the power consumption of a circuit which makes use some sort of data or address bus. Computing the hamming weight over a hypothetical value being processed yields the power model.

Hamming distance
The hamming distance model assumes that a device consumes more power when a bit changes than when it stays the same. The model assumes that no additional power is consumed when a bit does not change. The power consumption is computed by computing the difference between two subsequent intermediate values. On CMOS circuits this model is more realistic than the hamming weight model. However it also is more difficult to use, since it requires knowledge about two subsequent intermediate values rather than one. If we assume that an intermediate value is changed from 0 to our hypothetical value, then this model is equivalent to the hamming weight model.

Switching distance
The hamming distance model makes the assumption that a $1 \to 0$ change and a $0 \to 1$ change consume the same a mount of power. In practice this proves to be incorrect. For this reason the switching distance power model was introduced [20]. This model assigns a value of 1 to a $0 \to 1$ change and a value of Φ to a $1 \to 0$ change. This value is also referred to as the Switching Distance Factor. A comparison of these power models can be found in [21].

2.3.2 Simple power analysis

Simple Power Analysis (SPA) was introduced by Kocher in 1996 [22]. By analyzing a single power trace, it is often possible to distinguish between operations. If the execution path of an algorithm depends on some secret data the attacker can often recover this data just by looking at the power trace. Using the trace it is possible to recover the path which the algorithm took and using that the attacker can recover the secret data which was being processed. This attack only works if there is some kind of dependency between the data being processed and the execution path of the algorithm. Eliminating such a dependency is usually quite straightforward, therefore we shall study more advanced attacks in the next sections.

2.3.3 Differential power analysis

Differential power analysis (DPA) was introduced by Kocher et al. in 1999 [5]. DPA attacks use a large number of power traces and analyze the power consumption at a fixed moment in time. For a successful DPA attack the attacker needs to know which algorithm the DUT is running and the attacker needs to know either the ciphertext, the plaintext or some intermediate value of the algorithm. The attack consists of the following steps.

1. **Collecting power traces:** A large number N of power traces must be collected while the DUT is performing either an encryption or decryption

operation. All traces must be aligned such that every point on each trace corresponds to the same point in the algorithm. For the attack to be succesful a large number of different input values for the algorithm should be used.

2. **Modeling an intermediate value:** The attacker chooses an intermediate value of the algorithm which under attack. This intermediate value should be chosen such that its output can be predicted based on the input to the algorithm and a small part of the key. The following hypothesis is used in DPA; depending on the least significant bit (LSB) of this intermediate value, small variations in the power consumption may be observed at a certain point in time.

3. **Splitting traces:** Let p_i denote the plaintext input for trace i and let k_j denote a guess for a single byte of our key. For each trace and for every key guess the attacker computes the intermediate value denoted by $f(p_i, k_j)$. For every key guess k_j the traces are split into two groups depending on the value of the LSB of $f(p_i, k_j)$.

4. **Difference of means:** For every key guess the attacker computes the mean of each group of traces for every point on the trace. This should give a single trace for each group. Next the attacker computes the difference between these two traces, giving us a single trace for each key guess.

5. **Recovering the key:** In the traces obtained in the previous step some peaks indicating a large difference in power consumption between the two groups may be visible. The trace containing the largest peaks is most likely the one corresponding with the correct key guess. The above steps can be repeated for every key byte until the entire key has been recovered.

The above process describes the original DPA attack. Variations to this attack exist, such as the one described in [19], which uses the hamming weight rather than the LSB to separate traces.

2.3.4 Correlation power analysis

Correlation power analysis (CPA) is an attack similar to DPA, introduced by Brier et al. in [23]. The attack aims to model the power consumption of the DUT using the hamming distance or hamming weight power model. Similar to DPA, the key again is split into multiple subkeys and a power model is made for every key guess for each subkey. The attack consists of the following steps.

1. **Collecting power traces:** This should be done in the same way as in DPA. However generally far less traces are required for CPA.

2. **Modeling an intermediate value:** Similarly to DPA the attacker again needs to choose an intermediate value to attack. The influence of this intermediate value on the power consumption is modeled using one of the power models described in section 2.3.1.

3. **Pearson's correlation coefficient:** For each trace and key guess the attacker models the power consumption based on the input (or output). Using Pearson's correlation coeficient the attacker determines the correlation between each key hypothesis and the observed traces at every point

on the trace. Pearson's correlation coefficient is given by the following equation.

$$r_{i,j} = \frac{\sum_{d=1}^{n}(h_{d,i} - \bar{h_i})(t_{d,j} - \bar{t_j})}{\sqrt{\sum_{d=1}^{n}(h_{d,i} - \bar{h_i})^2 \sum_{d=1}^{n}(t_{d,j} - \bar{t_j})^2}} \tag{2.8}$$

In this equation:

- $i \in [0, 255]$ represents a guess for a single byte of the key.
- $r_{i,j}$ refers to the value of the correlation coefficient for key guess i at point j on the trace.
- n refers to the number of captured traces.
- $t_{d,j}$ refers to point j in trace d.
- $h_{d,i}$ refers to the modeled power value for key guess i and trace d.
- $\bar{t}_j$ represents the mean value of point j across all traces.
- $\bar{h}_i$ represents mean value of the modeled power value for key guess i across all traces.

Now for each key guess and point on the trace there is a correlation value which tells us how well our key guess correlates to the actual power consumption over all traces.

4. **Selecting a key guess:** For every subkey the attacker chooses the key guess which has the highest correlation at any point on the trace. I.e. $\max_{i\in[0,255],j\in[1,n]} r_{i,j}$.

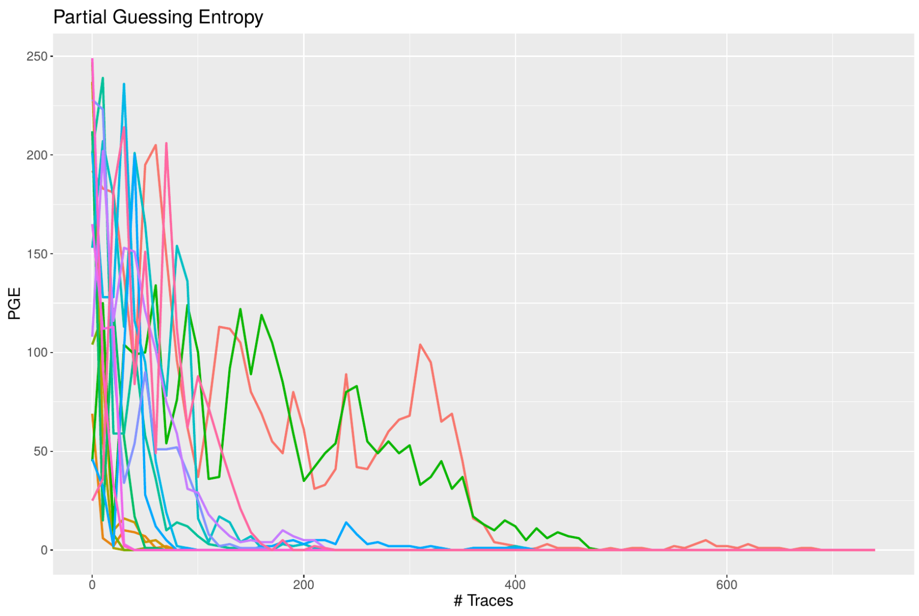


Figure 2.5: *The partial guessing entropy of an attack on an AES key. Each line corresponds to one byte of the key.*

Partial guessing entropy

The *partial guessing entropy* (PGE) can be used as an indication how many traces are needed to recover a key using a CPA attack. We use the PGE as defined by [24]. We can rank the guesses for each byte of the key based on their correlation value. The PGE is defined as the rank of the correct key. We can plot the rank of each key byte as a function of the number of traces used in the attack. An example of such a plot is given in figure 2.5. The PGE gives us the number of guesses required to guess the correct key.

2.3.5 Template attacks

Template attacks were introduced by Chari et al. in [25] as a very powerful attack requiring only a few traces from the target device. The attack is a profiling attack in which the attacker first creates a model of the DUT by experimenting with such a device that he has in his possession and uses the model to recover a secret key from the victim's device. The attack consists of the following steps.

1. **Collect profiling traces:** Using a copy of the device under attack the attacker need to capture a large amount of traces. These traces must use varying plaintexts and secret keys.
2. **Picking points of interest:** By studying differences in the traces, the attacker attempts to select a couple points at which the traces differ the most between different secret keys.
3. **Building a model:** Using the collected power traces the attacker builds a model for the power consumption of the device. The most basic way to do this is to create two equally sized groups of traces. One group corresponds to a certain bit of the key being equal to 1 and in the other group it is equal to 0. By taking the mean of both groups the attacker creates two templates. The model used in the template attack is a collection of at least two different templates corresponding to various private bits or bytes.
4. **Template matching:** Now on the victim's device the attacker captures a single power trace and choose the template which best matches the trace.

2.3.6 Online template attacks

Online template attacks (OTA) are a variant of template attacks introduced in [26] and [27]. In an OTA the template building and template matching phase is interleaved. If a target device performs a repetitive operation then the trace captured from the target device can be used to further improve an existing template

2.3.7 Horizontal attacks

Horizontal power analysis, introduced in [28], is another attack which only requires a single trace. Whereas *vertical* attacks such as DPA and CPA compare the same time sample across multiple traces, horizontal attacks compare multiple segments of the same trace. If a device performs a repetitive operation and the attacker can make some assumptions about the intermediate values, then a horizontal attack can be performed.

2.4 Test Vector Leakage Assessment

Test Vector Leakage Assessment (TVLA) is a testing method used to assess whether a device may be susceptible to side channel attacks. It was first introduced in 2011 by Goodwill et al. [8]. The test gives a level of statistical confidence whether the device under test (DUT) has exploitable leakage. It does not tell us whether it is possible to exploit this leakage in practice or how difficult it will be to exploit it.

The assessment is done by collecting two groups of traces from the DUT which is running some sort of cryptographic algorithm taking a secret key and plaintext as input. Every trace covers one execution of the algorithm. The secret key is the same in every trace. The plaintext switches between two options. The traces are grouped based on the plaintext. These two groups of traces are compared for similarity. If they contain statistically significant differences then that indicates some form of leakage. To check whether the two groups of traces are different *Welch's t-test* is used. This test is an adaptation of the Student's t-test which works better on samples with unequal sizes and variances. Welch's t-test is given in equation 2.9. We denote the mean of group j at point i on the trace by $\overline{\mu}_{i,j}$, the variance by $\sigma_{i,j}$ and the size of the group by N_j. The value of the t-test at point i on the trace is denoted by T_i.

$$T_i = \frac{\overline{\mu}_{i,1} - \overline{\mu}_{i,2}}{\sqrt{\frac{\sigma_{i,1}^2}{N_1} + \frac{\sigma_{i,2}^2}{N_2}}} \tag{2.9}$$

The statistic is calculated for every point in a trace. The null hypothesis is that the two groups have identical mean and variance. This hypothesis can be evaluated for every point in time on a trace. In literature a threshold of $T_i > 4.5$ is proposed to reject the null hypothesis. This implies 99.9999% confidence that the difference between the two traces is not caused by some random noise. I.e. for a random variable T following the Student's t-distribution: $\mathbb{P}(T_i > 4.5) < 10^{-5}$.

2.4.1 Non-specific TVLA

The test described above is also called a *specific TVLA*. It is specific in the sense that we create two groups based on a specific plaintext input value. The traces are categorized by one of the intermediate values of the algorithm, therefore the secret key must be known in order to perform this test. Furthermore we must know exactly which algorithm is being used on the DUT. This approach has some disadvantages. For instance, we may find some leakage which is only present under these specific plaintext inputs. Furthermore, the two plain texts might contain some bits which are equal, causing us to not discover some leakage.

In the case of a *non-specific TVLA* one group has a fixed plaintext and the other group has random plaintexts. Both groups again correspond to the same fixed secret key, but this time we don't need to know this key in order to group the traces. In [29] the authors argue that non-specific TVLA is superior to specific TVLA.

2.4.2 Higher order TVLA

TVLA can also be used to test for higher order side channel leakage, such as the attacks originally introduced by Messerges in [30]. Methods for higher order TVLA are described in [31].

2.4.3 The ISO 17825 standard

In 2016 the ISO 17825 standard named "Testing methods for the mitigation of non-invasive attack classes against cryptographic modules" [32] was published. This standard aims to provide a full testing methodology for conformance testing in the context of side-channel leakage. The standard recommends TVLA as the sole method to test for side-channel leakage.

This standard appears to be only generally available standard of its kind. Existing standards such as the Common Criteria and EMVCo only provide requirements in terms of side-channel resistance, but do not provide any testing methodology.

In [33] Whitnall and Oswell analyze the ISO 17825 standard and provide some well-versed criticism on some parts of the standard. They also provide some recommendations of their own to improve the standard. Their most important conclusions are that following the standard to the letter gives a very high probability of a false positive. To overcome this they recommend to use additional well established statistical methods to reduce the error rate and to supplement the test with a method to confirm the leakage which is found by the test.

2.4.4 TVLA confidence level

The original TVLA paper recommends a confidence level of $1 - 10^{-5}$ for the test. This implies a false-positive (type I error) rate of 10^{-5}. The only other formal recommendation for the confidence level is 0.95 in the ISO 17825 standard, which implies a lower false-negative (type II error) rate. This however poses another problem, which is explained in more detail in [33] and [34]. The confidence level which is used applies for every point on the trace. Therefore, the overall confidence level depends on the length of the trace. If we have a very long trace, i.e. millions of points, the type I error rate approaches 1. In [34] a proposal is given for choosing the confidence level based on the number of traces. This proposal is given by

$$\alpha' = 1 - (1 - \alpha)^{1/n},$$

in which α represents the overall confidence level, n the number of points on a trace and α' represents the confidence which should be use to evaluate the hypothesis at every single point on the trace.

2.5 Side-channel countermeasures

Countermeasures are modifications to a cryptographic implementation which aim to make a side-channel attack more difficult.

2.5.1 AES countermeasures

To protect an AES implementation from side-channel attacks a wide range of countermeasures have been developed. Most software-based countermeasures are based on one of the following principles:

1. Boolean or multiplicative masking of secret values [35], [36], [37], [38].
2. Secret sharing schemes (threshold cryptography) [39], [40], [41].
3. Randomising the order of operations.
4. Insertion of dummy instructions.

Similar countermeasures can also be implemented in hardware. In this thesis we only focus on software countermeasures. Arguably the first two countermeasures are the most effective, whereas the last two simply aim to complicate a side-channel attack rather than protecting fully against it.

2.5.2 Threshold cryptography

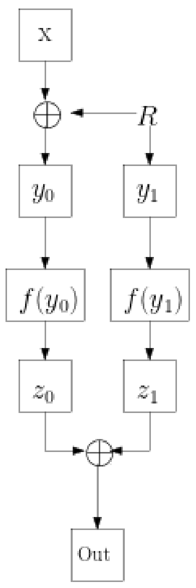


Figure 2.6: *A basic 2-share linear threshold scheme, R represents a randomly generated value.*

A threshold scheme, also known as a secret sharing scheme was introduced independently by Shamir [42] and Blakely [43]. The idea is to split a secret x in to n shares $y_0, y_1, ...y_{n-1}$, such that x can be easily reconstructed using $k \leq n$ shares, but having $k - 1$ shares reveals no information at all about x. In the context of threshold cryptography we extend this scheme to all cryptographic operations. Every computation is computed on at most $k-1$ shares, such that the output of each operation can never reveal any information about x. This property makes threshold schemes very useful as a countermeasure against side-channel attacks. Since choosing $k = n$ makes it the most difficult to reveal x, we will focus only on threshold schemes for which $k = n$. Furthermore we use a very simple sharing scheme which splits x in to shares using the XOR operation, i.e. $x = y_0 \oplus y_1 \oplus \cdots \oplus y_{n-1}$. A basic example of this threshold scheme is shown in figure 2.6.

2.5.3 ECC countermeasures

In ECC implementations most efforts are put into protecting the scalar multiplication algorithm. The double-and-add algorithm described in Section 2.1.2 contains a conditional branch on every bit of the scalar, which makes it vulnerable to timing side-channel attacks. To thwart this, most ECC implementations make use of a constant-time multiplication algorithm. An example of such an algorithm is the Montgomery ladder which is given in algorithm 2.2.

Randomized projective coordinates

A constant time multiplication algorithm may still be vulnerable to DPA attacks, such as the attack described in [44]. To protect against these attacks a commonly used countermeasure is the use of randomized projective coordinates. Projective coordinates such as defined in Section 2.1.2 have the nice property that they are not unique, i.e.:

$$(kX, kY, kZ) \equiv (X, Y, Z) \quad \forall k \in \mathbb{N}\backslash\{0\}. \tag{2.10}$$

Before every scalar multiplication a new radomized k can be chosen to randomize the coordinates. Therefore every trace will be different and a DPA attack will no longer work.

2.6 Oscilloscope bandwidth and sample rate

For our experiments we make use of a Digital Storage Oscilloscope (DSO). This type of oscilloscope digitally captures a signal by sampling it at a fixed sample rate. The sample rate of an oscilloscope is expressed in megasamples per second (MS/s) or gigasamples per second (GS/s). Based on the samples captured, a trace representing the original signal can be reconstructed. The Nyquist-Shannon sampling theorem [45] tells us that if a signal has no frequencies greater than B Hertz, that we can reconstruct the signal by sampling at a frequency of $2B$ samples per second. The bandwidth of an oscilloscope is defined as the maximum frequency which can be captured by the oscilloscope. If a signal contains frequencies beyond the bandwidth, the captured trace will be distorted.

Chapter 3

Experimental setup

3.1 Hardware

To perform our research we used the two side-channel evaluation setups described in this section.

3.1.1 Chipwhisperer

Chipwhisperer is an open-source hardware and software toolchain for side-channel power analysis and glitching attacks. For our analysis we used a Chipwhisperer lite board [46] as pictured in figure 3.1. This board consists of an OpenADC based oscilloscope used for capturing traces and a removable target board containing a STM32F3 ARM Cortex M4 CPU to be attacked. The target board contains convenient connections for analyzing the power consumption, capturing a trigger signal and performing clock glitching. From within the code running on the target we can raise a trigger pin which is connected to the trigger input of the scope. The Chipwhisperer's OpenADC scope synchronizes its sample rate with the clock of the target automatically. This allows us to easily capture well-aligned traces. In our experiments we found the following limitations of this device.

Maximum sample count: Due to limited resources on the board we can capture at most 24400 samples with the scope. For short algorithms this suffices, however for longer operations such as ECC this is not sufficient.

The sampling rate: The on board scope by default samples at 4x the clock speed of the target. Since the target runs at 7.37Mhz, this gives us a sampling rate of 29.538 MS/s. In theory the sampling rate can be configured up to 96 MS/s, however in our tests this caused the board to crash. Therefore it is still uncertain whether this actually is a possibility. The Chipwhisperer's documentation does not provide any further details on this. This sampling rate might be sufficient for our STM32F3 target, but for targets running at a higher clock speed it is too slow.

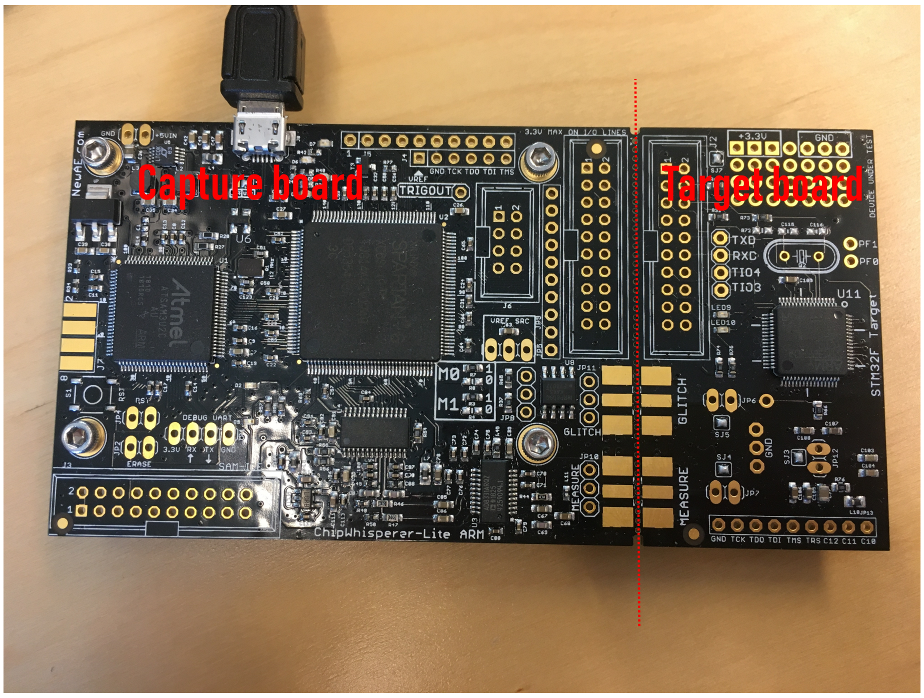


Figure 3.1: *Chipwhisperer based easurement setup*

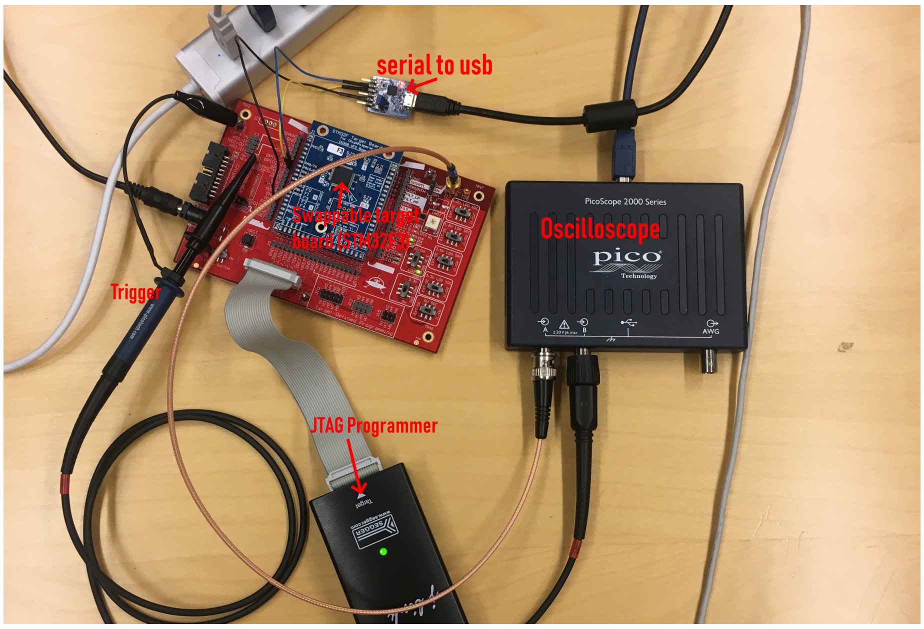


Figure 3.2: *Picoscope based measurement setup*

3.1.2 Picoscope based setup

In order to overcome some of the limitations of the Chipwhisperer we also built a setup based on the Picoscope 2207B. This still is a relatively low cost setup, but provides many more capabilities than the Chipwhisperer. The setup used is given in figure 3.2. Our target consists of a STM32F3 board mounted on a Chipwhisperer UFO board by NewAE. The UFO board facilitates side channel analysis of the target board by providing several facilities. It provides a stable 3.3V power source for the STM32F. Furthermore it contains a shunt resistor along with an amplifier to measure the current running through the target. Lastly it provides various connections to the target board, such as JTAG and the possibility to add an external clock signal.

The picoscope has a sample buffer of 500MS and can capture at a sample rate up to 1GHz. In practice, since we are using the two channels, our sample rate is limited to 500MS/s.

3.2 Verification

Before we perform any experiments using the aforementioned experimental setups, we need to verify that they function correctly. To do this we capture some traces using an AES implementation which is known to be vulnerable to side-channel attacks and describe the results here for both setups. The AES implementation which we attack is `tiny-AES-c` [47]. The methods described here will be used in all other experiments throughout this thesis, unless stated otherwise. We compiled the AES implementation using the `gcc-arm-none-eabi`

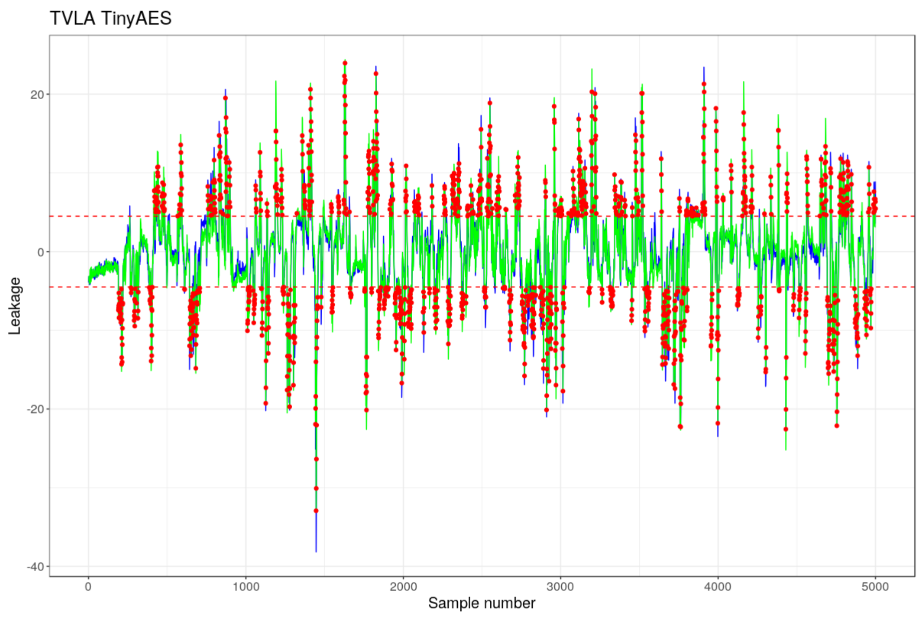


Figure 3.3: TVLA of TinyAES running on the Chipwhisperer, using 500 traces. One line is given for each run, the red points indicate leakage which was found in both runs.

compiler from the GNU ARM Embedded Toolchain [48]. We used the default compiler options for the STM32F3 target. For the Chipwhisper we used a sampling rate of 29.5 MS/s, on the Picoscope we used a sampling rate of 125 MS/s. We performed the TVLA as described in section 2.4. We captured 500 traces, each covering the full 10 rounds of AES, the results of the TVLA can be found in figure 3.3.

3.2.1 CPA attack

To verify that the leakage which was found in the TVLA can also be exploited we launched a CPA attack. For the CPA attack we use a power model based on the output of the S-box in the first round of AES. This same attack is described in detail in [19]. We denote the i^{th} byte of the plaintext input by p_i and the i^{th} byte of the key guess by k_i. According to the model, the power consumption P_i is given by

$$P_i = HW(S(p_i \oplus k_i)). \tag{3.1}$$

Using this power model we are able to successfully recover the AES key using around 50 traces on the Chipwhisperer. When using the Picoscope for the attack we need around 100 traces for a successful attack. This is most likely because the traces captured with the Chipwhisperer are aligned better than the traces captured with the Picoscope.

3.3 Sample rate considerations

For an optimal measurement setup it is important to choose the optimal sample rate. If we choose a sample rate which is too high then we store a lot of redundant data which leads to slower processing. If our sample rate is too low then we may not capture any leakage. On the Chipwhisperer the highest possible sample rate is 29.5 MS/s, which translates to 4 samples for every clock cycle. Since we only can capture at most 25000 samples with the Chipwhisperer, sampling at a lower rate does not make any significant difference in terms of storage.

The Picoscope can capture up to 64 million samples at a rate of 500MS/s. Since we used the Picoscope to capture longer traces, choosing the optimal sample rate makes a big difference. The bandwidth of the Picoscope is 70 MHz. By the Nyquist-Shannon sampling theorem it would not be useful to sample at a rate higher than 140 MS/s. However, according to the documentation of the Picoscope the sampling rate may need to be around 5 times greater than the bandwidth [49]. The maximum frequency of the signal from the target could also be lower than 70Mhz, which implies that a lower sample rate might also be sufficient.

Due to these uncertainties we decided to experimentally determine the optimal sample rate. It is important to note that the sample rate on the Picoscope can only be chosen from a set of predefined values: 32, 64, 125, 250 and 500 MS/s. To determine the optimal sample rate we performed a TVLA on the vulnerable AES implementation at all of these sample rates. Between 32, 64 and 125 MS/s we see a clear increase in the amount of leakage found by the TVLA. Between 125, 250 and 500 MS/s we do not see any difference. Therefore we chose 125 MS/s as the sample rate to use for the remainder of our experiments.

3.4 Environmental effects

If we capture a lot of traces subsequently, then we notice that they are not always vertically aligned when we overlay them. We observe that the mean power value slowly decreases over time. Our hypothesis is that this is caused due to the chip warming up during operation. To test this hypothesis we performed a couple of tests by cooling the chip to around $-20°C$ and capturing a set of traces. In this set of traces we see a significant change in the power value as the chip heats up to room temperature again. We did not research this effect any further as it was not relevant for our side-channel analysis. To compensate for this effect we regularize all our traces. Given a trace t of length n, in which t_j denotes the jth point on the trace, the regularized trace t' is given by the following equation.

$$t'_j = t_j - \frac{1}{n}\sum_{i=1}^{n} t_i \tag{3.2}$$

This results in all traces having a mean power value of 0.

3.5 TVLA considerations

In Section 2.4.4 we covered the fact that TVLA can produce false-positives on longer traces. To reduce the false-positive rate we use the following solution. All TVLA tests are performed twice. If leakage occurs at the same point in both tests, then we consider it to be actual leakage rather than a false-positive.

3.6 Filtering and alignment

In order to remove high frequency noise from our traces we can apply a low-pass filter to the traces. We experimented using a Butterworth low-pass filter [50]. In the CPA attack and TVLA measurements filtering is not necessary; the noise on the traces can be compensated by capturing more traces. Since we did not want to risk losing information by filtering we did not apply any filtering in any of the TVLA tests or CPA attacks. In the single trace attack described in Chapter 5 we did apply filtering, which is explained in more detail in Chapter 5.

Alignment

In most cases we found our traces to be aligned very well due to the accurate trigger setup and stable clock signal on the STM32F3. In CPA and TVLA measurements a slight misalignment of traces can be compensated by capturing more traces. To test how well the traces are aligned we used the Sum of Absolute Distances (SAD) metric. If we have two traces t_1 and t_0 of length n, the SAD value is given by the following sum.

$$\sum_{j=1}^{n} |t_{1,j} - t_{2,j}| \tag{3.3}$$

The jth point on trace i is given by $t_{i,j}$. A low SAD value indicates that the traces are aligned well. In practice we do not compute the SAD over the entire trace, but over a smaller segment. For instance a section of the trace which should be the same across all traces.

Chapter 4

Side channel analysis of AES

In this chapter we investigate the side-channel security of ARM's Mbed TLS library [6] and of several side-channel countermeasures. Mbed TLS is a very commonly used embedded TLS library and therefore interesting to attack. In this chapter we explore how resilient the AES implementation of this library is. We consider 128-bit AES in electronic code book (ECB) mode and the more commonly used counter (CTR) mode. Lastly we explore the effectiveness of several countermeasures.

Mbed TLS does not include any countermeasures against power-based side-channel attacks by default and therefore should be easy to attack using basic side-channel attacks.

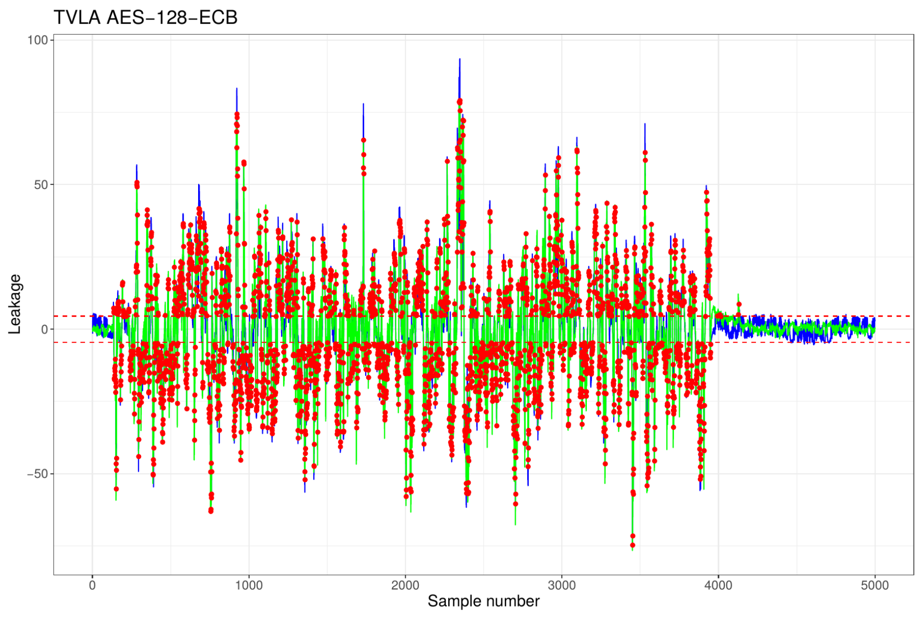


Figure 4.1: *TVLA of AES-128-ECB using 2500 traces. One line is given for each run, the red points indicate leakage which was found in both runs.*

4.1 Analyzing AES in ECB mode

We compiled the AES implementation of Mbed TLS using default compiler options for our STM32F3 target. For our measurements we used the Chipwhisperer, sampling at 29.5 MS/s. Next we performed the TVLA as described in Section 2.4. We captured 2500 traces, each covering the full 10 rounds of AES. The results of the TVLA can found in figure 4.1.

As expected, we observe multiple excursions above the 4.5 threshold. Therefore there is a very strong indication that side channel leakage is present. To verify that we can exploit this leakage we launch the CPA attack as described in Section 3.2.1.

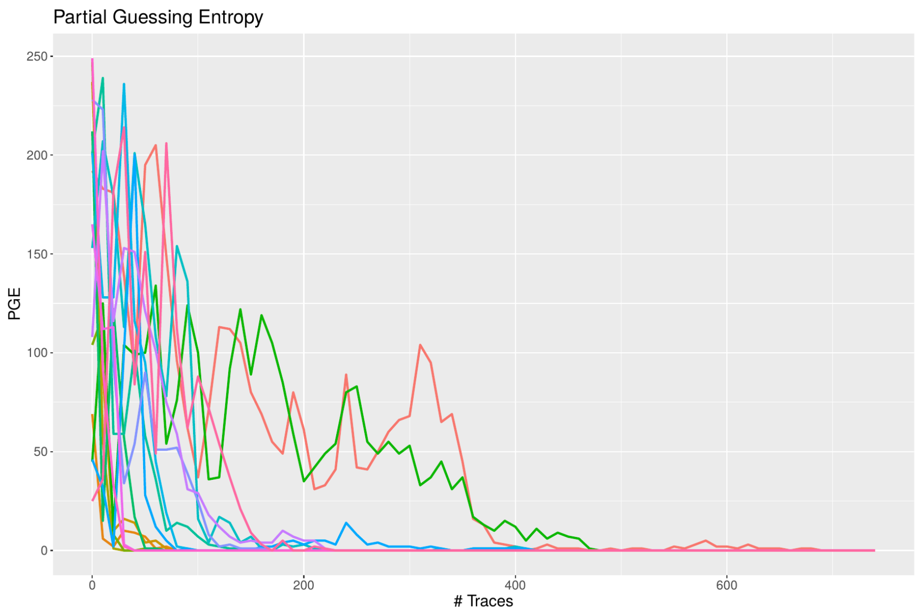


Figure 4.2: *PGE for the CPA attack on the Mbed TLS AES implementation. Each line represents one key byte.*

Using this attack we were able to recover the full key using around 500 traces. To illustrate how many traces are needed for this attack the PGE is given in figure 4.2. In figure 4.3 the correlation value of the correct key guess is plotted for every point on the trace. We observe many excursions at the beginning of the trace which appears to confirm that we actually are attacking the S-box output and not another part of the algorithm.

4.2 AES forward tables

Many modern AES libraries, including Mbed TLS, make use of forward tables. Since with this aproach the S-box is no longer explicitly computed, one might expect that the aforementioned CPA attack on the S-box would no longer work. However, since each entry in the forward table is a linear combination of multiple

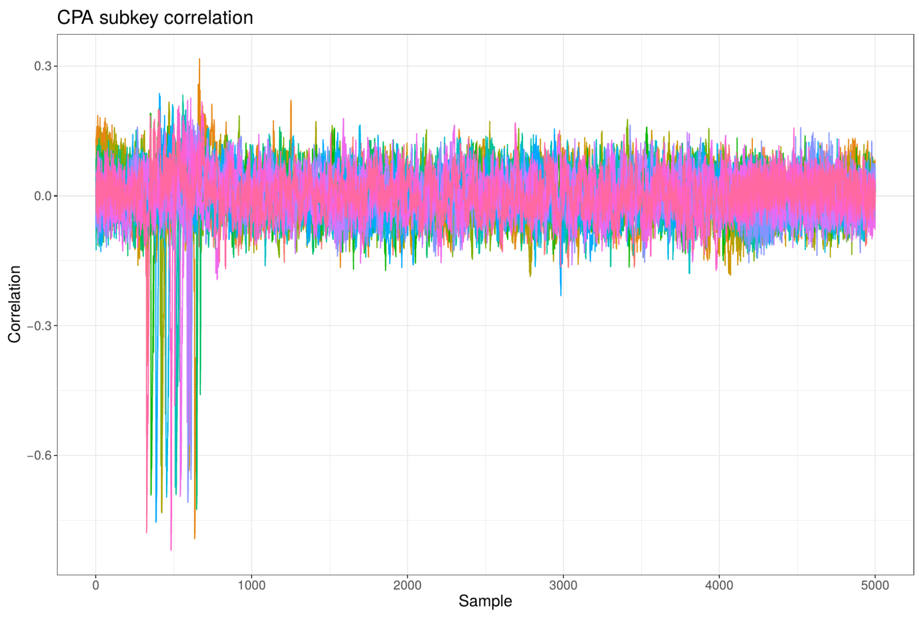


Figure 4.3: *The correlation of the best keyguess at every point in time. Every line corresponds to one byte of the key.*

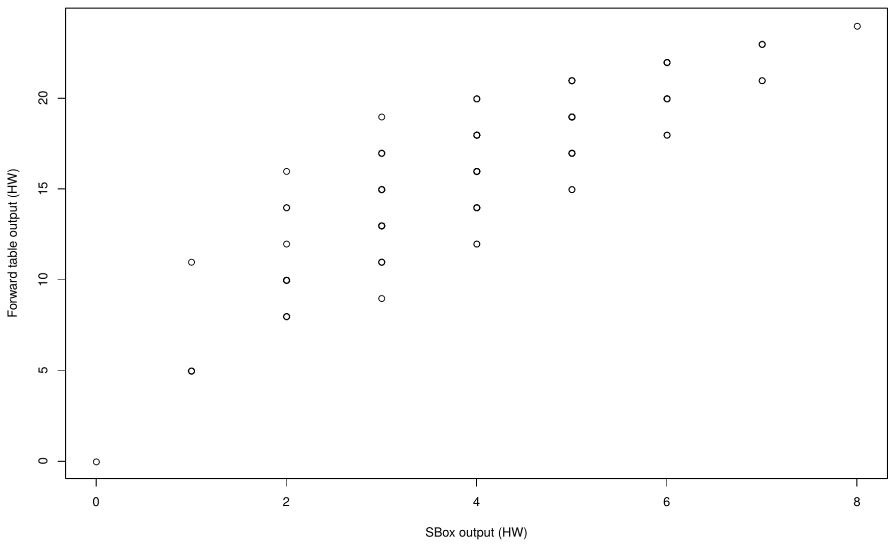


Figure 4.4: *The hamming weight of each output of the S-box plotted against the hamming weight of the corresponding forward table value.*

S-box values, there exists a strong correlation between the S-box output and the forward table output. This correlation can also be seen in figure 4.4.

The observed results imply that using a single attack we can attack both conventional AES implementations and implementations using forward lookup tables.

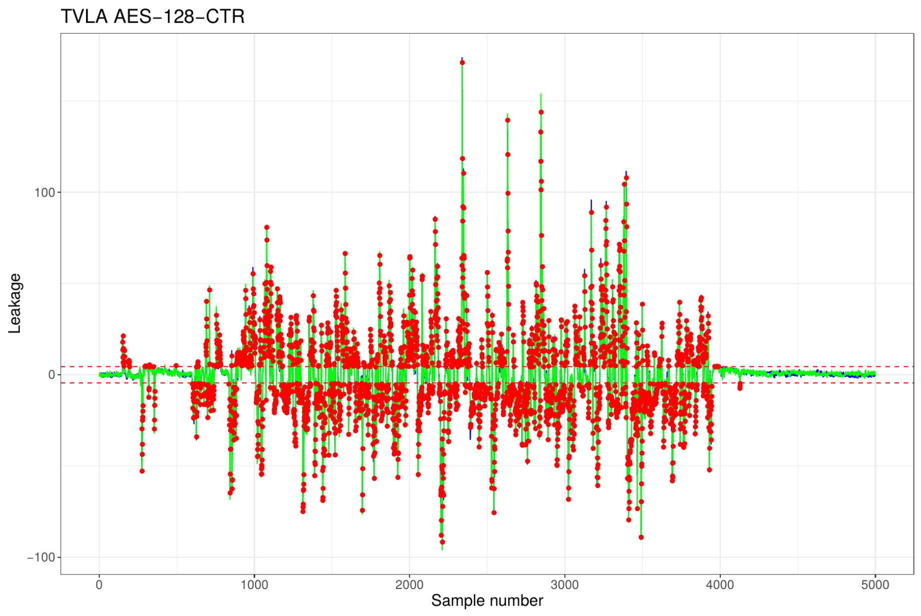


Figure 4.5: *TVLA of AES-128-CTR using 2500 traces. One line is given for each run, the red points indicate leakage which was found in both runs.*

4.3 Analyzing AES in CTR mode

Attacking AES in counter mode presents a few new challenges. In counter mode the first round will always be the same and the IV which is used is unknown to the attacker. For a succesful attack on AES in counter mode we need to gather one long consecutive trace rather than many small traces which only cover one iteration. Due to the limited sample length on the Chipwhisperer this was not a posibility. Therefore we chose for an approach which we will refer to as *simulated counter mode*, in which the IV is known to us. For this we the same AES-ECB implementation as used in Section 4.1. We again capture a large number of single-block traces. However, instead of providing the actual plain text as input we provide an IV combined with an incrementing counter value as input. This gives us power traces each corresponding to a consecutive iteration of AES in counter mode. As an added benefit this approach saves a lot of alignment work.

In order to perform a TVLA we also need to make some changes. In CTR mode AES acts as a stream-cipher. Therefore the input plaintext is only used in an XOR operation with the output of AES; it is not provided as input to the first AES round. If we simply apply standard TVLA, then we only test the leakage of this XOR operation, since the rest of the algorithm is independent of the plain text input. For the TVLA we use simulated counter mode, while making the counter value range from 0 through 9. The traces are then split into two groups; one group containing traces corresponding to counter value 0 and another equally sized group corresponding to a random sample of the other counter values. The resulting TVLA traces can be found in figure 4.5. Again we see multiple excursions above the 4.5 threshold. Therefore we expect that this implementation will also be vulnerable to side-channel attacks in CTR mode. An example of an attack which could be launched on this implementation is described by Ding et al. in [34].

4.4 Protecting Mbed TLS against side-channel attacks

To protect the AES implementation in Mbed TLS from the aforementioned attacks, we implement several countermeasures such as these described in Section 2.5.1.

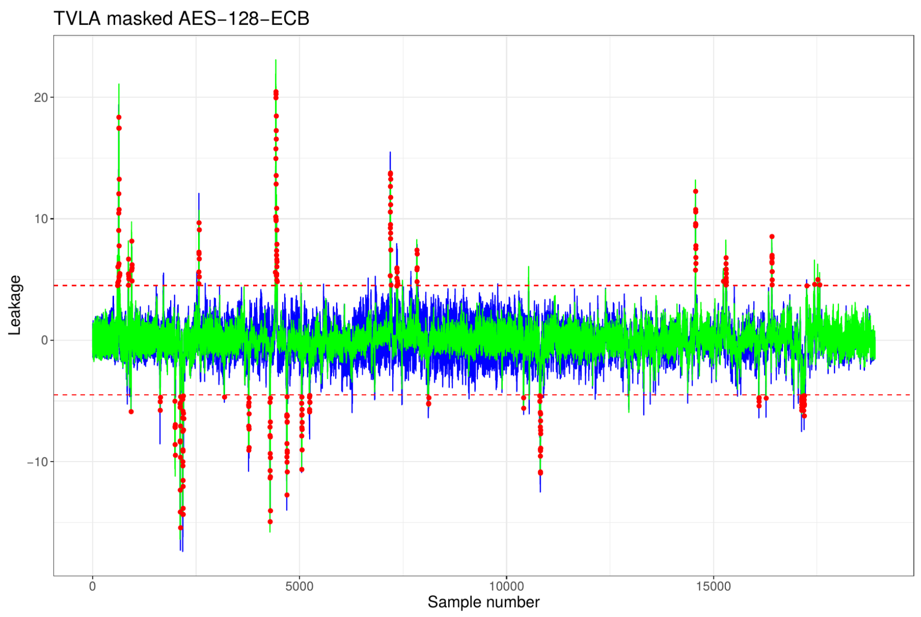


Figure 4.6: *TVLA of the AES-128-ECB with boolean masking using 10,000 traces. One line is given for each run, the red points indicate leakage which was found in both runs.*

4.4.1 A masked AES implementation

The first countermeasure is a relatively basic random boolean masking scheme. Our scheme is very similar to the scheme proposed in [35] and [36]. The main difference is that instead of applying it to a conventional implementation, we apply it to the forward table implementation of AES as described in Section 2.1.1. To mask this implementation we make use of the following distinct and uniformly random masking values. A single byte input mask $M \in \{0,1\}^8$ and

Implementation	Cycles/block	Memory footprint (B)
Unmodified mbed TLS	1004	4112
Masked mbed TLS $n = 1$ (4.4.1)	2346	4117
Fully threshold $d = 2$ (4.4.2)	184,361	1056
Hybrid threshold (4.4.3) $n = 1, d = 2$	27,376	8229

Table 4.1: *Performance figures for the various masked AES implementations.*

4-byte words M_0, M_1, M_2, $M_3 \in \{0,1\}^{32}$. For each forward table T_0, T_1, T_2 and T_3 we generate new masked forward table T'_k defined by

$$T'_k[x] = T[x \oplus M_s] \oplus M_{t_k}, \tag{4.1}$$

with its inverse given by:

$$T'_k[x \oplus M_s] \oplus M_{t_k} = T_k[x]. \tag{4.2}$$

Lastly we define an intermediate value P defined by:

$$P = M_0 \oplus M_1 \oplus M_2 \oplus M_3 \oplus (M \parallel M \parallel M \parallel M) \tag{4.3}$$

The ith AES round is defined by computing

$$\widetilde{RK}_{i,j} \oplus T'_0[\tilde{X}_j[0]] \oplus T'_1[\tilde{X}_{(j+1)\bmod 4}[1]] \oplus T'_2[\tilde{X}_{(j+2)\bmod 4}[2]] \oplus T'_3[\tilde{X}_{(j+3)\bmod 4}[3]] \oplus P \tag{4.4}$$

for all $j \in [0,3]$. For the first round we assign an XOR of the j^{th} input word with $(M_s \parallel M_s \parallel M_s \parallel M_s)$ to $\tilde{X}_j$ for all $j \in [0,3]$. XOR'ing with P results in the removal of the masks and the application of the input mask for the next round.

After every n blocks, the masks must be refreshed. If we do not do this, then an attacker could simply recover the masked values using DPA or CPA. The choice of n is a tradeoff between security and performance. Choosing a lower n arguably yields a more secure implementation, while a higher n yields a better performance. In our experiments we chose $n = 1$. Using this scheme, theoretically all intermediate values until the last round should be masked. To verify this we perform a new TVLA on the masked implementation. The results can be found in figure 4.6. We can see that the amount of leakage clearly is reduced a lot, but there still is a significant amount of leakage. However, we no longer are able to perform a CPA attack on this implementation. The leakage which we still see is possibly caused by secret information not being erased from memory properly after the masking. However, due to time constraints we did not verify this hypothesis. In future experiments we did pay more attention to properly erasing secret information from memory.

4.4.2 A threshold AES implementation

Another method which we used to mask our implementation is a threshold implementation as described in Section 2.5.2. This method was introduced in [39] as a provably secure method to mask AES against 1st order DPA. Our threshold implementation is uses the approach described by Rivain et al. [51]. It provides a masked AES implementation at any order.

A higher-order masked threshold implementation involves splitting all sensitive variables x in to $n = d+1$ shares, in which d is the order of side-channel resistence which we are trying to achieve. In general an implementation is considered dth order secure when every sensitive variable is masked using d random values.

Every function $f(x)$ is split up in to multiple functions $f_1(...), \dots, f_n(...)$. For these functions the following two properties must hold.

Correctness: I.e. $f_1(...) + \cdots + f_n(...) = f(x)$ must be true.

Independence: Every function $f_i(...)$ must depend on at most $n-1$ out of the n shares.

Uniformity: The output of every function f_i must be uniformely distributed. For linear functions, where $f(x_1 + x_2) = f(x_1) + f(x_2)$ these properties can be easily achieved. In AES all operations except for the S-box are linear. Therefore we shall mainly focus on masking the S-box.

Refreshing masks

Initially we set $x_0 = x, x_1 = 0, ..., x_n = 0$ for each secret variable. After this we apply our masking operation. This operation is shown in figure 4.7. In order to achieve the uniformity property, we can repeat this operation as many times as needed; the property $x_0 + x_1 + ... + x_n = x$ is always maintained.

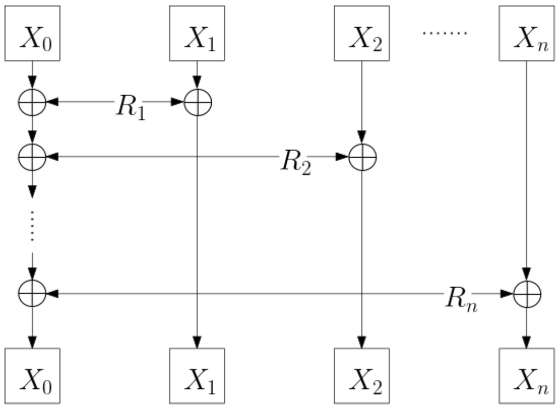


Figure 4.7: *The mask refresh operation. $R_1, ...R_n$ are randomly generated values.*

The AES S-box

The AES S-box consists of taking the multiplicative inverse over $GF(2^8)$ followed by an affine transformation. In $GF(2^8)$ the following two convenient properties hold.

$$x^{-1} \equiv x^{254}$$
$$(x_1 + x_2)^2 \equiv x_1^2 + x_2^2$$

This implies that we can compute the multiplicative inverse of x using a basic square and multiply algorithm. More specifically we will compute the following square and multiply sequence.

$$((x^2 \cdot x)^4 \cdot (x^2 \cdot x))^{16} \cdot (x^2 \cdot x)^4 \cdot x^2 \tag{4.5}$$

Since squaring is linear in $GF(2^8)$, all exponentiation in this calculation can be implemented using linear lookup tables and can therefore trivially be split in to multiple shares. The only non-linear operation left is the multiplication. Multiplication of $x \cdot y$ in shares can be done in the following manner. Given two vectors $x_0, x_1, ...x_n$ and $y_0, y_1, ...y_n$ as input we have intermediate shares defined by

$$t_{i,j} = (r_{i,j} \oplus x_i \cdot y_j) \oplus x_j \cdot y_i. \tag{4.6}$$

The values $r_{i,j}$ represent randomly generated numbers. If the computations are ordered according to the brackets, then the independence property holds for $n \geq 1$. Using these intermediate values we can compute the final shares of $x \cdot y$, which are given by the following sum.

$$o_j = x_j y_j \oplus \bigoplus_{i=0}^{j-1} t_{i,j} \oplus \bigoplus_{i=j+1}^{n} r_{j,i} \tag{4.7}$$

Using these shares the correctness property should hold, i.e.:

$$\bigoplus_{j=0}^{n} o_j = x \cdot y \tag{4.8}$$

A proof of this is given in appendix A.

Using equation 4.5 combined with this multiplication method we now can implement a threshold S-box for AES. Using this S-box we create a complete threshold implementation of AES. In this implementation we also made sure that unmasked secrets in memory are overwritten with random data after masked secrets have been created. We again performed a TVLA on this implementation. From this point on we used the Picoscope based setup for our analysis, since the sample buffer on the Chipwhispere was no longer large enough. The TVLA plot for our implementation can be found in figure 4.8. We still see some leakage at the beginning related to passing the plaintext to the masking operation. This leakage can not be trivially exploited.

4.4.3 The hybrid threshold scheme

The downside of a full threshold implementation can be seen in the performance figures in Table 4.1. Computing the S-box values consumes many clock cycles. To overcome the performance issue of a full threshold implementation we propose a hybrid scheme. In this threshold scheme we replace the S-box by a more performant, non-threshold, implementation. The S-box is replaced by a construction using a lookup table. A schematic overview of this implementation is given in figure 4.9. The S-box is now represented by a lookup table $S' : 0, 1^8 \rightarrow 0, 1^{16}$. For two shares we define this lookup table as follows.

$$S'[x] = (r, S[x \oplus M] \oplus r) \tag{4.9}$$

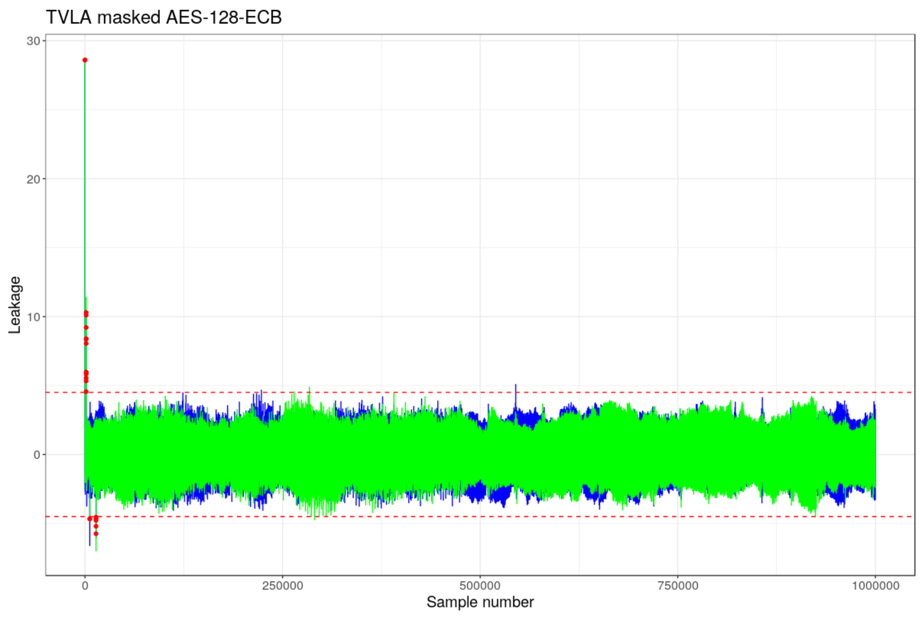


Figure 4.8: *TVLA of our threshold AES-128-ECB using 2500 traces. One line is given for each run, the red points indicate leakage which was found in both runs.*

M is a fixed mask and r is randomly generated for every entry. The fixed mask and the lookup table must be refreshed every n rounds, similar to the implementation described in Section 4.4.1.

At the S-box step M is added to the first share, after this the shares are XOR'ed with the first share to combine them in to a single masked value. This masked value is used to lookup $S'[x]$. The vector returned by $S'[x]$ represents the shares to be used for all further AES computations.

As shown in table 4.1, this implementation consumes significantly fewer cycles than the full threshold implementation.

To test whether this implementation provides the same level of side-channel resistance as the threshold implementation we performed a TVLA. The resulting TVLA plot is given in figure 4.10.

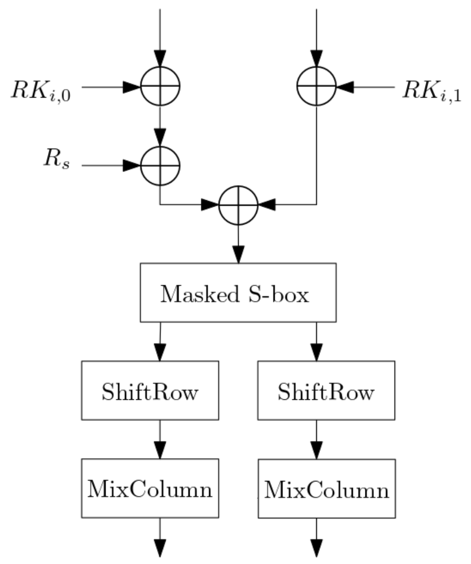


Figure 4.9: *The hybrid threshold scheme.*

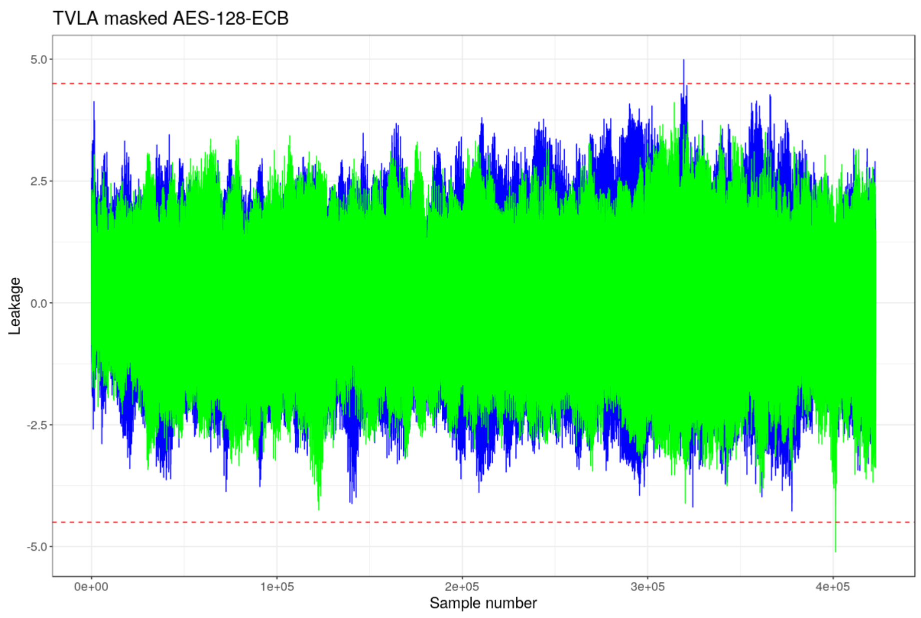


Figure 4.10: *TVLA of our hybrid threshold implementation using 10000 traces. One line is given for each run.*

Chapter 5

Side channel analysis of ECC

In this chapter we analyze the side-channel security of the MicroECC library [7]. This is a commonly used ECC library on embedded platforms with limited resources.

MicroECC contains multiple countermeasures against side-channel attacks. The main countermeasures are the use of randomized projective coordinates and a montgomery ladder for multiplication. This offers protection against (1st order) DPA attacks, since the coordinates used for the scalar multiplication are different in every trace. Furthermore, it even offers protection against most template attacks.

5.1 Attacking MicroECC

In MicroECC the Montgomery ladder is implemented the following way.

```
XYcZ_initial_double(Rx[1], Ry[1], Rx[0], Ry[0], initial_Z, curve);
for (i = num_bits - 2; i > 0; --i) {
  nb = !uECC_vli_testBit(scalar, i);
  XYcZ_addC(Rx[1 - nb], Ry[1 - nb], Rx[nb], Ry[nb], curve);
  XYcZ_add(Rx[nb], Ry[nb], Rx[1 - nb], Ry[1 - nb], curve);
}
```

The value of `initial_Z` is randomly generated. As a result of this the `XYcZ_addC` and `XYcZ_add` operations are also randomized. As intended with a Montgomery ladder, we do not expect to see any leakage during these operations. The `uECC_vli_testBit` function is the only operation which makes use of the secret scalar, after which the result (a single bit of the scalar) is used to compute the memory address to store the result of the point addition or doubling. Possibly we could find leakage during the computation of the memory address. To evaluate potential side-channel leakage we perform a specific TVLA test in which we vary the private key between two fixed private keys. These two private keys are chosen such that the LSB is different for each key. Since we are looking for leakage in an operation which only operates on a single bit of the private key, any other sort of TVLA test would be useless. We place a trigger at the `uECC_vli_testBit` line and capture 1000 traces for the TVLA. In order to ease

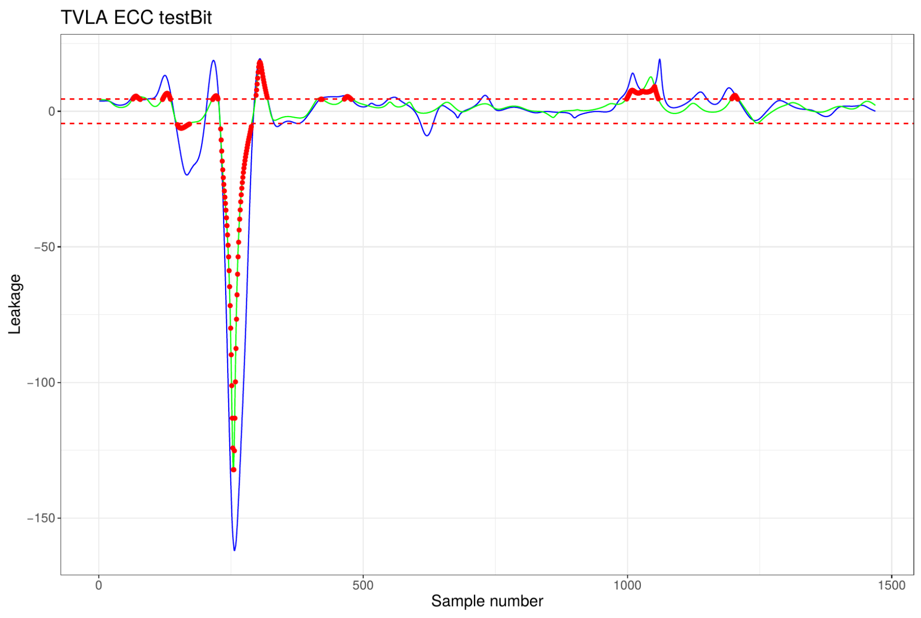


Figure 5.1: *TVLA plot of the uECC testBit function using an alternating bit. The TVLA was done using 1000 traces spread accross 2 runs. One line is given for each run.*

our attack we applied a butterworth low-pass filter with a cut-off frequency of 2KHz to all traces. The cut-off frequency of 2KHz was experimentally found to give the best results. Using these traces we compute a TVLA plot which can be found in figure 5.1. We also plot the mean of each group of traces which can be found in figure 5.2. As we can see in these figures, there is a clear dependency between the private key bit and the power consumption around sample 250. In this case we only tested this dependency for the first bit, in a more thorough test more bits should be considered. With this knowledge, a single trace of an ECC point multiplication is enough to recover bits of the secret scalar.

5.1.1 Finding the cause of the leakage

The following ARM assembly code is generated by the compiler when compiling MicroECC. This section of assembly code corresponds to the `uECC_vli_testBit()` line in the C code snippet given earlier. The `trigger_high()` function is used to raise a trigger pin to start our measurement.

```
bl          trigger_high
asr         param_4,r4,#0x5
ldr         param_3,[sp,#local_130]
uxth        r4,r4
ldr.w       param_3,[param_3,param_4,lsl #0x2]
mov         param_2,#0x1
and         param_4,r4,#0x1f
lsl.w       param_4,param_2,param_4
tst         param_4,param_3
itE         eq
mov.eq      r5,param_2
```

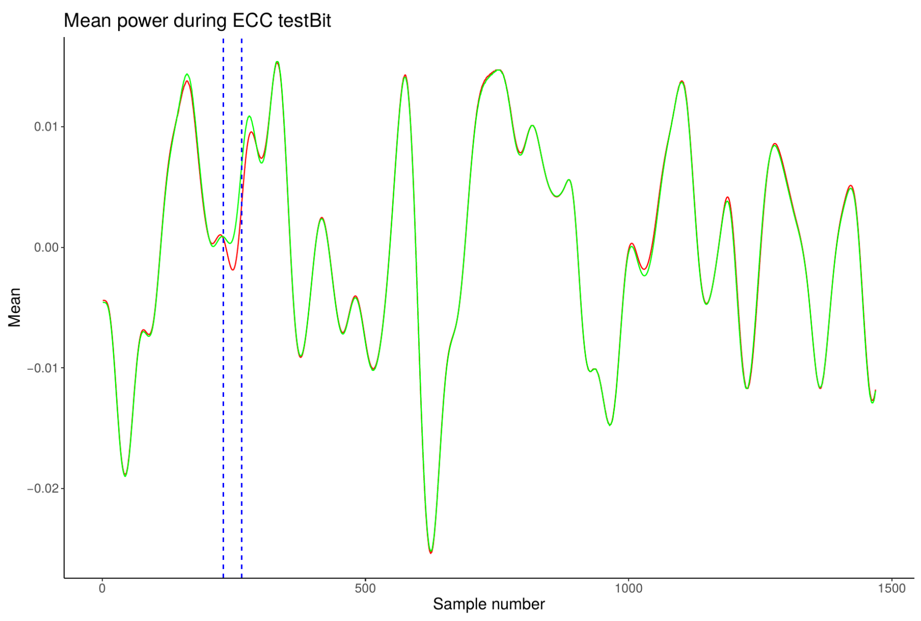


Figure 5.2: *Mean power consumption of the testBit function. The green line gives the mean of all traces for which the bit was 1. The red line gives the mean of all traces for which the bit was 0. Around sample 250 (marked by the dashed lines) we see a clear distinction between the two traces.*

```
12 mov.ne      r5,#0x0
13 rsb         r6,r5,#0x1
14 lsl         r6,r6,#0x5
15 lsl         r5,r5,#0x5
16 add.w       r9,r10,r6
17 add.w       r8,r10,r5
18 add         r6,r11
19 add         r5,r11
20 bl          trigger_low
```

If we study this assembly code, we observe a conditional branch at line 10. Branching is performed on `param_3`, which contains data from the secret scalar. In one case a register value `param_2` is moved in to `r5` (line 11), in the other case an intermediate value `0x0` is moved in to `r5` (line 12). Since these are slightly different operations, we expect this to be the main cause of the leakage which we see. Interestingly, this conditional is nowhere to be found in the original C code of the `uECC_vli_testBit()` function. As we can see below it only performs bitwise operations on the scalar (`vli`) and does not do any branching anywhere.

```c
/* Returns nonzero if bit 'bit' of vli is set. */
uECC_VLI_API uECC_word_t uECC_vli_testBit(const uECC_word_t *vli,
    bitcount_t bit) {
return (vli[bit >> uECC_WORD_BITS_SHIFT] & ((uECC_word_t)1 << (bit
    & uECC_WORD_BITS_MASK)));
}
```

Therefore we must conclude that the branching behavior was introduced by compiler optimizations. To verify this we compiled the library with compiler optimizations disabled. In this case no branch is introduced and no leakage can be found at the same point.

5.1.2 Leakage verification

In order to verify that this leakage is in no way related to or amplified by our trigger signal, we remove the trigger function from the code and check again whether we can find the same leakage. When we start moving the call to the trigger function around in the code, some unexpected changes happen to the assembly code generated by the compiler. When removing the trigger, the assembly code corresponding to the `uECC_vli_testBit()` function also changes. It even changes when we modify code below that line. An example of this behavior is given in figure 5.3.

```
XYcZ_initial_double(Rx[1], Ry[1], Rx[0],
      Ry[0], initial_Z, curve);
for (i = num_bits - 2; i > 0; --i) {
trigger_high();
nb = !uECC_vli_testBit(scalar, i);
trigger_low();
XYcZ_addC(Rx[1 - nb], Ry[1 - nb], Rx[nb
      ], Ry[nb], curve);
XYcZ_add(Rx[nb], Ry[nb], Rx[1 - nb], Ry
      [1 - nb], curve);
}
```

```
XYcZ_initial_double(Rx[1], Ry[1], Rx[0],
      Ry[0], initial_Z, curve);
for (i = num_bits - 2; i > 0; --i) {
trigger_high();
nb = !uECC_vli_testBit(scalar, i);

XYcZ_addC(Rx[1 - nb], Ry[1 - nb], Rx[nb
      ], Ry[nb], curve);
XYcZ_add(Rx[nb], Ry[nb], Rx[1 - nb], Ry
      [1 - nb], curve);
}
```

```
bl          trigger_high
asr         param_4,r4,#0x5
ldr         param_3,[sp,#local_130]
uxth        r4,r4
ldr.w       param_3,[param_3,param_4,
            lsl #0x2]
mov         param_2,#0x1
and         param_4,r4,#0x1f
lsl.w       param_4,param_2,param_4
tst         param_4,param_3
itE         eq
mov.eq      r5,param_2
mov.ne      r5,#0x0
rsb         r6,r5,#0x1
lsl         r6,r6,#0x5
lsl         r5,r5,#0x5
add.w       r9,r10,r6
add.w       r8,r10,r5
add         r6,r11
add         r5,r11
bl          trigger_low
```

```
bl          trigger_high
ldr         param_3,[sp,#local_130]
asr         param_4,r4,#0x5
uxth        r4,r4
ldr.w       param_3,[param_3,param_4,
            lsl #0x2]
str         r7,[sp,#0x0]=>local_140
and         param_4,r4,#0x1f
mov         param_2,#0x1
lsl.w       param_4,param_2,param_4
tst         param_4,param_3
itEET       ne
mov.ne      r6,#0x20
mov.eq      r6,#0x0
mov.eq      r5,#0x20
mov.ne      r5,#0x0
add.w       r9,r10,r6
add.w       r8,r10,r5
add         r6,r11
add         r5,r11
```

Figure 5.3: *Side by side comparison of the generated assembly code corresponding to the C code given above. The only difference in the C code on the right side is a lack of the line `trigger_low()`. In the assembly code various things change which already occur above the line which was removed.*

The most notable change is in the branch which previously caused a lot of leakage. This branch now performs two almost equivalent operations:

Before	After
`mov.eq   r5, param_2`	`mov.ne   r6, 0x20`
`mov.ne   r5, 0x0`	`mov.eq   r6, 0x0`
	`mov.eq   r5, 0x20`
	`mov.ne   r5, 0x0`

Due to this change we now no longer are able to distinguish a 0-bit or 1-bit at the previously found location on the trace.

5.1.3 A new attack without a trigger

In order to ensure that the trigger code does not influence the execution of the ECC multiplication we move all of the triggering code to the outside of the multiplication loop. The trigger is only called once before the start of the actual multiplication. We verified the correctness of the code by disassembling the compiled binary and comparing it with a version which did not contain any trigger at all. This time there were no differences in the section which performs the scalar multiplication. We therefore now can present an attack which should work on a unmodified version of MicroECC compiled with default compiler options.

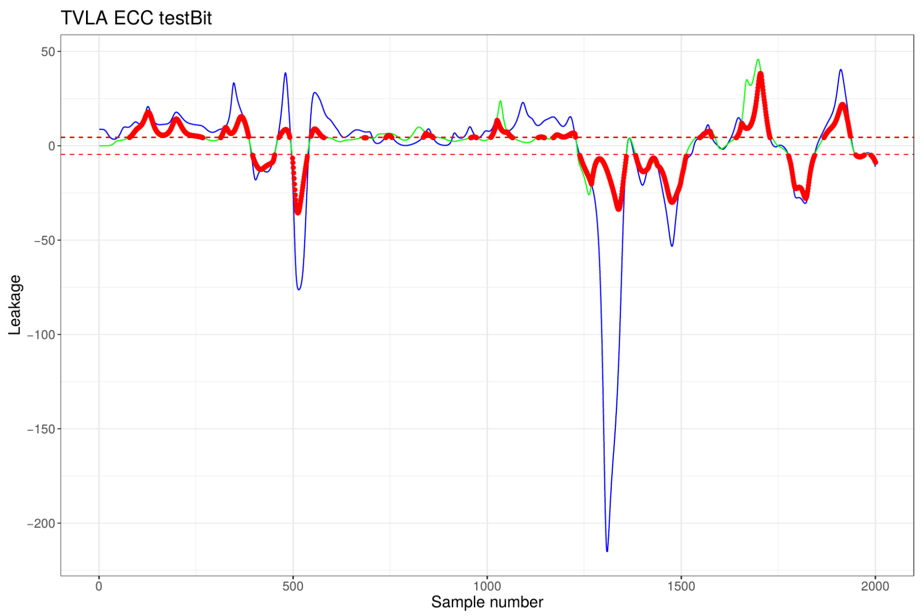


Figure 5.4: *TVLA of the uECC testBit function using an alternating bit. The TVLA was done using 1000 traces spread accross 2 runs. One line is given for each run.*

We again perform the same test as in Section 5.1. We increased the cut-off frequency of our filter to 4.5KHz as this gave us better results. The TVLA plot is given in figure 5.4. We also again plot the mean of each group of traces which can be found in figure 5.5. We see the greatest differences between the two traces starting at around 1200 samples. These captured traces could be used as

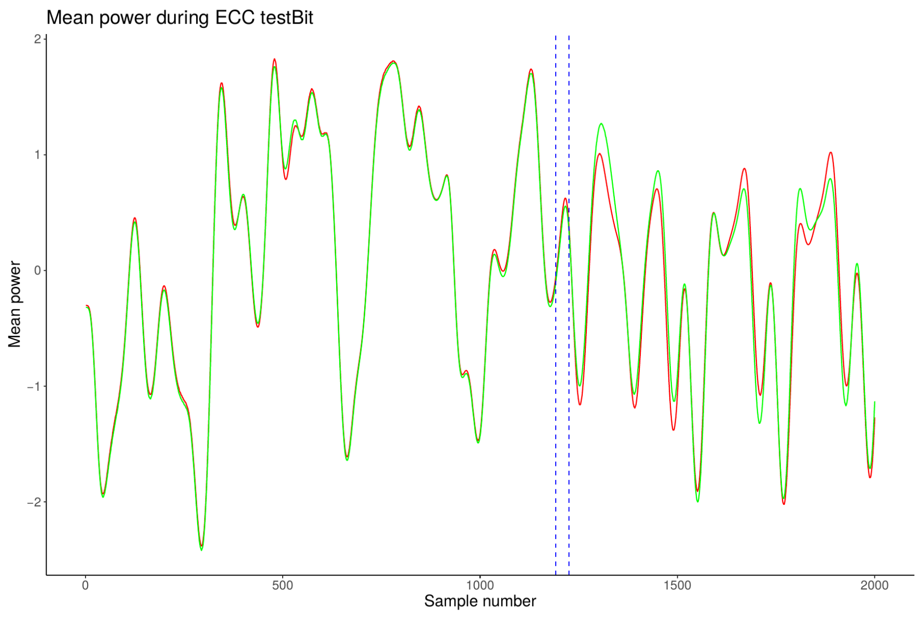


Figure 5.5: *Mean power consumption of the testBit function. The green line gives the mean of all traces for which the bit was 1. The red line gives the mean of all traces for which the bit was 0. The region between the dashed lines is the point of interest which we attack.*

a template for a template attack. However, we shall show that we can attack MicroECC without the need for a template.

5.1.4 A horizontal attack on MicroECC

To perfom the attack we capture one trace of an ECSM on the target the device. From this trace we extract the individual iterations of the Montgomery ladder loop. This is done by applying a low-pass filter to the trace with a cut-off frequency of 1KHz. This filter eliminates most differences between the iterations. Using the SAD metric from Section 3.6 we can compare whether two segments of the trace are similar. In this way we can search for repeating pattern in the trace and extract segments of 1000 samples corresponding to each iteration of the loop. We know that one loop iteration corresponds to around 400.000 samples. Therefore we search for segments which are aproximately 400.000 samples apart from eachother. Using this technique we can find a segment for almost all loop iterations. For a couple iterations we were not able to locate the segment. This is possibly caused by additional noise during the computation.

If we overlay all the segments, we obtain figure 5.6. If we take a closer look at the portion of the trace around sample 705 (figure 5.7, we can clearly see two distinct groups of traces. One group which corresponds to a 1-bit and one group which corresponds to a 0-bit. This separation can also be observed without coloring the traces, therefore it can be used to recover the bits of the private scalar. We used this attack to attack the first 58 bits of a private scalar used in an ECDH computation. We were able to recover 94% of the scalar bits with a 100% success rate. We repeated the attack with multiple public and private keys.

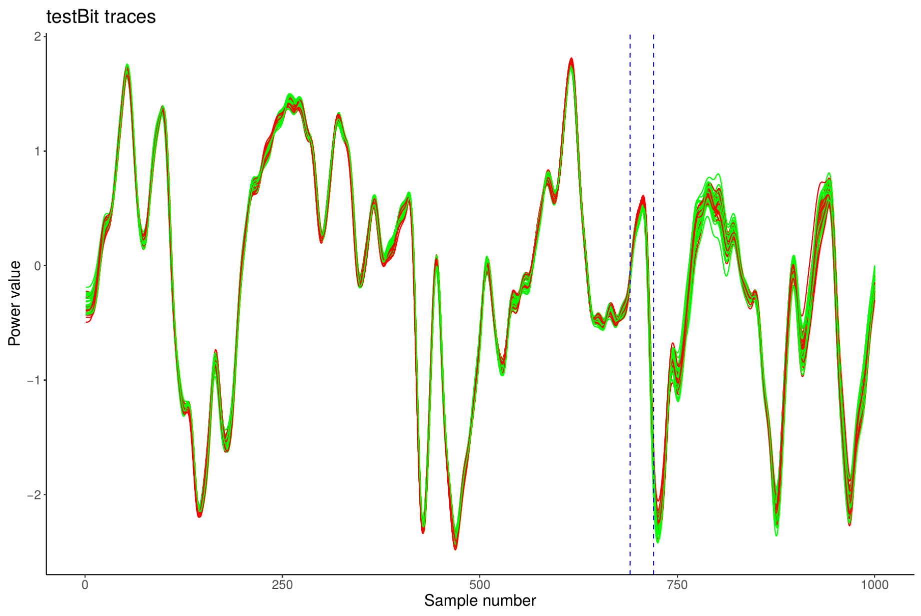


Figure 5.6: *Traces corresponding to the first 58 iterations of the scalar multiplication. Each line corresponds with 1 iteration. Lines corresponding to a 1-bit are green, lines corresponding to a 0-bit are red. The region between the dashed lines is the point of interest which we attack.*

The point which we are attacking corresponds to sample 1205 in the TVLA trace given in figure 5.4. The TVLA does indicate that there is leakage at this point, however the peaks are much larger at other points on the trace. The same can be seen in figure 5.5; the difference in means is much greater at other points. This raises the question whether this attack would not be more effective at another point on the trace. The answer to this question can be found if we look at the standard deviation between the different traces. The standard deviation between the different traces is given in figure 5.8. Here we see that at these points which show a large difference in mean there also is a large standard deviation. If we overlay the traces, this means that we will not see two clear groups of traces, but a wide spectrum of traces instead. Due to this effect the attack does not work at these points.

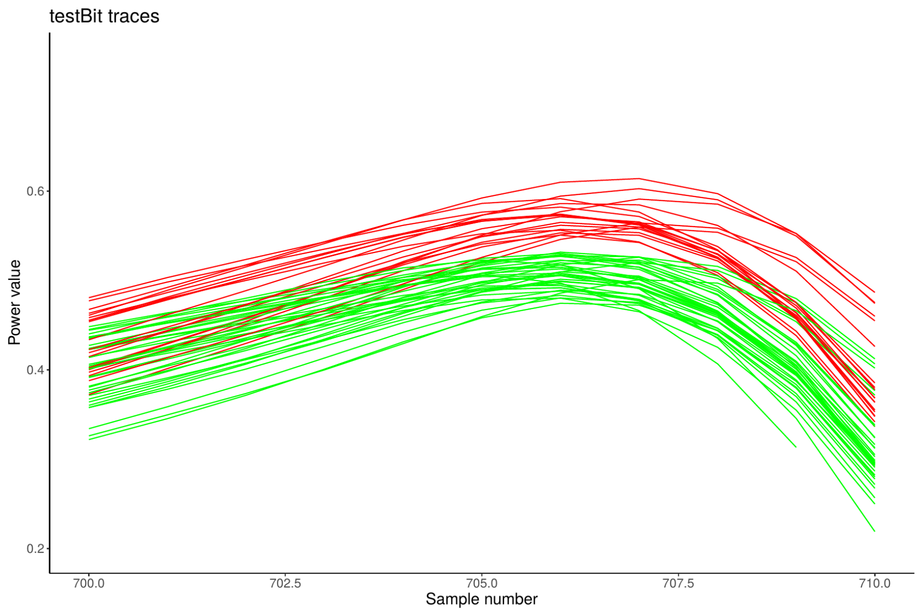


Figure 5.7: *Traces corresponding to the first 58 iterations of the scalar multiplication. Each line corresponds with 1 iteration. Lines corresponding to a 1-bit are green, lines corresponding to a 0-bit are red. The region between the dashed lines is the point of interest which we attack.*

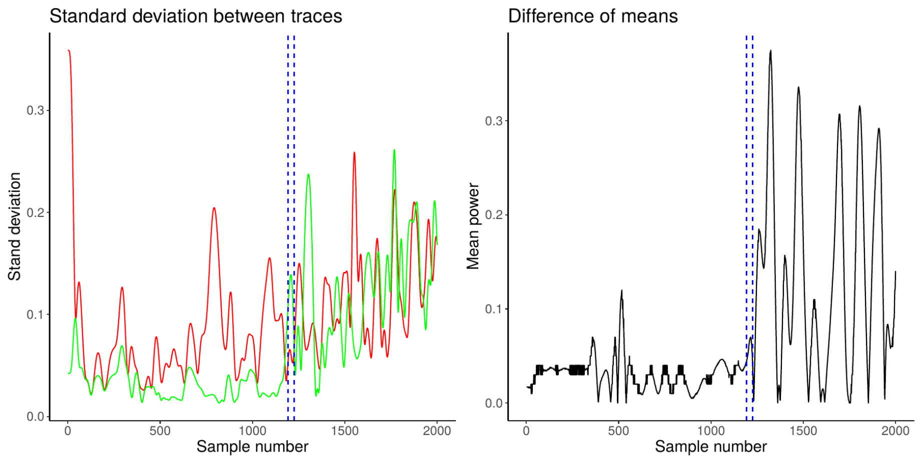


Figure 5.8: *Left: Standard deviation between traces in within the same group. The standard deviation for the group corresponding to a 1-bit are given in green, the group corresponding to a 0-bit in red. Right: The difference in means between the two groups of traces.*

Chapter 6

Conclusion

In this thesis we researched the side-channel security of two popular embedded cryptographic libraries: Mbed TLS and MicroECC. We first built two low-cost side-channel evaluation setups. We then successfully attacked the AES implementation in Mbed TLS. We showed which countermeasures were effective and how to use TVLA to verify this. We then also evaluated the side-channel security of MicroECC and were able to find a new side-channel vulnerability in this library.

6.1 TVLA conclusions

Our main question regarding TVLA was whether it was suitable as a general testing methodology for side-channel leakage. From our literature study we found that TVLA often gives false-positives, i.e. it exposes leakage which in no way can be exploited in a side-channel attack. Futhermore, TVLA is not suitable for testing asymmetric cryptography and very little work has been done on adapting it for this purpose. In our experiments we had the same findings as found in literature. In our analysis of AES we found leakage at many locations, whereas most known attacks only target very specific locations. In our analysis of ECC we had to adapt TVLA to use it on ECC. In the context of our horizontal attack we noticed another limitation of TVLA; it only tests for a difference in means between two groups of traces. In our attack a large difference of means is not very useful if the standard deviation between the traces is large.

6.1.1 Future work on TVLA

We gave a few pointers to improve TVLA as a testing method for resistance against template and horizontal attacks. Further research would be needed to develop a new testing methodology based on TVLA which can properly test for leakage which can be abused in a template attack or horizontal attack. The addition of the standard deviation to the test could be a good starting point for this.

6.2 AES conclusions

We can conclude that the AES implementation in Mbed TLS does not offer any protection against power-based side-channel attacks. However, it can easily be extended to include protection against side-channel attacks. The boolean masking scheme from Section 4.4.1 offers protection against first order side-channel attacks at a very low cost in terms of memory and clock cycles. It does require a very rigorous implementation approach; when implementing the masking scheme it is important to ensure that all intermediate values are also masked. The threshold scheme covered in Section 4.4.2 offers a provably secure masking of AES. The main advantage of this approach is that it can easily extended with multiple shares, offering protection against higher order side-channel attacks. Another advantage of this approach is that it is nearly impossible to implement it incorrectly. A major downside of this approach is the number of clock cycles which it consumes. Our un-optimized implementation takes around 184k clock cycles to encrypt a single block, whereas the original Mbed TLS does this in around 1k cycles. For resource constrained embedded devices such a large decrease in performance is not an option, making this not a very suitable software countermeasure. The hybrid scheme given in Section 4.4.3 offers the same level of protection as the threshold scheme, but does this at a lower cost in terms of clock cycles. The downside comes in terms of memory usage, it requires additional lookup tables which consume a significant amount of memory.

6.2.1 Future work on AES

Our work focused on creating proof-of-concept AES implementations containing the aforementioned countermeasures. All countermeasures were evaluated for protection against first order attacks using up to 10.000 traces. Due to time and resource constraints we did not do any tests using more than 10.000 traces, as these tests would take multiple days or weeks to complete and require significant amounts of data storage. For a higher level of assurance these tests should be repeated with more traces. The performance of the AES implementations also is an area which could be improved on. It could be investigated whether the hybrid scheme can be optimized enough to make it a viable option as a software-based countermeasure.

6.3 ECC conclusions

MicroECC required a new approach compared to AES. Due to the randomized projective coordinates DPA style attacks do not work. We therefore explored template attacks and horizontal attacks in the context of ECC. We found that we can recover almost the full ECC private key using a horizontal attack. We also discovered that in certain cases a compiler can generate assembly code which eases this attack.

6.3.1 Future work on ECC

It is not enitrely clear what causes the leakage which we are exploiting in our attack. One hypothesis is that it is caused by a memory access which depends on a secret value. Verifying this is a non-trivial operation and would need further work. Once the cause is known countermeasures for this attack should be explored and evaluated. A possible countermeasure introduced in [52] is storing data in memory addresses with the same hamming weight for two different addresses. Another possibility is the countermeasure introduced in [53], which randomizes the order of the iterations of the Montgomery ladder.

Bibliography

[1] *IoT market size worldwide 2017-2025 | Statista*. URL: https://www.statista.com/statistics/976313/global-iot-market-size/ (visited on 04/03/2020).

[2] *Flexible Key Provisioning with SRAM PUF | White Paper*. URL: http://go.intrinsic-id.com/flexible-key-provisioning-sram-puf-lp (visited on 04/03/2020).

[3] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. "Physical one-way functions". In: *Science* 297.5589 (2002), pp. 2026–2030. ISSN: 00368075. DOI: 10.1126/science.1074376.

[4] *Intrinsic ID - Home*. URL: https://www.intrinsic-id.com/ (visited on 04/03/2020).

[5] Paul Kocher, Joshua Jaffe, and Benjamin Jun. "Differential power analysis". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 1666. 1999, pp. 388–397. ISBN: 3540663479. DOI: 10.1007/978-1-4419-5906-5_196.

[6] *SSL Library mbed TLS / PolarSSL*. URL: https://tls.mbed.org/ (visited on 10/29/2019).

[7] *kmackay/micro-ecc: ECDH and ECDSA for 8-bit, 32-bit, and 64-bit processors*. URL: https://github.com/kmackay/micro-ecc (visited on 01/17/2020).

[8] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. "A testing methodology for side channel resistance validation". In: *NIST Workshop 2011*. 2011, pp. 1–12.

[9] Joan Daemen and Vincent Rijmen. "The Rijndael Block Cipher: AES Proposal". 2003. URL: http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf.

[10] NIST. "Processing Standards Publication 197: Advanced Encryption Standard". In: *Federal Information Processing Standards Publication* 197 (2001). URL: http://csrc.nist.gov/csor/.

[11] Morris Dworkin. "Recommendation for Block Cipher Modes of Operation Methods and Techniques". In: *National Institute of Standards and Technology Special Publication 800-38A 2001 ED* December (2001), p. 66. DOI: 10.6028/NIST.SP.800-38a.

[12] Neal Koblitz. "Elliptic Curve Public Key Cryptosystems". In: *MATHEMATICS OF COMPUTATION* 48.177 (1987), pp. 203–209. DOI: 10.1007/978-1-4615-3198-2.

[13] Peter L. Montgomery. “Speeding the Pollard and elliptic curve methods of factorization”. In: *Mathematics of Computation* 48.177 (1987), pp. 243–243. ISSN: 0025-5718. DOI: 10.1090/s0025-5718-1987-0866113-7.

[14] Alfred Menezes. *Elliptic Curve Public Key Cryptosystems*. Springer US, 1993. DOI: 10.1007/978-1-4615-3198-2.

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

[16] Ronald H Brown, Mary L Good, and Arati Prabhakar. “Federal Information Processing Standards Publication: digital signature standard (DSS)”. In: (1994).

[17] Elaine Barker, Don Johnson, and Miles Smid. “Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography”. In: *NIST Special Publication* 800.56A (2006). DOI: 10.6028/NIST.SP.800-56ar. URL: http://csrc.nist.gov/groups/ST/toolkit/key{_}management.html.

[18] Whitfield Diffie, Whitfield Diffie, and Martin E Hellman. “New Directions in Cryptography”. In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654. ISSN: 15579654. DOI: 10.1109/TIT.1976.1055638.

[19] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis attacks: Revealing the secrets of smart cards*. Springer US, 2007, pp. 1–337. ISBN: 0387308571. DOI: 10.1007/978-0-387-38162-6.

[20] Hongying Liu, Guoyu Qian, Satoshi Goto, and Yukiyasu Tsunoo. “AES key recovery based on Switching Distance model”. In: *3rd International Symposium on Electronic Commerce and Security, ISECS 2010*. 2010, pp. 218–222. DOI: 10.1109/ISECS.2010.55.

[21] Hassen Mestiri, Noura Benhadjyoussef, Mohsen Machhout, and Rached Tourki. “A Comparative Study of Power Consumption Models for CPA Attack”. In: *International Journal of Computer Network and Information Security* 5.3 (2012), pp. 25–31. ISSN: 20749090. DOI: 10.5815/ijcnis.2013.03.03.

[22] Paul C. Kocher. “Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 1109. Springer Verlag, 1996, pp. 104–113. ISBN: 3540615121. DOI: 10.1007/3-540-68697-5_9.

[23] Eric Brier, Christophe Clavier, and Francis Olivier. “Correlation power analysis with a leakage model”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3156 (2004), pp. 16–29. ISSN: 03029743. DOI: 10.1007/978-3-540-28632-5_2.

[24] François Xavier Standaert, Tal G Malkin, and Moti Yung. “A unified framework for the analysis of side-channel key recovery attacks”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 5479 LNCS. 2009, pp. 443–461. ISBN: 3642010008. DOI: 10.1007/978-3-642-01001-9_26.

[25] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. “Template Attacks”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes*

in Artificial Intelligence and Lecture Notes in Bioinformatics) 2523 (2003), pp. 13–28. ISSN: 03029743. DOI: 10.1007/3-540-36400-5_3.

[26] Lejla Batina, Łukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, and Michael Tunstall. "Online template attacks". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 8885. Springer Verlag, 2014, pp. 21–36. ISBN: 9783319130385. DOI: 10.1007/978-3-319-13039-2_2.

[27] Lejla Batina, Łukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, and Michael Tunstall. "Online template attacks". In: *Journal of Cryptographic Engineering* 9.1 (2019), pp. 21–36. ISSN: 2190-8508. DOI: 10.1007/s13389-017-0171-8.

[28] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. "Horizontal correlation analysis on exponentiation". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6476 LNCS. 2010, pp. 46–61. ISBN: 3642176496. DOI: 10.1007/978-3-642-17650-0_5.

[29] G Becker, J Cooper, E Demulder, G Goodwill, J Jaffe, G Kenworthy, T Kouzminov, A Leiserson, P Rohatgi, and S Saab. "Test Vector Leakage Assessment (TVLA) methodology in practice". In: 2013.

[30] Thomas S. Messerges. "Using Second-Order Power Analysis to Attack DPA Resistant Software". In: *Cryptographic Hardware and Embedded Systems*. 2000, pp. 238–251.

[31] Tobias Schneider, · Amir Moradi, Amir Moradi, and Moradi@rub De. "Leakage assessment methodology Extended version". In: 6 (2016), pp. 85–99. DOI: 10.1007/s13389-016-0120-y.

[32] "Information technology - Security techniques - Testing methods for the mitigation of non-invasive attack classes against cryptographic modules". 2012. URL: https://www.iso.org/standard/60612.html.

[33] Carolyn Whitnall and Elisabeth Oswald. *A Critical Analysis of ISO 17825 ('Testing methods for the mitigation of non-invasive attack classes against cryptographic modules')*. Tech. rep. URL: https://csrc.nist.gov/Projects/cryptographic-module-validation-program/.

[34] A Adam Ding, Liwei Zhang, Francois Durvaux, Francois Xavier Standaert, and Yunsi Fei. "Towards sound and optimal leakage detection procedure". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 10728 LNCS. 2018, pp. 105–122. ISBN: 9783319752075. DOI: 10.1007/978-3-319-75208-2_7.

[35] Thomas S. Messerges. *Securing the AES finalists against power analysis attacks*. Tech. rep. 2001, pp. 150–164. DOI: 10.1007/3-540-44706-7_11.

[36] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. "An AES smart card implementation resistant to power analysis attacks". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 3989 LNCS. 2006, pp. 239–252. ISBN: 3540347038. DOI: 10.1007/11767480_16.

[37] Mehdi Laurent Akkar and Christophe Giraud. "An implementation of DES and AES, secure against some attacks". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lec-*

ture Notes in Bioinformatics). Vol. 2162. Springer Verlag, 2001, pp. 309–318. ISBN: 3540425217. DOI: 10.1007/3-540-44709-1_26.

[38] Jovan D. Golić and Christophe Tymen. "Multiplicative Masking and Power Analysis of AES". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2523 (2003), pp. 198–212. ISSN: 03029743. DOI: 10.1007/3-540-36400-5_16.

[39] Johannes Blömer, Jorge Guajardo, and Volker Krummel. "Provably secure masking of AES". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3357 (2004), pp. 69–83. ISSN: 03029743. DOI: 10.1007/978-3-540-30564-4_5.

[40] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. "Towards sound approaches to counteract power-analysis attacks". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 1666. Springer Verlag, 1999, pp. 398–412. ISBN: 3540663479. DOI: 10.1007/3-540-48405-1_26.

[41] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. "A more efficient AES threshold implementation". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 8469 LNCS. 2014, pp. 267–284. ISBN: 9783319067339. DOI: 10.1007/978-3-319-06734-6_17.

[42] Adi Shamir. "How to Share a Secret". In: *Communications of the ACM* 22.11 (1979), pp. 612–613. ISSN: 15577317. DOI: 10.1145/359168.359176.

[43] G. R. Blakley. "Safeguarding cryptographic keys". In: *1979 International Workshop on Managing Requirements Knowledge, MARK 1979.* Institute of Electrical and Electronics Engineers Inc., 1979, pp. 313–317. ISBN: 9781509031818. DOI: 10.1109/MARK.1979.8817296.

[44] Jean Sébastien Coron. "Resistance against differential power analysis for elliptic curve cryptosystems". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 1717. 1999, pp. 292–302. ISBN: 354066646X. DOI: 10.1007/3-540-48059-5_25.

[45] Claude E. Shannon. "Communication in the Presence of Noise". In: *Proceedings of the IRE* 37.1 (1949), pp. 10–21. ISSN: 00968390. DOI: 10.1109/JRPROC.1949.232969.

[46] NewAE. *CW1173 ChipWhisperer-Lite - ChipWhisperer Wiki.* URL: https://wiki.newae.com/CW1173{_}ChipWhisperer-Lite (visited on 10/15/2019).

[47] *GitHub - kokke/tiny-AES-c: Small portable AES128/192/256 in C.* URL: https://github.com/kokke/tiny-AES-c (visited on 02/06/2020).

[48] *GNU Toolchain | GNU-RM Downloads – Arm Developer.* URL: https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads (visited on 03/27/2020).

[49] *Choosing an Oscilloscope.* URL: https://www.picotech.com/library/application-note/oscilloscope-tutorial (visited on 03/27/2020).

[50] S. Butterworth. "On the Theory of Filter Amplifiers". In: *The Wireless Engineer* (1930), pp. 536–541.

[51] Matthieu Rivain and Emmanuel Prouff. “Provably secure higher-order masking of AES”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6225 LNCS. 2010, pp. 413–427. ISBN: 3642150306. DOI: 10.1007/978-3-642-15031-9_28.

[52] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenaka. “Address-Bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2523 (2003), pp. 129–143. ISSN: 03029743. DOI: 10.1007/3-540-36400-5_11.

[53] Duc Phong Le, Chik How Tan, and Michael Tunstall. “Randomizing the montgomery powering ladder”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9311. 2015, pp. 169–184. ISBN: 9783319240176. DOI: 10.1007/978-3-319-24018-3_11.

Appendix A

Threshold multiplication scheme correctness proof

By induction we prove that

$$\bigoplus_{j=0}^{n} o_j = (x_0 \oplus \cdots \oplus x_n) \cdot (y_0 \oplus \cdots \oplus y_n) \tag{A.1}$$

given that x and y both are split in to n shares.

Base case: n=1 For $n = 1$ this clearly holds:

$$\begin{aligned}
\bigoplus_{j=0}^{1} o_j &= (x_0 y_0 \oplus \bigoplus_{i=1}^{1} r_{0,i}) \oplus (x_1 y_1 \oplus \bigoplus_{i=0}^{0} t_{i,1}) && \text{(A.2)}\\
&= x_0 y_0 \oplus r_{0,1} \oplus x_1 y_1 \oplus t_{0,1} && \text{(A.3)}\\
&= x_0 y_0 \oplus r_{0,1} \oplus x_1 y_1 \oplus (r_{0,1} \oplus x_0 y_1) \oplus x_1 y_0 && \text{(A.4)}\\
&= x_0 y_0 \oplus x_1 y_1 \oplus x_0 y_1 \oplus x_1 y_0 && \text{(A.5)}\\
&= (x_0 \oplus x_1)(y_0 \oplus y_1) && \text{(A.6)}\\
& && \text{(A.7)}
\end{aligned}$$

Induction step If our equation A.1 holds for some n, then we shall now show that it also holds for $n + 1$.

$$\begin{aligned}
\bigoplus_{j=0}^{n+1} o_j =& \bigoplus_{j=0}^{n+1}(x_j \cdot y_j \oplus \bigoplus_{i=0}^{j-1} t_{i,j} \oplus \bigoplus_{i=j+1}^{n+1} r_{j,i}) \\
=& \bigoplus_{j=0}^{n}(x_j \cdot y_j \oplus \bigoplus_{i=0}^{j-1} t_{i,j} \oplus \bigoplus_{i=j+1}^{n} r_{j,i}) \oplus \bigoplus_{j=0}^{n} r_{j,n+1} \oplus x_{n+1}y_{n+1} \oplus \bigoplus_{i=0}^{n} t_{i,n+1} \\
=& \bigoplus_{j=0}^{n} o_j \oplus \bigoplus_{j=0}^{n} r_{j,n+1} \oplus x_{n+1}y_{n+1} \oplus \bigoplus_{i=0}^{n} t_{i,n+1} \\
=& (x_0 \oplus \cdots \oplus x_n) \cdot (y_0 \oplus \cdots \oplus y_n) \oplus (r_{0,n+1} \oplus \cdots \oplus r_{n,n+1}) \\
& \oplus x_{n+1}y_{n+1} \oplus (t_{0,n+1} \oplus \cdots \oplus t_{n,n+1}) \\
=& (x_0 \oplus \cdots \oplus x_n) \cdot (y_0 \oplus \cdots \oplus y_n) \oplus (r_{0,n+1} \oplus \cdots \oplus r_{n,n+1}) \\
& \oplus x_{n+1}y_{n+1} \oplus ((r_{0,n+1} \oplus \cdots \oplus r_{n,n+1}) \oplus y_{n+1}(x_0 \oplus \cdots \oplus x_n) \\
& \oplus x_{n+1}(y_0 \oplus \cdots \oplus y_n)) \\
=& (x_0 \oplus \cdots \oplus x_n) \cdot (y_0 \oplus \cdots \oplus y_n) \\
& \oplus x_{n+1}y_{n+1} \oplus y_{n+1}(x_0 \oplus \cdots \oplus x_n) \oplus x_{n+1}(y_0 \oplus \cdots \oplus y_n) \\
=& (x_0 \oplus \cdots \oplus x_{n+1}) \cdot (y_0 \oplus \cdots \oplus y_{n+1})
\end{aligned}$$