Power analysis on AES and LoRaWAN

Robin Quetin

Promotor: Prof. Dr. Peter Quax
Co-promotor: Prof. Dr. Wim Lamotte
Supervisor: Mr. Pieter Robyns

A thesis submitted in partial fulfillment of the requirements for the degree of Master in Computer Science

23 August 2018

Abstract

For years cryptoanalysis has been a method to attack and determine the strength of a cryptographic algorithm. Side-channel attacks introduce an alternate method of attacking cryptographic algorithms by targeting the implementation of the computer system rather than the algorithm itself. One of these side-channel attacks is the power analysis attack and has shown to be successful in attacking various device implementations using cryptographic algorithms which are considered to be secure. One of these victims is the popular and well-used AES algorithm. AES is used in various contexts such as the encryption of Web traffic and wireless network communication. Power analysis attacks could thus provide a method to break the security of these communication channels.

However, the practicality and effectiveness of these attacks in practice are often undocumented. In this thesis, the author will try to answer to question: "Is the AES algorithm in modern IoT appliances vulnerable for power analysis attacks?" and "Are power analysis attacks usable in practice?". To answer these questions, a detailed description about the AES algorithm and power analysis attacks is provided in this thesis. Next, several case studies are performed to address the practicality and effectiveness of the power analysis attacks. During these case studies the author was able to conclude that the quality of the measurements directly affects the effectiveness of the power analysis attacks. The quality of the measurements relies on several conditions such as electrical noise and measurement set-ups. Due to problems involving these conditions, the attacks on modern IoT appliances could not be performed and a conclusion regarding the vulnerability in these appliances could not be formed.

Acknowledgements

I would like to thank my promotor Peter Quax and my supervisor Pieter Robyns for their support and feedback on my thesis. I would also like to thank my family, friends and my student colleagues for supporting me during the realisation of this thesis. In particular, I would like to thank to my parents, who allowed me to study at the University and supported me through every decision I have made.

Contents

List of Figures

List of Tables

Glossary

application key is a cryptographic key specified by the LoRaWAN specification. The key should be unique per device and is used to derive the session keys from in a LoRaWAN network setting. 21

application session key is a cryptographic key specified by the LoRaWAN specification. Up until version 1.0.2 of the protocol, its primary purpose was to act as the secret key for the AES encryption of application specific data in LoRaWAN MAC messages. 20

ATmega328P is a programmable AVR based microcontroller created by Atmel with an 8-bit processor. 41, 47, 50, 54, 58–62, 64

ciphertext is the result of performing one or more cryptographic algorithms on plaintext information. Ciphertext is often uninterpretable in its direct form; the ciphertext must be decrypted before it can be interpreted by humans or applications. 2

Diffie-Hellman is a key establishment algorithm originally conceptualized by Ralph Merkle and later described by Whitfield Diffie and Martin Hellman. It allows two entities to establish a shared secret key over an insecure channel, which can later be used for symmetric encryption algorithms. 3

Fast-Fourier transformation is a transformation which takes as an input a signal over a period of time or space and extracts the frequency components from the signal. Eventually the signal is converted from a time or space domain to a frequency domain. 40, 44

initialisation vector is a fixed-sized data vector usually containing random or pseudorandom values that will be used to initialize a cryptographic primitive. 13, 14

Internet-of-Things is a term used to describe all devices which are in a way networked to the Internet. 1, 12

LoRaWAN is a MAC protocol designed to work in combination with the LoRa physical protocol. 1, 18–21, 64, 67

Low Power Wide Area Network is a network type which allows network connectivity over a wide area while being optimized to use only a small mount of power.. 1

Message Integrity Code is a piece of information which allows the receiver of the message to verify the authenticity and integrity of the message. 21

network key is a cryptographic key specified by the LoRaWAN specification v1.1. The key should be unique per device and is used to derive the network session keys from in a LoRaWAN network setting. 21

network session key is a cryptographic key specified by the LoRaWAN specification. Up until version 1.0.2 of the protocol, its primary purpose was to act as the secret key for the AES encryption of network specific information in LoRaWAN MAC messages. 20, 21

nonce is an arbitrary vector of data that can only be used once. 17, 20

plaintext is unencrypted, interpretable information that is in a directly usable form. 2

Rijndael is the name for a cryptographic algorithm invented by Joan Daemen and Vincent Rijmen. It is often associated with the Advanced Encryption Standard which is effectively a subset of Rijndael. 6, 8–10

RSA is an asymmetric cryptographic system designed by Ron Rivest, Adi Shamir, and Leonard Adleman. 3, 4

Acronyms

AV_{cc} ADC voltage common collector. 58
V_{cc} voltage common collector. 58, 69
`AppEUI` Application Extended Unique Identifier. 20, 21
`AppKey` application key . 20, 21, *Glossary:* application key
`AppNonce` application nonce. 20
`AppSKey` application session key . 20, 21, *Glossary:* application session key
`DevEUI` Device Extended Unique Identifier. 20
`DevNonce` device nonce. 20, 21
`FHDR` frame header. 21
`FNwkSIntKey` Forwarding Network Session Integrity Key. 21
`FOpts` frame options. 20, 21
`FPort` frame port. 20, 21
`FRMPayload` frame payload. 20, 21
`JoinEUI` Join Extended Unique Identifier. 21
`MHDR` MAC header. 21
`NetID` Network ID. 20
`NwkKey` network key . 21, *Glossary:* network key
`NwkSEncKey` network session encryption key. 21
`NwkSKey` network session key . 20, 21, *Glossary:* network session key
`SNwkSIntKey` Serving Network Session Integrity Key. 21
`TxCh` transmission channel. 22
`TxDr` transmission data rate. 22
1O-DPA first-order differential power analysis attack. 34, 38

ABP Activation-by-Personalisation. 20
ADC analogue-to-digital converter. 41, 42, 58
AES Advanced Encryption Standard. i, 1, 3, 4, 6–10, 12, 16, 17, 20, 21, 23, 27, 31, 36–38, 47–52, 54–56, 59–64, 66, 67, 72, 74, 76
ASCII American Standard Code for Information Interchange. 47

BNC Bayonet-Neill-Concelman. 44

CBC Cipher Block Chaining. 13, 14, 16, 17, 27, 36–38, 50, 54, 55, 59, 60, 63, 72, 74, 76
CBC-MAC CBC MAC. 16, 17, 21
CCM Counter with Cipher Block Chaining-Message Authentication Code. 12, 16, 47, 64, 67
CFB Cipher Feedback. 13, 14
CIA triad Confidentiality-Integrity-Availability triad. 2, 4
CMOS Complementary Metal Oxide Semiconductor. 27, 28, 35, 40
CPA correlation power analysis. 34, 36, 37, 48, 53–57, 60, 61, 64, 67
CTR Counter. 15–17, 20, 37, 38

DDoS Distributed Denial-of-Service . 4, *Glossary:* Distributed Denial-of-Service

DES Data Encryption Standard. 6, 23

DH Diffie-Hellman . *Glossary:* Diffie-Hellman

DPA differential power analysis. 33, 34, 36, 38, 39, 66

DSO digital storage oscilloscope. 40, 41, 43, 47

ECB Electronic Codebook. 12

EM electro-magnetic. 49, 67

FFT Fast-Fourier transformation . 40, 44, *Glossary:* Fast-Fourier transformation

GND ground. 58, 59

GPIO General-Purpose Input Output. 43, 46, 58, 59, 62, 63, 76

HO-DPA higher-order differential power analysis attack. 34, 40

I^{2}C Inter-Integrated Circuit. 43

IoT Internet-of-Things . i, 1, *Glossary:* Internet-of-Things

IV initialization vector . 13, 14, 17, 37, 38, 54, 59, *Glossary:* initialisation vector

KVL Kirchhoff's Voltage Law. 69, 70

LPWAN Low Power Wide Area Network . 1, *Glossary:* Low Power Wide Area Network

LXI LAN eXtensions for Instrumentation. 43, 47, 48, 58

MAC message authentication code. 5, 16, 17

MD5 Message Digest Algorithm 5. 5

MHz megahertz. 40–42

MIC Message Integrity Code . 21, 22, 65, *Glossary:* Message Integrity Code

MSE mean squared error. 47

OFB Output Feedback. 14, 15

OTAA Over-The-Air Activation. 20

PGE partial guessing entropy. 48, 52–54, 56, 57, 60, 61, 63, 64

RS-232 Recommended Standard 232 . 43, 47, 58–62, *Glossary:* RS-232

S-box substitution box. 38

SCPI Standard Commands for Programmable Instruments. 47

SHA Secure Hash Algorithm. 5

SPA simple power analysis. 29–31, 34

SRAM Static Random Access Memory. 27, 28

TCP Transmission Control Protocol. 47

USB Universal Serial Bus. 43, 45, 47, 65

USBTMC USB Test & Measurement Class. 43, 47

VISA Virtual Instrument Software Architecture. 47

VXI-11 VME eXtensions for Instrumentation. 43

XOR exclusive OR. 6, 8–10, 12, 14, 15, 37–39

Chapter 1

Introduction

For several centuries cryptography has been a means to provide two parties with a secure method of communicating to each other over insecure communication channels. Until the last century the usage of cryptography was mainly limited to verbal and written communication. But with the introduction of the computer and publicly accessible networks, the role of cryptography has changed dramatically. Nowadays, cryptography is one of the key components in providing secure communication over the Internet. According to studies, over 60 percent of the websites accessed by Google Chrome is securely transferred using cryptographic algorithms [Goo18]. While Web traffic is the most commonly known example of how cryptographic algorithms are being used in a modern-day context, there are numerous other examples in which it is used to provide security over insecure channels such as wireless networks like WiFi and cellular networks.

One of the most commonly used cryptographic algorithms is the Advanced Encryption Standard (AES) algorithm. It is used in various applications such as the encryption of Web traffic and wireless protocols. While invented in 1999, the AES algorithm is still considered to be a secure encryption algorithm. Previous studies have not (yet) been able to break the algorithm using cryptanalysis. However, several years ago a new form of attacks called side-channel attacks have been introduced in the field of computer security. Instead of targeting the algorithm itself, these attacks focus on the implementation of the device. Side-channel attacks have shown to be successful in the past against algorithms that were considered to be cryptographically secure [KJJ99] [Ber05] [DKH13]. Since the implementation of devices is dependant on the manufacturer and often unregulated in terms of security, insecure devices can still exist even if the cryptographic algorithms are considered secure.

Because AES is so widely used in the Internet-of-Things context, this thesis will focus on the security aspect of AES against power analysis attacks. The main research questions of this thesis are: ”Is the AES algorithm in modern IoT appliances vulnerable for power analysis attacks?” and ”Are power analysis attacks usable in practice?”. In order to answer this question, the inner workings of the AES algorithm will be first described thoroughly in section 2.2. Chapter 3 introduces the concept of side-channel attacks. One category of side-channel attacks, the power analysis attacks, will be described in more detail. The power analysis attacks will also be tested in practice according to an incremental approach of case studies. The final goal is to perform the power analysis attacks on a network component which uses a wireless protocol intended for Internet-of-Thingss applications. For this final test, the LoRaWAN protocol, a promising Low Power Wide Area Network (LPWAN) protocol specifically designed for Internet-of-Thingss applications, will be the targeted wireless networking protocol. The LoRaWAN protocol will be further described in section 2.4.

Chapter 2

AES and LoRa(WAN)

2.1 Introduction to cryptography

To the general population it is often unclear how much cryptography is involved in daily applications. Since the existence of basic communication, people have sought to communicate with each other in such a way that unwanted individuals (also referred to as *adversaries*) could not overhear their conversations. One possibility to keep this communication private is by using cryptography. Cryptography is a field in information science that is dedicated to the practice and study of techniques in order to provide secure communication between two parties in the presence of adversaries [Lee90]. This field has become increasingly popular since the introduction of publicly accessible mediums such as radio and the Internet.

The most commonly known part of cryptography is probably the encryption process. Encryption is the process of transforming unencrypted data or plaintext into encrypted data or ciphertext [And14]. However, encryption alone is not able to provide secure communication. Modifying the original communication through interception or blocking communication could also lead to a breach in secure communication. In the field of computer security, experts often refer to the CIA triad as a guideline to address the level of security in a system [And14]. The CIA triad stands for the following principles:

- **Confidentiality:** confidentiality deals with the principle that the information or resources should only be available to the allowed entities [And14]. This also means that even if an entity is deemed trustworthy, it should not be able to access the information or resources if it was not granted access to the resource. Authentication and authorization checks play an important role in the realisation of this principle. Encryption itself can be a measure to extend confidentiality: if a malicious party is unable to use the information or resource due to the encryption, it will be also unable to access the information or resource.
- **Integrity:** integrity provides the involved parties with certainty that the information or resource is indeed transferred and remains identical during the delivery from sender to receiver [Sta05]. Measurements such as hashes provide the users with a means to validate if the information or resource has remained unchanged.
- **Availability:** availability revolves around the fact that resource or information should remain accessible by the allowed entities at all times [Sta05]. This concept is often difficult to place in the context of security. However, if valid users cannot access the resource or information, then the system itself cannot fulfil its purpose and will prevent end-users from performing their jobs.

2.1.1 Encryption and decryption

As mentioned before, encryption is the process of transforming *plaintext* into *ciphertext* [And14]. The process is often formulated as an algorithm and thus called an *encryption algorithm*. However, without a way to transform the encrypted information back to its original form, this process would make it impossible for others to interpret the information, even for a legitimate receiver. For that reason another algorithm called the *decryption algorithm* is also specified as a means to revert the scrambled information back to its original form. In cryptography, these algorithms are referred to as *ciphers* [And14].

To prevent that a cipher would always yield the same result for a given plaintext, ciphers will allow the user to specify extra parameters which will affect the obtained result [And14]. The most common example is the use of one or multiple secret keys. A secret key is a string of bits which will be used during the encryption process to transform the plaintext into ciphertext. Depending on the type of encryption algorithm, the secret key or a related decryption key will be shared amongst the receivers of the ciphertext. Only the owners of the decryption key(s) will then be able to decipher the ciphertext; the use of other keys will yield an invalid result.

Arguably the easiest way of using keys in an encryption context is by using the same keys for both the encryption and decryption process. This type of encryption is also referred to as *symmetric key* or *single-key encryption* [Sta05]. However, this introduces the issue of keeping the secret key(s) confidential between the trusted participants, as owning the secret key would allow the owner to both encrypt and decrypt the communication. This is especially difficult when the keys still need to be exchanged between the participants. Key exchange protocols such as Diffie-Hellman have been defined which can provide a secure method for establishing a secret key between participants [Mer78] [DH76]. As an alternative, encryption algorithms were invented which used separate keys for encrypting and decrypting data. Such algorithms are called *asymmetric key* or *public key encryption* algorithms. These algorithms generally provide the receiver with a secret or *private key* which is used to decrypt incoming data and provide the senders with one or more *public keys* derived from the private key which in turn can be used to encrypt the data. This results in a one-way encryption: using a single key pair the sender can only encrypt data and the receiver can only decrypt data.

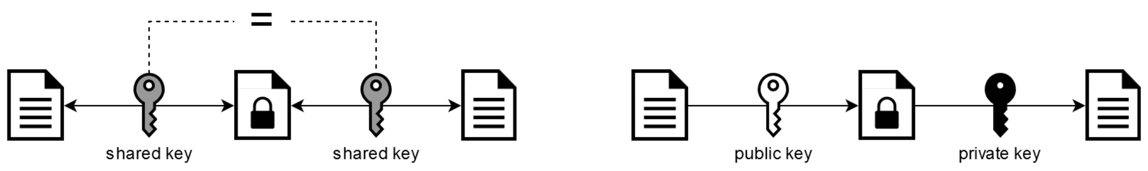


Figure 2.1: A schematic overview of how a symmetric key algorithm (left) and an asymmetric key algorithm (right) work. The arrows indicate the direction in which encryption and decryption are possible.

While it can be argued that the asymmetric encryption system provides better security due to the separate purpose of keys, it does come at the disadvantage of a higher computational cost. Key generation for public-key algorithms is often met with high computation times due to the mathematical problems they are based on. At the moment of writing, these mathematical problems have no efficient solution available [And14][Sta05]. Encryption and decryption of data blocks is also slower. A performance test to support this statement is given by listing 2.1. Another disadvantage of public-key algorithm is that they often require larger key sizes to provide an equivalent level of security compared to symmetric algorithms. In [Bar16], it was advised to use a 3072-bit key for integer-factorization related asymmetric algorithms such as RSA to provide the same algorithm strength as the AES symmetric algorithm provides with a 128-bit key. For these reasons, asymmetric encryption is primarily used to encrypt small portions of data (such as secret keys) and small battery-powered devices often use symmetric encryption algorithms instead to encrypt their data.

```
$ openssl.exe speed aes
...
Doing aes-128 cbc for 3s on 1024 size blocks: 385612 aes-128 cbc's in
    3.00s
...
Doing aes-192 cbc for 3s on 1024 size blocks: 320795 aes-192 cbc's in
    3.00s
...
Doing aes-256 cbc for 3s on 1024 size blocks: 275300 aes-256 cbc's in
    3.00s

$ openssl.exe speed rsa1024
Doing 1024 bit private rsa's for 10s: 59368 1024 bit private RSA's in
    9.98s
Doing 1024 bit public rsa's for 10s: 866784 1024 bit public RSA's in
    10.00s
```

Listing 2.1: Performance test performed with OpenSSL 1.0.2o to compare the performance between RSA (an asymmetric algorithm) and AES (a symmetric algorithm). AES performs better than RSA even when larger keys are being used.

2.1.2 Data integrity and authentication

While encryption provides a method to partially guarantee confidentiality[1]of communication between two parties, it will be insufficient to provide a completely secure way of communication. The CIA triad also mentions the integrity and availability as essential principles within a security context. Encryption alone cannot guarantee that the communication data will not be changed or will actually reach the receiver.

As stated in section 2.1, availability revolves around the fact that the resource or information should remain accessible by the allowed entities [And14]. Solutions to ensure availability are more often found in hardware and application software instead of cryptography. Common examples of such solutions include DDoS protection hardware and load-balancing hardware or cloud solution. However, cryptography can be a means to improve integrity in a security context. Encryption itself partially provides integrity as a successful decryption of a ciphertext will prove that the sender has the same key as or a key derived from the receivers secret key [Sta05]. However, this means that the receiver needs to have an idea about how the plaintext is supposed to be formatted (e.g. a correct English sentence). In case of random byte values without a checksum, this method cannot guarantee integrity since there is no fixed data pattern [Sta05]. Therefore, other cryptographic algorithms exist which are able to verify the authenticity and integrity of data regardless of their formatting. The two most common techniques to provide data integrity using cryptography will be explained in the following paragraphs.

Message digest

A message digest is a fixed-length data string which is calculated based on a given message [Sta05]. Message digests are also known as hash codes or hash values. To calculate a message digest, a hash function must be defined. A hash function is a one-way, keyless cryptographic

[1]Encryption is unable to provide authorization and access control

function which is able to transform data of a variable length to a fixed-length data string [Sta05]. The message digest for a given message remains the same regardless of the entity that performs it. However, if the original message was changed, the hash function would yield a different message digest. Common examples of hash functions are MD5 and the SHA hash functions.

While message digests can be used to provide data integrity, some security measures should be considered to prevent misuse by third parties. The first measure to be taken is choosing a strong hash function. A hash function should be able to generate a unique digest for a given message. Suppose it would be possible that two messages M and M' would yield the same digest. This would be called a collision [Sta05]. An attacker would be able to intercept message M and replace it with message M' while leaving the digest untouched. In this case, the receiver would not be able to detect the change in messages.

A second measure is that the hash code should be secured as well. Let $H(M)$ be the message digest for a message M and $E(M)$ be the encryption of M. If a message digest $H(M)$ is appended in plaintext to the ciphertext $E(M)$, then the attacker would have a way to verify if a guessed decryption of $E(M)$ is correct. Sending a hash of the ciphertext (i.e. $H(E(M))$) is not useful, since the ciphertext can still be tampered with as long as the hash is recalculated. An improved method of sending both a digest and a ciphertext is by appending the digest to the plaintext and encrypt both the plaintext and the digest (i.e. $E(M + H(M))$). However, even this form of securing a hash can be misused by changing the ciphertext, potentially even allowing it to be used for padding oracle attacks [Sut13]. For these reasons, the use of a MAC is preferred over message digests in a security context.

At last, it should also be noted that a message digest is unable to provide authentication since every entity is able to calculate the message digest while the result remains the same for a given message.

Message authentication code

Like the message digest, a message authentication code (MAC) is also a fixed-length block of data that is based on the original message [Sta05]. However, while message digests are calculated without the need of keys, the MAC assumes that both involved parties are in possession of a shared secret key. If is M the input message, k the shared secret key, $MacFunc$ the MAC function and mac the resulting MAC. A typical MAC calculation can then be defined as follows:

$$mac = MacFunc(M, k)$$

The sender performs the calculation with his secret key and appends the MAC to the message. The sender then sends the message to the receiver. Once received, the receiver will extract the message and the MAC and will then perform the same MAC calculation on the message with his secret key. If both MACs are equal to each other, then the authenticity and integrity of the message are verified.

While this method adds an authenticity check to the verification process, it can only be considered valid as long as the shared secret key remains a secret. Any third party acquiring the shared key will break the authenticity and integrity. Another point of concern is the re-use of the keys. While it is possible to use the same secret key for the encryption and the MAC calculation, it is advised to use separate keys for each operation. If this is not the case, then the possibility would exist that breaking one of the cryptographic systems would lead to compromising the other system.

2.2 Advanced Encryption Standard

Advanced Encryption Standard (AES) is a symmetric cryptographic specification designed by Vincent Rijmen and Joan Daemen [NIS01]. It is a subset of the Rijndael cipher and allows encryption using three key lengths: 128-bits, 192-bits, and 256-bits. AES was designed as a replacement of the older and then popular Data Encryption Standard (DES) in response to the proof that DES was breakable in less than a week due to a short key size (56-bits) and the evolution of raw computing power [Ele98][KPP+06]. In 2002, AES became a federal government standard [Com03] and has continued to become one of the most popular symmetric encryption methods to be used in data encryption.

2.2.1 Global overview of the AES encryption process

The AES encryption process assumes two input values: a 128-bit *block* of plaintext data and a shared secret key which can either be 128, 192, or 256 bits long [NIS01]. Both the plaintext data and/or the shared secret can be preprocessed before they are being used in the encryption process.

AES usually works with a two-dimensional array to perform all calculations on. This means that the original plaintext data is divided into rows and columns before execution [NIS01]. The resulting two-dimensional array is also called the state. Each column of the state contains a 32-bit word. Words can be regarded as a quadruple of bytes. An example of a state representation of the plaintext ASCII string ‘WE LOVE SECURITY’ is shown in figure 2.2.

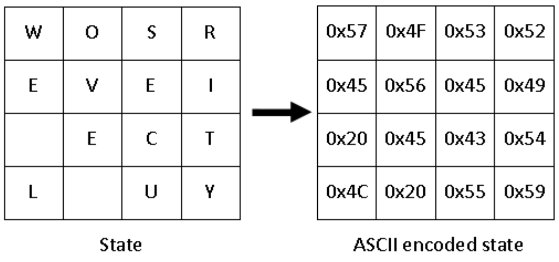


Figure 2.2: Example of a state representation of a plaintext ASCII string

The algorithm defines five functions which are used during the encryption process [NIS01] which are further explained in subsections 2.2.2 to 2.2.6:

- `KeyExpansion`: expands a short key to a number of round keys;
- `AddRoundKey`: bitwise XOR-operation between the round key and the state bytes;
- `SubBytes`: S-box based substitution of state bytes;
- `ShiftRows`: shifting the rows of the state array using the row index as the number of places to shift;
- `MixColumns`: mixes the columns of the state array.

These functions are executed in the following order:

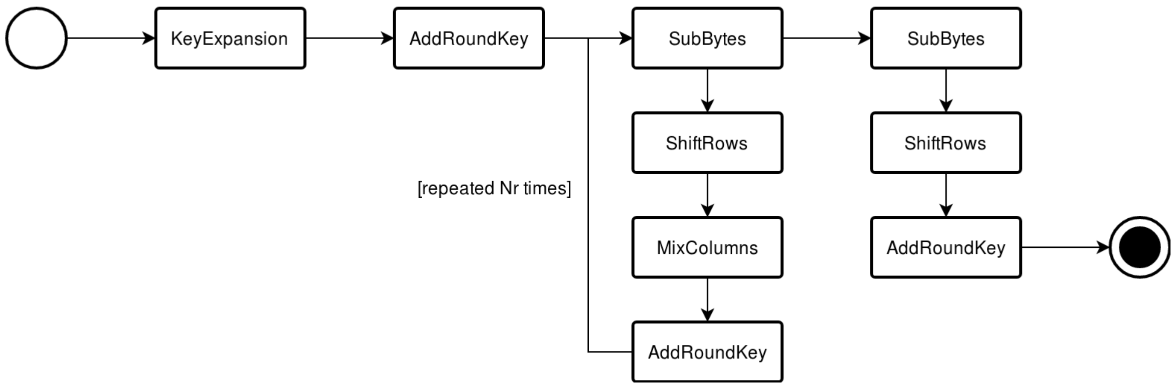


Figure 2.3: Execution steps for AES encryption

In the diagram, the sequence of $SubBytes \rightarrow ShiftRows \rightarrow MixColumns \rightarrow AddRoundKey$ is repeated Nr times. Nr is dependent on the key size; for the possible key sizes of 128, 192, and 256 bit the number of rounds will respectively be 10, 12, and 14 rounds [NIS01].

Galois fields

In order to explain the procedures of AES in more detail, the concept of Galois fields or finite fields needs to be explained first. In mathematics, a field is "an algebraic structure consisting of a set of elements for which the operations of addition, subtraction, multiplication, and division satisfy certain prescribed properties" [MM07]. The most commonly known field is the field of real numbers, which contains the numbers used for day-to-day calculations and represents a set of infinite decimal representable elements (e.g. 42, 3.14159, ...). The size of such a field is called an order, which is infinite for the field of real numbers.

A finite field is a field that is limited to a number of elements defined by its order [Gal30]. A finite field is also called a Galois field, named after the French mathematician Évariste Galois who introduced the concept of finite fields [Gal30]. The notation of a Galois field is defined as $GF(q)$, where q is the order of the field [HMV03]. One of the properties of Galois fields is that a Galois field of order q exists if and only if $q = p^n$ with p being a prime number and n a positive integer [Gal30].

In such a context, p and n are respectively referred to as being the *characteristic* and the *degree* of the finite field. If a finite field can be expressed as $GF(p^n) = 0, 1, \cdots, p^n - 1$, each element in the finite field can be expressed using its polynomial representation:

$$\forall z \in GF(p^n) : z = \sum_{i=1}^{n} a_{n-i}x^{n-i} \qquad \text{with } a_{n-i} \in \{0, \cdots, p-1\} \tag{2.1}$$

Finite fields must also define basic arithmetic operations. These operations include addition and multiplication. For finite fields, the addition can be described as an addition of the terms in the polynomial representations of the elements, with a reduction of the result using modulo the characteristic. For example, the addition of two elements in $GF(2^8)$ can be described as follows:

$$\begin{aligned}a + b &= \sum_{i=0}^{7}(a_i x^i + b_i x^i)\ mod\ 2 \\ &= ((a_7 + b_7)\ mod\ 2)x^7 + ((a_6 + b_6)\ mod\ 2)x^6 + ((a_5 + b_5)\ mod\ 2)x^5 \\ &\quad + ((a_4 + b_4)\ mod\ 2)x^4 + ((a_3 + b_3)\ mod\ 2)x^3 + ((a_2 + b_2)\ mod\ 2)x^2 \\ &\quad + ((a_1 + b_1)\ mod\ 2)x^1 + ((a_0 + b_0)\ mod\ 2)x^0\end{aligned} \tag{2.2}$$

Notice that the coefficients for each term in $GF(2^8)$ can either be 0 or 1. Assume that $a = 1$ and $b = 1$. If a and b were real numbers, the sum of a and b would yield a 2. However, in case of $GF(2^n)$, the number 2 cannot exist as a coefficient [MM07]. Therefore, the modulo 2 operation will be applied to the sum of coefficients when an addition is performed in $GF(2)$ (i.e., $a + b = \sum_{i=0}^{7}\left(((a_i + b_i)\ mod\ 2)x^i\right)$).

The multiplication in Galois fields also follows a specific procedure. Multiplication in a finite field is equal to the multiplication of the terms in the polynomial representation, followed by a reduction of the result using modulo a fixed irreducible polynomial that is used to describe the finite field. For the aforementioned finite field $GF(2^8)$, this irreducible polynomial is $x^8 + x^4 + x^3 + x + 1$. For example, the multiplication of $a = x^4 + x^2$ and $b = x^2 + x^0$ will initially result in $a\dot{b} = x^6 + x^4 + x^4 + x^0$. This result is further reduced using modulo $x^8 + x^4 + x^3 + x + 1$, but since the result is of a lower order than the modulo argument, the original result will remain the same.

Given these properties, one could observe that this would allow elements in Galois fields using 2 as the characteristic to be represented by polynomials similar to bit strings. The degree of the finite field would then determine the number of bits used in the bit string and the coefficients of the terms would represent bit values. For this reason, finite fields of the order 2^n are also called *binary fields* or *characteristic-two finite fields* [HMV03]. For obvious reasons, binary fields are often used in computer science to express arithmetic operations on a bit level. In the context of AES, $GF(2^8)$ will be used because the algorithm performs operations on a byte-per-byte level [NIS01]. The polynomial representation of a byte value in $GF(2^8)$ will be noted as $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0$. An example of polynomial representation of a byte in $GF(2^8)$ is given in figure 2.4. It should also be noted that, as demonstrated in 2.2, that an addition on two elements in a binary field can be seen as an XOR operation between the bit string representations of the elements.

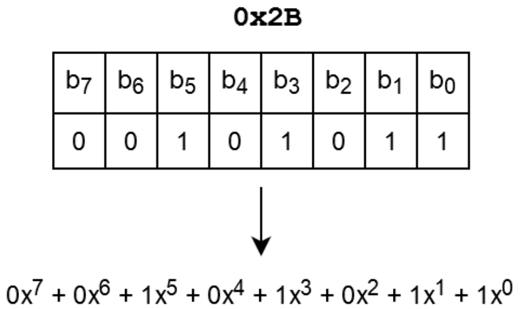


Figure 2.4: Conversion of byte 2B to a polynomial in $GF(2^8)$

2.2.2 KeyExpansion

`KeyExpansion` takes the cipher key and performs Rijndael's key schedule algorithm to generate new round keys which will be used during the `AddRoundKey` procedure [NIS01].

The procedure starts by creating an array w which will contain 32-bit words. Each word in this array can be described as $w[i]$, where i will indicate the position of the word in the array. In total the array will consist of $Nb \times (Nr + 1)$ words, where Nb is the (plaintext data) block size in terms of 32-bit words (4 for AES) and Nr is the number of rounds (dependent on the key size, see 2.2.1). Nk will also be defined as the number of 32-bit words that the cipher key contains. For example, a 128-bit key will yield $Nk = 128/32 = 4$.

The Nk first words of w will be filled in by the cipher key itself [NIS01]. The cipher key will be divided in portions of 32-bit words and these portions will be the first elements of word array w. The remaining words $w[i]$ will be based upon the XORed value of $w[i-1]$ and $w[i - Nk]$ until the array is filled with $Nb \times (Nr + 1)$ words. However, when $i \ mod \ Nk = 0$, $w[i - 1]$ will be preprocessed before it is used for the XOR-operation:

1. `RotWord` will shift the order of bytes one position to the left (i.e. $(b_0, b_1, b_2, b_3) \rightarrow (b_1, b_2, b_3, b_0)$)
2. `SubWord` will use the S-box to substitute the each of the four bytes in the word with the corresponding value in the S-box;
3. The result of the previous two operations will be XORed with $Rcon\lfloor i/Nk \rfloor$. As noted in [NIS01], "$Rcon[i]$ contains the values given by $[x^{i-1}, 00, 00, 00]$, with x^{i-1} being powers of x (...) in the field $GF(2^8)$."

Apart from this, AES will also perform an extra run of `SubWord` on $w[i - 1]$ before the XOR operation when $Nk > 6$ and $i - 4 \ mod \ Nk = 0$. Note that this is only the case when the cipher key is 256 bits long [NIS01].

Reverting `KeyExpansion` in a 128-bit key context

Since the `KeyExpansion` procedure exists exclusively of shifting, substitution and XOR operations, the results can be reversed since all of the operations are reversible:

- **Shifting:** shift the bytes in the opposite direction;
- **Substitution:** if the original substitution yields a unique result (as the Rijndael S-Box does), an inverse substitution lookup table can be built to reverse the substitution;
- **XOR operation:** An XOR operation of two values van be reverted by XOR-ing the result with one of the two values; the result will be the other value.

The full algorithm to perform a `KeyExpansion` reversal on a i-th round key is outlined in listing 2.2.

```
def reverseRoundKey(rndkey, round):
    initkey = list(rndkey)
    for rnd in range(round, 0, -1):
        # 128-bit key consists of 4 32-bit words
        for w_idx in range(3, 0, -1):  # 3 words remaining
            start_idx = w_idx*4
            end_idx = (w_idx + 1)*4
            prev_word = initkey[start_idx-4:end_idx-4]
            for b_idx in range(4):  # 4 bytes per word
                initkey[start_idx+b_idx] ^= prev_word[b_idx]

        # For i%Nk = 0 preprocessing is executed on the word i during
    encryption
        # i%Nk = 0%4 = 0 => first word (encryption) => last word (decryption)
        last_word = initkey[12:]

        # Shift one position to the left
        preproc_word = [last_word[1], last_word[2], last_word[3],
    last_word[0]]

        # Substitute each byte by its corresponding S-box value
        preproc_word = [aes_sbox[t] for t in preproc_word]

        # XOR first word with Rcon
        preproc_word[0] ^= aes_rcon[rnd]

        for idx in range(4):
            initkey[idx] ^= preproc_word[idx]

    return initkey
```

Listing 2.2: KeyExpansion reversal for an i-th round key

2.2.3 `AddRoundKey`

The `AddRoundKey` procedure adds a round key to the current state by a bitwise XOR operation [NIS01]. The round keys are pre-generated by the `KeyExpansion` procedure. The `AddRoundKey` procedure is executed Nr times during the algorithm. Per procedure call another round key is used to perform the addition. Each round key consists of Nb 32-bit words. Since $Nb = 4$ for the AES algorithm and a single column in the state array consists of 32-bit words, each word from the round key can be added to a column in the state. If w_0, w_1, w_2, w_3 describe the words of the round key and if $s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}$ describe the bytes in the column c of the state array, then the addition can be defined as:

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus w_c \tag{2.3}$$

2.2.4 `SubBytes`

The `SubBytes` procedure will substitute the values in the state array with the associated value from the Rijndael S-box [NIS01]. The Rijndael S-box or substitution box is a matrix of 2^8 elements where each element is associated to a specific byte value (e.g. the first element associates with byte `0x00`, the second with `0x01`, ...). When put in a 2-D matrix, the row index can be regarded as the value of the first nibble[2] of the input byte and the column index as the value of the second nibble. The output value corresponding to a certain input value can be calculated by the multiplicative inverse of the input byte in $GF(2^8)$, where 0 (having no multiplicative inverse) is mapped to 0. In order to prevent calculating the S-box value for each input byte, the S-box will be stored as a constant matrix by the AES implementation[3].

2.2.5 `ShiftRows`

The `ShiftRows` procedure will shift the rows in the 2-D representation of the state array to the left [NIS01]. The rows of the state array are shifted according to their row index i where $0 \leq i < 4$. Thus the first row will be not be shifted as $i = 0$, the second ($i = 1$) will be shifted one position to the left, and so on. Elements at left-most position of the row will be moved to the end of the row.

[2]A nibble is a group of 4 bits

[3]Calculating each value per input byte would result in performance loss and a possible security risk

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

$\rightarrow$

1	5	9	13
6	10	14	2
11	15	3	7
16	4	8	12

Table 2.1: Example of the `ShiftRow` procedure on the state array

2.2.6 `MixColumns`

The `MixColumns` step will transform the state array column-per-column by performing a multiplication modulo x^4+1 of its $GF(2^8)$ polynomial representation with a fixed polynomial $a(x) = 3x^3 + 1x^2 + 1x^1 + 2x^0$ [NIS01]. Suppose the $GF(2^8)$ polynomial representation of a state array's column can be noted as follows:

$$b(x) = b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0$$

Each coefficient represents a single byte of the column in the above equation. The multiplication of the state array's column with the fixed polynomial $a(x)$ will then result in:

$$\begin{aligned} c(x) &= a(x) \cdot b(x) \\ &= (a_3x^3 + a_2x^2 + a_1x^1 + a_0x^0) \cdot (b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0) \\ &= c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x^1 + c_0x^0 \end{aligned} \tag{2.4}$$

with:

$$\begin{aligned} c_0 &= a_0 \cdot b_0 \\ c_1 &= a_1 \cdot b_0 \oplus a_0 \cdot b_1 \\ c_2 &= a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2 \\ c_3 &= a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3 \\ c_4 &= a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3 \\ c_5 &= a_3 \cdot b_2 \oplus a_2 \cdot b_3 \\ c_6 &= a_3 \cdot b_3 \end{aligned}$$

Since $c(x)$ is a 7-term polynomial, it would imply that the multiplication would yield 7 bytes instead of the 4 bytes foreseen in the state array's column. The modulo $x^4 + 1$ operation is performed to solve this issue [NIS01] and will reduce $c(x)$ to a 4-term polynomial $d(x) = d_3x^3 + d_2x^2 + d_1x^1 + d_0x^0$ with:

- $d_0 = (a_0 \cdot b_0) \oplus (a_3 \cdot b_1) \oplus (a_2 \cdot b_2) \oplus (a_1 \cdot b_3)$
- $d_1 = (a_1 \cdot b_0) \oplus (a_0 \cdot b_1) \oplus (a_3 \cdot b_2) \oplus (a_2 \cdot b_3)$
- $d_2 = (a_2 \cdot b_0) \oplus (a_1 \cdot b_1) \oplus (a_0 \cdot b_2) \oplus (a_3 \cdot b_3)$
- $d_3 = (a_3 \cdot b_0) \oplus (a_2 \cdot b_1) \oplus (a_1 \cdot b_2) \oplus (a_0 \cdot b_3)$

Because $a(x)$ is a fixed polynomial for which the coefficients are known, $d(x)$ can be rewritten to a matrix multiplication (2.5) [NIS01]. Note that the matrix multiplication below shows the `MixColumns` step for a single column of the state array. By adding the remaining columns to the second matrix of the multiplication, the other columns will be processed in a similar way.

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{2.5}$$

The matrix multiplication will thus perform additions and multiplications in the $GF(2^8)$ domain. Additions for $GF(2^8)$ are equal to an XOR operation and are thus not complex to implement. However, the multiplication for two elements in $GF(2^8)$ is a complex operation [NIS01]. Many AES implementations will therefore use lookup tables for the multiplication step, because there are only three different coefficients in $a(x)$ and 256 possible input bytes for $b(x)$.

2.3 Block cipher modes of operation

In the previous section it was assumed that the plaintext input was a 128-bit data block. In practice the plaintext input could be much larger in size, effectively spanning hundreds of 128-bit blocks. In its core AES is only able to process blocks of 128 bits of data; extra processing is necessary in order to make AES capable of encrypting plaintexts larger than 128 bits. Therefore, AES and various other block ciphers are used in combination with a block mode of operation, an algorithm that uses a block cipher to provide an information service such as confidentiality or authenticity, regardless of the plaintext size [And14].

Since block cipher modes can perform operations on the plaintext, secret key, and/or ciphertext, they can also introduce security vulnerabilities or performance issues to the encryption process. In the following subsections an overview will be provided of the basic set of block cipher modes recommended by [Dwo01] and the block cipher mode CCM [Dwo04], which is commonly used in Internet-of-Things protocols [IEE11][BTS16][Gem17]. Throughout the explanations of the block cipher modes, two terms will be used namely the *input* and *output block*, which are respectively the block that is used as the input for the cipher and the resulting block after the cipher operation.

2.3.1 Electronic Codebook mode (ECB)

The Electronic Codebook (ECB) mode is arguably the simplest block cipher mode. ECB will perform the block cipher on a block-per-block basis without making changes to the plaintext, the ciphertext or the secret key [Dwo01]. Let P_i be the i-th block of plaintext and C_i be the i-th ciphertext block, then the encryption process using ECB can be described as:

$$C_i = Cipher(P_i, K) \qquad (\text{for } i = 1, \cdots, n)$$

The decryption process is effectively the inverse of the encryption process:

$$P_i = Decipher(C_i, K) \qquad\qquad (\text{for } i = 1, \cdots n)$$

Notice that in the context of ECB, the input block is the plaintext block and the output block is the ciphertext without any for of preprocessing. While this mode is quite effective and easy to use, it comes with the disadvantage that any given plaintext P will always yield the same ciphertext (e.g., if $P_i = P_j \Rightarrow P_i \oplus K = P_j \oplus K \Rightarrow C_i = C_j$) [Dwo01]. This makes the ciphertexts vulnerable to pattern recognition and replay attacks [4]. It is therefore not recommended to use this block cipher mode if this behaviour is unwanted [Dwo01].

2.3.2 Cipher Block Chaining mode (CBC)

The Cipher Block Chaining (CBC) mode is a block cipher mode which uses the ciphertext of the previously encrypted block to modify the current block of plaintext before encrypting it using the block cipher [Dwo01]. This prevents (to a certain degree) that a given plaintext block always yields the same ciphertext. However, it also introduces a problem: the first block of plaintext cannot be modified since there is no previously encrypted ciphertext block to use. To solve this problem, an initialization vector (IV) is used instead. An IV is a vector of random bits equal to the size of a regular block. The IV itself need not to be kept a secret, but should be unpredictable [Dwo01].

A complete overview of the algorithm is provided by figure 2.5. The CBC mode of operation can be described as follows. Let n be the number of plaintext blocks to encrypt, P_i be the i-th plaintext block, C_i be the i-th ciphertext block and IV be the initialization vector, then the encryption process will be:

$$\begin{aligned} C_1 &= Cipher(P_1 \oplus IV, K) \\ C_i &= Cipher(P_i \oplus C_{i-1}; K) && \text{(for } i = 2, \cdots, n) \end{aligned}$$

The decryption process will be the inverted operation, namely:

$$\begin{aligned} C_1 &= Decipher(C_1, K) \oplus IV \\ C_i &= Decipher(P_i; K) \oplus C_{i-1} && \text{(for } i = 2, \cdots, n) \end{aligned}$$

In this context, the input block will either be $P_1 \oplus IV$ or $P_i \oplus C_{i-1}$ depending on the position i of the plaintext block to process. However, the output block still remains the result of the cipher operation (i.e., the ciphertext block).

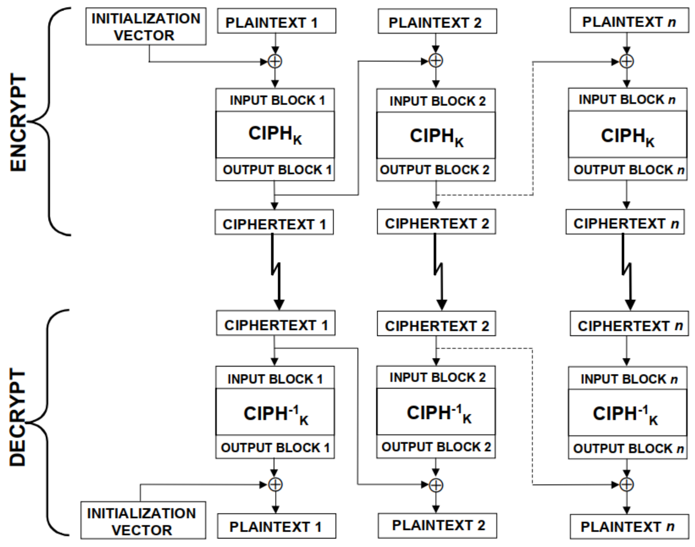


Figure 2.5: Overview of the CBC block cipher mode. Image from *Recommendation for Block Cipher Modes of Operation: Methods and Techniques* (2001) by Dworkin, Morris J..

2.3.3 Cipher Feedback mode (CFB)

The Cipher Feedback (CFB) is a block cipher mode of operation which, similar to CBC, uses the ciphertext of the previous block to prevent the encryption process from yielding the same

[4]Replay attacks are attacks which resend or delay valid messages with the intention of performing malicious or fraudulent actions.

ciphertext given a specific plaintext block. However, CFB differs from CBC at multiple key points. The first difference is found in the input blocks. In contrary to the previous block cipher modes, CFB will not use the plaintext block as a part of the input block [Dwo01]. Instead, the cipher is executed on the previous ciphertext block and afterwards XORed with the plaintext block.

A second difference with CBC is that CFB is also able to further randomize the input blocks used for the cipher operations [Dwo01]. It does so by using only a part of the ciphertext block to form a new input block. For this reason, CFB will also require an integer parameter s to indicate how much bits of the previous ciphertext block will be used to form the new input block. The s most significant bits of the previous ciphertext block will be prepended with the $b - s$ least significant bits of the previous input block (with b being the block size in bits). The consequence of limiting the number of ciphertext block bits to be used for the next input block is that only s usable bits of ciphertext will be yielded per block. This causes the plaintext and ciphertext to be divided in segments of s bits, thus requiring more processing to complete the encryption process [Dwo01].

Finally, the CFB mode can be described as follows. Let n be the number of plaintext segments to encrypt, $P_i^{\#}$ the i-th plaintext segment, $C_i^{\#}$ the i-th ciphertext segment, IV the initialization vector, s the integer parameter and I_i and O_i be the i-th input and output block, then the encryption process can be described as:

$$\begin{aligned}
I_1 &= IV \\
I_i &= LSB(I_{i-1}, b-s)|MSB(C_{i-1}^{\#}, s) && \text{(for } i = 2, \cdots, n) \\
O_i &= Cipher(I_i, K) && \text{(for } i = 1, \cdots, n) \\
C_i^{\#} &= P_i^{\#} \oplus MSB(O_i, s) && \text{(for } i = 1, \cdots, n)
\end{aligned}$$

The decryption process will be the inverted operation, namely:

$$\begin{aligned}
I_1 &= IV \\
I_i &= LSB(I_{i-1}, b-s)|MSB(C_{i-1}^{\#}, s) && \text{(for } i = 2, \cdots, n) \\
O_i &= Cipher(I_i, K) && \text{(for } i = 1, \cdots, n) \\
P_i^{\#} &= C_i^{\#} \oplus MSB(O_i, s) && \text{(for } i = 1, \cdots, n)
\end{aligned}$$

2.3.4 Output Feedback mode (OFB)

The Output Feedback (OFB) is a block cipher mode of operation which combines the principle of output blocks with the CBC mode [Dwo01]. Like the CBC mode, OFB requires an initialization vector to randomize its output for a given plaintext. The IV however is not directly applied to the plaintext, but is first encrypted using the encryption algorithm. This process yields an output block that will be used for an XOR operation on the plaintext block. This results in the ciphertext block for the first plaintext block.

The second block uses the output block of the previous block encryption as an input block for the encryption algorithm. Therefore, no information of the plaintext or ciphertext is required to calculate the output block for the second block encryption. The second output block will then be used as the input block for the third block encryption. The process is repeated for each plaintext block that needs to be encrypted, with the output block of the previous block encryption (O_{i-1}) being the input block for next block encryption (I_i). This makes it possible to calculate the output block for the OFB mode prior to the availability of the plaintext blocks [Dwo01].

Let n be the total number of blocks, IV be a random IV, K the initial secret key, I_i the ith input block, O_i the ith output block, P_i the ith plaintext block and C_i the ith ciphertext block, then the encryption process can be defined as:

$$\begin{aligned}
I_1 &= IV \\
I_i &= O_{i-1} && \text{(for } i = 2, 3, ..., n) \\
O_i &= Cipher(I_i, K) && \text{(for } i = 1, 2, ..., n) \\
C_i &= P_i \oplus O_i && \text{(for } i = 1, ..., n-1) \\
C_n^* &= P_n^* \oplus MSB(O_n, u) && \text{(with } u \text{ the bit length of } P_n^*)
\end{aligned}$$

The decryption process will be the inverted operation, namely:

$$\begin{aligned}
I_1 &= IV \\
I_i &= O_{i-1} && \text{(for } i = 2, 3, ..., n) \\
O_i &= Cipher(I_i, K) && \text{(for } i = 1, 2, ..., n) \\
P_i &= C_i \oplus O_i && \text{(for } i = 1, ..., n-1) \\
P_n^* &= C_n^* \oplus MSB(O_n, u) && \text{(with } u \text{ the bit length of } C_n^*)
\end{aligned}$$

2.3.5 Counter mode (CTR)

The Counter (CTR) mode is very similar to the OFB mode with the sole exception that a counter and nonce are used to generate the output blocks instead of the same IV for each block [Dwo01]. The combination of counter and nonce must be different for each block and each message to avoid repeated use of the same output block. This is achieved by incrementing the counter value for each block of the message.

The CTR mode is also able to make a block cipher behave like a stream cipher. In order to this, CTR will treat the last block of the message differently from the rest. The CTR mode will generate the candidate key stream like in the previous runs, but will only use an equal number of significant bits as the plaintext block needs for the output block. This way the plaintext block and the output block will have the same number of bits, which can be XORed with each other and will result in a ciphertext block with an equal bit length as the plaintext block.

An overview of the CTR mode is provided by figure 2.6. Finally, the CTR mode can be described as follows. Let n be the total number of blocks, N be a random nonce, K the initial secret key, O_i the ith output block, P_i the ith plaintext block and C_i the ith ciphertext block, then the encryption process is:

$$\begin{aligned}
T_i &= N + i && \text{(for } i = 1, 2, ..., n) \\
O_i &= Cipher(T_i, K) && \text{(for } i = 1, 2, ..., n) \\
C_i &= P_i \oplus O_i && \text{(for } i = 1, ..., n-1) \\
C_n^* &= P_n^* \oplus MSB(O_n, u) && \text{(with } u \text{ the bit length of } P_n^*)
\end{aligned}$$

The decryption process is performed the same way as the encryption process, except that the plaintext block and the ciphertext block are swapped in the XOR operation:

$$\begin{aligned}
T_i &= N + i && \text{(for } i = 1, 2, ..., n) \\
O_i &= Cipher(S, T_i) && \text{(for } i = 1, 2, ..., n) \\
P_i &= C_i \oplus O_i && \text{(for } i = 1, 2, ..., n-1) \\
P_n^* &= C_n^* \oplus MSB(O_n, u) && \text{(with } u \text{ the bit length of } C_n^*)
\end{aligned}$$

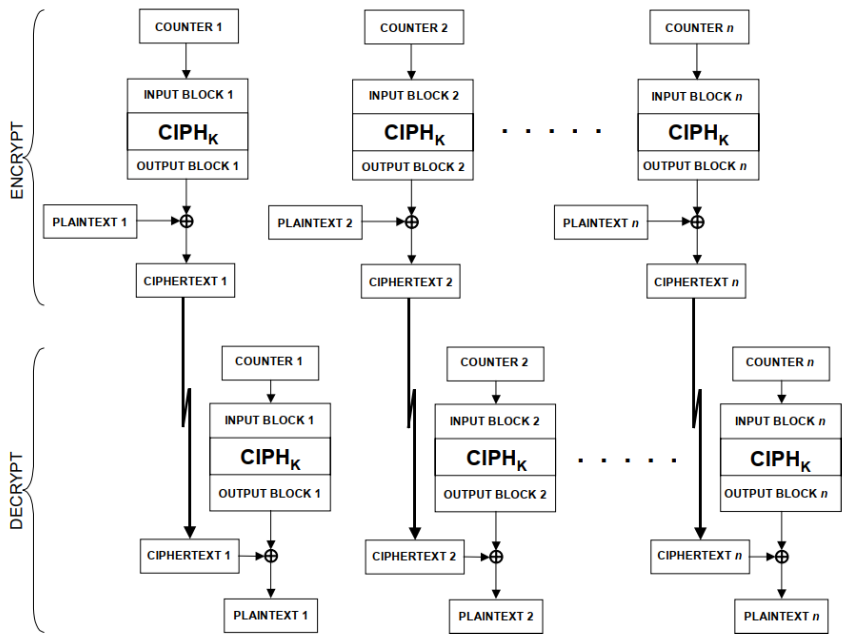


Figure 2.6: Overview of the CTR block cipher mode. Image from *Recommendation for Block Cipher Modes of Operation: Methods and Techniques* (2001) by Dworkin, Morris J..

2.3.6 Counter with Cipher Block Chaining-Message Authentication Code (CCM)

The Counter with Cipher Block Chaining-Message Authentication Code (CCM) mode is a block cipher mode of operation which also provides authenticity to the encrypted data. It combines the aforementioned CTR mode with the CBC-MAC algorithm [Dwo04]. Figure 2.7 gives an overview of the encryption-generation and decryption-verification process.

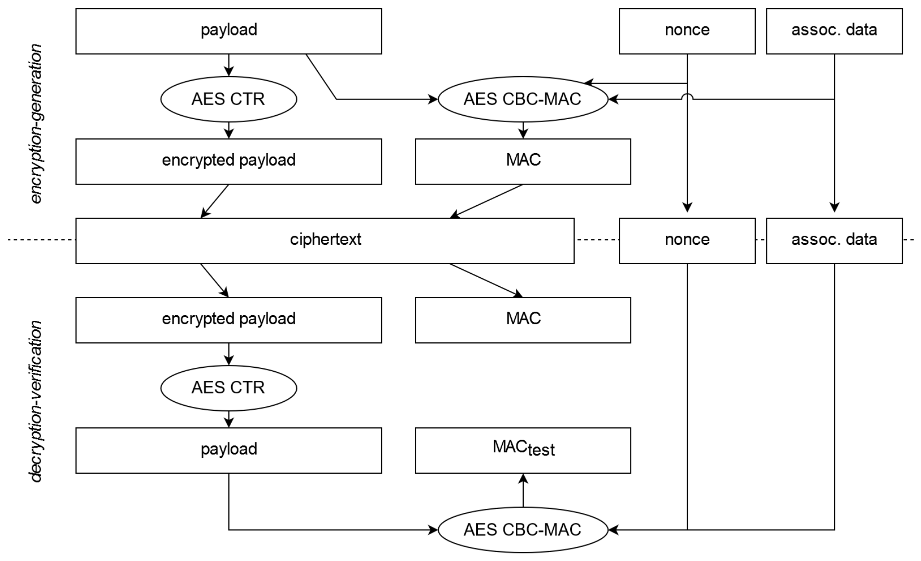


Figure 2.7: Overview of AES in CCM mode. The encrypted message is authenticated and unmodified if the original MAC and MAC_{test} are the same after the decryption-verification step.

The encryption-generation process starts by generating the MAC using the CBC-MAC algorithm. The CBC MAC (CBC-MAC) algorithm is a MAC algorithm which uses AES in combination with the CBC mode and a fixed zero-only IV to generate a MAC. The CBC-MAC algorithm identifies several input values:

- A nonce N;
- The associated data A, which is data which will be authenticated but not encrypted by the AES algorithm in CTR mode. Examples of this form of input are headers and flags;
- The payload P, which is the data which will be authenticated and encrypted. This is usually the confidential data.

The CBC-MAC will thus serve as an authentication and integrity measure for the whole message and not only the confidential data. This requires all the relevant data to be packed into one data stream. An input formatting function $InputFormat$ will therefore be defined to ensure that both the sender and receiver pack the data in a common format.

Let $Cipher(x, K)$ the cipher (AES) which encrypts data x with key K, T_i the counter for block i, s the desired size of the MAC in bits, n the size of the payload in bits, $InputFormat$ be the input formatting function, and r the number of blocks generated by $InputFormat$,. The CBC-MAC algorithm can then be described as follows [Dwo04]:

$$\begin{aligned}
m &= \lceil n/128 \rceil \\
O_i &= Cipher(T_i, K) && \text{(for } i = 1, ..., m) \\
B_i &= InputFormat(N, A, P) && \text{(for } i = 1, ..., r) \\
Y_1 &= Cipher(B_1, K) \\
Y_i &= Cipher(B_i \oplus Y_{i-1}, K) && \text{(for } i = 2, ..., r) \\
T &= MSB(Y_r, s) \\
W &= (T \oplus MSB(O_1, s)) \\
U &= O_2 ... O_m \\
V &= P \oplus MSB(U, n) \\
C &= V \parallel W
\end{aligned}$$

Notice that calculations 3 until 7 are related to the the calculation of the CBC-MAC and calculations 8 and 9 are related to AES CTR encryption. The decryption-verification process is mostly the inverse operation. Let l be the length of the ciphertext:

$$
\begin{aligned}
V, W &= \mathit{Split}(V, W) \\
m &= \lceil (l-n)/128 \rceil \\
O_i &= \mathit{Cipher}(T_i, K) && \text{(for } i = 1, ..., m\text{)} \\
U &= O_2 ... O_m \\
V &= MSB(C, (l-n)) \\
P &= V \oplus MSB(S, (l-n)) \\
W &= LSB(C, s) \\
T &= W \oplus MSB(0_1, s) \\
B_i &= \mathit{InputFormat}(N, A, P) && \text{(for } i = 1, ..., r\text{)} \\
Y_1 &= \mathit{Cipher}(B_1, K) \\
Y_i &= \mathit{Cipher}(B_i \oplus Y_{i-1}, K) && \text{(for } i = 2, ..., r\text{)} \\
T_t &= MSB(Y_r, s) \\
ok &= (T = T_t)
\end{aligned}
$$

2.4 The LoRaWAN protocol

In the introduction it was mentioned that a wireless protocol called LoRaWAN would be targeted for the power analysis tests. The choice for the LoRaWAN protocol was made for several reasons.

With the exponential growth of Internet connected device it becomes more plausible that a larger variety of devices will become connected to the Internet. At first, the Internet was primarily used by immobile computers and mainframes, allowing the use of cables to connect the devices to the Internet. With the introduction of portable devices such as the laptop and smartphone, a need for a more mobile solution started to grow amongst the population. This lead to the adoption of wireless technologies such as WiFi and cellular networks, providing Internet access to mobile devices through radio waves. While these technologies provided a wireless and portable method of connecting devices to the Internet, they also came with several disadvantages such as limited range, high power consumption and high set-up costs. These are especially problematic for devices with small batteries such as sensor devices. As a response, various low-power wireless radio technologies have been created to address these issues.

One of these low-power wireless technologies is LoRa. LoRa is a wireless connectivity technology which uses unlicensed MHz ISM radio frequency bands to exchange data [AF17]. LoRa is designed to be used for long ranges and high endurance on battery-powered devices, allowing data communication up to a range of 10 km in rural areas and for an operation cycle of several years using regular AA batteries [AF17]. These advantages do come at the price of a lower data rate: where WiFi and cellular networks are capable of transferring data at a data rate of several megabits per second, LoRa will only be able to transfer data at several kilobits per second.

However, LoRa itself is only a physical layer protocol; it provides a method for physically sending signal through radio frequencies, but it does not define the protocols to interconnect devices [AF17]. Therefore LoRa is often used in combination with the LoRaWAN protocol which serves as a MAC protocol for LoRa enabled devices. While the MAC protocol is primarily created to interconnect devices within the network, it also defines the measurements used to ensure secure data exchange over LoRaWAN networks.

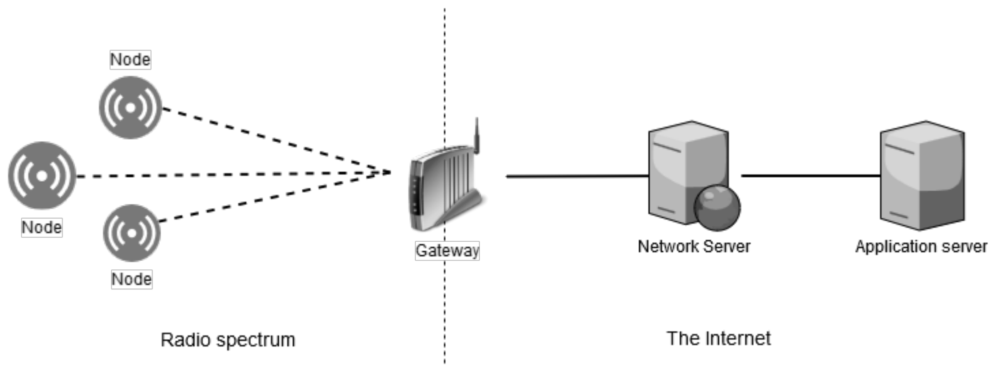


Figure 2.8: Structure of a basic LoRaWAN network

Figure 2.8 shows the typical structure of a LoRaWAN network [The18]. The end devices or *nodes* are the entities which will retrieve data from the environment such as sensor data. They will send their data using the LoRa radio technology to Internet connected *LoRaWAN gateways* which are basically LoRa enabled relay stations capable of sending and receiving LoRa packets. If the gateway is part of the LoRaWAN network which the node is trying to contact, it will be able to relay the packet to a LoRaWAN network server which will further handle the LoRaWAN MAC functionalities. The data is then often transferred to an application server which will process and optionally respond to the data.

The key features which make LoRaWAN an interesting target for side-channel attacks are the following:

- **Low (deployment) costs** Since LoRa uses the unlicensed ISM radio frequencies, entities are able to deploy their own devices without having to pay license costs unlike other wireless technologies such as cellular technologies. LoRaWAN radio modules are also relatively cheap (less than 10 euros) to purchase. This improves accessibility to the network and thus allows the technology to be adapted more easily, resulting in a large user base which can be targeted;
- **Long range and low power consumption** The combination of long range connectivity and low power consumption makes it more plausible for LoRaWAN enabled end devices to be left unmonitored for longer periods of time. This makes it easier to probe said devices, since it is more likely that the devices are left unsupervised.

2.4.1 LoRaWAN security

Before the LoRaWAN protocol can be targeted using side-channel attacks, a better understanding of the security measures are needed. Since LoRa uses radio technology for data transfer, it is susceptible to interception by third-parties. The specifications and inner workings of the LoRa protocol are proprietary and not made available for the public, although attempts have been made and were successful in reverse engineering the protocol [Kni16][Rob18]. To prevent that third-party entities would be able to intercept and decode the radio signals (i.e., breaking confidentiality) and to prevent that the original payload could be modified unintentionally (i.e., breaking integrity), LoRaWAN uses an encryption scheme based on the algorithm described in IEEE 802.15.4/2006 Annex B [LoR15].

The exact working of this encryption scheme are subject to change from specification to specification. At the time of writing, the latest final specification of LoRaWAN is version 1.1. This version introduced extra security measures compared to the previous version (1.0.2). However, the LoRaWAN device that will be used in the case studies does not support this

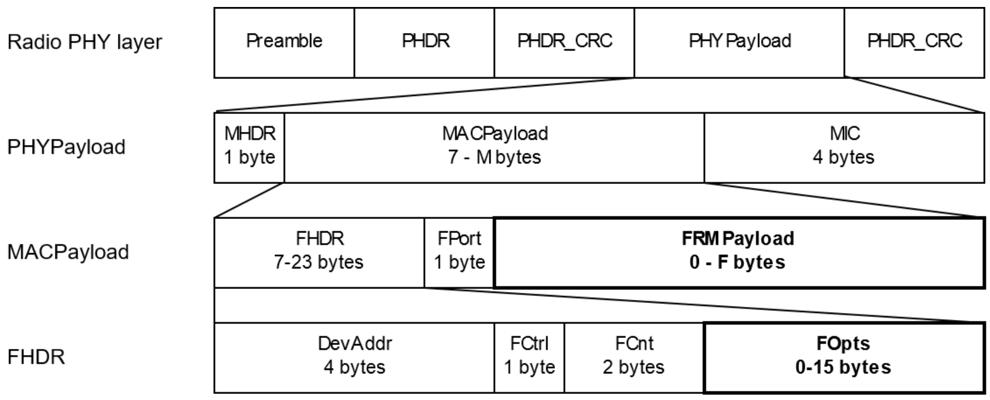


Figure 2.9: The format of a LoRaWAN MAC message according to the LoRaWAN specification v1.0.1 [LoR15]. The payload defined by the developer resides in `FRMPayload`. In version 1.0.1, only `FRMPayload` is encrypted, while both `FRMPayload` and `FOpts` are encrypted according to version 1.1.

specification yet. For this reason, the focus in this thesis will be turned to the latest supported specification by the device, version 1.0.1.

In version 1.0.1, the LoRaWAN protocol uses two secret keys to encrypt its traffic: the application session key (`AppSKey`) and the network session key (`NwkSKey`). How the device obtains these keys is dependent on the way it joins the network [LoR15]. LoRaWAN defines two join methods: Over-The-Air Activation (OTAA) and Activation-by-Personalisation (ABP). The ABP method is straightforward: `AppSKey` and `NwkSKey` are directly stored into the device memory. However, in case of the OTAA method, `AppSKey` and `NwkSKey` are dynamically generated by the network server [LoR15]. During the join procedure, the node will provide `AppEUI`, `DevEUI` and `DevNonce` to the network server. `AppEUI` and `DevEUI` can be chosen freely by the developer although both should be unique within the network. `DevNonce` is randomly generated per session context by the node. The network server will use these three values to generate an application nonce (`AppNonce`). The application nonce is then sent back to the node where it will be used in combination with `AppKey` and `DevNonce` to generate the session keys. The node stores both keys for the duration of the session context. The exact calculation for the session keys is performed as follows [LoR15]:

$$\texttt{NwkSKey} = aes128\_encrypt(\texttt{AppKey}, \texttt{0x01}|\texttt{AppNonce}|\texttt{NetID}|\texttt{DevNonce}|pad_{16})$$
$$\texttt{AppSKey} = aes128\_encrypt(\texttt{AppKey}, \texttt{0x02}|\texttt{AppNonce}|\texttt{NetID}|\texttt{DevNonce}|pad_{16})$$

After completing the join process, the keys can be used for data communication. The application session key is used to encrypt application data; this is usually the payload that has been defined by the developer to be sent to the cloud. The network session key is used for encrypting the information that is relevant for the node and the gateway; this is usually network related information [LoR15]. Figure 2.9 shows the format of a data-containing LoRaWAN MAC message[5]. According to the LoRaWAN specification 1.0.1 [LoR15], the frame payload (`FRMPayload`) is first encrypted using the AES CTR algorithm with a 128-bit key. The choice of encryption key depends on the frame port (`FPort`) value:

- **0:** `FRMPayload` contains MAC commands (i.e., network related information), thus `NwkSKey` is used as the encryption key;
- **1...223:** `FRMPayload` contains application-specific data, thus `AppSKey` is used as the encryption key;
- **224:** indicates that the LoRaWAN message is a test message and will also use `AppSKey` as the encryption key;

- **225...255:** reserved for future usage. AppSKey will be used as the encryption key.

Once the frame payload has been encrypted, a Message Integrity Code (MIC) is generated from the LoRaWAN message, more specifically the part that encapsulates MHDR, FHDR, FPort, and FRMPayload [LoR15]. For the MIC calculation, the AES CBC-MAC algorithm is used with a 128-bit key (NwkSKey). The calculation of the MIC is defined as follows:

$$
\begin{aligned}
cmac &= aes128\_cmac(NwkSKey, B_0|msg) \\
MIC &= cmac[0...3]
\end{aligned}
$$

B_0 in the previous formula is constructed from information about the message itself. Figure 2.10 shows the format of B_0 [LoR15]. After the payload is encrypted and the MIC is calculated and added to the message, the message is sent to the LoRaWAN gateway. The gateway will be configured to relay the message to the network server. Since the network server possesses the NwkSKey, it will be able to verify the message using the MIC and decrypt the relevant network related information. The encrypted payload is then relayed to the application server for which the message is destined to. Finally the application server, which possesses the AppSKey, will be able to decrypt the payload.

0x49 1 byte	0x00000000 4 bytes	Dir 1 byte	DevAddr 4 bytes	FCntUp or FCntDown 4 bytes	0x00 1 byte	Length 1 byte

Figure 2.10: The format of the data block B_0 that is used during the calculation of the MIC. Dir indicates whether the message is an uplink (0) or downlink message, while FCntUp or FCntDown specifies the frame uplink or downlink message counter.

Differences in version 1.1

In the newer LoRaWAN specification (v1.1), several security measures have been added [LoR17]:

- **More cryptographic keys:** version 1.1 defines more cryptographic keys to fit the purpose that each needs to fulfil. Apart from the aforementioned application key (AppKey), a network key (NwkKey) is now also defined per end-device. This ensures that the application and network session keys are generated independently of each other. The network session key (NwkSKey) is also further split into multiple keys according to their purpose: FNwkSIntKey, SNwkSIntKey and NwkSEncKey;
- **JoinEUI instead of AppEUI:** AppEUI is in name changed with JoinEUI, but its purpose remains the same;
- **DevNonce cannot be freely chosen:** previously, DevNonce could be freely chosen by the developer as long as it was unique in the network. Version 1.1 states that DevNonce is now a counter that gets incremented with each join and is persisted over power-cycles as long as JoinEUI remains the same;
- **FOpts is also encrypted:** apart from the FRMPayload, FOpts is now also encrypted. The encryption key that is used to encrypt this field is NwkSEncKey;
- **Other keys for MIC calculation:** instead of NwkSKey, SNwkSIntKey is now used to calculate the MIC for downlink messages to both gls:lorawan 1.0 and 1.1 network servers, FNwkSIntKey for uplink messages to LoRaWAN 1.1 network servers;
- **B_0 is changed for MIC calculation:** when calculating the MIC for LoRaWAN 1.1 network servers, a block format represented by figure 2.11 is used instead.

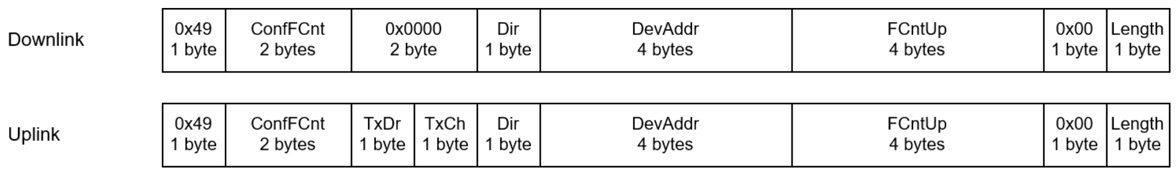


Figure 2.11: The format of the data blocks B_1 that are used during the calculation of the MIC. The data blocks for uplink messages define two extra attributes `TxDr` and `TxCh`, which respectively indicate the transmission data rate and channel.

Chapter 3

Power analysis attacks

While AES is considered to be a strong cryptographic algorithm, it will not guarantee that a cryptographic device will be able to provide secure data encryption. Over the past years, researchers have performed attacks on the system implementation of cryptographic devices with the hope of attacking and breaking otherwise secure cryptographic algorithms. These attacks have been named *side-channel attacks* and have been known to be successful against algorithms such as DES and AES. In this chapter, an introduction to side-channel attacks will first be given to construct an overview about how these side-channel attacks can be used to attack cryptographic devices. Several popular forms of side-channel attacks will be briefly described to give the reader an idea of which side-channels can be used during these attacks. Afterwards, the basic forms of power analysis attacks and techniques related to performing power analysis attacks will be explained in more detail. Finally, several case studies will be performed to address the practicality and effectiveness of the power analysis attacks in practice.

3.1 Side-channel attacks

The security of a system is often affected by the knowledge an attacker has about the system. Take for example that an attacker would like to gain access to information in a secured facility. Knowing which security system is used allows the attacker to closer investigate the system and benefits the attackers chances to infiltrate the secured facility and retrieve the information. In this situation, the attacker would have access to the security details of the facility. But assume that the attacker would not have access to the security details. Would that make the facility secure all of the sudden? There is no certain answer, since the facility could be insecure due to other (external) factors.

A similar situation is also applicable to cryptographic systems. In a cryptographic system, there are also multiple layers on which information about the cryptographic system can be gathered. In general, the more information an attacker is able to gather from a system, the easier it will become to attack the system. Figure 3.1 shows the information layers available inside and outside a cryptographic system.

The various layers indicate where the information can be retrieved from. The following layers are identified:

- **Communication media:** information from radio signals and other physical media is gathered in this layer which is not part of the cryptographic device itself. These physical media are mostly public accessible, which makes it the easiest layer to gather information from. An example of gathering information from this layer is signal fingerprinting and packet sniffing;

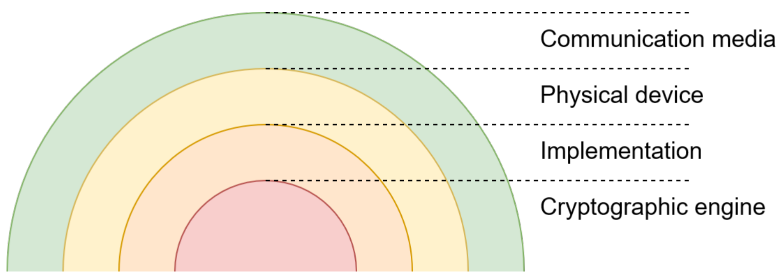


Figure 3.1: The information layers available inside and outside a cryptographic system. The accessibility of the information is represented by the layers from easily accessed (out-most layer) to very difficult to access (inner-most layer).

- **Physical device:** in this layer the information comes from the physical properties of the cryptographic device itself e.g., power consumption of a specific chip or acoustic from the CPU fan. However, custom implementations could prevent information leakage from physical properties;
- **Implementation:** the implementation of the device is often crucial to the security of a system. A bug or vulnerability in the implementation could weaken or break the security of a cryptographic device. Having more information about the implementation of a system is often more difficult to retrieve, but could be critical in attacking the device;
- **Cryptographic engine:** information from the cryptographic engine will provide the attacker with information about the used cryptographic algorithms and possibly the cryptographic keys used by the engine.

The goal of the attacker is to retrieve information about the cryptographic keys in order to decrypt the confidential information. However, as displayed by figure 3.1, this information is often the most difficult to access. Therefore, the attacker may choose to rather focus on the other, more accessible information layers first. The combination of information from several layers could then lead to information about the cryptographic keys. This concept did not apply in the older forms of cryptography from before the age of computers. Up until the last century, cryptography was a mostly a manual operation. A person would have to use an algorithm or a cryptographic device to encrypt the message to ciphertext. The medium was also less accessible, because communication often happened through physical objects such as sealed paper or boxes. Therefore, the strength of the cryptographic system was and still is primarily determined by the strength of the cryptographic keys and the mathematical correctness of the implementation. Since the introduction of the computer and wired and radio communication channels, cryptography shifted from being a manual operation to a more automated operation. Hardware implementations for encrypting and decrypting confidential information became more common.

However, since 1943 it has been known that hardware implementations are prone to leak information about the encrypted data [NSA72]. In 1996, Kocher et al. [Koc96] pointed out that the implementations of modern-day cryptographic algorithms (both in terms of hardware and software) were also a factor to consider when addressing the level of security hardness [Koc96]. Several physical properties of a cryptographic device such as the time or the power needed to encrypt a certain piece of data, could lead to information leakage. By specifically attacking these physical properties, a malicious person could retrieve information such as the cryptographic keys of the encryption algorithms, which would in turn undermine the security of the cryptographic system. These attacks were named *side-channel attacks*, since these properties were observable outside the cryptographic system.

In a real-life situation a vast variety of side-channels would be involved in the usage of a cryptographic device. However, not all of them can be used in a realistic manner when attacking a device. When considering side-channel attacks, one could categorize the attacks based on a level of intrusiveness or destructiveness. For example, some side-channel attacks would require deconstruction of the cryptographic device in order to observe specific parts of the system. In case of a power analysis, this could be the process of opening the enclosure and removing several electronic components such as capacitors which are known to interfere with power measurements. These intrusive side-channel attacks are more likely to be observed by the owners and could therefore lead to the identification of the attack(er). However, there are some side-channel attacks which are less intrusive and destructive like the electromagnetic attacks.

In the following subsections, various forms of side-channel attacks will be briefly described. Afterwards, power analysis attacks will be described in more detail. Power analysis attacks can be a powerful method to attack cryptographic devices. The cryptographic algorithms used by the devices are often paired with intensive calculations such as searching large prime numbers or performing repetitive algebraic calculations. Since heavy computation is often met with an increased power consumption and fluctuations in power consumption, the analysis of the device's power consumption could lead to revealing information about the communication that's being encrypted.

3.1.1 EM analysis attack

The electromagnetic analysis attack is a side-channel attack which focuses on the electromagnetic output of an electrical device during operation time. The electromagnetic attack often targets the electromagnetic radiation that is output by electronic components upon power consumption. The origin of the electromagnetic radiation is found on the level of electrons. Because electrons are charged particles, they emit a magnetic field around them. The amount of electrons moving in the medium will therefore determine the strength of the magnetic field generated in the medium. Given that the magnetic field is strong enough, which corresponds to an adequate current flowing through the medium, the magnetic field will also be observable outside the medium. The electromagnetic analysis attack will target the fluctuations in the strength of the magnetic field to determine the change of the current flow in the targeted device. Similar to the power analysis attack, the electromagnetic analysis attack will also try to correlate the electromagnetic fluctuations and the associated changes in current to the bit changes in the device's memory.

3.1.2 Timing attacks

The timing attack is a side-channel attack which focuses on time-based properties to determine vulnerable information [ZF05]. During a timing attack the attacker will measure the time which is required to perform certain operations using different inputs. Based upon the recorded times the attacker then tries to uncover information which will lead to discovering the vulnerable information.

In general there are two types of timing attacks: the conventional timing attack and the cache timing attack. The conventional timing attack will simply measure the time that is needed to perform a specific (part of the) operation. This attack is mostly used on algorithms that have different runtimes based upon different inputs.

The subtype of timing attacks, the cache timing attack, will also measure the time that is needed to perform a certain (part of an) operation. However, cache timing attacks specifically target the time that is needed for the system to store and use values in their cache memory [Ber05]. Cache memory is a special type of memory which is optimized for fast access by the system, thus improving the performance of the system. The downside of cache memory is that it is very expensive to manufacture compared to the slower main memory. The result is that many systems have a limited amount of cache memory available and are thus required to swap variables between the cache memory and the slower, but larger main memory. When a system tries to execute an operation, it will have to check whether the required variables are available in the cache memory. When the variable is present in the cache memory, the system will be able to retrieve it relatively quickly from the cache memory. This is also called a cache hit [Ber05]. However, if the variable is not present in the cache memory, the system would have to access the relatively slow main memory to retrieve the variable. This is also called a cache miss. While the outcome of the algorithm would unlikely be affected by the absence of the variable in the cache memory, the difference in execution time would be noticeable. This difference in access time forms the base for the cache timing attack. Two notable examples of cache timing attacks are the Meltdown [LSG+18] and Spectre attacks [KHF+18].

3.1.3 Fault attack

Fault attacks are a form of attacks which focus on particular faults in device implementations. In general, there are two types of fault attacks which can be determined: computational fault attacks and input related fault attacks. The first type focuses on computational faults occurring in the cryptographic device [ZF05]. These can be randomly or intentional be induced by modifying physical properties or requirements of the device. One example is inducing faults by changing the supply voltage abruptly.

The second type of fault attacks, the input related fault attacks, are used when the device has shown to follow special routines or be incapable of handling certain inputs. The attacker could use these inputs to gain information about the cryptographic system or could potentially render it unusable for genuine users. An example of an input related fault attack is the fault attack on RSA signature implementation using the Chinese-Remainder-Theorem [BDL97]. By comparing the differences between correct and faulty signatures, a secret modulo N could be derived which is used on the input data to form the signatures. Once N is known, other signature can be forged and thus the cryptosystem will be broken. The use of faulty input thus is the foundation of the attack.

3.1.4 Acoustic attack

Acoustic attacks rely on sound waves to extract information about the cryptographic system. Acoustic attacks in the field of computer security are relatively new, but have shown to be successful in the past [GST14] [GST17]. The technique works by correlating the emitted sound waves of a computer to the operation it is performing. The source of the sound is often found with the low frequency sounds emitted by the processor. Although the processor often operates at much higher clock frequencies, the execution of certain long operations can cause a characteristic acoustic spectral signature to be generated by the processor [GST17]. The timing of the low frequency sound emission can also be associated to the timings of the operations. An example of a successful acoustic attack is the RSA key extraction attack performed by Genkin et al in [GST14].

3.1.5 Error message attack

Error message attack are side channel attack which try to extract information from a cryptographic device by analysing the error messages that are received in response to invalid events or inputs [ZF05]. The most commonly form of attacks using this pattern are the padding oracle attacks. These attacks use padding errors to guess the otherwise inaccessible information. Padding oracle attacks on AES CBC have been shown to be successful in the past. This attack relies on the error responses by the receiver to determine the last plaintext byte associated with a given ciphertext byte. Once the last byte is known, the second-to-last byte can be determined in a similar way, and the process is repeated until the whole plaintext is known.

3.2 The design of a cryptographic device

Before the power analysis attacks can be explained in more detail, the fundamentals upon which it relies need to be explained first. Cryptographic devices are constructed using multiple electronic components. In general, a cryptographic device will have one or more processors and memory which will be used to perform the cryptographic operations. The processors can be dedicated cryptographic processing units or a general-purpose processing units. In high-level terms the difference between these two types of processors is that the first one will only perform cryptographic operations while the latter could be also be used for other operations. While both processors have different purposes, their designs often consist of a common component: the Complementary Metal Oxide Semiconductor (CMOS) cell. CMOS cells are the smallest components which are able to store and manipulate logic data values (i.e., bits).

To further explain how this is relevant for power analysis attacks, the inner workings of a CMOS cell and their inner components have to be explained first. Processors are able to process data that is represented by the aforementioned logic data values. But from a physical perspective, these data values are represented by electricity. The logic data values in CMOS cells are typically defined by the voltage inside the components: a voltage level equal to the components supply voltage V_{DD} represents a value of 1, while a supply voltage equal to the ground voltage GND represents a 0. To manipulate these values, CMOS cells use a network of transistors to form so called *logic gates*. Logic gates provide binary operations on one or more binary inputs in order to produce a single binary output. These logic gates are implemented using transistors which can be used as electrical switches, allowing a current to flow between two entry points (the drain and source) based on the voltage between the source entry and a third entry point (the gate) within the transistor. Depending on the gate-source voltage of the transistor and the type of transistor, a current will flow between the drain and source of the transistor. Using n- and p-channel transistors, numerous logic gates can be constructed such as the NAND gate displayed in figure 3.2.

Whenever a transistor is in the "off" state, the resistance between the source and drain is extremely high, thus only a very low current will be flowing between the source and drain. To switch the "off" state to the "on" state, a voltage higher than the threshold voltage must be applied to the gate of the transistor. In the "on" state, the resistance between the source and drain is defined by the transistors specifications, often referred to as $R_{DS(on)}$. However, the complementary use of transistors in CMOS designs will always cause at least one transistor between V_{DD} and GND to be in the "off" state, effectively creating a high resistance on the path and thus limiting the current flow to be a very small point during static operations. It is only during the switching of states that a high current will flow through the cell, since the switching will momentarily cause a short circuit between V_{DD} and GND [Fai83]. Consequently, a CMOS cell will only consume an adequate amount of power whenever the input values change.

The power consumption characteristics of a CMOS cell will directly be associated with the power analysis attacks since most microprocessors are designed with CMOS logic gates, and use SRAM and static caches which are also commonly implemented with CMOS cells

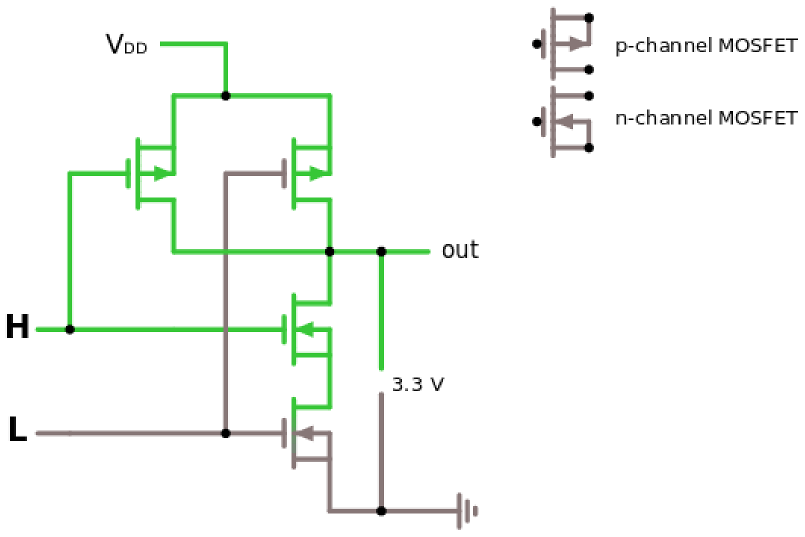


Figure 3.2: An example of a CMOS NAND cell implemented using p- and n-channel MOSFETs. The threshold voltage for all transistors is equal to 0 V, allowing direct application of input voltages and source voltage (V_{DD}) on the transistors.

(figure 3.3 shows an example of a CMOS based SRAM cell). Because both the processing and storage of data in a cryptographic device will involve the use of CMOS technology, it is safe to assume that the execution of cryptographic algorithms, which will involve changes in bit values in the CMOS cells, will have a direct effect on the power consumption of the cryptographic device.

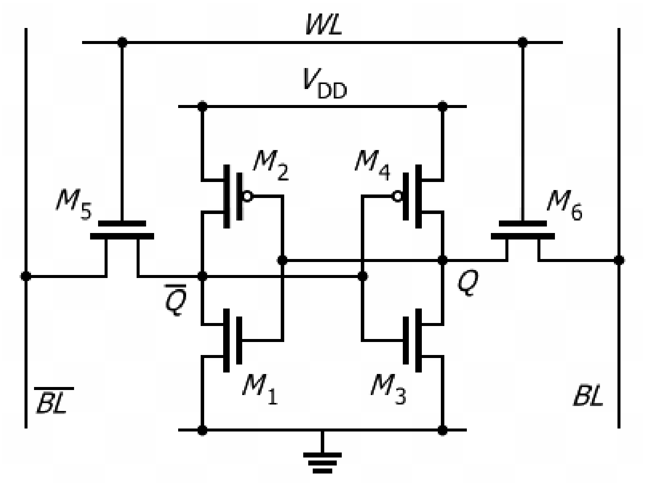


Figure 3.3: Design of a CMOS SRAM cell using 6 transistors [1]

[1]Image source: `https://en.wikipedia.org/wiki/Static_random-access_memory#/media/File:SRAM_Cell_(6_Transistors).svg`

3.3 Simple power analysis (SPA)

The first type of power analysis attacks that will be described is the simple power analysis (SPA) attack. Kocher et al. described simple power analysis as "a technique that involves directly interpreting power consumption measurements collected during cryptographic operations" [KJJ99]. Using SPA, the attacker would thus try to extract the cryptographic key(s) from the measured power consumption traces using a minimal amount of data processing. While the main goal of an SPA attack is to retrieve the cryptographic key(s), it is also possible that other valuable information can be retrieved from the power traces. Several examples include:

- Identification of loops (i.e. identical power consumption patterns) which, for example, can lead to the determination of key lengths;
- Identification of memory accesses (i.e. often lower power consumption) which can be used to identify certain steps within the algorithm;
- Identification of non-intentional repetitive steps (i.e. identical power consumption patterns) which can (for example) be used to identify the re-usage of keys;
- Identification of 'special' events in algorithms (i.e. anomaly in power consumption patterns) which can (for example) be used to derive the secret key(s).

An SPA attack can be performed in several ways depending on the number of traces that are available. A brief description of the most popular methods will be described in the following subsections.

3.3.1 Visual inspection

One of the most straightforward methods to perform an SPA attack is to visually inspect the power traces. In most cases, cryptographic algorithms are sequentially executed by the microprocessor (i.e., instructions are performed in a predefined fixed order). This means that a sequence of instructions is likely to generate the same power consumption characteristics if it is executed repeatedly. On the other hand, some instructions might be known to have a high power consumption such as memory IO operations. These characteristics are often referred to as patterns. By identifying the patterns in a power trace, the attacker could potentially extract information about the cryptographic system.

Kocher et al. demonstrated that a visual inspection of the power consumption traces could be used to target the DES algorithm. The overall steps of the DES algorithm are explained by listing 3.1. Performing a simple power analysis allowed Kocher et al. to determine the number of rounds used by the algorithm as seen in figure 3.4(a). Further analysis also allowed Kocher et al. to identify the number of rotations in the key scheduling algorithm per round. Figure 3.4(b) shows the power consumption of round 2 and 3 of the DES algorithm. In the second round the key scheduling algorithm will rotate the round key once (visible by only one peak in the trace), while the third round will rotate the round key twice (visible by the two peaks in the trace). However, the cryptographic system was not completely broken after the simple power analysis; it only revealed information that could be used to help with other attacks [KJJ99].

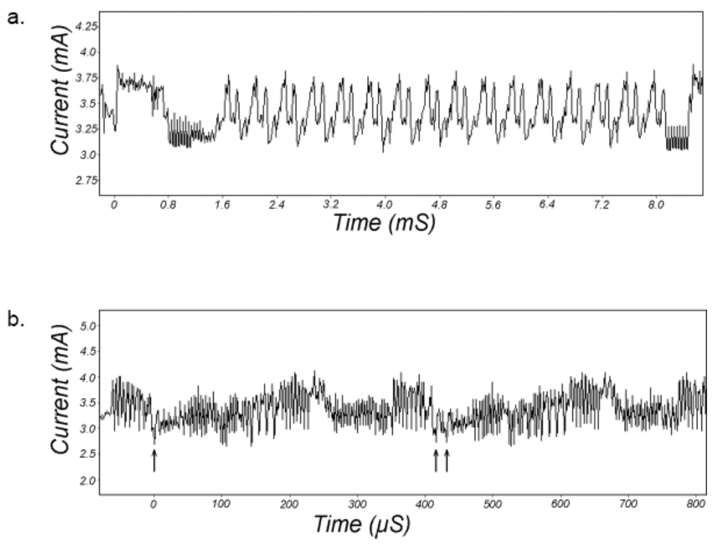


Figure 3.4: Power consumption traces of the DES encryption cipher measured by Kocher et al in [KJJ99]. Subfigure (a) focuses on the overall algorithm with its distinguishable 16 rounds, while subfigure (b) focuses on the round key rotation of rounds 2 and 3.

Visual inspection of power traces does have a couple of requirements before it can be used. First of all, the attacker will need to have a decent knowledge of the internal workings of the cryptographic device. Certain microcode could introduce operand-dependent power consumption, which would increase the device's vulnerability [KJJ99]. Secondly, the attacker must know or derive what kind of encryption algorithm is used by the cryptographic device before he can associate characteristics in the power trace to an algorithms properties. Even when the algorithm is known, there is still the possibility that an SPA attack is unable to extract useful information from the power traces. Kocher et al. [KJJ99] noted that symmetric cryptographic algorithms were less vulnerable to SPA attacks, because in most cases they do not produce as much power variation as asymmetric cryptographic algorithm do.

```
function festel(data, round, key):
    # data = half block of data (32 bits)
    # round = current round
    # key = original secret key
    roundkey = keysched(key, round)
    ext_data = expand(data)
    temp = xor(ext_data, roundkey)
    b = sbox(temp)
    permute(b)

function des_encrypt(pt, key):
    # pt = 64 bits of plaintext data
    # key = 64 bits secret key
    permute(pt)
    a = pt[0:32]
    b = pt[32:64]

    for 16 rounds:
        festel(b)
        a = xor(a, b)
        swap(a, b)

    ct = concat(a, b)
    invPermute(ct)
```

Listing 3.1: Pseudocode describing the global operations of the DES encryption cipher

3.3.2 Template attacks

A second type of SPA attacks that will be explained are the template attacks. Templates attacks build further on the characterization of power consumption by cryptographic operations. While visual inspection depends on the visual representation of the power consumption trace, template attacks will use the statistical properties to characterise the power traces. In a way, template attacks can be seen as the first steps into a machine learning approach of analysing the power consumption traces. A template attack is executed in two steps: the template building step and the template matching step.

Template building

A template attack will need to build templates in order to categorize new power traces. The categorization will afterwards be used to determine the input values (e.g. plaintext, secret key, ...). However, for this categorization to be accurate, a large number of sample data will be needed to form a correct and fine-grained categorization procedure. In machine learning, this sample data is often referred to as training data.

The template building step is therefore the first step to be executed in a template attack. This step requires a large number of power traces along with the plaintext or ciphertext and the used keys. For example, if the first key byte of an AES 128-bit algorithm was targeted and every possible data value was considered, it would take 256^2 power consumption models for the template attack (one model per possible key byte value) [MOP07]. For each model, a couple of thousand power traces would be required to assure an adequately accurate model. However, the number of power consumption models can be reduced if another template is chosen. For example, if the template is chosen to be the result of an operation such as $SBox(d_i \oplus k_j)$, with d_i a data byte value and k_j a key byte value, then the number of templates would be reduced to 256 possibilities (all possible resulting byte values) [MOP07].

However, by mapping multiple inputs to a single power model, the efficiency of the power model will be decreased. As a result, a reduced template will be unable to find the correct key byte given a single power trace from the target device.

Collecting these traces is also not evident as the attacker needs both the data and the key for the template. The most viable approach in doing this is by retrieving an exact copy of the cryptographic device and configuring the device to process chosen plaintexts using chosen secret keys. The power traces are then characterized using statistical properties. In [MOP07], the mean vector m and the covariance matrix C are suggested as characterization properties. However, it should be noted that the covariance matrix grows quadratically with the number of data points in the trace [MOP07]. In the next section it will become clear that a large covariance matrix will influence the processing time during the template matching. Thus, it is advised to determine key points in the power traces which are most unique to a power trace. One possible method of determining unique points over a number of traces is by calculating the sum of distances (see appendix C). After the template building step, each pair of data and key values (d_i, k_j) will be linked to a template $(m(t_{d_i,k_j}), C(t_{d_i,k_j}))$.

Template matching

The second step involves matching the templates to the power traces captured from the device under attack. A frequently used metric to compare sets of data points is the probability density of the normal distribution. The probability density of the normal distribution is defined as follows:

$$f(X|\mu, \sigma^2) = \frac{1}{\sqrt{(2\pi)^k \sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.1}$$

with k the number of data points in X. Adjusted to the available templates $(m(t_{d_i,k_j}), C(t_{d_i,k_j}))$, the metric is calculated as follows [MOP07]:

$$\begin{aligned} f(t|m, C) &= \frac{1}{\sqrt{(2\pi)^k C}} e^{-\frac{(t-m)^2}{2C}} \\ &= \frac{1}{\sqrt{(2\pi)^k C}} e^{-\frac{(t-m)^T C^{-1} (t-m)}{2}} \end{aligned} \tag{3.2}$$

The attacker then proceeds by calculating the probability densities for the power trace(s) captured from the target device for every available power consumption model. Calculating the probability density requires several matrix multiplications to be performed. As mentioned in 3.3.2, the number of data points used for building the templates will affect the size of the covariance matrix C and the mean vector m. Considering the number of probability density calculations that have to be performed, the matching process can be considerably reduced by having smaller covariance matrices. This means that using less data points to generate the templates will have a positive effect on the duration of the template matching.

Once all the probability densities have been calculated for a power trace, the highest probability density will be selected and its arguments (i.e., the power consumption model) will be considered the best guess for this power trace. If a direct power consumption model is used (i.e., a model for each pair (d_i, k_j)), processing a single trace from the target device might yield the correct key byte as a result. Other models will need more power traces in order to derive the correct key bytes.

3.4 Differential power analysis (DPA)

In [GP99], Goubin defined a differential power analysis (DPA) as "an attack that allows to obtain information about the secret key (...) by performing a statistical analysis of the electric consumption records measured for a large number of computations with the same key". Concretely this means that differential power analysis will try to extract the key given a) a set of input values, b) a set of power consumption graphs and c) the output values. The process can (abstractly) be described by the following steps:

1. The attacker targets a certain part of the algorithm and starts collecting the power consumption curves (denoted as E_i) and the output ciphertexts (denoted as C_i) until the targeted part of the algorithm has executed its computations;
2. The attacker defines a selection function $D(C, j, K) \rightarrow b_j$ which simulates the encryption process done by the targeted part of the algorithm on the j-th bit of a ciphertext C that would yield an output b_j using the secret key K.
3. The attacker then generates a curve that represents the mean of the collected power consumption curves. This will be called the Mean Curve in further mentions;
4. The attacker then sets a hypothetical key K_h and uses this key to evaluate $D(C, j, K)$ for each collected C_i.
5. The attacker is now left with combinations of $(D(C_i, j, K_h), E_i) \rightarrow b_j$ where b_j can only be a limited number of possible values. Given the fact that the same sequence of calculations should be done to generate the same output (otherwise the calculations would be irreversible), the attacker can then form groups of combinations which yield the same (expected) output b_j.
6. The attacker can now generate the mean curves for each group and differentiate these curves with each other. The result is a differential trace ΔE.

Kocher et al. [KJJ99] formed a formula which describes this differential trace ΔE as follows:

$$\begin{aligned}\Delta E[t] &= \frac{\sum_{i=1}^{m} D(C_i, j, K) \cdot E_i[t]}{\sum_{i=1}^{m} D(C_i, j, K)} - \frac{\sum_{i=1}^{m} (1 - D(C_i, j, K)) \cdot E_i[t]}{\sum_{i=1}^{m} (1 - D(C_i, j, K))} \\ &\approx 2\left(\frac{\sum_{i=1}^{m} D(C_i, j, K) E_i[t]}{\sum_{i=1}^{m} D(C_i, j, K)} - \frac{\sum_{i=1}^{m} E_i[t]}{m}\right)\end{aligned} \tag{3.3}$$

If K_h is incorrect, then the selection function D will yield random results which are uncorrelated to the actual output. Because the creation of groups is dependent on the result of D, it will cause the groups to formed at random as well. Regarding 3.3, this would result in the differential trace to converge to zero since $m \rightarrow \infty$. However if K_h proves to be correct, it will yield a differential curve which also converges to zero, but shows clear power consumption differences when the targeted bit was accessed or manipulated. The attacker then knows the guessed key K_h was indeed correct.

3.4.1 Higher-order differential power analysis

The DPA attack described above only mentions the use of one intermediate value during the evaluation of the power traces. However, the reliance on this single value also means that the success of the attack depends entirely on the correct interpretation of the intermediate value. Using this knowledge, several cryptographic algorithms have implementations which try to prevent basic power analysis attacks from being successful [MOP07]. This involves modifying the typical power consumption by performing special operations during the execution of the algorithm. These operations will be further explained in 3.7.

However, these countermeasures can be bypassed by extending the aforementioned differential power analysis attack. Until now, the DPA attacks were assumed to be executed on the original power traces without any preprocessing. Such an approach is also called a first-order differential power analysis attack (1O-DPA), since the attack only consists of a single attacking step. However, when the previously mentioned countermeasures are in place, this direct approach will be unsuccessful since the traces contain information related to the processing of misleading intermediate values. Messerges [Mes00] introduced a variant of the original DPA attack which extended the DPA attack by exploiting the leakage derived from two or more intermediate values instead of a single one. This variant was called the higher-order differential power analysis attack (HO-DPA).

Usually higher-order DPA attacks are constructed using a preprocessing stage and an attacking stage. Like the attacking stage, the preprocessing stage is also focussed on a specific operation. Thus, a specific time window can be used during the preprocessing step, namely the time window in which the targeted operation takes place. The following actions are usually taken for the preprocessing stage:

1. Determine the time window in the traces where the vulnerable procedure takes place;
2. Use the interval I from the time window to form all possible pairs of points within the window without introducing duplicates. The order of points in the pair are irrelevant and duplicate pairs are not allowed (i.e. let $v_x, v_y \in T_i$ and $x \neq y$ then $(v_x, v_y) = (v_y, v_x)$);
3. Create new traces with each trace containing the absolute difference of two points formed by the previously formed pairs (i.e. $a = \|v_x - v_y\|$).

The attacking phase is then executed similarly to the first-order differential power analysis attack attack, with the main difference being the change of intermediate values that are used for the evaluation.

3.5 Correlation Power Analysis (CPA)

While differential power analysis (DPA) provides an alternative to the simple power analysis (SPA), it is not an ideal method in every situation where an attacker tries to break the cryptographic system. In [BCO04], Brier et al. stated that there were three major assumptions made when deploying the DPA attack:

- **Word space assumption:** DPA assumes that the prediction of the target bits or bytes is independent of non-targeted bits.
- **Guess space assumption:** DPA assumes that when using wrong hypothetical key information, the predicted values will not depend on the values obtained when using the correct key information. However, if these values are even partially correlated, it could result in unexpected peaks because of additional noise distorting the correct result. These unintended peaks are also called 'ghost peaks' and can be regarded as false positives;
- **Time space assumption:** DPA assumes that a DPA peak will happen only when the value bit is explicitly handled. However it could be that a peak occurs during the algorithm while it is not explicitly handling the value bit.

To overcome these problems, Brier et al. introduced a new power analysis variant called the correlation power analysis (CPA) which uses a leakage model based on Hamming distances.

3.5.1 Coupling Hamming distances to power consumption

The Hamming distance is a metric which is used to express the number of transitions between $0 \rightarrow 1$ and $1 \rightarrow 0$ between two states of machine words over a period of time [MOP07]. For example, assume we have words $R = \texttt{0x11001010}$ and $S = \texttt{0x00101100}$. The Hamming distance between words R and S of length n could then calculated as follows:

$$H(R, S) = \sum_{i=0}^{n} (1 \text{ if } S_i \neq R_i) = 5 \tag{3.4}$$

In section 3.2 it was explained that CMOS cells consume an adequate amount of power when the transistors within the cells are forced to switch between on and off states. The transistors in the CMOS cell must switch states when their logic input values change. Since the Hamming distance expresses the number of bit transitions in a machine between states, it can be assumed that the power consumption of CMOS cell is related to the Hamming distance over an equal time period [BCO04]. This however assumes that the power consumption needed to flip a bit from 0 to 1 is equal to the power consumption of flipping a bit from 1 to 0. Brier et al. stated that these assumptions are quite realistic [BCO04].

Brier et al. also assumed that a linear relationship existed between the power consumption of the device and the Hamming distance between the two states. Given this assumption and by using the Hamming distance as defined by 3.5.1, the number of bits which have been flipped over (the part of) the encryption process can be described as $H(R, S)$. It should be noted that it is difficult to determine exactly what the current state of bits is in the chip's memory at a certain moment; therefore $H(R, S)$ is often replace by hypothetical values. If it can be assumed that the chip has a certain base power consumption b which is unrelated to the flipping of the bits and that the power consumption of flipping the bits is linearly related to the number of bits that are being flipped by a factor a, then the total power consumption during the encryption process can be defined as:

$$W = aH(R, S) + b \tag{3.5}$$

The main question concerning this formula is which states should be chosen as the reference state R and input state C. Ideally, the states before and after the algorithm has accessed or modified critical words are chosen as reference and input states.

3.5.2 Improving the guessing

Since the Hamming distances and the power consumption are put in a linear model, a relationship between the variables should exist. In [BCO04], Brier et al. used the Pearson correlation coefficient (PCC) to describe this relationship:

$$\rho_{W,H} = \frac{cov(W, H)}{\sigma_W \sigma_H} \tag{3.6}$$

The main advantage of the correlation coefficient is that if the model is correct, the correlation coefficient should be maximized when the noise variance is minimized. During the CPA attack, each possible change between two states is considered as a guess. The correlation coefficient can be used as an indication of how well a guess 'fits' within the model. Thus each guess R' for the reference word R can be tested by the following formula:

$$\rho_{W,H(R',S)} = \frac{cov(aH(R, S) + b, H(R', S)}{\sigma_W \sigma_{H(R',S)}} \tag{3.7}$$

After expanding the formula further, Brier et al. came up with the following formula for the correlation coefficient test:

$$\rho_{W,H(R',S)} = \rho_{W,H(R,S)} \frac{m-2k}{m} \tag{3.8}$$

where m is the number of bits in the word and $k = H(R, R')$ or simply the number of bits different between R and R'. This formula leads to three possible situations:

- $k > \frac{m}{2} \Rightarrow \rho W, H(R', S) < 0$: the guessed state change is badly correlated to the observation and thus likely to be wrong;
- $0 < k \leq \frac{m}{2} \Rightarrow 0 \leq \rho_{W,H(R',S)} < 1$: the guessed state change is correlated to the observation and thus a good candidate;
- $k = 0 \Rightarrow \rho_{W,H(R',S)} = 1$: the guessed state change is strongly correlated to the observation and most likely to be the actual reference string.

The total number of guesses can thus be greatly reduced by using the correlation coefficient.

3.6 Vulnerable points in the AES algorithm

3.6.1 Simple Power Analysis and template attacks

In terms of simple power analysis attacks, one could attempt to perform a visual inspection of the power traces in order to identify key parts of the algorithm in the power trace. However, directly retrieving the secret key from the visual inspections is very unlikely to occur. Further processing of the power traces using Hamming weights and distances have proven to be successful in the past though [Man02][CMW14]. These attacks focused on deriving (a part of) the secret key from the `KeyExpansion` step. Clavier et al. were even able to perform the attacks on AES implementations which had countermeasures in place to prevent power analysis attacks [CMW14]. Template attacks and other profiling attacks have been shown to be successful while not specifically targeting specific parts of the AES algorithm [OM07]. However, it should be noted that they require an extensive number of power traces in order to be able to create the templates or profiles.

3.6.2 Differential Power Analysis

Differential and correlation power analysis attacks will use the differences in power consumption to derive the bit changes in the memory of the target device. However, to know the differences in memory is only the first step in deriving the secret key; the bit changes still need to be linked to an operation which involves the use of the secret key [MOP07]. It is therefore that a DPA and CPA attack will target a function $f(d, k)$ where k is the secret key and d is input data on which the secret key is used. In general, there are two options of input data that the attacker could use: the plaintext data that was encrypted (for example, fixed headers of a message protocol) or the ciphertext itself [MOP07].

In CBC mode

Suppose the AES algorithm in CBC mode with a 128-bit key is targeted. In case the plaintext is chosen as the input data d, the CPA attack can focus on the initial steps of the encryption process. The first steps of the encryption process include `KeyExpansion`, `AddRoundKey`, and `SubBytes`. `KeyExpansion` will generate round keys which will be used during the AddRoundKey step [NIS01]. However, the first round key is always the unmodified secret key, effectively making the first AddRoundKey an XOR operation between the unencrypted data stream and the secret key. The following SubBytes operation could thus be seen as a lookup operation of every byte in the XORed data stream against the AES S-box. Since this operation only modifies the bits in the state array, it is often chosen as a point to attack. In this case, the target function for the calculation of intermediate values will be $f(d_i, k_i) = SBox(d_i \oplus IV_i \oplus k_i)$ with d_i being the ith byte of plaintext, IV_i being the ith byte of the IV and k_i being the ith key byte.

It should be noted that in educational examples the IV is almost always set to an all-zero IV, effectively removing the IV from the XOR operation during the calculation of the intermediate value. However, if implemented correctly, the encryption process would use a random IV for each message. It is assumed that this IV is unknown to the attacker, since it could be a fixed value stored onto the device or it could be encrypted during the AES CTR encryption. Therefore, another approach is used to overcome this issue.

In [Jaf07], Jaffe was able to find the secret key of an AES CTR encryption without knowing the counter. Note that the counter in AES CTR is used in a similar way as the IV is used in the CBC mode of AES, namely to randomize the input data[2]. The same principle as Jaffe described can be used for the CBC mode with an unknown IV. First a standard CPA attack is performed on a block B_m. Instead of finding the secret key k, a hypothetical modified key $k' = k \oplus C_{m-1}$ will be found instead, with C_{m-1} being the ciphertext of the previous block. At first this modified key seems not useful, since it can only be used for encrypting block m. However, remember that the first thing that happens when encrypting a block B_m is the AddRoundKey procedure. For a block B_m, this equals to $AddRoundKey(a, b) = a \oplus b \Rightarrow AddRoundKey(k, P_m \oplus C_{m-1}) = k \oplus P_m \oplus C_{m-1}$. Since the XOR operation is associative, $k \oplus C_{m-1}$ can be taken apart and be replaced with the hypothetical modified key k'. Thus the AddRoundKey procedure can be expressed as $AddRoundKey(k', P_m)$. With this in mind, the hypothetical modified key can be considered as the first round key of the encryption of block B_m and a new CPA attack can be performed on the second round. The calculation of the intermediate value used for this attack is described in listing 3.2. The result of this attack will be the second round key, which can be reverted to the initial key using the algorithm in section 2.2.2.

```
def aes_intermediate(key_guess, plaintext, byte_pos, round1_key):
    state = [plaintext[i] ^ round1_key[i] for i in range(16)]
    state = sub_bytes(state)
    state = shift_rows(state)
    state = mix_columns(state)
    return sbox(state[byte_pos] ^ key_guess[byte_pos])
```

Listing 3.2: Algorithm to calculate the intermediate result for AES CBC round 2

Another possible point of attack is the last round of the AES encryption. Since the last rounds of the algorithm are attacked, the ciphertext will used as the input data instead of the plaintext [MOP07]. This point of attack is often considered when hardware acceleration is available in the microchip. Depending on the hardware implementation, hardware acceleration could in theory be able to perform a single round of AES in a single clock cycle [dO18]. This makes it more difficult to detect possible bit changes in the memory. However, the last round of AES is different than the other rounds; it does not perform a MixColumns operation. Since the MixColumns operation adds complexity to the AES rounds, the last round would be less complex and thus easier to analyse than the other rounds. This attack

[2] $N + ctr$ will change the input block for each block during CTR mode. $P \oplus IV$ will change the plaintext before it is used as the input block during CBC mode.

strategy will thus target the last round of AES, which includes the `SubBytes`, `ShiftRows`, and `AddRoundKey` operation between the 9th and 10th round states. The calculation of the intermediate value for this point in the algorithm depends on reversing these operations, which is described in listing 3.3. Effectively, the result of the calculation is the difference of the processed input byte between 9th and 10th round of AES [FAA+17]. Note that the IV is not required in this attack strategy, since the IV is only applied to the plaintext at the start of the algorithm.

```
def aes_intermediate(keyguess, ciphertext, byte_pos):
        state10_byte = ciphertext[CPABase.INVSHIFT_undo[byte_pos]]
        state9_byte = aes_inverse_sbox[ciphertext[byte_pos] ^ keyguess]
        return state9_byte ^ state10_byte
```

Listing 3.3: Reverse operations for the 10th round of AES on a given ciphertext byte and a key guess

In CTR mode

Attacking AES in CTR mode is different from the previous attacks. As explained in 2.3.5, the CTR performs the cipher using the secret key on a counter. This result is then XORed with the plaintext byte. If the attacker is able to identify the initial counter, the previous attack on the S-box in the first round can be adapted to using the current counter instead. The resulting target function will then be $f(ctr_i + b, k_i) = SBox((ctr_i + b) \oplus k_i)$ with ctr_i being the ith byte of the initial counter, b the incremental value corresponding to the number of successful processed blocks and k_i the ith byte of the secret key. Attacking the last round of the encryption process is also a possibility, although slightly different from the last round attack on CBC mode. When attacking the last round, one must take into account that the ciphertext is the result of the XOR operation of the plaintext byte P_i and the output block byte O_i. Attacking the last round of AES requires the result of the cipher which in this case is the output block O_i. In theory, the XOR operation can be reverted to obtain the output block:

$$C_i = O_i \oplus P_i \Rightarrow O_i = C_i \oplus P_i$$

However, this requires both the plaintext P and the ciphertext C to be known. An attack on the S-box operation of the first round is therefore more plausible in practice. This attack strategy does require the initial counter to be known. In [Jaf07], Jaffe was able to determine a method to perform a first-order differential power analysis attack on AES in CTR mode without having to know the initial counter. The attack can be described briefly as follows:

1. For the data acquisition, a minimum of four rounds of the AES algorithm must be captured and a total of T encryptions must be captured [Jaf07];
2. A DPA attack is performed against the first round of the AES algorithm to obtain the 15th and 16th output bytes for the `SubBytes` operation;
3. The encryptions T_i are then selected for which the first fourteen bytes of the input block remain constant throughout the `AddRoundKey` and `SubBytes` operations. This allows the attacker to use a hypothetical output Z_1 for the `SubBytes` operation of the first round [Jaf07];
4. The remaining operations of the round, `ShiftRows` and `MixColumns`, are performed on the hypothetical output values of Z_1. This provides the attacker with a hypothetical input block X_2 for the `AddRoundKey` of the second round. For the known values

$X_{2,1} \cdots X_{2,8}$, an error term $E_{1,j}$ will be symbolically added, with j being the byte position corresponding to the byte of Z_1 for which it is added. The other input bytes $X_{2,9} \cdots X_{2,16}$ are unknown, but constant [Jaf07];

5. `AddRoundKey` is performed on the hypothetical input block X_2 (i.e., $X_2 \oplus K_2$), preserving unknown variables such as the round key K_2 and the error terms $E_{1,j}$ determined in the previous step;
6. Perform a new DPA attack on the second round to acquire the outputs of the `SubBytes` operation for the second round. This will introduce error terms $E_{2,j}$, which are the same fixed error terms that occurred in step 4 [Jaf07]. Therefore, the error terms of the first eight input bytes of X_2 are negated by the XOR operation and the results of the first eight bytes from the `SubBytes` operation of the second round will be known correctly (not hypothetical); the other eight bytes will be unknown but constant;
7. Steps 4, 5, and 6 are repeated to reveal the remaining eight unknown bytes, resulting in the complete and correct output bytes for the `SubBytes` operation of the third encryption round [Jaf07];
8. The `ShiftRows` and `MixColumns` operations are applied to the correct output of the previous step to acquire the correct input for the `AddRoundKey` procedure of the fourth round;
9. A last DPA attack is performed on the fourth round to acquire the correct round key for the fourth encryption round;
10. The resulting round key is reversed using the procedure described in section 2.2.2 to result in the initial secret key.

3.7 Countermeasures

Several countermeasures exist to prevent simple power analysis and differential power analysis from being successful. The first countermeasure that will be discussed is masking. Masking is the process of randomizing the intermediate values of a cryptographic algorithm. It involves a vector of mask values which is used to process the original data before, during and/or after the critical and potentially vulnerable encryption steps are executed. This allows the designer to 'trick' the attacker into believing other input values are being used during the encryption steps, which would result in different intermediate values being used during the critical encryption steps. Since power analysis attacks rely on these intermediate values, the attacker would be presented with misleading information which would influence the efficiency of the power analysis dramatically.

Another well used countermeasure is hiding. Hiding is a technique which tries to break the relation between power consumption and the processed values. There are two possible ways to achieve this. The first one is by completely randomizing power consumption of the device. For this to happen, the device will have to consume a random amount of power per cycle on top of its regular usage. The second method is by levelling the overall power consumption to a constant level. This means that the power consumption of the device remains the same independent of the operations that were performed and data values involved.

While both approaches sound promising, it is not so evidently implemented as a solution: perfect randomization or equalization cannot be reached in practice [MOP07]. Several solutions have been suggested over the past which do try to achieve a similar result. These solutions have been sorted into two categories: the time related and amplitude related solutions. Time dimension related solutions include the addition of dummy operations and the shuffling the operations that can be run independently. Amplitude dimension related solutions include increasing and reducing the noise[3]generated during the execution of the operations. However, both approaches can often be bypassed by higher-order differential power analysis attacks.

3.8 Measuring power consumption

All of the above methods require traces of power consumption from the device to extract the key information from the device.

To accomplish this, a digital storage oscilloscope (DSO) can be used. A regular oscilloscope is an electronic measuring device capable of measuring the electrical quantities (usually voltage) as a function of time [IEE00]. A DSO makes it possible to store these quantities in memory for analysis and data transfer. An oscilloscope is able to measure the electrical quantities via a tool called a probe. A probe features two important connections: the probe pin, which is usually responsible for measuring the electrical quantities, and a ground clip which is directly connected to the oscilloscopes ground and serves as a reference point for the measurements. The probe is connected to the electrical circuit using the probe pin and optionally the ground clip. In most cases the oscilloscope is used to measure voltages in an electrical circuit. In this scenario the potential difference between the probe pin and the ground clip will determine the voltage value.

For the power analysis measurements the current of a device must be measured since the current flow in the CMOS cell will increase temporarily if one or more values change. Most oscilloscopes cannot read electrical current values directly. In order to measure the current, a resistor can be placed in series with either the power or ground input of the targeted component. This provides the attacker with a voltage U_r across the resistor. Since the current is equal across all components in a series connection, the current I can be calculated from the voltage U_r using Ohm's law:

$$I = \frac{U_r}{R} \tag{3.9}$$

Since resistors placed in series will affect the voltage to other components in the series connection, it will be necessary to keep the resistor small in terms of resistance. The higher the resistance of the resistor is, the higher the voltage over the resistor will become within the series connection, potentially causing the other devices to receive an insufficient amount of voltage. This statement is proven in appendix B. On the other hand, when the resistor is too small, it will become more difficult to measure voltage differences with the measuring device since the voltage differences over the resistor will become smaller and the oscilloscope is limited in the resolution of voltages it can measure.

While inserting the resistor gives the attacker a method of measuring the current changes in an electrical circuit with an oscilloscope, there are still other aspects that need to be taken into account.

Bandwidth and sample rate

In signal processing, it is assumed that each signal can be represented by a sum of sinusoidal functions multiplied by specific coefficients [MOP07]. The sinusoidal functions present in the signal have a specific frequency f at which it occurs in the signal. Figure 3.5, for example, shows a sum of two sinusoids, one at 14 MHz and the other at 20 MHz. By using a Fast-Fourier transformation (FFT), these frequencies can be extracted from a signal. The distance between the lowest and highest frequency component in a signal is called the bandwidth of a signal [MOP07]. Oscilloscopes have a limited range of bandwidth they can

[3]Reducing the noise actually means that the differences in power consumption for different data values are mitigated as much as possible by use of filters, etc.

measure. Most modern oscilloscopes are able to handle signals with bandwidths of several 100 MHzs [MOP07]. The advertised bandwidth of a digital oscilloscope is often equal to the maximum frequency that can be measured by the oscilloscope. The reason for this is because oscilloscopes have no trouble measuring low-frequency signals. Therefore, the bandwidth of an oscilloscope is equal to $f_{max} - 0$.

Another requirement to keep in mind during the measurements, is the minimal sample rate that has to be used. Most digital oscilloscopes have sample rates of a couple of hundred million samples per second or megasamples per second. The sample rate indicates how many times per second the oscilloscope is able to measure the amplitude of the signal. For example, if the oscilloscope has a sample rate of 500 megasamples, it will be able to measure the amplitude of the signal every 2 ns. While the sample rate and bandwidth are closely related to one another, they can have a different meaning when used in the specifications of digital oscilloscopes [dme17]. Digital oscilloscopes sometimes advertise to be able to capture signals with bandwidths higher than the sample rate (for example 100 MHz bandwidth using a 40 megasamples per second sample rate). A digital oscilloscope is able to do this when a periodic signal is detected. The digital oscilloscope will then use sub-sampling to reconstruct the higher frequency signals from multiple measurements. During power analysis attacks, the voltage measurements are prone to change based on the input that the device has to process, so the sample rate is often of more relevance to the measurements than the advertised bandwidth.

Both the sample rate and the actual bandwidth are important when considering measuring power consumption of an electronic device. Microprocessors such as the ATmega328P usually run at a clock speed ranging from 1 to 20 MHz. This means that the transistors within the processor are able to switch 1 to 20 million times per second. Since the oscilloscope is supposed to measure the current changes involved with the logic value changes within the microprocessor, it will (in theory) need to be able to measure at a rate of at least the clock speed of the microprocessor.

However, it should be noted that measuring samples of a signal only provides a linear change between the samples. Since voltage signals are actually sinusoids, this can cause a problem when the sample rate is too low or the signals frequency too high. Figure 3.5 shows what the result of using various sample rates can cause. This is also confirmed by the Shannon-Nyquist theorem; it states that the sample rate f_s should be at least $2f_{max}$ [MOP07]. However, it should be noted that electrical components are occasional capable of generating unwanted high-frequency noise signals. Therefore, it is often recommended to find the dominant frequency (e.g. the highest operating frequency of relevant electrical components in the circuit) and to use a sample rate equal to twice or more the dominant frequency [MOP07].

Resolution

A third requirement that the oscilloscope has to fulfil is the minimum resolution. A digital storage oscilloscope uses an analogue-to-digital converter (ADC) to convert analogue samples to digital values. However, an ADC is often limited in the number of output values it can use to map analogue values to. This is also called the ADCs resolution. For example, an 8-bit ADC will be able to output 2^8 distinct values. These output values will not directly represent an absolute voltage value, but an integer value. This integer value will be used to calculate the absolute voltage value based on the voltage range applied by the oscilloscope. For example, a voltage range from 20 mV to 276 mV results in a 1 mV step size for the possible output values on an 8-bit ADC. For an output value i, the absolute voltage can then be calculated as follows:

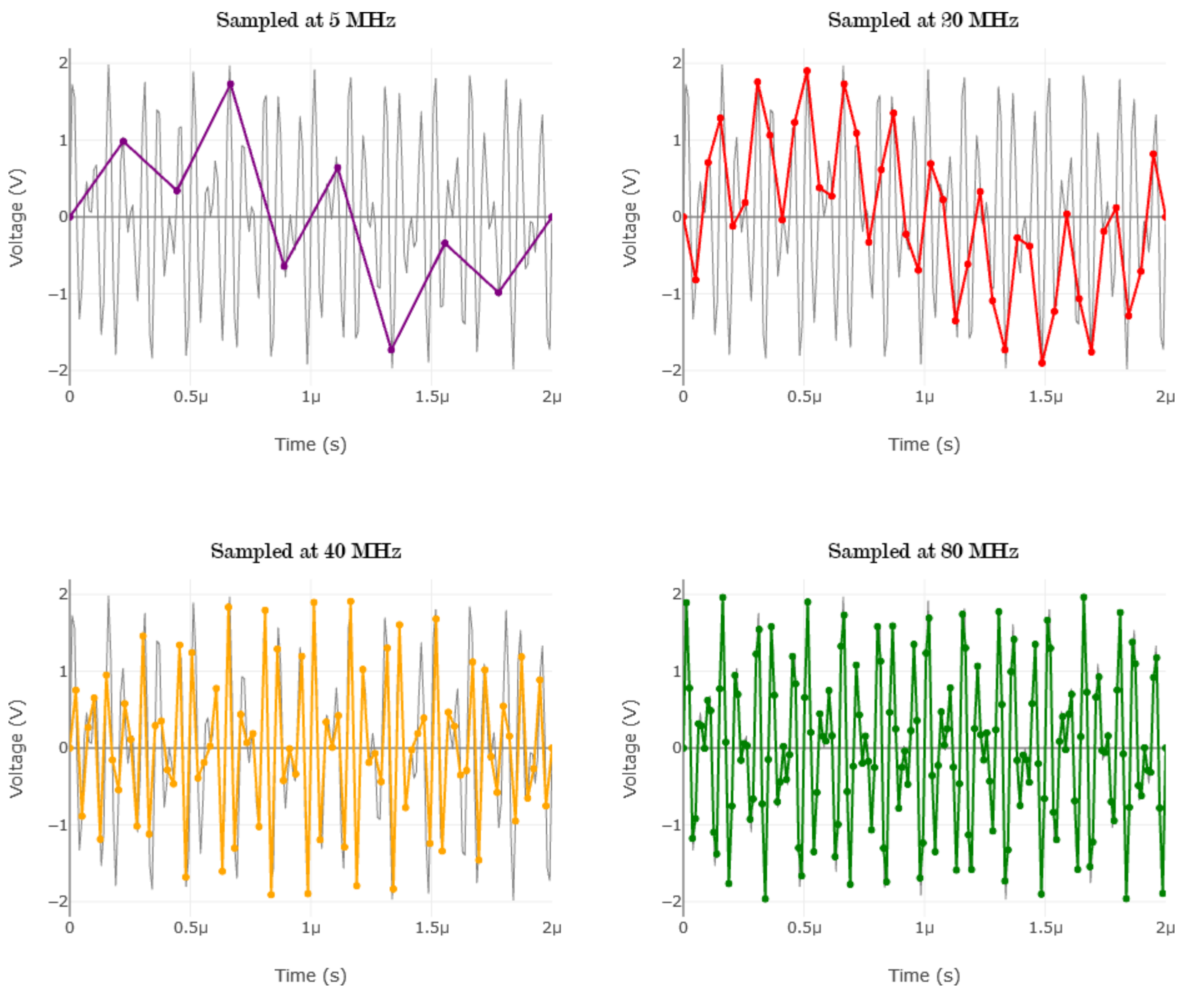


Figure 3.5: Example of how different sample rates influence the reproducibility of the signal. The simulated signal is generated by the function $f(t) = sin(20 \cdot 10^6 \cdot 2\pi t) + sin(14 \cdot 10^6 \cdot 2\pi t)$. The highest frequency component is thus 20 MHz. Note that the characteristics of the signal are best saved when a sample rate of 80 MHz is used, i.e. when $f_s > 2f_{max}$.

$$\begin{aligned} V &= V_{min} + i \cdot \frac{V_{max} - V_{min}}{2^8} \\ &= 20 \text{ mV} + i \cdot \frac{276 \text{ mV} - 20 \text{ mV}}{256} \end{aligned}$$

Since analogue signals are able to produce an infinite number of distinct values within a range, the ADC could (and will) map multiple distinct values to a single digital representable value. This could cause a problem during the measurements when the voltage differences are too small to map to other output values. Decreasing the voltage range on the oscilloscope could solve this problem, but could also cause analogue values outside this range to be cut off. A second problem is that the upper and lower bounds of the ranges are limited on most oscilloscopes. Therefore, it could be possible that a large offset (e.g. 5 V when a 100 mV scale is used) cannot be applied on the oscilloscope. This would make it impossible to measure small voltage differences above or below the offset even if the resolution would allow it. This is often one of the reasons why the resistor in a measuring set-up is place at the ground pin: when using a probe referenced to the ground, the voltage measured at the ground pin is often near to 0 V, thus eliminating the need to apply an offset on the oscilloscope.

Triggers

An oscilloscope shows measured samples over a period of time. Since the internal storage of an oscilloscope is limited, it will be unable to store samples over a long period of time. Therefore the oscilloscope is often set up to save information over a specific time window. However, when performing a power analysis attack the attacker is usually interested in a specific time window where a specific event occurs. In order to do so, an oscilloscope features a triggering mechanism. Triggers allow the oscilloscope user to define an event or restriction which "triggers" the oscilloscope into measuring and storing the samples within a certain time window. For example, suppose the interested part of a signal starts with a short peak voltage of 5 V. An oscilloscopes trigger could then be set to an edge trigger with threshold value 5 V. Whenever a peak of 5 V would occur in the real-time measurements, the oscilloscope will perform measurements to complete the current time window, but afterwards will stop measuring new samples and let the current time window of samples remain in the memory.

Determining which trigger is most useful for measurements often depends on the power consumption characteristics of the target device and the implementation. Most power analysis set-ups use a communication protocol bus as a point of triggering. Some oscilloscope are capable of decoding communication protocols such as RS-232 and I^2C allow a trigger to be set on the occurrence of specific data values in this communication protocol. Other methods include the combined use of edge, pulse and slope triggers and trace characteristics to trigger the readings at a specific time. This also includes trigger on GPIO output values.

Data storage and transfer

In the previous subsection it was already mentioned that the internal memory of an oscilloscope is often limited. For this reason, DSOs often provide ways of storing or transferring trace data to other devices. Most modern DSOs provide an USB connection to either connect the oscilloscope to a PC or a USB storage device to the oscilloscope. Depending on the implementation and the software provided by the manufacturer this could be a straightforward operation. However, sometimes the software or implementation could be a limiting factor during measurements. Manufacturers therefore often include remote access functionality for the oscilloscope, using protocols such as LXI, VXI-11, and USBTMC to provide developers with low-level functionality to the oscilloscope.

3.8.1 Measuring issues

Even when the equipment is configured and capable of measuring the signal in theory, there are still issues that could arise during the measurements. Several issues that have been experienced during the experiments in the case studies are described in this subsection.

Electrical noise

Electrical noise or interference are undesirable signals which distort or interfere wth the desired signals [VBB04]. The presence of electrical noise could cause the oscilloscope to measure distorted power consumption values. These distorted values could decrease the effectiveness of the power analysis attacks described in the previous sections. Therefore it is adamant that the electrical noise in a circuit is kept to a minimum.

Electrical noise can be cause by numerous things, for example faulty power supplies or components and cross talk from other devices. Vijayaraghavan [VBB04] stated that for electrical noise to exist in a circuit, three contributing factors must be in place: a source of electrical noise (e.g. faulty component or other devices on the power grid), a mechanism coupling the source to the affected circuit (e.g., the power grid itself) and a circuit conveying sensitive communication signals. While there is no concrete communication signal targeted during power analysis measurements, the voltage signal that must be captured using the oscilloscope can be seen as a "communication signal" since it will be interpreted by the oscilloscope and later by one of the power analysis attack algorithms.

To mitigate the electrical noise from the affected circuit, the attacker has several options [VBB04]. The first option is to find the source of the electrical noise and remove it or its coupling mechanism. There are various sources known to cause interference in a circuit. Examples include electrical motors, fluorescent tubes and welding equipment. However, there may also be other causes for devices to become a source of noise; badly manufactured or ageing capacitors are also notorious for generating noise in otherwise stable electrical circuits. For this reason, it is often difficult to locate all noise sources. An alternative option is to separate the affected circuit from other circuits and provide an isolated and stable power supply. Lab grade power supplies and battery packs can be used to achieve this.

Another option is to filter the noise from the measured signal. This requires signal analysis to determine which frequencies are dominant and which are redundant. A Fast-Fourier transformation (FFT) of the measured signal could provide more insight into these frequencies. When referring to noise in terms of frequencies, there are three types of noise to be determined [VBB04]:

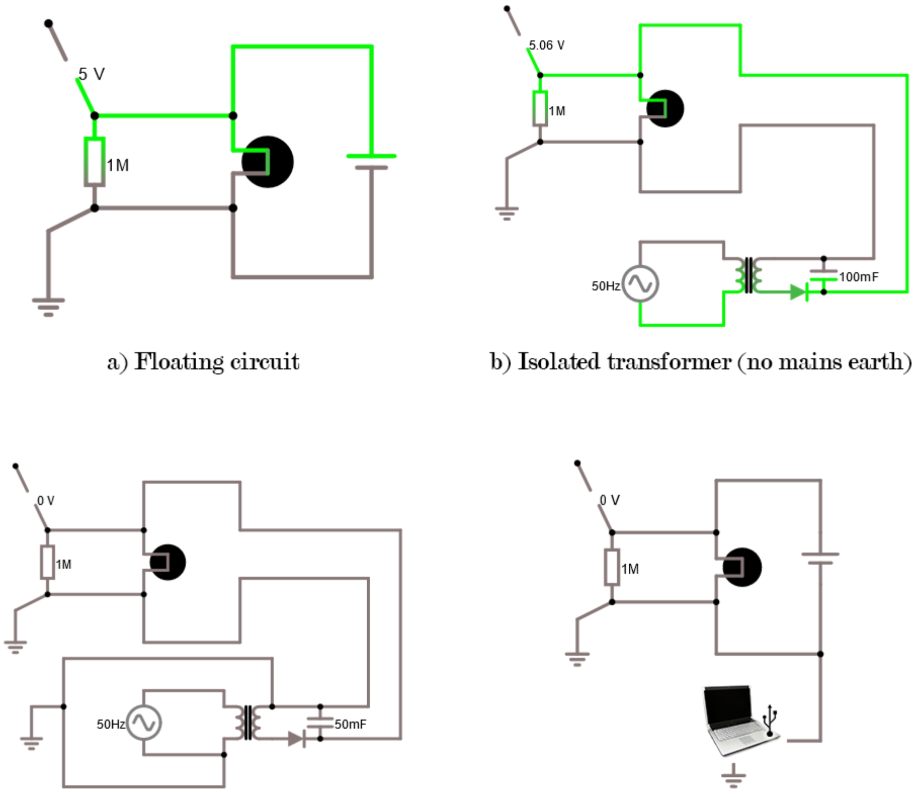
- **Wideband noise:** wideband noise often occurs in various amplitudes and frequencies and are thus difficult to filter from the signal without loss of information from the desired signal;
- **Impulse noise:** impulse noise is defined by a sudden peak or drop in the signal, often occurring for a limited time of period in the signal. This type of noise could also cause loss of information to the original signal, but is less widespread than the wideband noise. If the impulse occurs in a periodic manner and are fixed in amplitude, then a notch filter could still be able to filter most of the noise from the measured signal;
- **Constant frequency noise:** constant frequency noise may be the easiest noise to filter from the measured signal. Low-pass, high-pass and band-pass filters can be used to filter this type of noise with reasonable results.

Ground loops

IEEE states that ground loops occur "when two or more points in the electrical system, that are nominally at ground potential, are connected by a conducting path such that either or both points are not at the same ground potential" [IEE00]. The probes of the oscilloscope that will be used for the case studies are connected to the oscilloscope via a BNC connector. The BNC connector is a connector type used for coaxial cables. Like the coaxial cable, a BNC connector has two conduction parts: the centre core, which is a conducting wire in the centre of the cable, and a shield which is a woven layer of small wires or a metallic foil on the outside of the isolated layer around the centre core. The shield of the coax cable and BNC connector are typically kept at ground potential while a signal carrying voltage is applied to the center core. For an oscilloscope, this means that the ground clip of the probe is connected to the ground potential of the scope, which is usually the mains earth. Note that the path between the ground clip of the scope and the mains earth is a low impedance path, which means a current can flow through the path with almost no resistance [Jon12].

The problem of ground loops occurs when differential probing is performed using a single probe. Differential probing with a single probe means that the ground clip is inserted into the electrical circuit which needs to be measured. This can cause the ground clip to be connected to a potential different from the ground potential (e.g., the 5 V supply voltage).

When this happens the mains earth line will act as a conductor wire with a different potential applied to one end [Jon12]. This can potentially cause a short circuit if the circuit is closed between the scopes ground pin and another device with the ground connected to the same ground wire. An example of such a construction is given in figure 3.6c.



a) **Floating circuit** b) **Isolated transformer (no mains earth)**

c) **Grounded transformer** d) **USB powered**

Figure 3.6: Measurement set-ups for common electrical circuits involving grounding. The first two constructions are not susceptible for ground loops, where the last two constructions can potentially be shorted by a ground loop.

At first, this does not seem to be a problem since no earth-grounded power supply is applied to the development board in the measuring set-ups. However, the issue lies with the "hidden" earth-ground on the USB connection that is used for the serial communication [Jon12]. While the USB connection can provide a 5 V DC current to the development board, it is not an isolated power supply. The ground pin of the USB connection is connected to the internal ground line of the computer^4. This ground line is mostly connected to the mains earth, effectively causing the ground pin of the USB connection to act as a mains earth connection. Figure 3.6d shows an example of this scenario.

Several solutions exist to prevent these scenarios from happening. In case of the USB scenario, a laptop could be used instead of an AC powered computer. However, the power supply of the laptop could still provide an earth-grounded connection for the USB ports on the laptop. To validate this, the resistance between the ground connector of the laptop supply and the ground pin of the power supply cable can be measured using a multimeter [Jon12]. Figure 3.7 shows how this measurement is performed. If the measurement reads anything else than the maximum impedance of the multimeter, then it indicates that the laptop is mains earth grounded. In this case it is advised to disconnect the power supply when performing measurements.

Another solution is to use an isolated differential probe instead of a regular probe to

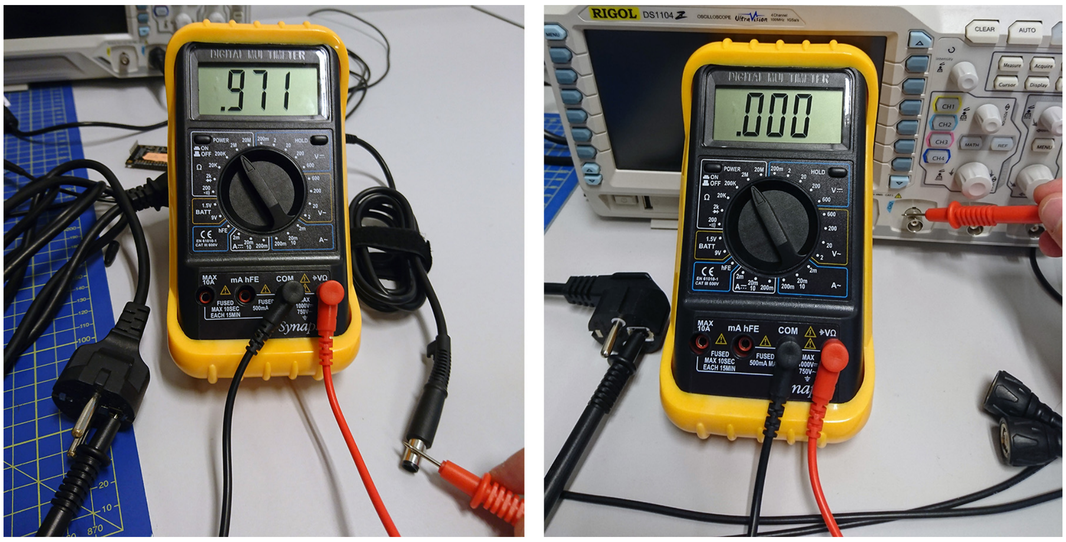


Figure 3.7: Verification of a grounded laptop power supply and the ground of the oscilloscope. The laptop power supply features a resistance of 1 MΩ, which is considered enough to discharge any regular current at an interval of less than a second. The oscilloscope however has a low impedance path to the mains earth.

perform the measurements [Jon12]. The ground clip of a differential probe is isolated from mains earth, effectively preventing another potential to reach the mains earth line. However, purchasing a differential probe can be costly. If multiple regular probes are available, a differential probe can be simulated by two regular probes. Instead of inserting the ground clip of the first probe to the electrical circuit, the measuring tip of the second probe is inserted. The two measurements can then be subtracted from each other to achieve the same measurements as with the differential probe. This set-up also eliminates potential noise from the power supply from being recorded.

Misaligned traces

When performing power analysis measurements with an oscilloscope, a common measuring strategy is to use a trigger signal that can be found right before or after the cryptographic operation takes place. The two most used trigger signals are from either direct power readings or communication signals [MOP07].

Triggering on direct power signals is often very difficult. It requires a reoccurring pattern to be present in the power consumption trace that is observable closely before or after, or during the cryptographic operation. Most trigger values for this type of signals are based on peaks or pulses. However, noise can also be a cause for triggers to be fired during measurements. A second problem is that most devices do not show a clear power consumption pattern that can be linked to the cryptographic operation [MOP07].

As an alternative, one of the communication signals is often used to fire a trigger. For example, a GPIO pin can be momentarily set to high (1) just before the cryptographic operation starts. The oscilloscope notices the change in signal and starts to measure the voltage data after the trigger signal. However, communication IO is often not performed synchronously with the clock signal of the processor [MOP07]. It is also possible that other instructions are performed before the cryptographic operations that are variable in duration. Therefore, it is possible that there is a variable delay between the trigger signal and the power consumption samples of interest.

In this scenario, the traces will need to be realigned. A possible method is to use a reference pattern and shift other traces according to this reference pattern. Metrics such as the mean squared error (MSE), the distance between maximum values, or the correlation coefficient can be used to find the number of position shifts that yields the best results. In this thesis, the correlation coefficient is used as the primary metric for the alignment of traces.

3.9 Case studies

Since there are various parameters which could cause a power analysis to fail, an incremental approach was opted in order to test the various parts involved in a power analysis. The first tests involve verifying the attack algorithm by using test datasets which were proven to be vulnerable for a power analysis (see section 3.9.3). Afterwards, different set-ups using an actual device were used to verify that the power consumption measurements were performed correctly. The target device was an ATmega328P microcontroller. The next step was to implement the AES algorithm in CCM mode on the ATmega328P using the same construction in order to retrieve traces and modify the attack algorithm to perform correctly under the CCM mode. The final step was to perform the attack on a Microchip RN2483 LoRa transmitter.

3.9.1 Measuring equipment

A Rigol DS1104Z digital storage oscilloscope (DSO) was used to measure the power consumption of the devices in the case studies. The Rigol DS1104 provides the following features:

- 4 input channels, allowing up to four probes to be connected to the oscilloscope;
- Maximal bandwidth of 100 MHz;
- Maximal sample rate of 1 gigasamples per second;
- Deep Memory, allowing up to 12 million points to be stored in-memory;
- Remote access via USBTMC and LAN connectivity;
- Extensive trigger support;
- Real-time mathematical functions;
- Support for decoding communication protocols such as RS-232.

The measured points are initially stored in the main memory of the oscilloscope. The Rigol oscilloscope supports remote control via USB and Ethernet. First it was opted to use the USB port to transfer the data to the host computer via USB flash drive. The Rigol oscilloscope supports exporting data in waveform format (.wmf) and in comma-separated text format (.csv). The waveform format is encoded with a proprietary format that is susceptible for changes across firmware versions. The waveform format is also not described in the programmer's and user manual, which makes decoding the format a difficult process. This made it impractical to use the waveform format for prolonged usage. The comma-separated text format was also not practical in use: the oscilloscope needed to convert the byte values from the internal memory to ASCII readable characters, which took 10 to 15 minutes given the limited processing power of the oscilloscope. Thus it was opted to use remote access over the network to retrieve the voltage samples. The oscilloscope provides two ways to communicate with it: either by installing a device driver which can be used as a back-end service to communicate according to the VISA specification, or via a Telnet-like TCP connection using the LXI specification. Both specifications use the syntax standard SCPI to format commands and data for the measuring instruments. Basic implementations exists for the Rigol oscilloscope in Python, but are rather limited in usability. To increase the usability of the oscilloscope in Python, a new library based upon the LXI specification was developed for the Rigol DS1104Z by the author.

3.9.2 Implementation details

The implementation of the correlation power analysis (CPA) is written in Python 3 and was adapted from the algorithm used by the power analysis tool ChipWhisperer [New16][New13]. The algorithm from ChipWhisperer itself is derived from the CPA algorithm described in section 3.5. One noticeable detail about the implementation is that it can use different attack strategies to target different parts of the AES algorithm. For first-order CPA attacks, the only difference is often the leakage value which will be used to determine the appropriate Hamming distance for calculating the correlation coefficient. Like in the ChipWhisperer implementation, this implementation will use attack strategy classes where each class defines the calculation for the leakage value in a function `aes_intermediate`. An example of an attack strategy class is presented by listing 3.4.

Apart from the calculation of the leakage value, some attack strategies will also yield a specific round key instead of the initial secret key. Therefore, an attack strategy class will also provide a method `postprocessKeyguess`. This method will process the outputted round key and return the initial secret key. In case of an attack strategy that is targeted towards the final round of AES, this method will reverse the round key of the 10th round to the initial key. An explanation of how this procedure works is described in section 2.2.2.

During the attack, the partial guessing entropy (PGE) values are calculated for each key byte guess if the secret key was known prior to the attack. The PGE value is a metric which indicates how many successive guesses are needed to determine the true key byte value. Positive PGE values indicate that there are n incorrect guesses higher ranked than the actual key byte value. Consequently; a PGE value of 0 indicates that the guess was correct.

At the end of the procedure, the key guess and the corresponding PGE values for each key byte guess are returned. The implementation also provides the means to plot the correlation coefficient per sample for each key guess, which shows the position in the trace where the key guess was found to be most fitting.

```python
class FirstOrderLastRoundCiphertextAttack(CPABase):
    @staticmethod
    def aes_intermediate(keyguess, ciphertext=None, byte_num=None, **kwargs):
        state10_byte = ciphertext[CPABase.INVSHIFT_undo[byte_num]]
        state9_byte = aes_inverse_sbox[ciphertext[byte_num] ^ keyguess]
        return state9_byte ^ state10_byte

    @staticmethod
    def postprocessKeyguess(fullkeyguess):
        return reverseRoundKey(fullkeyguess, 10)
```

Listing 3.4: Example of an attack strategy class

3.9.3 Case study: power analysis using test datasets

Since incorrect power consumption measurements could cause a power analysis attack algorithm to fail, it was opted to first implement a correct attack algorithm using reference data instead of self-recorded traces. In order to test the implementation of the CPA attack algorithm, three test datasets were used which were already tested for CPA attacks and were found to be vulnerable. An overview of the datasets is provided in table 3.1.

Dataset 1: avr_aes128_picoscope6403D.zip[5]					
# traces	2000	**Samples/trace**	16000	**Sample rate**	312 MS/s
Algorithm	AES CBC	**Key size**	128	**Target round**	First

Dataset 2: sakurag-picoscope6403d-4000.zip[6]					
# traces	4000	**Samples/trace**	1500	**Sample rate**	156 MS/s
Algorithm	AES CBC	**Key size**	128	**Target round**	Last

Dataset 3: ASCAD.h5[7]					
# traces	10000	**Samples/trace**	700	**Sample rate**	2 GS/s
Algorithm	AES CBC	**Key size**	128	**Target round**	First

Table 3.1: Overview of test datasets used to verify CPA algorithm

The first two datasets (`avr_aes128_picoscope6403D.zip` and `sakurag-picoscope6403d -4000.zip`) follow the file structure used by the power analysis tool ChipWhisperer. The file structure provides the following files:

- `<prefix>_traces.npy`: an array of traces with each trace containing a number of voltage values;
- `<prefix>_textin.npy`: an array of input strings used during the encryption process. Each input string is linked to one of the previously mentioned power traces in sorting order of position and each input string is 128 bits long;
- `<prefix>_textout.npy`: an array of output strings (ciphertexts) resulting from the encryption process. Like the input strings, each output string is linked to one of the previously mentioned power traces in order of position and each output string is 128 bits long;
- `<prefix>_knownkey.npy`: a single secret key which was used for all encryption processes. This is the key that should be obtained from the CPA attack.

`<prefix>_keylist.npy` and `config_<prefix>_.cfg` are also included in the datasets, but will not be used in the power analysis attacks.

The third dataset is a test dataset provided by ANSSI and CEA. The ASCAD dataset was constructed by measuring EM signals coming from an ATmega8515 while performing a masked implementation of the AES algorithm. The associated paper [PSB+18] states that the third key byte should be targeted for non-profiling attacks. It also mentions that there is no first-order leakage in the unmasked S-box output $SBox(p[3] \bigoplus k[3])$. However, the paper also states that it is possible to calculate the masked output after the first round by using $SBox(p[3] \bigoplus k[3]) \bigoplus r_{out}$. r_{out} is the output mask byte which was also provided in the dataset per trace.

The dataset itself is represented by an H5 data file which contains the trace values and other information about the traces. The data file provides two groups of traces: the attack traces and the profiling traces. The profiling traces are originally meant to create machine learning profiles, the attack traces are meant for the actual attacking phase. For this case study, the attack traces will be targeted.

Unlike the previous two datasets, the ASCAD dataset is differently formatted. Using h5py, the dataset is represented as a dictionary-like object which includes the following

[7]`https://www.assembla.com/spaces/chipwhisperer/documents/auoySCKlGr44oRacwqjQXA/download/auoySCKlGr44oRacwqjQXA`

[7]`https://www.assembla.com/spaces/chipwhisperer/documents/c9R60mJrSr44noacwqjQWU/download/c9R60mJrSr44noacwqjQWU`

[7]`https://github.com/ANSSI-FR/ASCAD`

entries:

- `traces`: an array of traces with each trace containing a number of voltage values;
- `labels`: an array of labels that are assigned to the traces for classification. This data is specifically meant for machine-learning purposes and will not be used in this case study;
- `metadata`: an array of metadata structures for each trace with each structure containing the following fields:
  - `plaintext`: the 128-bit input string used for the encryption process;
  - `key`: the 128-bit key used for the encryption process;
  - `ciphertext`: the 128-bit ciphertext resulting from the encryption process;
  - `masks`: the different mask vectors that were used to obfuscate the data values during the encryption process;
  - desync: the amount of positions that the trace were shifted to simulate jitter (misalignment).

Dataset 1: `avr_aes128_picoscope6403D.zip`

The first dataset is a set of power traces from an ATmega328P using a software implementation to perform AES CBC encryptions. The microcontroller is clocked at 7.37 MHz. The power trace was recorded using a PicoScope 6403D digital oscilloscope. During the measurements, a construction with a shunt resistor inserted before the V_{cc} pin was used. The description of the dataset suggests an attack on the first round of the AES algorithm.

A visual inspection of the trace was first performed to verify that the power trace represented the power consumption of an AES encryption. A good indication is often the presence of 11 peaks or drops in the trace which corresponds to the 11 rounds in the AES algorithm. However, a visual inspection is not always conclusive; the peaks could also be generated by noise from other components. At the same time, other noise could also cause the distinctive pattern to be obscured. Figure 3.8 shows the power consumption of a single AES encryption from the first test dataset. Through visual inspection several recurring patterns of peaks and drops can be observed in the power consumption trace. However, the number of these patterns is much higher than the expected number of rounds that are performed by the AES algorithm. Therefore, this first inspection is unable to conclude if the power trace is indeed the power trace of an AES encryption.

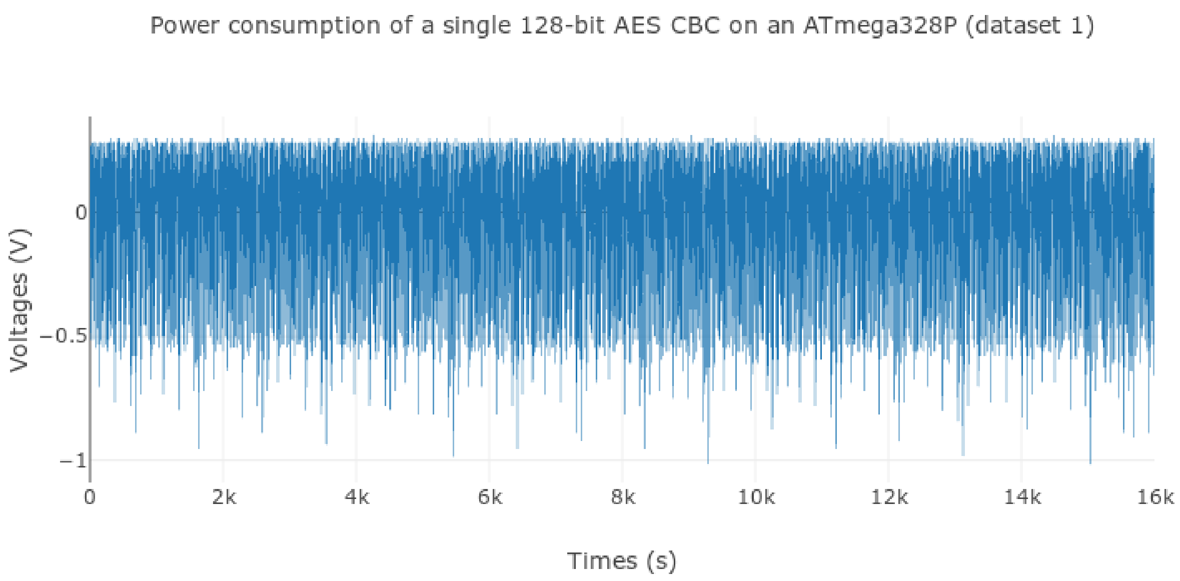


Figure 3.8: Power consumption trace of a single AES encryption from the first test dataset. Note that there are several peaks vaguely observable, but their number does not relate to the number of rounds in the AES algorithm.

As a result, a frequency analysis was performed to determine the dominant frequencies in the power trace. Figure 3.9 shows the distribution of frequencies within the trace. Note that the frequency range in the distribution goes from 0 Hz to 156 MHz. This is effectively half the sample rate at which the signal was recorded. This is due to the Shannon-Nyquist theorem which was explained in section 3.8. The distribution shows that the first dominant frequency lies around 8 MHz, which is near the advertised clock speed of the microcontroller (7.37 MHz). The other dominant frequencies are multiples of the first dominant frequency and can therefore be described as harmonic frequencies of the latter. Apart from this fact, the frequency analysis also shows high amplitudes for the frequencies before the first dominant frequency. Therefore a low-pass filter was applied to the power trace to filter unwanted frequencies. The result of this low-pass filter is provided by figure 3.10. While the signal drops are more clearly after applying the filter, the distinct pattern of the AES encryption is still not visible.

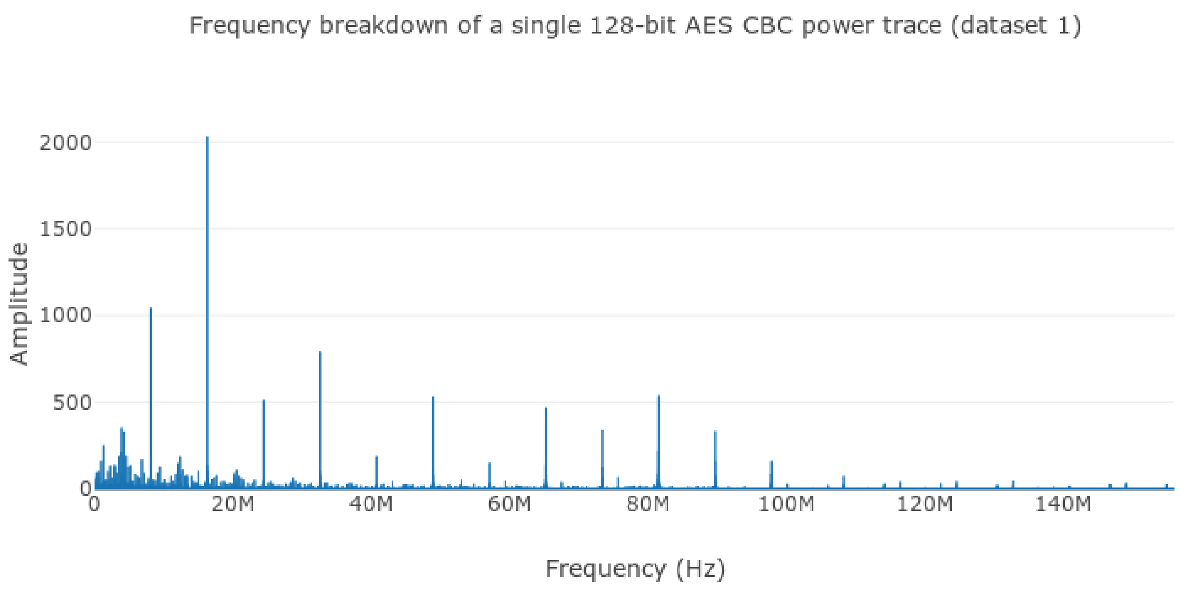


Figure 3.9: Frequency analysis for the power trace of a single AES encryption from dataset 1.

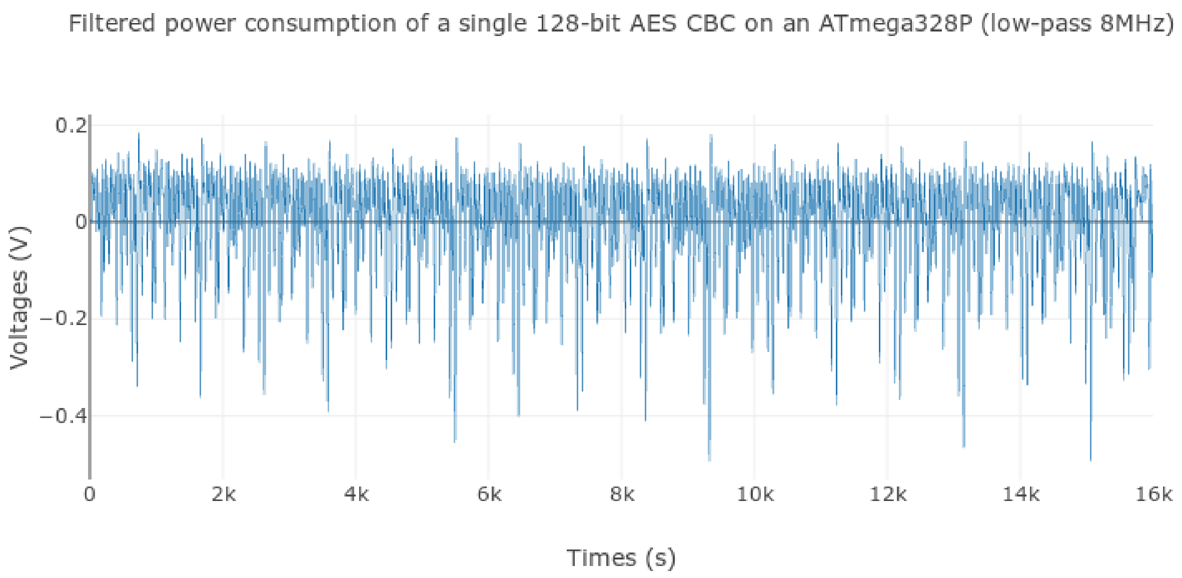


Figure 3.10: Filtered power consumption trace of a single AES encryption from dataset 1 using a 8 MHz low-pass filter

A first-order CPA attack was then performed on both the unfiltered and filtered power trace. The description of the dataset noted that the first round was best targeted for a power analysis attack. The attack was thus based on the Hamming Distance model with the Hamming distances being the number of bits changing during the `AddRoundKey` and `SubBytes` procedure of the first AES round (i.e. $SubBytes(P_0 \bigoplus K_0)$). The calculated intermediate value was set to $SBox[p[i] \bigoplus k_h[i]]$ with $p[i]$ being the plaintext byte at position i and $k_h[i]$ being the guess for a key byte at position i in the key. The attack was first performed on the complete dataset (i.e. using all 1000 traces). This resulted in the correct guess for the secret key as presented by listing 3.5. The PGE values for all guesses are equal to zero, meaning that the attack will always be successful using 1000 traces from the same set-up.

```
Known key:
2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

processing.analysis.functions [DEBUG]: Subkey  0, hyp = 00: 0.14590
processing.analysis.functions [DEBUG]: Subkey  0, hyp = 01: 0.13911
processing.analysis.functions [DEBUG]: Subkey  0, hyp = 02: 0.15942
...
processing.analysis.functions [DEBUG]: Subkey  0, hyp = 2b: 0.93816
...
Found a new byte guess (hex: 2b, ascii: +)
...
Found a new byte guess (hex: 3c, ascii: <)

Full key guess: 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
PGE: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Listing 3.5: Result from executing the CPA attack on the first dataset.

The attack can further be evaluated using less traces for the CPA attack. Figure 3.11 shows an overview of the changes in PGE values per key byte guess when incrementing the number of traces by 1. The results show that the minimum required number of traces is equal to 22. This gives an indication that the device is very vulnerable for power analysis attacks; the number is very low compared to the hundreds or thousands of traces that are usually required.

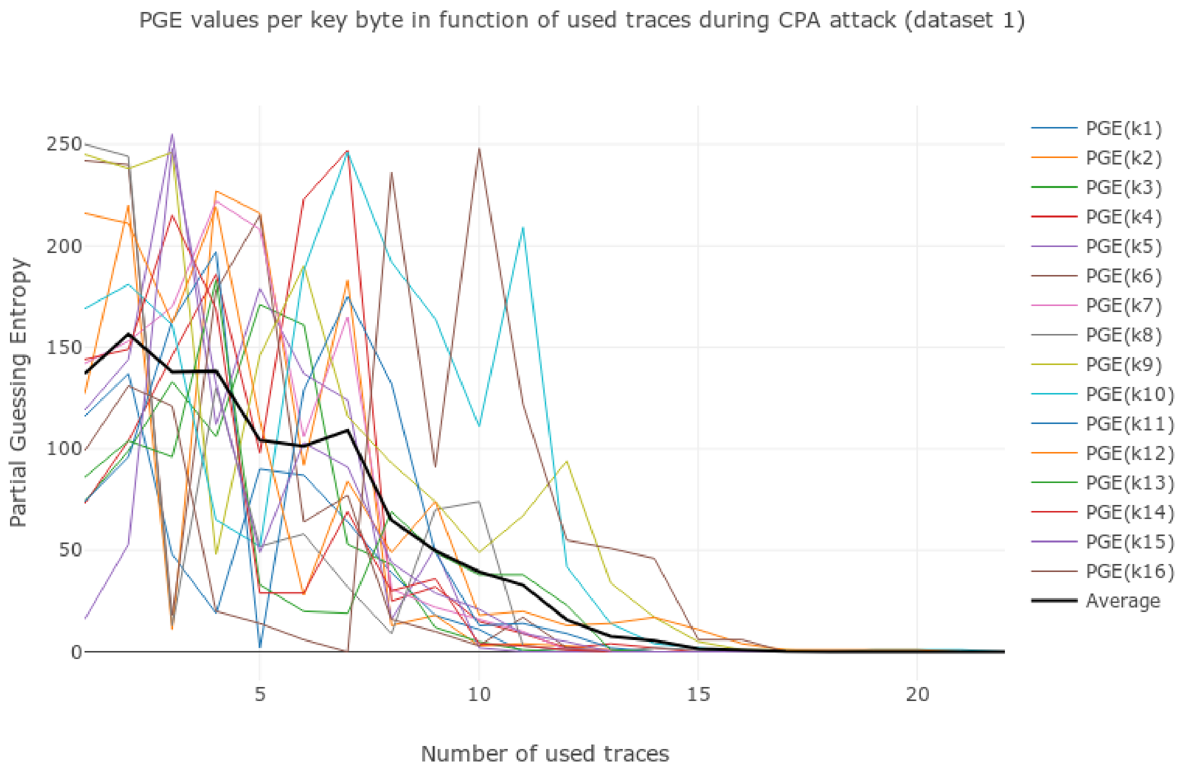


Figure 3.11: The changes in PGE values for each key byte in function of the used number of traces in the CPA attack. The PGE values decline as more traces are used during the CPA attack. All PGE values have reached 0 when the attack is performed with 22 traces.

Dataset 2: `sakurag-picoscope6403d-4000.zip`

The second dataset is a set of power traces from an ATmega328P using a hardware implementation to perform AES CBC encryptions. The microcontroller is clocked at 1.5 MHz. The power trace was recorded using a PicoScope 6403D digital oscilloscope. The construction for this measurement is unknown. The description of the dataset suggests an attack on the last round of the AES algorithm.

The visual inspection of the power traces show a much clearer pattern of the AES encryption than the first dataset. Figure 3.12 shows a trace from a single AES encryption. The eleven drops in the trace indicate the eleven rounds of AES that are being executed. Since the pattern is clearly recognizable, it was unnecessary to perform a frequency analysis and filter the unwanted frequencies from the trace.

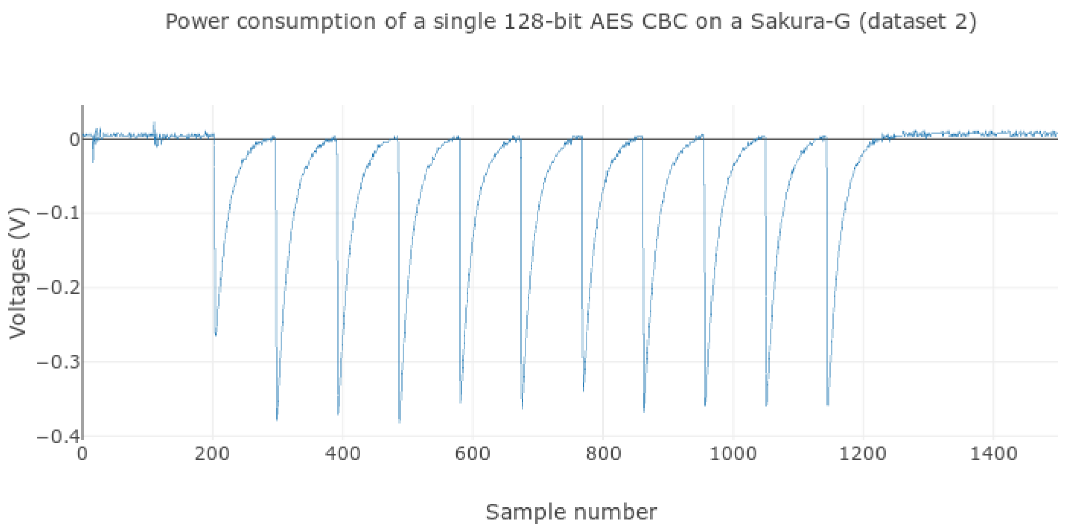
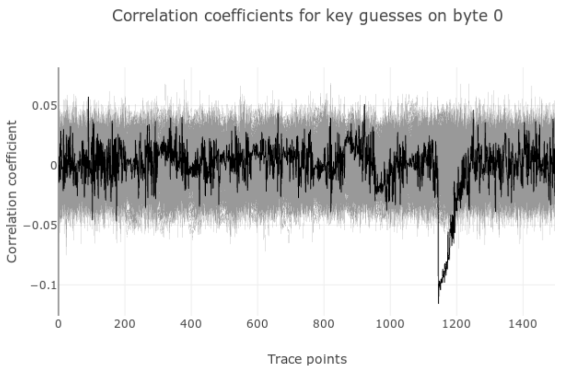


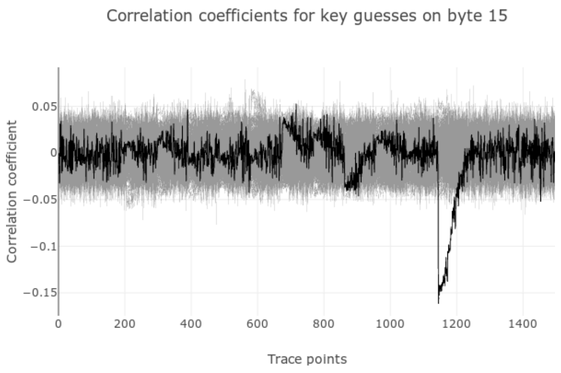
Figure 3.12: Power consumption trace of a single AES encryption from the second dataset. The voltage drops in the trace correspond to the number of rounds in the AES algorithm.

Thus, a CPA attack was performed on the dataset. While the description of the dataset suggests an attack on the last round of AES, an attack on the first round was also performed to compare the effectiveness in this case. Listing 3.6 shows the results of both attacks. The CPA attack on the first round of AES yielded negative results, showing an average PGE of 167.6875 across all guesses. The possibility of a non-zero IV was considered, but performing a manual AES encryption on the plaintext using the provided secret key and an all-zero IV still resulted in the provided ciphertext. The second attack however was able to guess the correct secret key from the first try. During the second attack, the correlation coefficients for each key byte were plotted. All high correlation coefficients were situated at the end of the trace, indicating that the trace was indeed a complete AES encryption with 10 consecutive rounds (also see figure 3.13. Therefore it is believed that the first round simply did not contain any leakage in the power consumption trace. The reason behind this is to be found in the way that the input and output values of the `SubBytes` are stored in the registers of the device; the values are not stored in the same registers [Car15]. This means that there is no Hamming distance to be measure between the input and output values, rendering the intermediate value calculation for the first round invalid.

Afterwards the PGE values were also calculated in function of the number of traces. The results of this calculation show that the CPA attack is successful when more than 2700 traces are used. Thus, the device is more resistant against the CPA attack than the previously targeted device or the measurements were less precise than the previous ones.



(a) first key byte



(b) final key byte

Figure 3.13: The correlation coefficients plotted in function of time for the first and last key guess while using a CPA attack targeted at the first round of AES. The darker line indicates the correlation coefficients for the correct key guess. It should be noted that both lines peak at the end of the trace.

```
# Targeting the first round of AES
Known key:
00 01 02 03 04 05 06 07 08 f7 15 88 09 cf 4f 3c

Found a new byte guess (hex: 85, ascii:  )
...
Found a new byte guess (hex: 87, ascii:  )

Full key guess: 85 98 36 e2 72 0d 50 16 0c dc d4 7d 36 bf 42 87
PGE: [202, 160, 231, 216, 98, 212, 198, 209, 47, 49, 105, 99, 194, 179,
    252, 232]
```

```
# Targeting the last round of AES
Known key: 00 01 02 03 04 05 06 07 08 f7 15 88 09 cf 4f 3c
Last round key: e6 19 d4 c4 7d ad 82 9d c0 a6 13 ed 36 6d 3a 3b

Found a new byte guess (hex: e6, ascii: æ)
...
Found a new byte guess (hex: 3b, ascii: ;)

Round key guess: e6 19 d4 c4 7d ad 82 9d c0 a6 13 ed 36 6d 3a 3b
First round key guess: 00 01 02 03 04 05 06 07 08 f7 15 88 09 cf 4f 3c
PGE: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Listing 3.6: Results of two CPA attacks on the second dataset. The first attack, which is focused on the first round of AES, yields a wrong guess for the secret key, while the second attack on the final round yields the correct key guess.

Test dataset 3: `ASCAD.h5`

The third dataset contains two sets of power traces of an ATmega8515 using a secured software implementation to perform AES CBC encryptions. The microcontroller is clocked at 4 MHz. The construction for this measurement is unknown. The description of the dataset suggests an attack on the first round of the AES algorithm. According to the associated paper, only the third key byte can be revealed using a first-order power analysis attack due to the masked implementation.

A visual inspection of a single power trace (figure 3.14) shows no particular characteristics of the AES encryption. This is because the dataset only contains trimmed traces which are focused on the third key byte of the AES encryption. The trace in figure 3.14 thus represents only the third round of the encryption. From the visual inspection it can also be determined that there is little to no interference from other signals. Therefore, a frequency analysis and filtering are not necessary. Next, the same CPA attack from the first dataset was performed on the dataset. However, this time the attack was unsuccessful (also see listing 3.7). Due to the masked implementation of the AES algorithm, it was necessary to add the mask byte to the intermediate value, thus resulting in an intermediate value $SBox(p[i] \bigoplus k[i]) \bigoplus r_{out}$. Using this calculation as the intermediate value, a standard CPA attack could again be performed exclusively on the third key byte. Using the modified intermediate value, the CPA attack was successful. The results of the attack are provided in 3.7. The PGE results for the third key byte are provided in figure 3.15 and show that the attack is successful after only 40 traces.

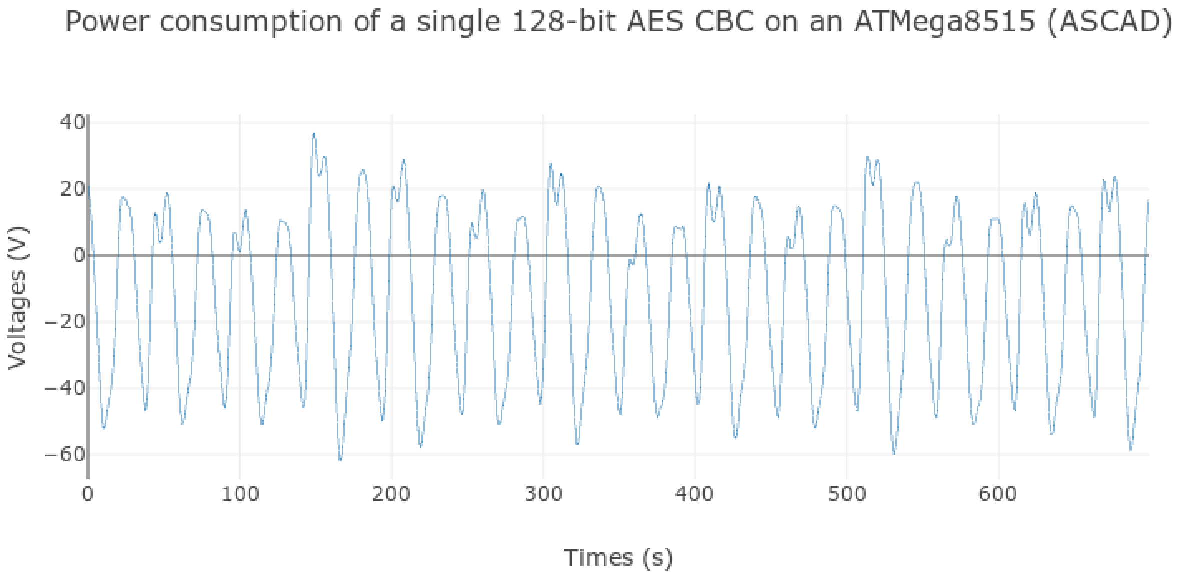


Figure 3.14: Power consumption trace of a single AES encryption from the ASCAD dataset.

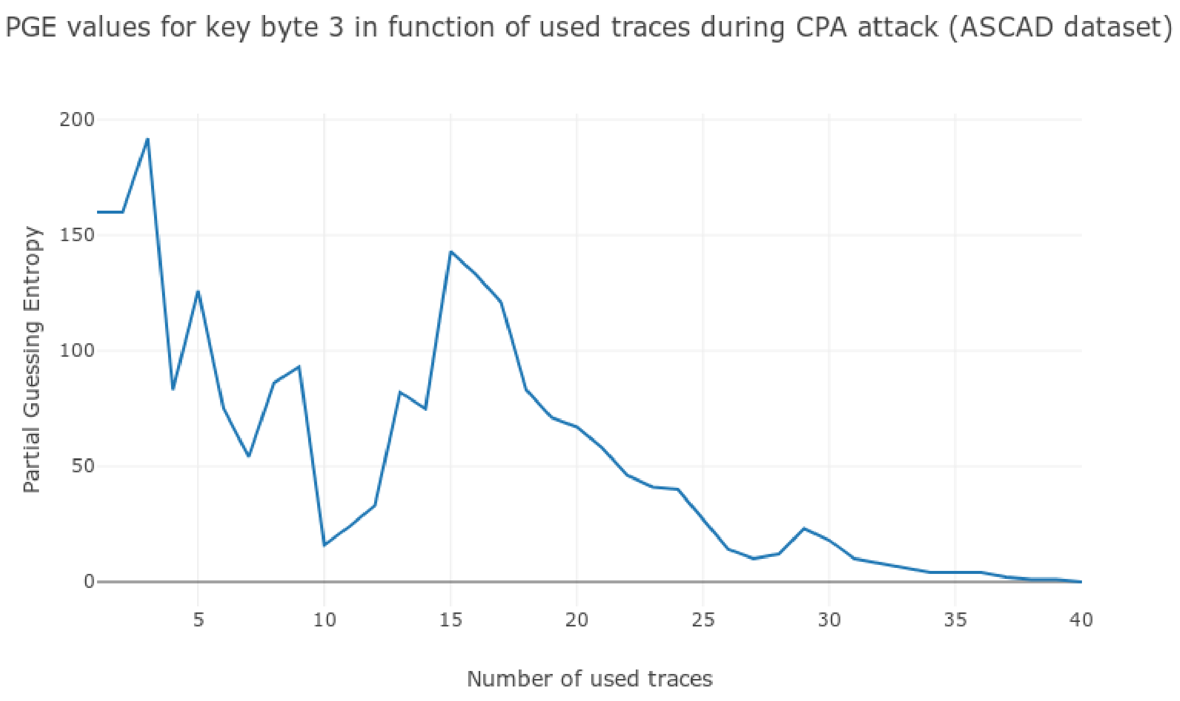


Figure 3.15: The changes in PGE values for the third key byte in function of the used number of traces in the CPA attack. The PGE value has reached 0 when the attack is performed with 40 traces.

```
# Attack strategy: First Round Plaintext Attack
processing.analysis.functions [INFO]: Processing 10000 trace(s)...
Known key:
4d fb e0 f2 72 21 fe 10 a7 8d 4a dc 8e 49 04 69

Found a new byte guess (hex: 83, ascii:  )
Full key guess: 0f 0f 83 0f 0f 0f 0f 0f 0f 0f 0f 0f 0f 0f 0f 0f
PGE: [0, 0, 67, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

# Attack strategy: Unmasked First Round Plaintext Attack
processing.analysis.functions [INFO]: Processing 10000 trace(s)...
Known key:
4d fb e0 f2 72 21 fe 10 a7 8d 4a dc 8e 49 04 69

Found a new byte guess (hex: e0, ascii:  à)
Full key guess: 0f 0f e0 0f 0f 0f 0f 0f 0f 0f 0f 0f 0f 0f 0f 0f
PGE: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Listing 3.7: Results of performing two CPA attacks on the ASCAD dataset. The first implementation did not make use of the mask byte and was unsuccessful in guessing the correct key byte. The second attempt did include the mask byte value and was successful.

3.9.4 Case study: testing the implementation using self-measured data

The next case study involves self-measuring the power traces of the AES algorithm on a microcontroller. The target microcontroller in this case study is the Atmel ATmega328P, which is clocked at 16 MHz using an external oscillator. A minimal breadboard set-up was chosen to prevent interference from non-essential components. The ATmega328P is powered using the 5 V GPIO pin of a RaspBerry Pi 3B version 1.2. The Raspberry Pi itself is used to retrieve the traces from the oscilloscope via LXI and to send the random plaintexts to the microcontroller during the second measurement option.

In the next subsection the various steps that were taken to measure the power consumption will be described in more detail. First the basic measurement set-up will be discussed. Then a measuring attempt using the RS-232 signal as trigger is described and it is explained why this attempt failed. At last, a second solution is presented which is able to capture the signals correctly.

Basic measurement set-up

To measure the power consumption of the microcontroller, one must insert a shunt resistor in the electrical circuit which will power the microcontroller. The ATmega328P features two V_{cc} and GND connectors. However, pins 20 (AV_{cc}) and 22 (GND) are only used for the integrated ADC in the ATmega328P; the processor of the ATmega328P only uses pins 7 (V_{cc}) and 8 (GND).

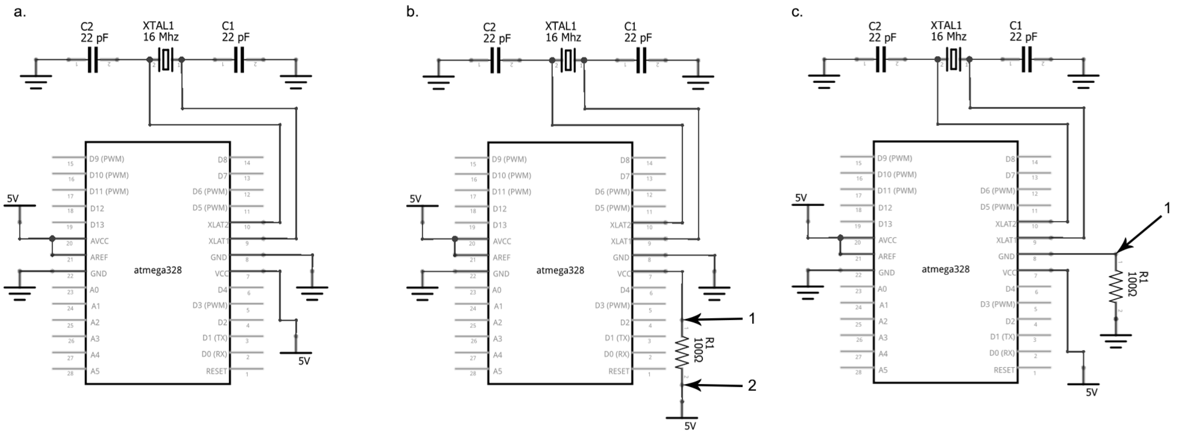


Figure 3.16: Overview of breadboard set-ups for the ATmega328P. Subfigure (a) shows the minimal breadboard set-up without measurement additions. Subfigure (b) shows the measurement set-up for V_{cc} probing and subfigure (c) shows the measurement set-up for GND probing.

Thus, there are two options that can be chosen. The first option is to insert a shunt resistor in series with the V_{cc} input and to connect the probe between the resistor and the V_{cc} pin. Figure 3.16b shows this option applied to the ATmega328P. The main advantage of this measurement method is that it will be less influenced by the internal capacitors in the microprocessor. However, the main disadvantage is that a voltage almost equal to the required input voltage of the microprocessor is measured if only one probe is used. This could present a problem since oscilloscopes have a limited vertical offset which can be applied to the measurements. This disadvantage can however be countered by using a second probe which is inserted behind the resistor, or a differential probe. The two voltage readings are then subtracted from one another. The result is the voltage used only by the resistor, which is still proportional to the power consumption of the microcontroller.

The second option is to insert the resistor in series with the GND pin and connect the probe between the GND pin and the resistor. This set-up is preferred when only one probe is available since the voltage measurements behind the GND will be closer to 0 V, effectively countering the problem that is present in the first option. Figure 3.16c shows this option applied to the ATmega328P set-up.

For this case study, the second option was chosen. To make sure that this set-up worked, a simple Arduino programme (see appendix D) was created which performed an AES CBC encryption with a fixed 128-bit secret key and plaintext. The output voltage of GPIO pin `PB0` (pin 8 in the Arduino IDE) was set to high during the encryption. This way the oscilloscope could be set-up with a simple rising edge trigger on the GPIO pin to capture the encryption process. Figure 3.17 shows the result of using the single trigger mode on the oscilloscope.

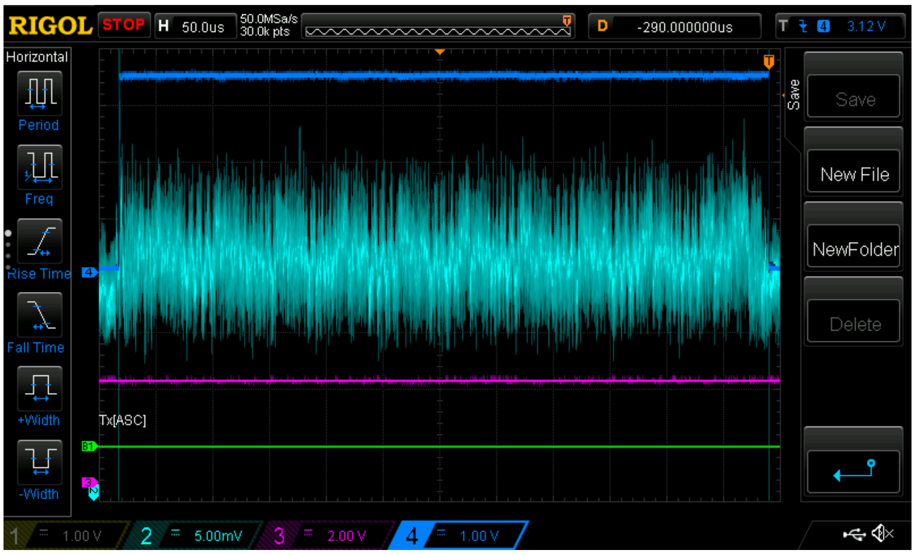


Figure 3.17: Power trace of a single AES CBC encryption on an ATmega328P measured by a Rigol DS1104Z oscilloscope. The dark blue line indicates the GPIO signal that was used as the trigger signal.

First attempt: RS-232 as a trigger

In a first attempt, the ATmega328P was programmed to perform the following instructions in a loop:

- Generate random plaintext data of 16 bytes long on the ATmega328P;
- Send a newline character (`\n`) over serial communication which serves as the operation starting the trigger pulse;
- Perform the AES CBC encryption on the plaintext data with a fixed zero-only IV;
- Send the plaintext data over serial communication to the PC.

The illogical order of sending the newline character and the rest of the plaintext is due to the way the trigger for this measurement set-up works. The RS-232 trigger of the oscilloscope was unable to fire a trigger based on the data in the signal. Therefore a pulse trigger was used instead. The trigger position of a pulse trigger starts at the end of the samples for which the condition was valid. In other words, when a pulse trigger is used for which the condition states that the pulse must be at least x microseconds above a certain voltage level, then the trigger position will be placed at the end of the pulse instead of the beginning. In this case study, the pulse trigger triggers when the pulse stays at a voltage level of 3 V or higher for longer than 100 µs. During the visual inspection of the traces (figure 3.18), it can be seen that there is a moment between sending the newline character and the rest of plaintext where the RS-232 signal outputs a logic value 1 (i.e., an idle signal). This signal will represent the pulse for the pulse trigger.

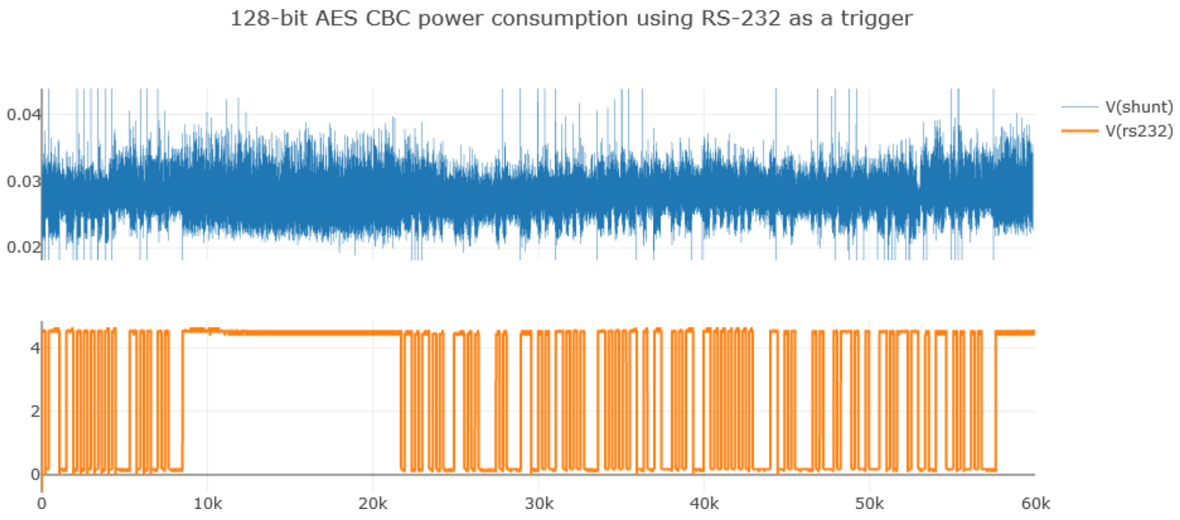


Figure 3.18: Power trace of a single AES CBC encryption on an ATmega328P using the RS-232 trigger.

The oscilloscope parameters and the Arduino programme used in this set-up are described in appendix G and E. After the traces have been captured, the relevant information needed to be extracted from the traces. Since the ATmega328P was responsible for the generation of the plaintexts, the plaintexts were sent over RS-232 and captured by the oscilloscope during the measurements. Therefore, both the RS-232 signal and the actual power consumption values of the AES encryption needed to be separated and analysed individually. Similar to the way the oscilloscopes trigger worked, the edges of the RS-232 signal were detected in Python and used to extract the power consumption during the AES encryption. The RS-232 signal itself was further processed using the python package `ripyl` to extract the plaintext. The plaintext was then saved along the power consumption trace of the encryption.

Since the pulse trigger is not an accurate trigger to use, the power consumption trace of the encryption needed to be aligned after extraction. An alignment method based on the correlation coefficient was used to achieve this. The results of this alignment were suboptimal: from the thousand traces that were recorded only 301 traces were shifted within a 5 % range relative to the reference trace. With the remaining 301 traces a CPA attack was attempted, but resulted in an incorrect guess of the secret key. Figure 3.19 shows the PGE values obtained from the CPA attacks on the power traces. The average PGE value does not seem to drop during both attacks. This is an indication that the CPA attack is ineffective: during a successful attack the average PGE value should drop as seen in the case studies using the test datasets.

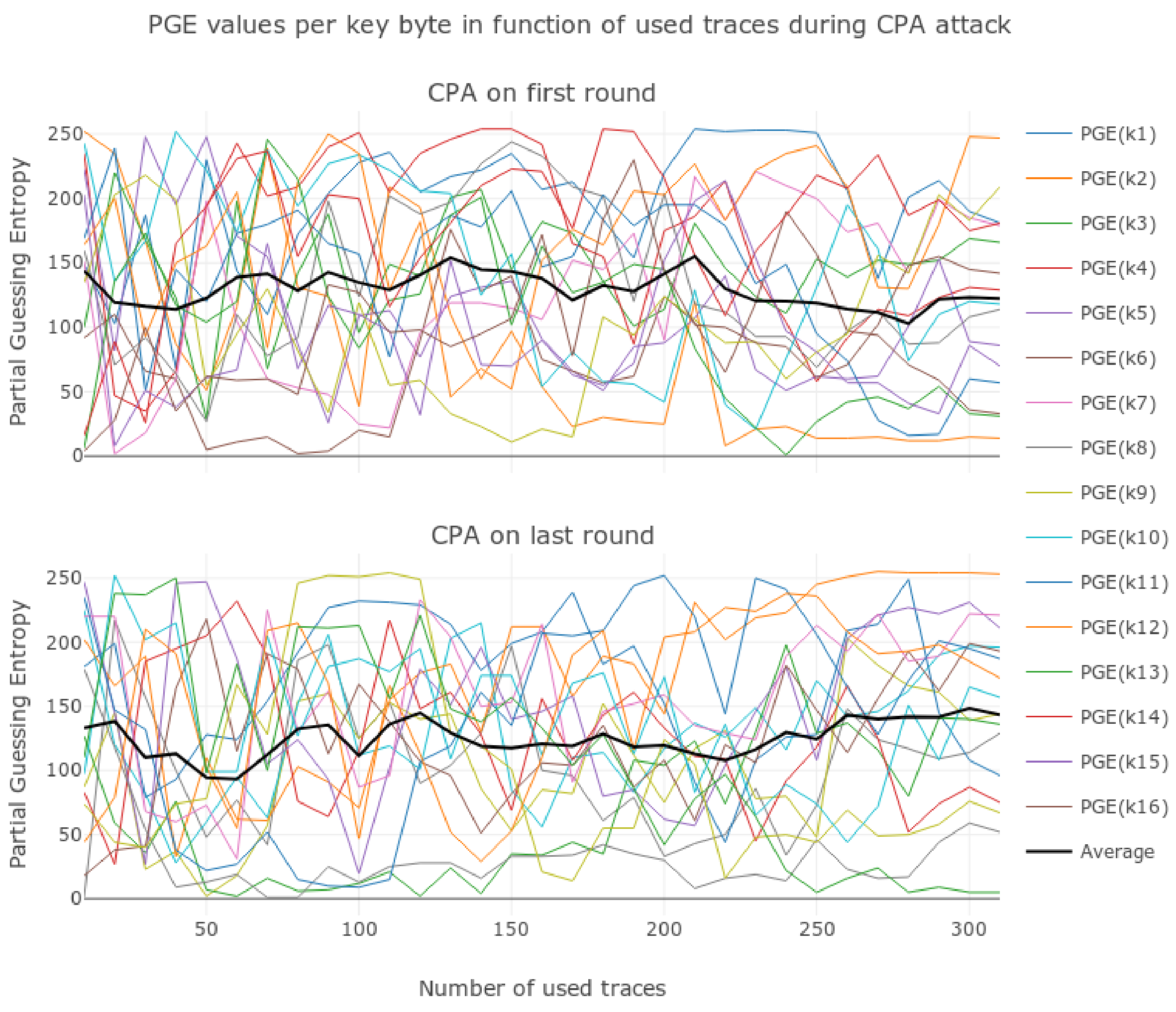


Figure 3.19: PGE values for both the CPA attacks on the first and last round of the AES algorithm. Both averaged PGE values do not appear to change dramatically, indicating an unsuccessful attack.

Further research showed that the RS-232 signal of the ATmega328P could not be used as a reliable trigger for the oscilloscope. The code in the Arduino sketch and the processing of the RS-232 communication are executed in parallel by the ATmega328P. This is visible in figure 3.20. Therefore, the second measurement set-up was chosen instead.

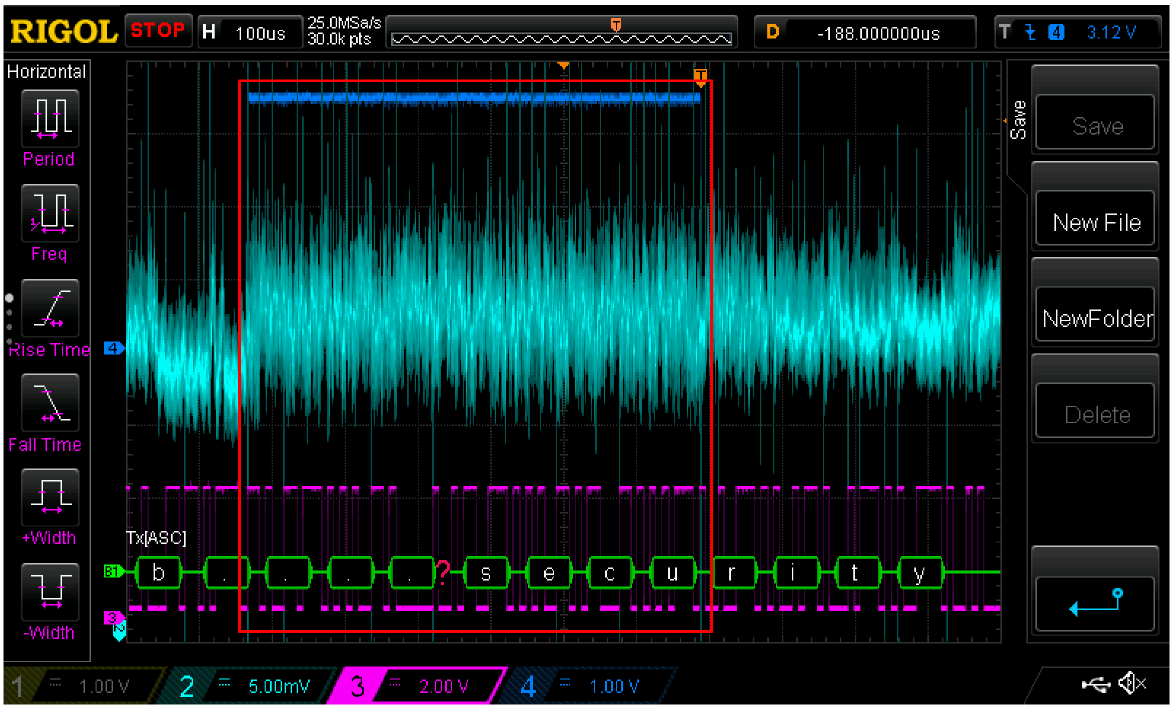


Figure 3.20: Asynchronous RS-232 communication captured during the AES encryption process. The dark blue line indicates the GPIO signal that was used as the trigger signal. Note that the RS-232 signal runs in parallel with the AES encryption, which will influence the measurements.

Second attempt: GPIO as a trigger

Due to the inaccuracy of the trigger in the previous set-up, a trigger based upon the GPIO output by one of the GPIO pins on the ATmega328P used instead. The GPIO pin on the ATmega328P would be set to high during the encryption of the plaintext. To mitigate the intrusion of RS-232 communication during the measurements, the ATmega328P was programmed to only accept new plaintexts via the RS-232 communication. Once the plaintexts were received, the ATmega328P would perform continuous encryptions using the same secret key and the provided plaintext. The complete Arduino programme and the oscilloscope parameters used for this set-up can be found in appendices F and H. Using this set-up, the only power consumption change could then be addressed to the encryption process and the change in GPIO output.

Initial visual inspection of the power traces shows little to no change in regard to the previous set-up. Figure 3.21 shows the power trace of a single encryption captured using this measurement set-up. Ten peaks can be observed in the trace. By using the averaging function on the oscilloscope, it can be observed that a) the traces remain the same when the same key and plaintext are used for the encryption process, and b) that the average trace changes when the plaintext changes. These two observations lead to the conclusion that the trigger works as intended and allows the oscilloscope to measure the encryption process.

Using the original non-averaged traces, a frequency analysis was performed to identify unwanted frequencies. The analysis showed the first dominant frequency to be in range of 16 MHz, which is equal to the clock speed of the microcontroller. A low-pass filter with a 18 MHz cut-off was then applied to the power consumption trace. This resulted in a negligible difference between the original and the filtered trace. Since the frequency analysis also showed dense peak around 8 MHz. A band-pass filter was applied to the original data using a range between 7.8 MHz and 8.2 MHz. Figure 3.22 shows the result of this filter. In this filtered trace the ten peaks observed during the visual inspection are now more clearly visible, but still cannot be associated with the ten rounds of AES.

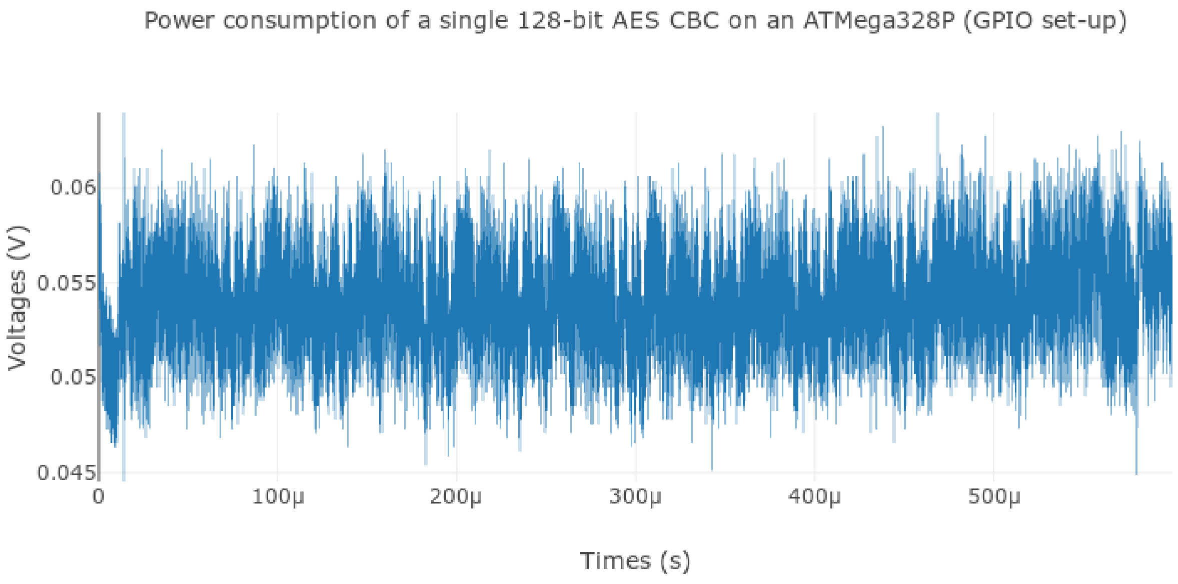


Figure 3.21: Power consumption trace of a single 128-bit AES CBC encryption using the GPIO trigger

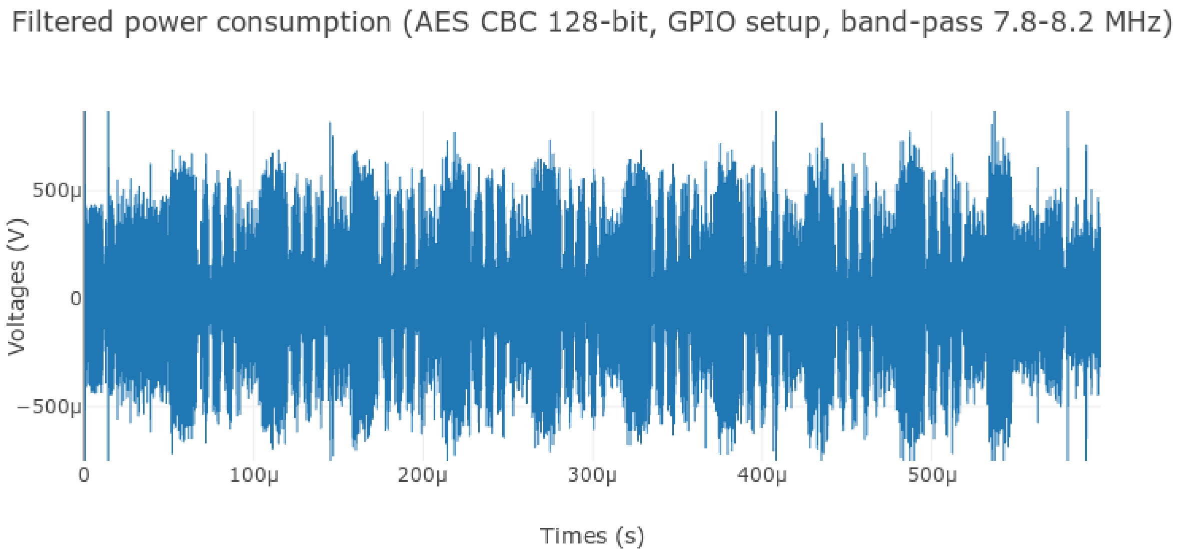


Figure 3.22: Filtered power consumption trace of a single 128-bit AES CBC encryption using the GPIO trigger. A band-pass filter with a range of 7.8 MHz and 8.2 MHz was used to filter the original trace.

Thus an attempt to attack the power traces was made. Since both the plaintext and ciphertext are known, both the attack on the first round and on the final round can be performed. However, both attacks were unable to find the secret key. Figure 3.23 shows the PGE values during both attacks on the traces. Like with the previous measurement set-up, the PGE values do not seem to fall. Therefore it is believed that both attacks will be ineffective in finding the secret key even with more traces. An explanation why the attacks are ineffective remains unknown. A possible cause can be electrical noise from other devices, since the power grid on which the measurements were performed is used by other devices. A faulty laptop supply or tube lights could also cause interference that is difficult to detect.

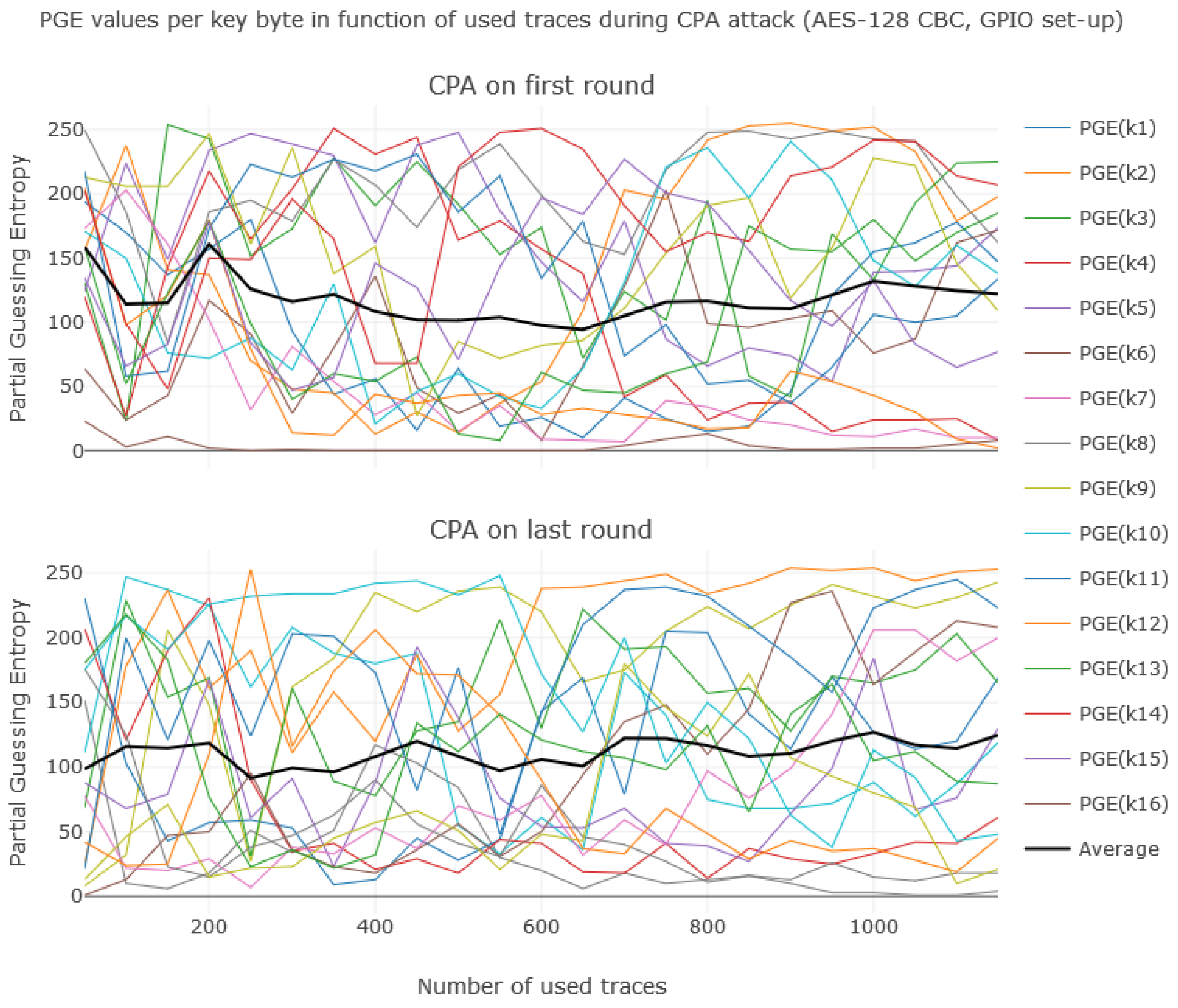


Figure 3.23: PGE values for both the CPA attacks on the first and last round of the AES algorithm. Both averaged PGE values do not appear to change dramatically, indicating an unsuccessful attack.

3.9.5 Case study: attack on AES CCM

The goal of this case study was to modify the original CPA attack algorithm to be capable of attacking an AES CCM implementation on an ATmega328P. However, due to the issues experienced in the previous case studies, this case study could not be performed.

3.9.6 Case study: attack on a Microchip RN2483 LoRa transmitter

For this case study, a Microchip RN2483 LoRaWAN transceiver was targeted. The RN2483 is a fully-certified 433/868 MHz module based on wireless LoRa technology[Mic17]. The chip features a complete LoraWAN class A stack, which means it is able to perform the encryption process on-chip. The concrete internal clock speed of the chip is not known. Therefore it is assumed that the internal clock speed of the chip is lower than 50 MHz, since it requires limited processing power to perform the encryption and LoRaWAN processing. The device also features an UART interface which can be used as a serial communications port. The chip itself is soldered on a RN2483 breakout board by Azzy'S Electronics to simplify connecting the RN2483 to a USB serial cable and a 3.3 V power supply.

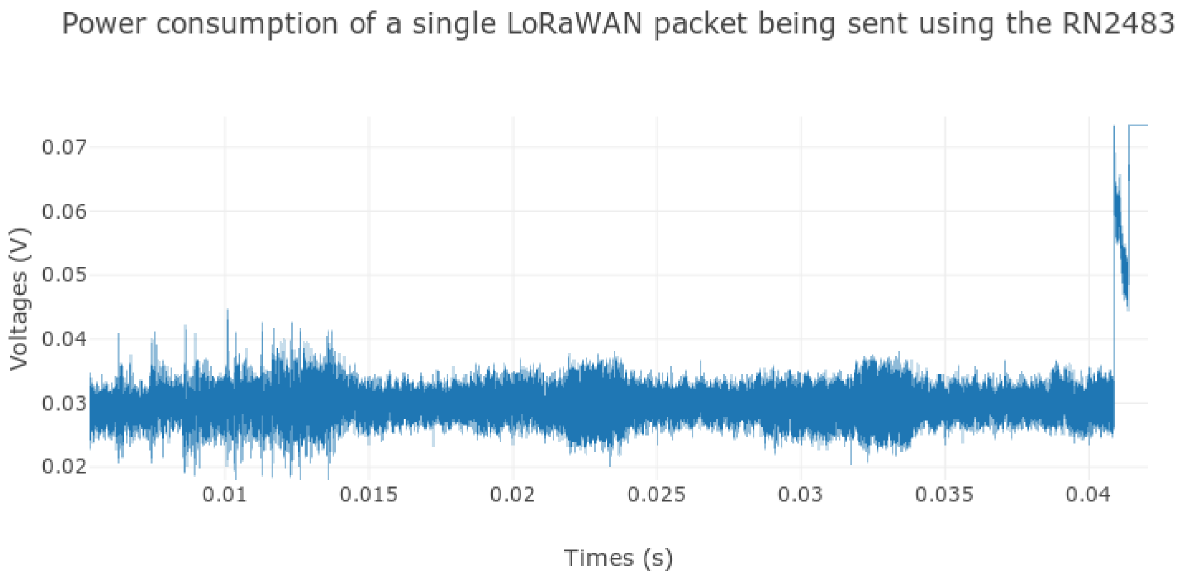


Figure 3.24: Power consumption of the pre-processing done by the Microchip RN2483 before a LoRa packet is sent.

Due to the problems experienced with the previous case studies, a detailed analysis could not be performed on the RN2483 transceiver. In one of the preliminary tests, a trace of the full preprocessing done by the RN2483 transceiver was captured using a pulse trigger set on the `RX` pin of the transceiver. Figure 3.24 shows the result of this measurement. The trace shows that the process starts with a series of heavy calculations. Then a repeating pattern with a period of less calculations followed by a period of a high number of calculations can be observed from the trace. One could try to associate these two patterns with the encryption of the payload and the generation of the MIC, but this cannot be validated at the moment; further measurements and analysis of the power consumption are needed to form further conclusions.

Chapter 4

Conclusion

Throughout this thesis, an introduction to power analysis attacks and in general side-channel attacks has been given. During the research surrounding the subjects, it became clear that even cryptographically strong algorithms can be prone to implementation related vulnerabilities. The power analysis attacks discussed in this thesis are the essential techniques from a vast collection of power analysis attacks; every year new forms and techniques are introduced which improve the current techniques.

That said, it became clear throughout the case studies that performing power analysis attacks requires a decent knowledge of cryptography, electrics and electronics, signal processing and statistics. The measuring equipment required to perform this type of side channel attack is also expensive and difficult to operate in non-ideal environments. It is therefore doubtful that power analysis attacks while be used by individuals and non-state actors in practice.

To the question "Are power analysis attacks usable in practice?", a clear verdict cannot be given at the moment of writing. Using various test datasets it was shown that, if implemented and measured correctly, power analysis attacks can be effective in finding the cryptographic key of an AES enabled device. The power analysis attacks performed on the test datasets were able to find the complete secret key using less than 10000 power consumption traces. With the first dataset, it was even possible to derive the secret key from 22 power traces. However, it should be noted that these results were only obtained using DPA attacks. One of the strong points of DPA attacks is the smaller number of power traces that is required to perform a successful attack. Other forms of power analysis attacks like the template attacks require an impractical amount of information from the target device, which makes the practicality of the method sometimes doubtful. However, in case of DPA attacks, the effectiveness is strongly influenced by the quality of the power consumption traces. Incorrect measurement set-ups, misaligned traces or electrical noise can have a direct impact on the outcome of the DPA attacks. These issues arose during the case study using self-measured traces. It should be noted that these measurements were performed in a lab-environment: measurements in this setting are more optimal than in-field settings.

4.1 Future work

Since several problems occurred during the case studies, practical examples could not be performed with success. The problems during the case studies were closely related to the measurements using the oscilloscope and the lack of precisely documented measurement set-ups used in other papers. Future work may thus include studying other measurement set-ups and other techniques related to signal processing. More documentation on the basics of power analysis attacks is also required, as the author experienced that practical examples were scarce or often incomplete.

Furthermore, the initial plan of attacking the AES CCM algorithm and the LoRaWAN protocol could not be performed due to the previously mentioned issues. Provided that the measurement issues can be solved, an attack on the AES CCM algorithm would be a first step to undertake. In [ROSW16], an attack on AES CCM was shown to be successful, thus the first steps into attacking the LoRaWAN would be to successfully reproduce the attack performed in [ROSW16]. However, reproducing this attack will not guarantee a successful attack on the LoRaWAN protocol; the implementation of the AES CCM algorithm is performed differently from the ZigBee standard [ROSW16][LoR15] and power consumption could possibly be influenced by other components.

Finally, it was noted that the power analysis attacks used in this thesis were limited to the essential techniques. Further research into other power analysis techniques and the closely related EM attacks could be researched and considered in the future as an alternative to the CPA attacks used in this thesis.

Appendix A

AES S-Box

		last 4 input bits															
first 4 input bits		**00**	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**0a**	**0b**	**0c**	**0d**	**0e**	**0f**
	00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Appendix B

Proof about the effect of resistor to voltage of series connected device

This proof shows the reasoning behind the principle that the voltage through a shunt resistor will proportionally increase according to the current changes in a series connected device. Throughout the proof, the construction in figure B.1 will be used as a reference. This construction includes a single power supply V_{cc}, a device d, and a shunt resistor s.

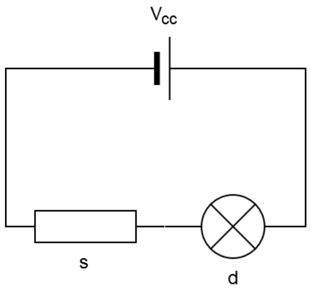


Figure B.1: Single power supply electrical circuit with a device d and a shunt resistor s.

According to Kirchhoff's Voltage Law (KVL), the directed sum of electrical potential difference (i.e., voltages) in a closed circuit is equal to 0:

$$\sum_{k=1}^{n} U_k = 0 \tag{B.1}$$

Applied to the construction given by B.1, this equals to the following sum:

$$\sum_{k=1}^{n} U_k = 0 \Rightarrow U_{cc} + U_d + U_s = 0 \tag{B.2}$$

As mentioned before, the voltage between two points in an electrical circuit is the difference in electrical potential between two points in the circuit. The electrical potential is defined to be the amount of work that is needed to move a positive charge from a given point to a reference point (often the ground potential). From a physical point of view, electrons flow from the negative side of a power supply to the positive side, carrying a charge along the way. When a low amount of work is needed to move the electrons across a path, the difference in electrical potential between the points will be minimal. However, a path that requires more work to move the electron (i.e., a resistor), will induce a high potential difference between a given distance.

According to this knowledge, a component with a high resistance will cause a voltage drop in the circuit. This also means that these resisting components can be seen as negative voltages, changing the previous sum of voltages in the circuit to the following equation:

$$|U_{cc}| - |U_d| - |U_s| = 0 \Leftrightarrow |U_{cc}| = |U_d| + |U_s| \tag{B.3}$$

In 3.2 it was mentioned that when logic values in CMOS cells transit from 0 to 1 and vice versa, a short circuit is created for a short amount of time, effectively decreasing the overall resistance (R_d) and the potential difference (U_d) of device while increasing the current flow (I_d) through the device. However, Kirchhoff's Voltage Law still applies to the circuit. Since U_d is decreased, the shunt resistor will need to "fill the gap" by increasing its potential difference U_s. This fact is also represented in Ohm's law ($R = U/I$) since the shunt resistor specifies a fixed amount of resistance and an increase in current requires a proportionate increase in voltage to keep the resistance equal.

Thus, this proves that a current increase caused by the device will also lead to a voltage increase over the shunt resistor, thus proving that the voltage over the shunt resistor is proportionate to the current flowing through device d.

Appendix C

Finding points of interest using sum of distances

In order to find interesting points for a template, the attacker must first aggregate traces belonging to the same operation and then search for points that differ the most across all different operations. This process can be done in three steps. The first step is aggregating the traces which perform the same operation. A commonly used aggregation method is to for a trace that represents the mean of the same-operation traces. This can be performed as follows. Let $T_{o,i,k}$ be the kth sample of the ith power trace for an operation o, $M_{o,k}$ be the kth sample of the mean trace M of an operation o and N_o be the number of traces captured for an operation o, then M can be defined as:

$$M_{o,k} = \frac{1}{N_o} \sum_{i=1}^{N_o} T_{o,i,k} \tag{C.1}$$

The second step is to search for points that differ the most amongst the different operations. Once the mean traces for all operations have been calculated, the mean traces are analysed for points in which they differ most amongst each other. A simple but effective metric is the sum of difference. Let $M_{o,k}$ be the kth sample of the mean vector M for an operation o, D_k the sum of difference for the kth sample, then the sum of differences can be expressed as follows:

$$D_k = \sum_{i}^{N_o} \sum_{j}^{N_o} |M_{i,k} - M_{j,k}| \tag{C.2}$$

Once D has been fully calculated, the sum of difference can be analysed for its maximum values. The highest values in D will correspond to points which differ greatly amongst the operation. However, picking only one point could prove to be insufficient: if the sum of difference reached its high value because of a great difference between a fraction of the mean traces, then only that fraction of operation would be identifiable amongst each other by that point. Thus, other operations might eventually be seen as the same operation. It is therefore advised to search for multiple points of interest to ensure the combination of points will lead to a better identification of individual operations.

Another point of attention is that the interesting points should not be picked near to each other, since chances are high that they will greatly differ for the same operations as other nearby points. It is therefore advices to set a threshold for the minimum distance that two points of interest must have in order to be considered unique.

Appendix D

AES test programme for ATmega328p

```
/* AES implementation is provided by Arduino-libs
 * Source: https://github.com/rweather/arduinolibs */
#include <Arduino.h>
#include <AES.h>
#include <Crypto.h>
#include <CBC.h>

uint8_t key[] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
uint8_t iv[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
uint8_t pt[] =
    {119,101,32,108,111,118,101,32,115,101,99,117,114,105,116,121};
int len = 16;
int trig_pin = 8;

void setup() {
  // put your setup code here, to run once:
  pinMode(trig_pin, OUTPUT);
}

void loop() {
  CBC<AES128> cbc;
  cbc.setKey(key, len);
  cbc.setIV(iv, len);
  uint8_t ct[len];

  while (true) {
    // Wait a small time between encryptions
    delayMicroseconds(10);
    digitalWrite(trig_pin, HIGH);
    cbc.encrypt(ct, pt, len);
    digitalWrite(trig_pin, LOW);
  }
}
```

Listing D.1: A test programme for Arduino-compatible microcontrollers which performs AES CBC encryptions with a fixed 128-bit secret key and plaintext

Appendix E

AES CBC-128 programme for ATmega328p using RS-232 as trigger signal

```
/* AES implementation is provided by Arduino-libs
 * Source: https://github.com/rweather/arduinolibs */
#include <Arduino.h>
#include <AES.h>
#include <Crypto.h>
#include <CBC.h>
void gen_random(char* s, const int len);

uint8_t key[] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
uint8_t iv[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
uint8_t pt[] =
    {119,101,32,108,111,118,101,32,115,101,99,117,114,105,116,121};
int len = 16;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial.setTimeout(10);
}

void gen_random(char *s, const int len) {
    static const char alphanum[] =
        "0123456789"
        "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        "abcdefghijklmnopqrstuvwxyz";

    for (int i = 0; i < len; ++i) {
        s[i] = alphanum[rand() % (sizeof(alphanum) - 1)];
    }

    s[len] = 0;
}

void loop() {
  // put your main code here, to run repeatedly:
  CBC<AES128> cbc;
  uint8_t ct[len];

  while (true) {
    // Wait a small time between encryptions
    delayMicroseconds(10);
    cbc.setKey(key, len);
    cbc.setIV(iv, len);
    gen_random(pt, len);
    Serial.println();
    cbc.encrypt(ct, pt, len);
    for (int i=0; i<len; i++)
      Serial.write(pt[i]);
  }
}
```

Listing E.1: The encryption programme for Arduino-compatible microcontrollers which performs AES CBC encryptions with a fixed 128-bit secret key and a random plaintext generated on the microcontroller. Serial communication is used to send the random plaintext to a PC and is also used as a trigger signal for the oscilloscope.

Appendix F

AES CBC-128 programme for ATmega328p using GPIO as trigger signal

```
#include <Arduino.h>
#include <AES.h>
#include <Crypto.h>
#include <CBC.h>

uint8_t key[] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
uint8_t iv[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
uint8_t pt[] =
    {119,101,32,108,111,118,101,32,115,101,99,117,114,105,116,121};
int len = 16;
int trig_pin = 8;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial.setTimeout(10);
  pinMode(trig_pin, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  CBC<AES128> cbc;
  cbc.setKey(key, len);
  cbc.setIV(iv, len);
  uint8_t ct[len];
  byte temp[len+1];

  while (true) {
    // See if new plaintext is submitted
    int readlen = Serial.readBytesUntil('\n', temp, len+1);
    cbc.setKey(key, len);
    cbc.setIV(iv, len); // Ensure that IV is always all-zeros
    if (readlen == len) {
      memcpy(pt, temp, len*sizeof(uint8_t));
//      // Used for verification of the input processing
//      for (int i=0; i<len; i++) {
//        Serial.write(pt[i]);
//      }
//      Serial.write("\n");
    }

    // Wait a small time between encryptions
    delayMicroseconds(10);
    digitalWrite(trig_pin, HIGH);
    cbc.encrypt(ct, pt, len);
    digitalWrite(trig_pin, LOW);
  }
}
```

Listing F.1: The encryption programme for Arduino-compatible microcontrollers which performs AES CBC encryptions with a fixed 128-bit secret key and plaintext provided by a PC over serial communication. GPIO pin 8 is used to output a trigger signal for the oscilloscope.

Appendix G

Rigol DS1104Z parameters - ATmega328p using RS-232 trigger

```
Model:DS1104Z
SN:DS1ZA155002892
Manufacturer:RIGOL TECHNOLOGIES
Board Ver:0.1.1
Firmware Ver:0.2.3.11
BOOT Ver:0.0.0.12
CPLD Ver:1.1
SoftWare Ver:00.04.04.SP1


    DSO Vertical System
CH1:Off
Scale:1.000000V
Position:0.000000V
Coupling:DC
Bandwidth Limit:OFF
Probe Ratio:10X
Unit:V

CH2:On
Scale:0.005000V
Position:-0.064900V
Coupling:DC
Bandwidth Limit:OFF
Probe Ratio:1X
Unit:V

CH3:Off
Scale:2.000000V
Position:-7.280000V
Coupling:DC
Bandwidth Limit:OFF
Probe Ratio:1X
Unit:V

CH4:On
Scale:2.000000V
Position:4.120000V
Coupling:DC
Bandwidth Limit:OFF
Probe Ratio:1X
Unit:V

```

```
    DSO Horizontal System
Delay:Off
Time Mode:YT
Time Scale:2.000000e-04s
Delay Time Scale:5.000000e-07s
Time Offset:5.120000e-04s
Delay Time Offset:0.000000e+00s

    DSO Acquire System
Acquire Mode:Normal
Memory Depth:600000pts
Average Num:16
Sampling Rate:125000000Sa/s

    DSO Trigger System
Trigger Mode:Pulse
Trigger Source:CH4
Trigger Edge Slope:Falling
Trigger Sweep:Auto
Trigger Coupling:DC
Trigger Noise Reject:Off
Trigger HoldOff:1.600000e-08s

CH1 Level:0.000000V
CH2 Level:-0.089900V
CH3 Level:3.080000V
CH4 Level:1.280000V

Pulse Condition:Positive More
Pulse High Time:0.000010s
Pulse Low Time:0.000002s

Slope Condition:Positive More
Slope High Time:0.000002s
Slope Low Time:0.000001s
Slope Win:Win Up
Slope Level1:4.000000V
Slope Level2:0.000000V

Video Polarity:Positive
Video Sync:All Lines
Video Standard:NTSC
Video Line:1

Runt Polarity:Positive
Runt Condition:Do not care
Runt Win:Win Up

Windows Type:Rising
Windows Pos:Enter
Window Time:0.000001s

NCycle Edge:Rise
NCycle Time:0.000001s
NCycle Num:2

Pattern CH1:X
Pattern CH2:X
Pattern CH3:X
Pattern CH4:X

Delay A:CH1
Delay B:CH2
Delay A Slope:Rising
Delay B Slope:Rising
Delay Range:More
Delay High:0.000002s
Delay Low:0.000001s

TimeOut Slope:Rising
TimeOut:1.600000e-08s

Duration Type:More
Dura. High:0.000002s
Dura. Low:0.000001s

Setup/Hold Clk:CH2
Setup/Hold Data:CH1
Setup/Hold Slope:Rising
Setup/Hold Patt.:H
Setup/Hold Type:Setup
Setup Time:0.000001s
Hold Time:0.000001s

RS232 Source:CH4
RS232 Type:Frame Start
RS232 Stop Bit:1
RS232 Parity:Odd
RS232 Data Bit:8
RS232 Baudrate:115200
RS232 Data:16

IIC Clock Source:CH1
IIC Data Source:CH2
IIC Type:Start
IIC Address:1
IIC Direction:Read
IIC Address Length:7
IIC Byte Length:1
IIC Data:82

SPI SCLK:CH1
SPI SDIO:CH2
SPI Mode:CS
SPI CS Mode:Low
SPI Edge:Rise
SPI Timeout:0.000001s
SPI Data Length:8
SPI Data:82

    LA System
D0~D7  Threshold Type:TTL
D0~D7  Threshold Value:1.400000V
D8~D15 Threshold Type:TTL
D8~D15 Threshold Value:1.400000V
D0~D7  Status:0000 0000
D8~D15 Stauts:0000 0000
```

Appendix H

Rigol DS1104Z parameters - ATmega328p using GPIO trigger

```
Model:DS1104Z
SN:DS1ZA155002892
Manufacturer:RIGOL TECHNOLOGIES
Board Ver:0.1.1
Firmware Ver:0.2.3.11
BOOT Ver:0.0.0.12
CPLD Ver:0.0
SoftWare Ver:00.04.04.SP1


    DSO Vertical System
CH1:Off
Scale:1.000000V
Position:0.000000V
Coupling:DC
Bandwidth Limit:OFF
Probe Ratio:10X
Unit:V

CH2:On
Scale:0.005000V
Position:-0.055000V
Coupling:DC
Bandwidth Limit:OFF
Probe Ratio:1X
Unit:V

CH3:On
Scale:2.000000V
Position:-7.280000V
Coupling:DC
Bandwidth Limit:OFF
Probe Ratio:1X
Unit:V

CH4:On
Scale:5.000000V
Position:10.300000V
Coupling:DC
Bandwidth Limit:OFF
Probe Ratio:1X
Unit:V

```

```
    DSO Horizontal System
Delay:Off
Time Mode:YT
Time Scale:5.000000e-05s
Delay Time Scale:5.000000e-07s
Time Offset:-2.800000e-04s
Delay Time Offset:0.000000e+00s

    DSO Acquire System
Acquire Mode:Normal
Memory Depth:30000pts
Average Num:16
Sampling Rate:50000000Sa/s

    DSO Trigger System
Trigger Mode:Edge
Trigger Source:CH4
Trigger Edge Slope:Falling
Trigger Sweep:Single
Trigger Coupling:DC
Trigger Noise Reject:Off
Trigger HoldOff:1.600000e-08s

CH1 Level:0.000000V
CH2 Level:-0.080000V
CH3 Level:3.080000V
CH4 Level:1.800000V

Pulse Condition:Negative More
Pulse High Time:0.000002s
Pulse Low Time:0.000002s

Slope Condition:Positive More
Slope High Time:0.000002s
Slope Low Time:0.000001s
Slope Win:Win Up
Slope Level1:10.000000V
Slope Level2:0.000000V

Video Polarity:Positive
Video Sync:All Lines
Video Standard:NTSC
Video Line:1

Runt Polarity:Positive
Runt Condition:Do not care
Runt Win:Win Up

Windows Type:Rising
Windows Pos:Enter
Window Time:0.000001s

NCycle Edge:Rise
NCycle Time:0.000001s
NCycle Num:2

Pattern CH1:X
Pattern CH2:X
Pattern CH3:X
Pattern CH4:X

Delay A:CH1
Delay B:CH2
Delay A Slope:Rising
Delay B Slope:Rising
Delay Range:More
Delay High:0.000002s
Delay Low:0.000001s

TimeOut Slope:Rising
TimeOut:1.600000e-08s

Duration Type:More
Dura. High:0.000002s
Dura. Low:0.000001s

Setup/Hold Clk:CH2
Setup/Hold Data:CH1
Setup/Hold Slope:Rising
Setup/Hold Patt.:H
Setup/Hold Type:Setup
Setup Time:0.000001s
Hold Time:0.000001s

RS232 Source:CH4
RS232 Type:Frame Start
RS232 Stop Bit:1
RS232 Parity:Odd
RS232 Data Bit:8
RS232 Baudrate:115200
RS232 Data:16

IIC Clock Source:CH1
IIC Data Source:CH2
IIC Type:Start
IIC Address:1
IIC Direction:Read
IIC Address Length:7
IIC Byte Length:1
IIC Data:82

SPI SCLK:CH1
SPI SDIO:CH2
SPI Mode:CS
SPI CS Mode:Low
SPI Edge:Rise
SPI Timeout:0.000001s
SPI Data Length:8
SPI Data:82

    LA System
D0~D7  Threshold Type:TTL
D0~D7  Threshold Value:1.400000V
D8~D15 Threshold Type:TTL
D8~D15 Threshold Value:1.400000V
D0~D7  Status:0000 0000
D8~D15 Stauts:0000 0000
```

Bibliography

[AF17] Gildas Avoine and Loïc Ferreira. Rescuing LoRaWAN 1.0. Cryptology ePrint Archive, Report 2017/651, 2017. `https://eprint.iacr.org/2017/651`.

[And14] Jason Andress. *The Basics Of Information Security - Understanding The Fundamentals Of Infosec In Theory And Practice*. Syngress, 2nd edition, 2014.

[Bar16] Elaine B. Barker. SP 800-57: Recommendation for key management, part 1: General (rev. 4). Technical report, National Institute of Standards and Technology (NIST), Gaithersburg, MD, United States, 2016.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, pages 37–51, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[Ber05] Daniel J. Bernstein. Cache-timing attacks on AES. *University of Illinois*, 2005.

[BTS16] Bluetooth Core Specification v5.0. techreport, Bluetooth SIG, 2016. Retrieved on July 9, 2018.

[Car15] Tim Carstens. The msel orp system-on-a-chip, 2015. Retrieved on August 23, 2018.

[CMW14] Christophe Clavier, Damien Marion, and Antoine Wurcker. Simple power analysis on AES key expansion revisited. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, volume 8731, pages 279–297, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[Com03] Committee on National Security Systems. National policy on the use of the advanced encryption standard (AES) to protect national security systems and national security information, 2003.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, nov 1976.

[DKH13] Anh Do, Soe Thet Ko, and Aung Thu Htet. Electromagnetic side-channel analysis on Intel Atom processor. *Worcester Polytechnic Institute*, 2013.

[dme17] dmercer. Bandwidth vs sample rate and the ADALM1000, 2017. Retrieved on August 22, 2018.

[dO18] Greg d'Eon and Colin O'Flynn. Tutorial CW305-2 Breaking AES on FPGA, 2018. Retrieved on August 7, 2018.

[Dwo01] Morris J. Dworkin. Recommendation for block cipher modes of operation: Methods and techniques. techreport, National Institute of Standards & Technology (NIST), Gaithersburg, MD, United States, 2001.

[Dwo04] Morris J. Dworkin. Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality. Technical report, National Institute of Standards & Technology (NIST), Gaithersburg, MD, United States, 2004.

[Ele98] Electronic Frontier Foundation. *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1998.

[FAA$^+$17] Shah Fahd, Mehreen Afzal, Haider Abbas, Waseem Iqbal, and Salman Waheed. Correlation power analysis of modes of encryption in AES and its countermeasures. *Future Generation Comp. Syst.*, 83:496–509, 2017.

[Fai83] Fairchild Semiconductor. CMOS, the ideal logic family. *Application Note 77*, (77), 1983.

[Gal30] É. Galois. Sur la théorie des nombres. *Bulletin des Sciences mathématiques*, XIII:428, 1830.

[Gem17] Gemalto, Actility and Semtech. LoRaWAN^{TM} security: Full end-to-end encryption for IoT application providers, 2017.

[Goo18] HTTPS encryption on the web – google transparency report, 2018.

[GP99] Louis Goubin and Jacques Patarin. *DES and Differential Power Analysis: The "Duplication" Method*, pages 158–172. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[GST14] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 444–461. Springer, 2014.

[GST17] Daniel Genkin, Adi Shamir, and Eran Tromer. Acoustic cryptanalysis. *J. Cryptology*, 30(2):392–443, 2017.

[HMV03] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography.* Springer-Verlag, Berlin, Heidelberg, 2003.

[IEE00] The authoritative dictionary of IEEE standards terms, seventh edition. *IEEE Std 100-2000*, pages 1–1362, December 2000.

[IEE11] IEEE standard for local and metropolitan area networks–part 15.4: Low-rate wireless personal area networks (lr-wpans). *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)*, pages 1–314, September 2011.

[Jaf07] Joshua Jaffe. A first-order DPA attack against AES in Counter mode with unknown initial counter. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2007.

[Jon12] David L. Jones. EEVblog #279 - How NOT To Blow Up Your Oscilloscope! [YouTube video], 2012. Last acccessed on Aug. 3, 2018.

[KHF$^+$18] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2018.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 388–397, London, UK, UK, 1999. Springer-Verlag.

[Kni16] Matt Knight. Reversing LoRa, 2016.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology — CRYPTO '96*, pages 104–113. Springer Berlin Heidelberg, 1996.

[KPP$^+$06] Sandeep Kumar, Christof Paar, Jan Pelzl, Gerd Pfeiffer, and Manfred Schimmler. Breaking ciphers with COPACOBANA –a cost-optimized parallel code breaker. In *Lecture Notes in Computer Science*, pages 101–118. Springer Berlin Heidelberg, 2006.

[Lee90] Jan Leeuwen. *Handbook of theoretical computer science: Algorithms and complexity*, volume 1. Elsevier, 1990.

[LoR15] LoRaWAN$^{\text{TM}}$ specification v1.0.1. Technical report, LoRa Alliance, 2015.

[LoR17] LoRaWAN$^{\text{TM}}$ specification v1.1. Technical report, LoRa Alliance, 2017.

[LSG$^+$18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[Man02] Stefan Mangard. A simple power-analysis (SPA) attack on implementations of the AES key expansion. In *International Conference on Information Security and Cryptology*, volume 2587, pages 343–358. Springer, 2002.

[Mer78] Ralph C. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299, apr 1978.

[Mes00] Thomas S. Messerges. Using second-order power analysis to attack DPA resistant software. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, pages 238–251, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[Mic17] RN2483 - Wireless Modules. techreport, Microchip Technology Inc., 2017.

[MM07] G. L. Mullen and C. Mummert. *Finite fields and applications.* American Mathematical Society Mathematics Advanced Study Semesters, Providence, R.I. University Park, Pa, 2007.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks.* Springer US, 1 edition, 2007.

[New13] NewAE Technology Inc. ChipWhisperer - the complete open-source toolchain for side-channel power analysis and glitching attacks, 2013.

[New16] Tutorial B6 Breaking AES (Manual CPA Attack), 2016.

[NIS01] NIST-FIPS Standard. Announcing the advanced encryption standard (AES). *Federal information processing standards publication*, 197:51, 2001.

[NSA72] TEMPEST: A signal problem. *Cryptologic Spectrum*, 1972.

[OM07] Elisabeth Oswald and Stefan Mangard. Template attacks on masking – resistance is futile. In *Cryptographers' Track at the RSA Conference*, pages 243–256. Springer, 2007.

[PSB$^+$18] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cécile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. *IACR Cryptology ePrint Archive*, 2018:53, 2018.

[Rob18] Pieter Robyns. LoRa reverse engineering and AES EM side-channel attacks using SDR, 2018.

[ROSW16] Eyal Ronen, Colin O'Flynn, Adi Shamir, and Achi-Or Weingarten. IoT goes nuclear: Creating a ZigBee chain reaction. *IACR Cryptology ePrint Archive*, 2016:1047, 2016.

[Sta05] William Stallings. *Cryptography and Network Security (4th Edition).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2005.

[Sut13] Graham Sutherland. A quick crypto lesson – why "MAC then encrypt" is a bad choice, 2013. Retrieved on 06 August, 2018.

[The18] The Things Network. Network architecture, 2018. Retrieved on 06 August, 2018.

[VBB04] G. Vijayaraghavan, Mark Brown, and Malcolm Barnes. Electrical noise and mitigation. In G. Vijayaraghavan, Mark Brown, and Malcolm Barnes, editors, *Practical Grounding, Bonding, Shielding and Surge Protection*, pages 102 – 131. Newnes, Oxford, 2004.

[ZF05] Yongbin Zhou and Dengguo Feng. Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing. *IACR Cryptology ePrint Archive*, 2005:388, 2005.