NTNU
Norwegian University of
Science and Technology

Improving side channel attack resilience for IoT devices

Håkon Prestegård

Master of Science in Electronics
Submission date: June 2018
Supervisor: Kjetil Svarstad, IES
Co-supervisor: Frode Pedersen, Nordic Semiconductor

Norwegian University of Science and Technology
Department of Electronic Systems

Master Thesis Assignment

Assignment title:

Improving side channel attack resilience for IoT devices

Assignment text:

IoT devices need built-in security features, as indicated in the IoT Security Compliance Framework. These are typically implemented as a SW function or combination of HW and SW. Even if implemented correctly and resilient to logical attacks, the implementation can be subject to a low-cost side channel attack. This can make secrets vulnerable if they reside in IoT end nodes, which are often installed in uncontrolled environments, where they can be subject to tampering.

The project should study how vulnerable typical MCU-based IoT end nodes are to side channel attacks, with primary focus on SW execution of security protocols. A recommendation for methods to mitigate side channel vulnerability should be provided.

Assignment proposer / Co-superviser

Frode Pedersen - Nordic Semiconductor

Supervisor

Kjetil Svarstad - NTNU

Sammendrag

Sikkerhet i Tingenes Internet er viktigere enn noen sinne. Nye produkter kommer i salg hver dag. Verden er, og vil bli enda mer sammenkoblet enn tidligere. Denne oppgaven fokuserer på motstandsdyktighet mot strømforbrukanalyse, en undergruppe av sidekanalsangrep. For å utføre disse angrepene er det nødvendig med fysisk tilgang til produktet man ønsker å angripe. Strømforbrukanalyse baserer ser på at det er mulig å hente ut nyttig informasjon gjennom strømforbruket til produktet. Krypteringsnøkler er typiske mål for slike angrep.

Denne oppgaven fokuserer på testing av forskjellige hypoteser gjennom måling. ChipWhisperer platformen er brukt for måling av forskjellige mikrokontrolleres strømforbruk samtidig som mikrokontrolleren utfører kryptering av ulike meldinger med AES kryptering. Ulike mottiltak mot strømforbrukanalyse er testet, slik som å gjøre strømmålingene ujusterte, utføre kryptering fra RAM i stedet for flash, bruke forskjellige prosessnoder (40 og 90 nanometer) og måling på ulike forsynigspunkter til mikrokontrolleren. *Correlation power analysis (CPA)*, en type av *differantial power analysis (DPA)*, er brukt til å hente ut nyttig informasjon gjennom å analysere strømmålingene. Resultatene oppnådd gjennom laboratoriearbeid viser at når signal-støyforholdet er forverret trengs det flere strømmålinger for å hente ut nok informasjon til å korrekt annslå AES nøkkelen i bruk. De største resultatene funnet er at når to mikrokontrollere med veldig lik arkitektur er brukt, er det en 700% forbedring i antall strømmålinger når man bruker 40 nanometer prosessnode i stedet for 90 nanometer prosessnode, og at ved høyere klokkefrekvenser er det mer nyttig informasjon i strømmålingene, det vil si færre målinger er nødvendig for å hente ut nøkkelen i bruk.

Abstract

Security in Internet of Things (IoT) applications is more important than ever. New devices are pushed to the market every day. The world is, and will be more connected then ever before. This thesis focus side channel resilience against a subset of side channel attacks, namely power analysis attacks. These attacks needs hardware access to the IoT device to be possible. The concept of power analysis attacks is to extract useful information through analysing power consumption. Encryption keys are typical targets.

This thesis focus on testing different hypotheses trough measurement. The ChipWhisperer platform is used for measuring different microcontrollers power consumption while encrypting different messages using the AES encryption algorithm. Different counter measures against power analysis attacks are tested, such as dis-aligning traces, executing encryption code from RAM instead of flash, using different process nodes (40 and 90 nanometer) and measuring on different power supplies or decoupling capacitors. Correlation power analysis (CPA), a type of differential power analysis (DPA), is used when analysing the captured power traces. The results obtained through experimental work shows that when the signal to noise ratio (SNR) is worsened, more power traces need to be captured in order to fully extract the used AES key. Main results are a 700% improvement in number of traces when using MCUs with very similar architecture, moving from 90 nanometer to 40 nanometer, and how higher clock frequencies increases the amount of useful power leakage.

Table of Contents

List of Tables

List of Figures

Abbreviations

MCU	=	Microcontroller
SPA	=	Simple Power Analysis
DPA	=	Differential Power Analysis
CPA	=	Correlation Power Analysis
IoT	=	Internet of Things
SNR	=	Signal to noise ratio
FFT	=	Fast Fourier Transform
XOR	=	Exclusive or

Chapter 1

Introduction

1.1 Motivation

As the world becomes more connected with more devices pushed to the marked each day, the need for a better understanding of security for IoT is severe. Security is not necessarily something which is easy to understand or to obtain in a given system. Now as there are more vendors of IoT end products than ever before many unfortunately have rather critical security breaches. Some examples of this are the GPS watches for children [10]. Here information was sent in plain text which lets anyone track someone else's children through spoofing. Another example is the Philips Hue light bulb, where it was possible with physical access to attack the device and upload new firmware to the microcontroller in the light bulb [22]. The light bulb would then communicate with other light bulbs up to 400 meters away and spread this malicious firmware, which again would spread the firmware even further. This is a good example of a typical chain reaction inflecting a whole network. The result was that the attacker could control a whole city's lights of this kind. In both examples the consequences were critical for both the user and the supplier, and not to mention very costly for the suppliers. These examples are only two of a long list of similar cases where the security breaches could be avoided if some guideline were followed.

Side channel attacks of different kinds have existed for many years, so has also general power analysis attacks. More recently, from the late 1990's, another more specific power analysis attack form has evolved. This is the differential power analysis (DPA) attack. This is an attack form which is a non-invasive attack. This means that the attack does not influence the target victim, making it hard or even nearly impossible to detect. There are different forms of DPA attacks, which have different characteristics and qualities. Power analysis attacks are used to extract information about cryptographic keys. There are many devices which are vulnerable against power analysis attacks. The attacks are cheap to perform, requires little computing power and are relatively easy to understand and perform.

NewAE Technology [17] have made different series of measuring equipment for measuring microcontrollers power consumption, as well as software and tutorials for using their equipment. These different equipment series are called ChipWhisperer. ChipWhisperer is currently available in two flavours, a *lite* edition [7] which costs 325 USD and a *pro* version [8] which cost 3500 USD. Both are available as kits with practice microcontroller targets suitable for rehearsing different attacks before moving on to more relevant and more modern microcontrollers and even high end IoT devices. Everything they deliver is highly plug and play and easy to use. ChipWhisperer's strength is that the analog measuring part of the equipment is able to detect even very small changes in power consumption. ChipWhisperer also have fast data transferring unit which makes it possible to capture traces with a high number of samples. Given the previous concerns and this equipment, or any other equipment with this ease of use and low price makes it very relevant and important to seek a higher resilience against side channel attacks.

1.2 Scope, Limitations and Objectives

The main goal of this thesis is to achieve a better understanding on how to improve resilience against side channel attacks, mainly power analysis. This is done through testing and experimenting with different software and measuring techniques for different microcontrollers (MCU). In order to be able to try out and verify different hypotheses a lot of work has been done to learn and understand how different attacks work, and how to perform them. This project is only half a year long, making it hard to test, verify and manufacture hardware. Therefore the experiments done in this project will only focus on increasing side channel resilience through software techniques and different measurement setups, etc. Some hardware techniques will still be discussed, but with no verification to test whether they have an impact on the security or not. The results of this thesis will be an overview on how different approaches in firmware and measurement impact different MCUs side channel resilience. The main contribution is the discussion made upon the comparison of how the different firmware techniques and measurement setups affect MCU leakage.

This project has many objectives. In addition to learn how to attack and the theory behind the different attacks, learning how to create PCBs, program and debug different MCUs, etc., two iterative objectives sticks out. The work flow will mainly switch between developing attacks for the victims current firmware, and then if and when the cryptographic key is extracted, make changes in the victims firmware and try to make it harder to be attacked. A much used cryptographic software suite called MBED TLS [16] created by ARM will be used as a reference point for the cryptographic operations. Much effort has also been put into developing and running the final tests which all use the same attack in order to produce result with equal conditions.

1.3 Outline

Chapter 2

Cryptography

Cryptography has been used for ages. It is a way to make your data only readable to wanted receivers. The data's content will not be available for others than who you have shared your secret key or other method to decrypt the data with. There exists a lot of different cryptographic schemes, however some have been proven not to be as secure as when first introduced. To achieve a new, standardised, highly secure and efficient encryption algorithm, the National Institute of Standard and Technology [3] held a competition where cryptographers could submit their algorithms. The competition was won by the Belgian cryptographers Vincent Rijmen and Joan Daemen in 2001. In 2002 the U.S. government started using the new standard for encryption called Advanced Encryption Standard (AES). AES is today the most used encryption because of its high level of security, but also because of its ability to run efficient on different hardware from 8-bit smart cards to high end computers. This makes it suitable for IoT, and is thus the only encryption algorithm considered in this thesis.

2.1 Advanced Encryption Standard (AES)

AES is a block cipher. As every block cipher the AES algorithm takes two inputs: A block of data with length *n* bits, and a key with length *k* bits. There is a deterministic relationship between input and output. For AES the input data and key can be 128, 192 or 256 bit long independent of each other. The Federal Information Processing Standards (FIPS) are standards published by the U.S. government. These are made for public use, and especially non-military -U.S. government agencies. FIPS-197 [9] was announced in 2001. It specifies how the U.S. government should use AES. FIPS-197 specifies that the data blocks used in AES always are be 128 bits, but key length could be 128, 192 or 256 bit. In this thesis only the FIPS standard AES algorithm is referenced. The input data block is separated into bytes, giving 16 sub-bytes.

In general the AES algorithm consists of four stages:

1. SubBytes
2. ShiftRows

3. MixColumns
4. AddRoundKey

Figure 2.1 shows the program flow of the AES encryption algorithm. The four stages combined are called a round, and for AES-128 there are 10 rounds, for AES-192 there are 12 rounds and for AES-256 there are 14 rounds. AES-nn denotes key bit length. Remember we are in this thesis always working on 128 bit data block size. The algorithm begins with a AddRoundKey stage before the first round starts, and ends with a round where MixColumns stage is left out. The intermediate data values are called state, e.g. the state after the first round is the intermediate values of the cipher after the first round. For AES-128 the state of the cipher can be thought of as a 4x4 matrix where each cell is one byte of the state.

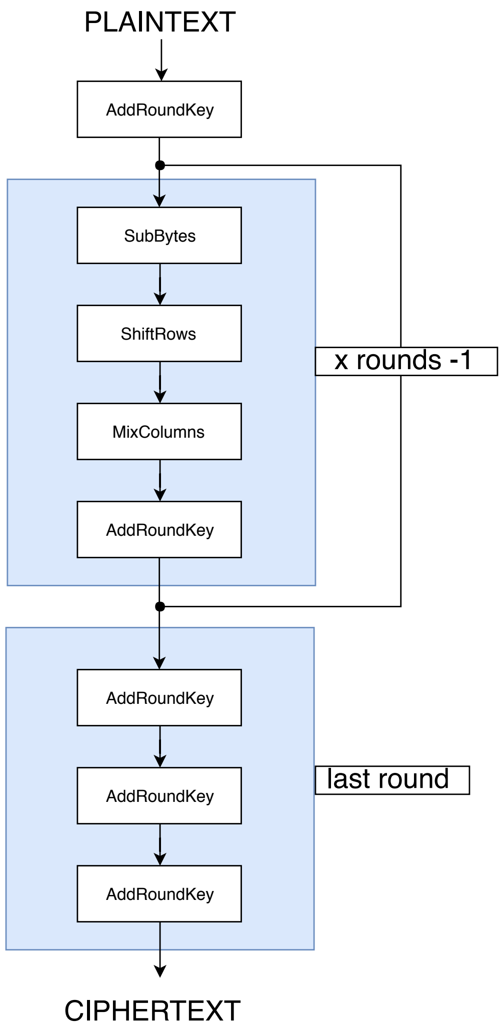


Figure 2.1: AES encryption overview.

2.1.1 SubBytes

In this stage the state is substituted byte for byte. This stage is basically a look-up table where each byte is substituted with another one depending on its value. There is a one to one dependency where a fixed byte value will be changed to some other fixed value. Figure 2.2 shows how this stage works. This stage is also referred to as the S-Box look-up.

2.1.2 ShiftRows

In this stage each row in the state matrix is shifted a certain offset. The first row is unchanged, the second row is shifted once to the left, the third row is shifted twice to the left

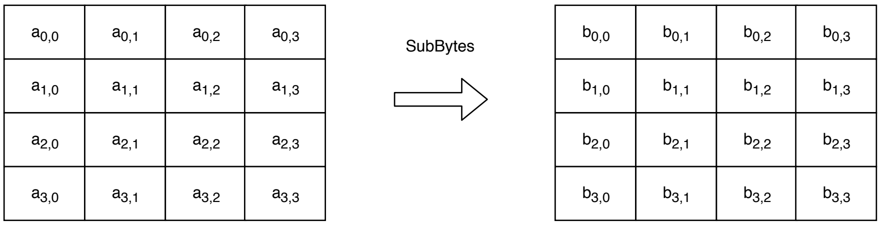

SubBytes
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

Figure 2.2: AES SubBytes stage overview

and the fourth row is shifted three times to the left. Figure 2.3 shows how this stage works.

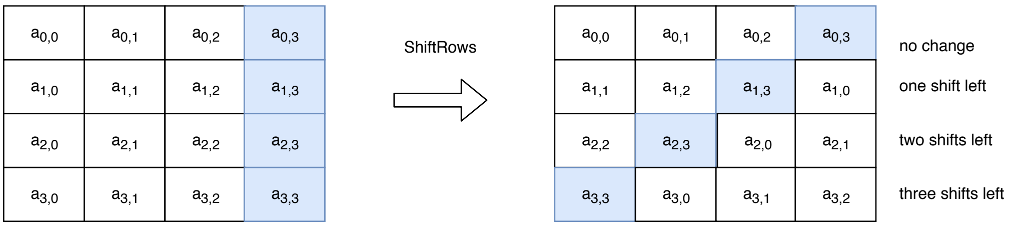

ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$
no change
one shift left
two shifts left
three shifts left

Figure 2.3: AES ShiftRows stage overview

2.1.3 MixColumns

In this stage each column in the state matrix is transformed using an invertible linear transformation. Here each input byte affect all output bytes. This and the SubByte stage provides diffusion for the cipher. The following matrix is multiplied with each column in the state. This gives a new state where all the columns values are changed

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

Figure 2.4 shows how the MixColumns stage works.

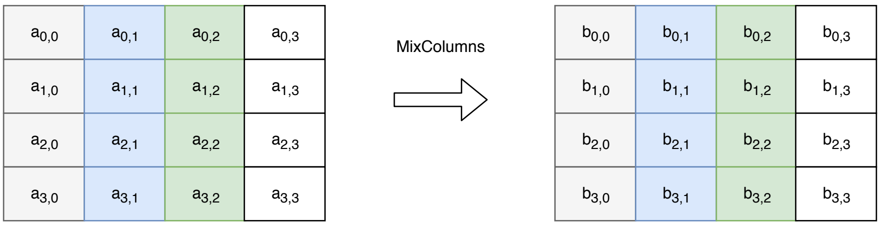
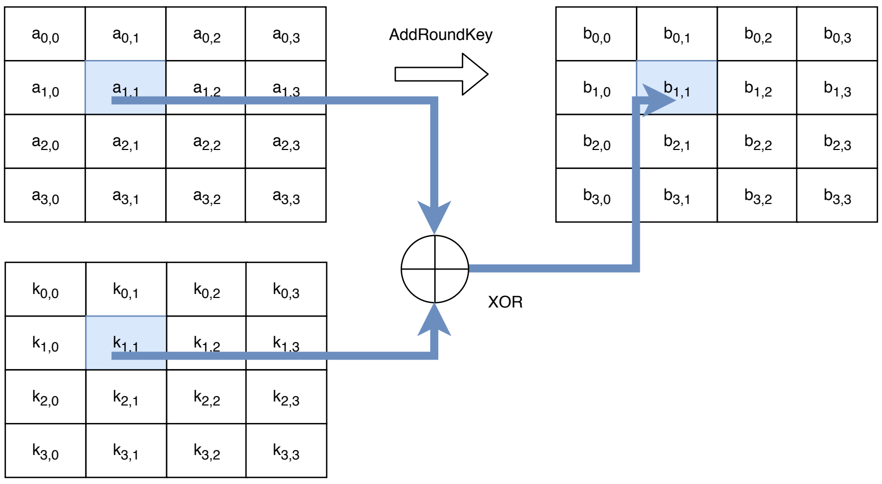

MixColumns
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

Figure 2.4: AES MixColumns stage overview

2.1.4 AddRoundKey

In this stage the state and the subkey is combined. Each byte in the state is XORed with the corresponding byte of the key. The key used in the different rounds are derived from the main key, which is used in the initial round, using the Rijndael's key schedule [4]. Figure 2.5 show how the AddRoundKey stage works.



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
AddRoundKey
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,0}$ $k_{2,1}$ $k_{2,2}$ $k_{2,3}$
$k_{3,0}$ $k_{3,1}$ $k_{3,2}$ $k_{3,3}$
XOR

Figure 2.5: AES AddRoundKey stage overview

2.1.5 Modes

Electronic Code-Book (ECB) mode is used in this thesis. This is the most basic mode of AES. This mode uses the same key every time for encrypting/decrypting data. Other modes of AES use different variants of key initialisation in order to make it harder to obtain the encryption key through logic analysis. Initialisation of the encryption key could be to XOR the last produced ciphertext or some other shared secret with the encryption key. This will make it harder for an attacker to figure out what the correct encryption key is since each encryption will use different keys.

2.1.6 Weaknesses

A clear weakness of AES regarding power analysis attacks is in the first AES round where the plaintext is XORed with the encryption key and used in the S-Box look-up. Power analysis attacks are described in chapter 3. This is the only place where the encryption key is exposed directly and used together with the plaintext. No masking nor diffusion of the plaintext or the encryption key before this.

Chapter 3

Side Channel Attacks

Side channel attacks are any attacks which gain information from the physical implementation of a device. This could e.g. be to extract information from an encryption process, where the information is neither the input plaintext nor the output ciphertext. Some examples of side channel attacks are timing attacks, power analysis, glitching attacks and H-field analysis, etc. In this thesis the attack focus will be on different variations of power analysis. General information about power analysis is presented first, then some of the most relevant attack methods are described in more detail. A severe danger of side channel attacks is that it is possible to e.g. extract full AES encryption keys with equipment which cost from 325 USD, see the ChipWhisperer Lite [7] for a low cost measuring equipment suited for power analysis. This makes more traditional security measures suited against e.g. probing or using electron microscope worth a lot less. Examples of more traditional security measures can be seen in the IoT Security Foundation's security compliance framework [11]. They provide a thorough list of traditional IoT security requirements.

3.1 Measurement

For power analysis attacks a good signal to noise ratio (SNR) is essential when measuring the target MCU's power consumption. With a higher SNR fewer traces are needed in order to have a successful attack. A trace is a series of samples. For power analysis a trace must at least contain enough samples to cover the cryptographic operation of interest. SNR can be influenced by counter measures added in both hardware and firmware, as well as bad measuring equipment. When the SNR is good it is easier to differentiate traces from one another and find needed correlations for the running attacks. In order to achieve good trace measurements some considerations are needed.

To achieve as little as possible of added noise it is essential to measure the power consumption both physically and electrically as close as possible to the target MCU. Measurement is typically done as shown in Figure 3.1 where a shunt resistor is placed between the target MCU and itss power supply. It is also possible to place the shunt resistor between the target MCU and ground. This is a basic setup where the power consumed by the entire MCU is measured directly.

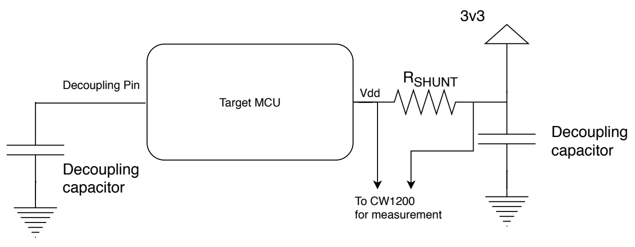


Figure 3.1: Measurement setup with shunt resistor placed between main 3v3 power supply and target MCU.

In order to improve SNR it is suitable to isolate the power consumed by the digital core which executes the cryptographic operations of interest. Most modern IoT MCUs have different power domains for digital core and other peripheral modules. It is usually possible to power the digital core through an external power supply. When this is done, measurement as shown in Figure 3.2 is possible.

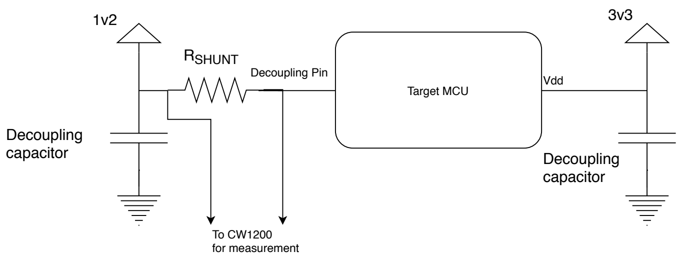


Figure 3.2: Measurement setup with shunt resistor placed between digital core power supply and target MCU.

In a circuit with MCUs there are typically several decoupling capacitors. These try to smooth out the power consumption, hence it is important to measure as close as possible to the target MCU. Naturally it is suitable to remove as many decoupling capacitors as possible to reduce the smoothing of the power consumption if it is not possible to measure after the decoupling capacitors closer to the MCU. In order to increase the ease of use for modern MCUs there are often internal voltage regulators, regulating down the main power supply of approximately 3.3 Volt down to around 1.2 Volt for the digital core. Now only one power supply is needed in stead of two. The output of the internal regulator needs a decoupling capacitor. When measuring power consumed by the digital core it is possible to insert a shunt resistor between the decoupling capacitor and the decoupling pin as shown in Figure 3.3. The decoupling pin usually serves two purposes, decoupling the internal regulator or serve as an input which provide power to the digital core.

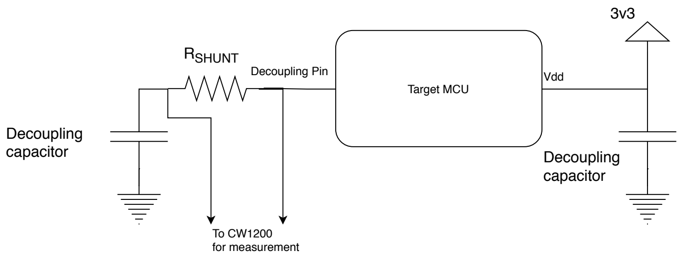


Figure 3.3: Measurement setup with shunt resistor placed between internal voltage regulator output decoupling pin and decoupling capacitor.

A small hack in a system where the internal regulator is enabled, is that it is often possible to apply a slightly higher voltage than the output from the internal regulator on the decoupling pin. No power is drawn from the regulator when a higher voltage is applied externally. This will for practical use disable the internal regulator. When this is done the measurement will be equal to the setup in in Figure 3.2. This can be exploited and possibly increase measurement SNR.

3.2 Simple Power Analysis (SPA)

Paul Kocher and his colleagues, Jaffe and Jun have done some leading work in the field of power analysis. They have written two papers together on differential power analysis (DPA), thus they describe SPA in both papers. In their first paper [12] published in 1999 SPA is described as such:

> *Simple Power Analysis (SPA) is a technique that involves directly interpreting power consumption measurements collected during cryptographic operations. SPA can yield information about a device's operation as well as key material.*

SPA could be used to collect information about the targets cryptographic implementations by e.g. interpret how many rounds are used during encryption/decryption. SPA is the simplest form of power analysis. Kocher, et al. [13] from 2011 states that

> *Simple power analysis is a collection of methods for inspection power traces to gain insight into a device's operation, including identifying data-dependent power variations. SPA focuses on examining features that are directly visible in a single power trace evident by by comparing pairs of power traces. SPA can e.g. recover key information by monitoring program flow.*

For further understanding of SPA, as well as DPA it is recommended to study both [12] and [13].

3.3 Differential Power Analysis (DPA)

DPA is an attack method which is much more powerful attack than SPA. In addition to large scale power variations found with SPA, DPA searches for correlations between different traces. There are several different DPAs. Similar for all DPAs are that they use statistical measures to find correlations between data-dependency and measured traces. For encryption use, DPAs depend on knowing either plaintext or ciphertext. E.g. DPAs search for correlation between an input plaintext and power consumption. DPA attacks were first introduced with the difference of means attack in 1999 by Kocher, et. al. [12]. Later, in 2004 the correlation power analysis attack was introduced Brier, et. al. [6]. Correlation power analysis is used in this thesis. The difference of means attack is briefly explained as a basis to show the differences and advantages with the correlation power analysis attack. Before applying a DPA attack a set of measured traces and knowing which cryptographic algorithm is in use is needed. A typical attack assumes that one MCU, a target MCU, is encrypting or decrypting some payload where power consumption is measured while the cryptographic operation is executed. The great power of DPAs is that they can treat cryptographic keys byte for byte, instead of e.g. 128 bit (16 byte) at once. For AES-128 there are 16 subkeys and 256 possibilities per subkey, giving $16 * 256 = 16 * 2^8 = 2^{12}$ checks, which is much better than 2^{128} which is needed for a brute force attack. More traces will give a clearer correlation. For more information about CPA attacks and background theory, it is recommended to read NewAE Technology theory page on CPA [18].

3.3.1 Difference of means attack

A much used methodology introduced by Kocher, et al. in [12], and later described in [13], is the difference of means attack. This attack searches for correlations by dividing the measured traces into two bins and comparing the the difference between the bins. For AES-128 search process is repeated 16 times, once for each byte of the plaintext- and key pair. For each byte, all 256 subkey combinations are tested. The process is equal for all subkey guesses. To divide the traces into two bins, a selection function is needed. When working with AES encryption, the selection function for dividing the traces could typically be to use the value of LSB for the *nth* byte's S-BOX output in the first round. Since the plaintext is known, it is possible to calculate what the LSB of the S-BOX will be for a given subkey guess.

An AES-128 example:

1. Start with byte 0 and with a guessed subkey value 0x00.
2. Calculate S-BOX output LSB value for every trace in the data set. In each trace a random plaintext is used, hence this calculation must be done for each trace separately.
3. Divide the traces into either bin 0 or bin 1 determined by the value of the S-BOX output LSB value.
4. Calculate the mean of each bin for each sample point, giving one mean-trace for each bin.
5. Compare the two mean-traces by subtracting the mean-trace from bin 1 with the mean-trace from bin 0.

6. Use the compared result-trace to look for correlations. In order to find the correct sample point for the guess-and-check while not needing to know about the implementation, different places in time are tested. I.e. scan through the compared result-trace and look for peak values. Select the highest peak. If the guessed subkey is wrong the compared result-trace will likely be close to zero for all sample points. However if there is a correlation between the guessed subkey and the measured traces there will be a peak in the compared result-trace at some sample point. Even very small differences in power consumption will be noticeable since the means are subtracted from each other in the compared result-trace.
7. Do this for all 256 possible subkey guesses.
8. Select the subkey with the highest peak value in its respective compared result-trace. This will most likely be the best guess for the given subkey.
9. Do this for all 16 subkeys.

More traces will give a better result. The difference in correlations found when guessing with wrong vs. correct subkey will increase as the number of traces grows. When a high number of traces are used, the compared result-traces found using wrong guesses will likely be entirely flat.

3.3.2 Correlation Power Analysis (CPA)

The CPA is a form of DPA. It differs a bit from the difference of means attack when searching for correlations. CPA uses a power model. This model is used to say something about the power consumption given a specific plaintext and key combination. CPA attacks have many models for expressing this. The two most common power models are the Hamming Weight and the Hamming Distance models.

The Hamming Weight power model assumes that the power consumed is proportional to the amount of bits set high on the internal data bus. This is a naive approximation. The power consumed in MCUs is often more correlated to the number of transitions from low to high, or high to low, than the amount of internal data bus lines held high. However the Hamming Weight works surprisingly well. The Hamming Distance power model proposed by Brier, et. al. [6], is a more complex model than the Hamming Weight power model. The Hamming Distance uses the number of bits that changes between each state change. A state change is the change in intermediate value when a cryptographic operation moves from one stage to another. In this thesis however, the Hamming Weight is used as power model for CPA. NewAE Technology uses by default the Hamming Weight as power model. The Hamming Weight works out of the box with both 32 and 8-bit MCUs. The Hamming Distance did not work for directly for 32-bit MCUs, hence the Hamming Weight is preferred.

A small note on CPA vs. DPA; Shown in [14], CPA is more efficient than standard DPA. That means, CPA needs in general less traces than DPA to achieve the same analysis results. However, there are some cons and corner cases where DPA could be more useful than CPA shown in Brier, et. al. [6]. The main drawback of CPA compared to DPA is that CPA has a more demanding power model. If the characterisation of the power model does not fit, the results of the attack could be useless. For standard DPA attacks there are less parameters, hence less that could go wrong. In this thesis CPA with Hamming Weight as correlation function is used, as this was found in general to be the most efficient attack.

NewAE Technology's theory page about CPA [18] and their tutorial page about CPA [19] is used as a basis in this thesis for understanding the inner workings of the attack. NewAE Technology describe how CPA work in the following way, cited from [18]:

> 1. *Write down a model for the victim's power consumption. This model will look at one specific point in the encryption algorithm. (ie: after step 2 of the encryption process, the intermediate result is x, so the power consumption is f(x).)*
> 2. *Get the victim to encrypt several different plaintexts. Record a trace of the victim's power consumption during each of these encryptions.*
> 3. *Attack small parts (subkeys) of the secret key:*
>   - *(a) Consider every possible option for the subkey. For each guess and each trace, use the known plaintext and the guessed subkey to calculate the power consumption according to our model.*
>   - *(b) Calculate the Pearson correlation coefficient between the modelled and actual power consumption.*
>   - *(c) Do this for every data point in the traces.*
> 4. *Decide which subkey guess correlates best to the measured traces.*
> 5. *Put together the best subkey guesses to obtain the full secret key.*

In other words this means that for every recorded trace, the calculated power consumption is compared to the measured trace for every possible subkey, for all subkeys. The subkey guess with the highest correlation is most likely the correct subkey. To make this explanation a bit clearer for the reader, an AES-128 encryption example is given below.

When working with AES encryption the power model could use the *nth* byte's S-BOX output in the first round as input. Remember, to execute a DPA either plaintext or ciphertext must be known. Assuming the plaintext is known, it is possible to calculate what, e.g. the Hamming Weight of the S-BOX output will be for a given subkey guess. For AES-128 search process is repeated 16 times, one time for each byte of the plaintext- and key pair. For each byte, all 256 subkey combinations are tested. The process is equal for all subkey guesses.

When finding the correlations for the different subkeys the following is a general, common procedure:

1. Start with subkey 0.
2. Start with the first trace in the data set of measured traces and the subkey guess 0x00.
3. Work through the whole trace, i.e. every sample point, and calculate correlations between the calculated power consumption (the Hamming Weight) and the actual power consumption. Do this for every trace in the data set.
4. For each sample point, sum the found correlations together.
5. Find the sample point where the summed correlation is, in absolute value, largest. There could be more than one sample point which shows a definite correlation, due to noise or other reasons. However, only the largest value is relevant.
6. Repeat for all subkey guesses (0x00 to 0xff).

7. Select the subkey guess with the largest summed correlation value. This will likely be the correct subkey.
8. Repeat for all subkeys (0 to 15).

Each subkey guess has a correlation value attached to it, saying something about how good the guess is. Comparing this the best guess to the second best guess says something about how much better the best subkey guess fits rather than the second best. If e.g. the two highest ranking guesses for a subkey have almost equal correlation values, the quality of the best guess is weaker than normal since the second best guess is almost equally good.

3.3.3 Trace Alignment and pre-processing

There are many good qualities with DPA and CPA. Both attacks are really good at finding correlations between input and power consumption, even with really small power consumption differences. However all DPAs have a severe weakness; the attacks are based upon the assumption that traces are aligned. This means that the attacks assume that the traces used in the attack are aligned, such that cryptographic operations happen at the same sample point in all traces in the given data set. When this is not the case, e.g. because some interrupts happen and makes the MCU spend time on other operations while it should encrypt some data, or when the design intentionally tries to dis-align the power leakage, the attacks suffer when trying to find correlations. Many DPA attacks does not need to know anything about the cryptographic implementation besides which algorithm is used, which is a positive side effect. In order to choose the best sample point for each subkey, the attacks search through the entire trace set to find out where the traces and the guessed subkeys best match each other. When the traces are dis-aligned, the summed correlations for the different traces will not accumulate in the same sample point. This makes the output value used when selecting the best guess for the subkey not that useful, because there will not be a distinct sample point which sticks out as a clear candidate for the correct sample point with a high correlation.

In order to deal with such dis-alignment, there are numerous ways to pre-process a given trace set before analysing it with CPA or DPA. One much used and efficient method is to use the fast Fourier transform (FFT) to find the frequency spectre of some part of a trace, and try to align the rest of the traces thereafter. This method starts by selecting start and stop sample point on a reference trace, e.g. the first trace in the data set, and analyse with FFT. This forms a windows which is used as a golden sample. The rest of the traces in the trace set are analysed with FFT and the algorithm tries to match some window in the trace under analysis with the reference window. If this is found, the trace is corrected by shifting the found window such that its frequency spectre matches the frequency spectre in the reference window.

3.4 Limitations of side channel attacks in more realistic environments

There are some limitations about these attacks worth noticing. To succeed with CPA attacks, some factors must be accompanied for. In a given application, either the plaintext or the ciphertext must be known. This could be accomplished by either interfacing with the MCU through a physical interface like UART as used in this thesis, or if the MCU is using radio through packet sniffing. There must be enough cryptographic operations happening within a relevant time frame, or else the data will no longer be interesting. A simple examples is that it is not interesting to find out tomorrow that a pizza was ordered yesterday.

In a real application, AES is typically used in counter mode or cipher block chaining mode. The counter mode uses a counter value and a number only used once to initialise the encryption, and the cipher block chaining mode uses the last produced ciphertext to initialise the key. Both modes usually starts the initialisation at zero. Meaning that if the MCU which does the encryption is reset, the initialisation will start over again from zero. The attacks depend on the same key, or a key derived from the root key, being used each time the encryption occurs. The AES mode used in this thesis, the electronic code book mode, uses the same key for every encryption. This helps to make it easier to execute the attacks, but is not necessary. The target MCU could have been reset between each encryption in order to force the MCU to initialise the key from zero again before each encryption. This will make it possible to find the key being used, even though an AES mode with a key initialisation scheme is used. This gives that if the attacker has the opportunity to control reset of the target MCU, CPA attacks will still be a real threat.

Chapter 4

Hypotheses

In this chapter this thesis's hypotheses are presented. These hypotheses are created through qualified guesswork and experiences made when working through the NewAE Technology's tutorial and exercises series. NewAE Technology have made a tutorial and exercises series that one can follow in order to learn enough to continue hacking on your own. In the beginning of the work on this thesis the whole tutorial and exercises series where completed. This was done in order to learn about the different parameters that it is possible to tune in the equipment, but also to see and understand the difference between different attacks and vulnerabilities of different MCUs. Some experiences richer with practical testing and hacking, as well as having read much literature in the field, it is now possible to point out some countermeasures and measuring setups that will be interesting to test and is within scope of this thesis. For all hypotheses ARM's MBED TLS crypto suite's [16] software implementation of AES-128 is the basis for the hypothesis under test.

The three first hypotheses are pure software, there are only changes in the cryptographic algorithm running. These are based on random noise and cycles added as an attempt to lure the attacker. Hopefully a larger amount of traces will be needed in order to successfully extract full key information. In order to effectively succeed with attacks, detailed knowledge of the firmware implementation should be known in order to adjust the attack suitably.

The fourth hypothesis is hardware/software dependent. Here the code execution is changed from running in flash to RAM. Perhaps leakage is less occurring due to RAM using less power than flash and having faster loading times.

The six last hypotheses are hardware dependent only. Here different process nodes, 90nm and 40nm, are compared against each other, single core against dual core MCU, and other measures are made to compare SNR with or without standard operation of regulators and capacitors. When working through the tutorial series, mainly two different MCUs were used. First the ATXmega128D4, see datasheet for more information [5], was used. This is a MCU known for its weaknesses against power analysis attacks. The architecture is said to be really course, and it uses a 180 nano meter process node. Further on in the tutorial series STM32Fx, x in range 1-4, is used. These MCUs use 90 nm technology node. The main experience made when running the same firmware and analysis is that a severe larger number of traces are needed in order to fully extract the used encryption key, most often approximately 10 times more traces were needed.

4.1 Hypothesis 0 - Baseline implementation of AES, proof of concept

In order to establish a reference number of traces and guesses a test of using the standard MBED TLS implementation and the normal measuring position is needed.

4.2 Hypothesis 1 - Dis-align and fool with random number of added dummy AES rounds

When executing a successful power analysis attack it is easier if the measured power traces are aligned in time, hence DPA attacks are based upon perfectly aligned traces. When traces are dis-aligned the correlations found will be less than if every trace was perfectly aligned. The hypothesis is that if random cycles, which executes operations which looks like real operations, are added, more traces are needed in order to succeed with the attack. Hopefully it should be possible to add enough, and smart enough dummy cycles such that the number of traces needed to succeed with the attack increase noticeable. Examples of dummy operations could be S-BOX look-ups. For AES this is as explained earlier the most common place to extract key information. If the dummy S-BOX look-ups added have a somewhat smart input data and key, they will produce some correlations which will make it harder to select the correct S-BOX look-ups in analysis afterwards. In order to increase execution time for the encryption as little as possible it is suitable to add as few extra cycles as possible, but still enough to cause needed dis-alignment. Some trial and error will be done in order to find some point where effects of added cycles are sufficient. The firmware implemented for this hypothesis is based upon adding a random number of dummy AES-rounds before the first real round starts. The dummy rounds will use a fixed random key, such that the analysis will hopefully be fooled into finding this key the most likely rather than the real key. There should also be added some random amount dummy rounds randomly spread between the rest of the rounds in order to make it harder to find useful correlations in the later rounds. Now the traces will both be dis-aligned and using another dummy key.

4.3 Hypothesis 2 - Random noise

When searching for correlations in traces it is easier with a good signal to noise ratio (SNR). When a lot of noise occurs it can be hard to e.g. find the correct start of cryptographic operations and align the traces thereafter. In order to find correlations with few traces a good SNR is needed. The hypothesis is that it will be harder to succeed with an attack when SNR is worsened. Examples of methods to worsen SNR could be to compute random different arithmetic operations while executing cryptographic operations. It is important that the noise is random each time cryptographic operations are executed in order to not be able to easily filter out the extra added noise.

4.4 Hypothesis 3 - Light weight masked implementation

Masking is a method where something, e.g. a ciphers intermediate value, is changed to some other value using a some scheme. This is a known counter measure for AES. The results obtained when using standard attacks will not be useful. The data found in the captured traces will indeed come from a S-BOX output, but the S-BOX look-up is feed-ed with masked values. The S-BOX in the masked implementation can also contain other values than regular S-BOXes due to the masking. This hypothesis is harder to test and verify due to the need for implementing a good enough attack that will be suited for a masked AES implementation. This is already done in hardware with quite good results, see Mangard, et. al. [15]. Here Mangard, et. al. implements three hardware crypto accelerators. They need a higher amount of traces to extract the used key compared to a standard implantation. They find that when the masking scheme is figured out, the masking does not add any value.

4.5 Hypothesis 4 - Code execution from RAM instead of flash

RAM uses less power than flash, and in also a lot quicker than flash. When the AES code is executed from RAM the S-BOX look-up will not be loaded from flash as normal. The look-up will happen in RAM, such that the power invasive flash dump will not happen. This will make the look-up use less power, hence a smaller SNR. The hypothesis is that there will be less leakage if the cryptographic algorithm is run from RAM instead of flash. In order to do this the linker script used must be updated such that memory regions for AES functions will be written to RAM.

4.6 Hypothesis 5 - Measuring from internal regulator's decoupling pin instead of 1v2 external power supply

Many MCUs have different power domains, with different voltage levels. Many typical IoT devices have a core voltage around $1 \sim 1.2$ Volt. There are often internal voltage regulators regulating main input voltage, typically 3.3 Volt down to the core voltage. Similar for these MCUs is that they all need a decoupling capacitor from the internal regulator's output pin. Adding a shunt resistor between the regulator's decoupling pin and the decoupling capacitor is a typical way of measuring power. When the internal regulator is enabled, and providing power for the digital core, noise will be introduced. Depending on the regulator, linear or switching, different levels of noise will be added. It is very hard to compete with a clean, stable, highly decoupled external power supply in terms of noise. The hypothesis is that if the internal regulator is enabled and the MCU's core is supplied from the regulator a worse SNR is achieved, hence more traces are needed in order to succeed with an attack than if the core is powered by an external power supply.

4.7 Hypothesis 6 - Measuring from 3v3 external power supply instead of 1v2 external power supply

When measuring on the 3v3 main input on a MCU the digital core's power consumption is not isolated. The measurement will contain power consumed by different parts of the MCU. This could be power hungry peripherals such as serial communication, DAC, radio, etc. When the power measured is influenced by and mixed with the power from the rest of the MCU the SNR will become lower. The hypothesis is that there will be a significant difference in number of traces needed when measuring on the 3v3 main power supply to the whole MCU, compared to measuring on an external power supply to the digital core.

4.8 Hypothesis 7 - Different process nodes

Some of the leakage, or the quality of the leakage used to extract key information, could be caused by the underlying process node. An effort has been made in order to find two MCUs with very similar architecture but different process node in order to test if there is a difference in leakage quality. The hypothesis is that there is a difference, and that the quality of the leakage is better for a larger process node. That is, it should harder to find correlations with a smaller process node. The reason for this hypothesis is that for CMOS devices, dynamic power consumption decreases with smaller process nodes while static power consumption increases. This will make the leakage from cryptographic operations blend more with the rest of the power consumption making it harder to extract only the power consumed by the cryptographic operations.

4.9 Hypothesis 8 - Different clock frequencies

Different MCUs could have a different internal frequency responses. They could act a filters, resulting in different amount of useful leakage. E.g. if using a slow frequency the useful leakage could be filtered out internally, and not be visible when measuring over a shut resistor outside the MCU.

4.10 Hypothesis 9 - Remove decoupling capacitors

A method often used when measuring power traces for attacks is to remove as many decoupling capacitors as possible in order to achieve a better SNR. The power signature will be more bouncy, and will maybe then contain more information. The hypothesis is that you need fewer traces in order to succeed with an attack when more decoupling capacitors are removed.

4.11 Hypothesis 10 - Dual core microcontroller

If the cores of a dual core MCU shares power domain there will be a lot more activity than a normal MCU with only one core. The hypothesis is that the SNR is a lot worse for dual core MCUs than for normal single core MCUs. This hypothesis could be tested with different implementation. First a clean implementation with only the encryption happening, but both cores running. Then a more noisy implementation like in Hypothesis 3 where more activity will be added to first one core, then both cores.

4.12 Hypotheses 11 - Controlling reset

There are counter measures possible for making it harder for the attacker to control reset. When the target MCU boots, it should be able to detect that it was recently reset. An interesting and effective counter measure is to use a start-up delay. The delay will be determined by how long time it has been since last reset. If the MCU has been on for a long enough time no delay will be added at the next start-up. This is the normal operation. To determine if the MCU has been on long enough a counter could be used. The counter should be initialised with a value at start-up, and it should count downwards until zero is reached. If zero is reached, no delay will be added the next start-up. However, when the MCU is frequently reset the counter will not be able to count to zero. The value left in the counter should be used start-up delay. This will lead to a delay being added each time the MCU is booting up. In order to make it even harder for the attacker to do the attack and measure the necessary traces, the delay added should increase each reset. If there is a delay present, the counter should add the default value to the present value. This will make the attacker have to wait even longer the next time the MCU is reset. The added delays could also be added such that the delay time grows exponentially. If an attacker need e.g. 1000 traces, this could be captured in a few minutes without the start-up delay. With the start-up delay, the attacker will be forced to use months or years to be able to capture the needed traces. Such a counter must not be reset due to a power cycle of the MCU. It needs a non-volatile memory. This means a memory which is not erased when power is removed. The flash in MCUs is non-volatile, but has a limit on the number of writes it can handle before it is no longer working. This limit makes flash not directly suited for such an application. It will be up to MCU architects to find a solution to the memory challenges. The hypothesis is that it will be too time demanding to successfully attack an MCU where the attacker does not control how the MCU resets.

Chapter 5

Tool chain

A lot of effort has been put into developing a suitable tool chain to be able to meet the requirements of this thesis. There are several components used in this project. Much effort has been made in understanding and figuring out how the different components work in order to produce or achieve the wanted output. Figure 5.1 shows the different components used and how they are connected. Above the dotted line are different hardware components, while below are different software components. In order to streamline the process of measuring power and analyse the traces several automation scripts are made.

5.1 Power supply and measurement

In order to find leakages useful for extracting encryption keys some measuring equipment suited for this task was needed. A few notes on measurement requirement are that the measurement equipment in use must have a high sensitivity and resolution as well as an adjustable gain in order to find even the smallest differences, a trigger I/O line input to known when to start sampling, and be capable to transfer large amounts of data from the equipment to a connected computer. NewAE Technology's ChipWhisperer Pro (CW1200) fulfils these demands as well as having a lot of other features. The ChipWhisperer Pro comes with a good graphical software and a thorough tutorial series in order to learn how to use the equipment properly and the theory and practical approaches to available attacks.

The ChipWhisperer Pro is used to measure different MCUs' power consumption, see [8] for more information and figure 5.2 to have a look. This is a rich featured measurement equipment suited for measuring small differences in power consumption of MCUs. It is also possible to e.g. measure H-field emissions, executing voltage and clock glitch attacks, and much more. A nicely made target test board, the CW308 UFO Target [20], is made to work together with CW1200. This board lets one easily run attacks on different MCUs. This board, the red board in figure 5.3, has a lot of features. It is possible to add different fixed voltages through a set of linear voltage regulators, add an adjustable voltage through a potentiometer, measure on different places, synchronise clocks, communicate over UART and I/O lines and also program and debug with an external debugger.

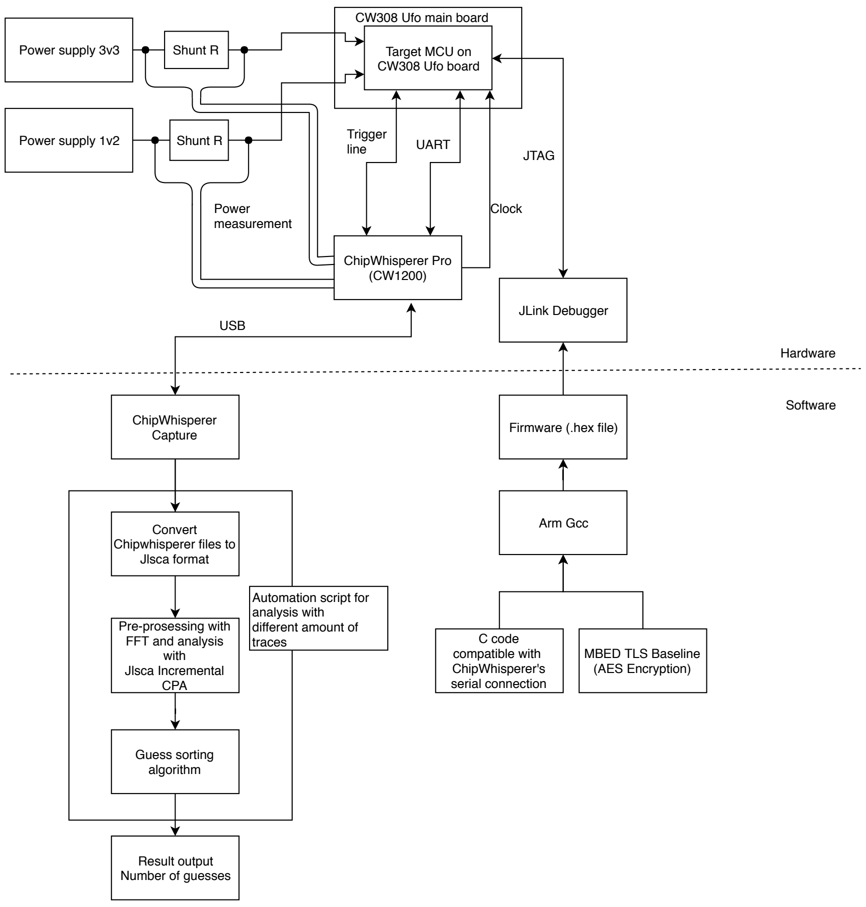
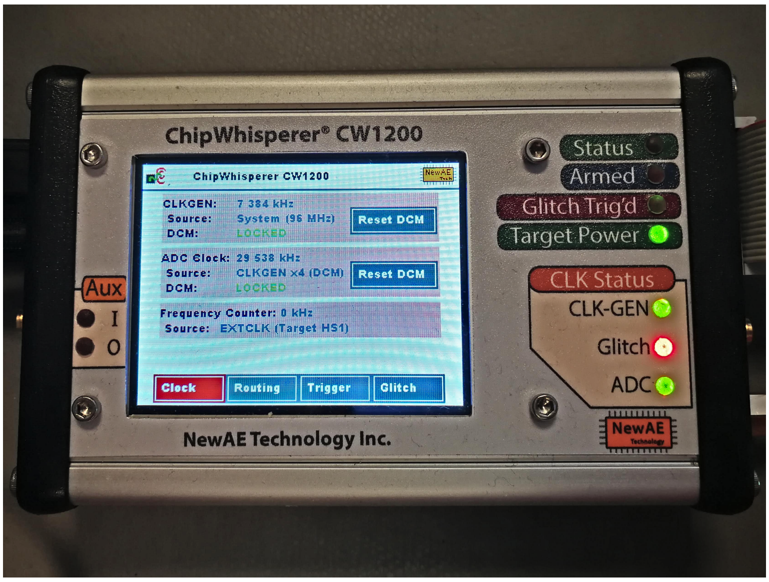

CW308 Ufo main board
Power supply 3v3
Shunt R
Target MCU on CW308 Ufo board
Power supply 1v2
Shunt R
Trigger line
UART
JTAG
Power measurement
Clock
ChipWhisperer Pro (CW1200)
JLink Debugger
USB
Hardware
Software
ChipWhisperer Capture
Firmware (.hex file)
Convert Chipwhisperer files to Jlsca format
Arm Gcc
Automation script for analysis with different amount of traces
Pre-prosessing with FFT and analysis with Jlsca Incremental CPA
C code compatible with ChipWhisperer's serial connection
MBED TLS Baseline (AES Encryption)
Guess sorting algorithm
Result output Number of guesses

Figure 5.1: Overview of the different components used in this project.



CLKGEN: 7 384 kHz
Source: System (96 MHz)
DCM: LOCKED

ADC Clock: 29 538 kHz
Source: CLKGEN x4 (DCM)
DCM: LOCKED

Frequency Counter: 0 kHz
Source: EXTCLK (Target HS1)

Figure 5.2: CW1200 ChipWhisperer Pro. Used to measure power and communicate with different MCUs under attack.

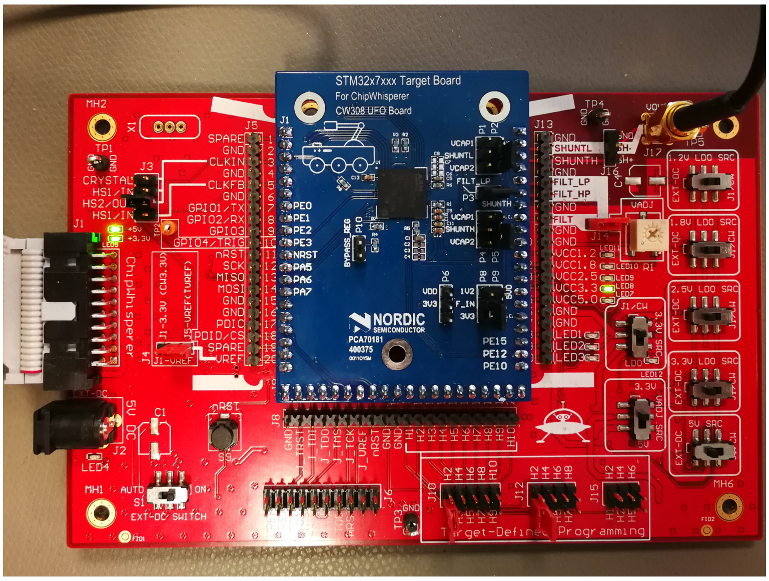


Figure 5.3: CW308 Ufo Main Target board. Used to easily change between different MCU targets.

5.2 Clock and sampling

To ease the process of measurement, CW1200 has two nice features for clock synchronisation. When the target MCU is driven from an external crystal or internal oscillator and has the possibility to use an I/O line to source a clock output, CW1200 can use this as an input and synchronise upon this. CW1200 can also provide clock to the target MCU. This is done in the testing in this thesis. A clock frequency of 7.37 MHz is used as default since this is the default clock frequency in the ChipWhisperer tutorial series. The CW1200 can sample once or four times per clock cycle. In this thesis sampling four times per clock used is selected. When sampling once per clock cycle no correlations were found, compared to having extracted an entire key with the same number of traces using four samples per clock cyckle. Four sample per clock sample thus provides more efficient traces than with only one sample per clock. The CW1200 has a maximum sample rate of 105 Mega samples per second. This means that in order to sample four times per clock the target MCU in use can have maximum clock frequency of 26.25 MHz.

5.3 Target MCUs STM32X7 and Cypress PSoC6

In order to test the different hypotheses, three MCUs are used; STM32F7, STM32H7 and Cypress PSoC6. The STM32X7 MCUs have an equal system architecture, but different process node. The F7 MCU is made in 90 nm, while the H7 is made in 40 nm. These MCUs are currently state of the art for IoT solutions, making them very suitable for testing different hypotheses. Both MCUs have an Arm Cortex M7 core. The Cypress PSoC6 MCU is a dual core MCU with an Arm Cortex M0 core and an Arm Cortex M4 core. The MCU is made in 40 nm. This MCU is currently only available for engineering samples and will likely be used in high end IoT solutions in the years to come. The MCUs support the JLink JTAG interface for programming and debugging.

5.4 Custom PCBs for power measurement

In order to measure the power consumption on desired MCUs, custom PCBs had to be made. These custom PCBs allows the attacker to select where to measure power; 5v0, 3v3, or 1v2 inputs, or on the internal regulator's decoupling pin. These PCBs also allow the attacker to choose different amount of decoupling capacitors and change the value of the shunt resistors that the power consumption is measured over easily with a soldering iron. Figure 5.4 shows the PCB made for the STM32X7 targets, and figure 5.5 shows the PCB made for the PSOC6 target. The STMX7 have equal pin-outs.

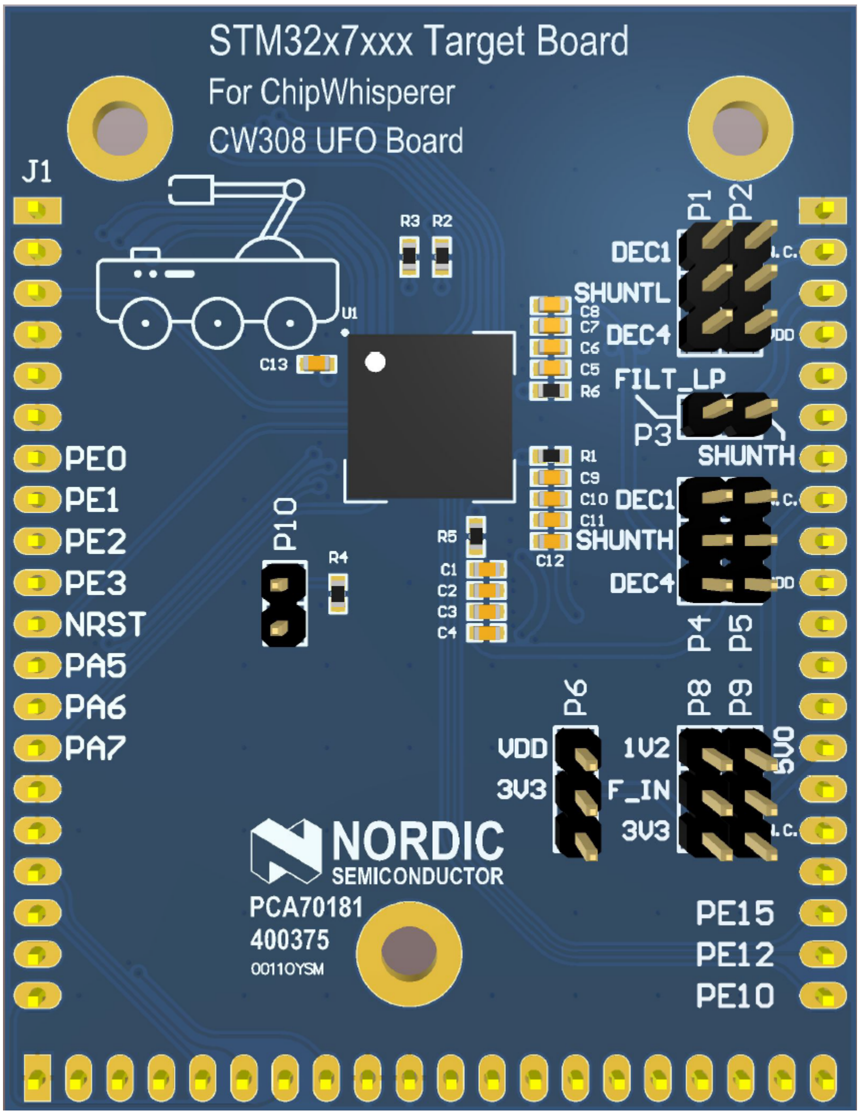


Figure 5.4: Custom pcb for STM32F7 and STM32H7 used for power measurement using the CW308 UFO Target

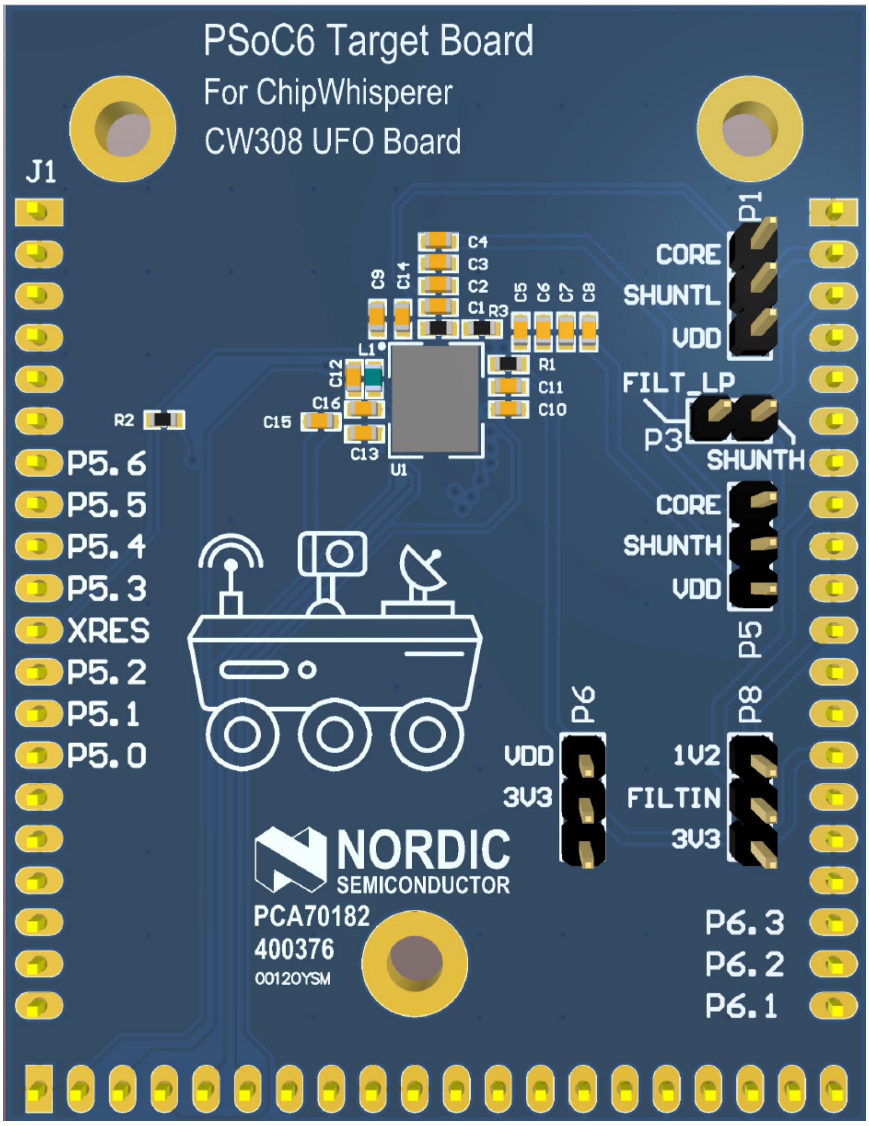


Figure 5.5: Custom pcb for Cypress PSOC6 used for power measurement using the CW308 UFO Target

Chapter 6

Implementations of target firmware

6.1 AES-128 from MBED TLS

The AES-128 implementation from MBED TLS [16] is used as a baseline in this thesis. This is a much used implementation, and is considered secure and efficient. It it is developed by ARM, which now is the leading processor core developer for IoT MCUs. The MBED TLS security suite is implemented for 32bit MCUs. The code in the next section shows MBED TLS's AES-128 implementation. This will be used as a baseline for the first three hypotheses from chapter 4. Only changes regarding the first S-BOX look-up will be shown in code listing in the next section, because this is the chosen focus of attack in this thesis. Listed in the appendix are both the MBED TLS AES standard encryption implementation and an updated linker script in order to execute code from RAM.

6.2 AES-128 with dummy rounds

The AES implementation starts with The first `AES_FROUND` in the first iteration in the for loop in line 15 is the first AES round. Here the state, X0 to X3, which is currently the plaintext XORed with the round-key, is used as input to a full AES round. This is the most interesting point of attack with CPA.

Code listing 6.1 shows part of the modified version of the MBED TLS's AES implementation used to test hypothesis 1. Starting in line 11 a dummy key is declared with its proper key schedule for all 10 rounds. The next step is XORing the plaintext and the dummy-key in lines 30 to 33, than XORing the plaintext and the round-key in lines 36 to 39. FROUND stand for forward round, as we are encrypting here instead of decrypting (reverse). The for loop starting in line 43 will add some dummy AES rounds before the first real AES round. Zero, two, four or six round will be added. The for loop starting in line 52 will add some dummy AES rounds after the first real AES round. This is added to try to fool the attacker with both dis-alignment between the different traces, and possible correlations for the dummy key in use.

Listing 6.1: C code which shows most major changes done for H1.

```c
int mbedtls_internal_aes_encrypt( mbedtls_aes_context *ctx,
                                  const unsigned char input[16],
                                  unsigned char output[16] )
{
    int i;
    uint32_t *RK, X0, X1, X2, X3, Y0, Y1, Y2, Y3;
    // Declare variables for dummy use
    uint32_t *D_RK, D_X0, D_X1, D_X2, D_X3, D_Y0, D_Y1, D_Y2, D_Y3;

    // Dummy key is expanded using AES key scedule
    uint32_t dummy_key[44] = \
    {
    0x014a956, 0xf4476a4f, 0x9ef9e283, 0x9654c9b6, \
    0x209797c6, 0xd4d0fd89, 0x4a291f0a, 0xdc7dd6bc, \
    0xdd61f240, 0x09b10fc9, 0x439810c3, 0x9fe5c67f, \
    0x00d5209b, 0x09642f52, 0x4afc3f91, 0xd519f9ee, \
    0xdc4c0898, 0xd52827ca, 0x9fd4185b, 0x4acde1b5, \
    0x71b4dd4e, 0xa49cfa84, 0x3b48e2df, 0x7185036a, \
    0xc6cfdfed, 0x62532569, 0x591bc7b6, 0x289ec4dc, \
    0x8dd359d9, 0xef807cb0, 0xb69bbb06, 0x9e057fda, \
    0x66010ed2, 0x89817262, 0x3f1ac964, 0xa11fb6be, \
    0xbd4fa0e0, 0x34ced282, 0x0bd41be6, 0xaacbad58, \
    0x94daca4c, 0xa01418ce, 0xabc00328, 0x010bae70  \
    };

    RK = ctx->rk;
    D_RK = dummy_key; // Points to start of random_rk array

    // Xor input with dummy key
    GET_UINT32_LE( D_X0, input,  0 ); D_X0 ^= *D_RK++;
    GET_UINT32_LE( D_X1, input,  4 ); D_X1 ^= *D_RK++;
    GET_UINT32_LE( D_X2, input,  8 ); D_X2 ^= *D_RK++;
    GET_UINT32_LE( D_X3, input, 12 ); D_X3 ^= *D_RK++;

    // Xor input with real key
    GET_UINT32_LE( X0, input,  0 ); X0 ^= *RK++;
    GET_UINT32_LE( X1, input,  4 ); X1 ^= *RK++;
    GET_UINT32_LE( X2, input,  8 ); X2 ^= *RK++;
    GET_UINT32_LE( X3, input, 12 ); X3 ^= *RK++;


    // Add dummy Forward rounds
    for(uint8_t dummy=0;dummy<rand()%4; dummy++)
    {
      AES_FROUND(D_RK, D_Y0, D_Y1, D_Y2, D_Y3, D_X0, D_X1, D_X2, D_X3 );
      AES_FROUND(D_RK, D_X0, D_X1, D_X2, D_X3, D_Y0, D_Y1, D_Y2, D_Y3 );

    }
    // First real Forward round
    AES_FROUND(RK, Y0, Y1, Y2, Y3, X0, X1, X2, X3 );

    for(uint8_t dummy=0;dummy<rand()%4; dummy++)
    {
      AES_FROUND(D_RK, D_Y0, D_Y1, D_Y2, D_Y3, D_X0, D_X1, D_X2, D_X3 );
      AES_FROUND(D_RK, D_X0, D_X1, D_X2, D_X3, D_Y0, D_Y1, D_Y2, D_Y3 );
    }
    // Second real Forward round
    AES_FROUND(RK, X0, X1, X2, X3, Y0, Y1, Y2, Y3 )
    ...
    ...
```

Chapter 7

Implementations of hardware setup and analysis software for power analysis attacks

7.1 Measurement scheme

The measurement setup in this thesis is based on the measurement setup in the ChipWhisperer's tutorial series. The target MCU communicates through a UART with the CW1200. The CW1200 is controlled by the ChipWhisperer capture software. Before each encryption, a key and a plaintext are sent from the capture software to the target MCU. The CW1200 now waits for its trigger input line to go high. The target MCU sets its trigger output line high when it is ready to start encrypting. The CW1200 now starts sampling as many samples as specified by the attacker in the capture software. The samples captured are called a trace. For each hypothesis, 64000 traces are captured. It takes approximately 45 minutes to capture 64000 traces. For each trace a random plaintext is used, while the same key is used every time. This corresponds to the AES electronic code book mode. The traces, as well as used key and plaintexts, are fed into the analysis attack which is the next step in the attack process. Figure 7.1 and Figure 7.2 shows the power consumption for the first AES round when using the AES baseline firmware implementation on the STM32H7 with different clock frequencies.

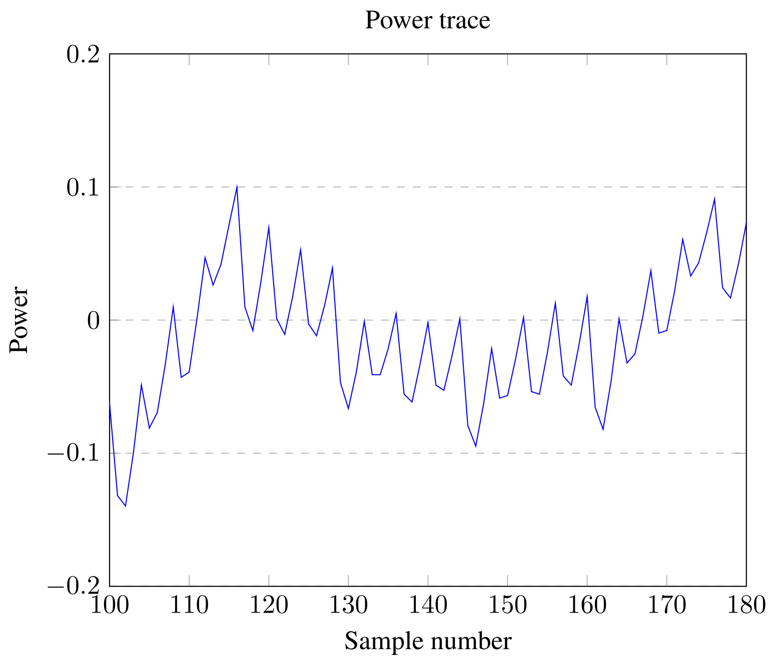


Figure 7.1: Power trace showing power consumed by STM32H7 with AES baseline implementation. Clock frequency 26.25MHz.

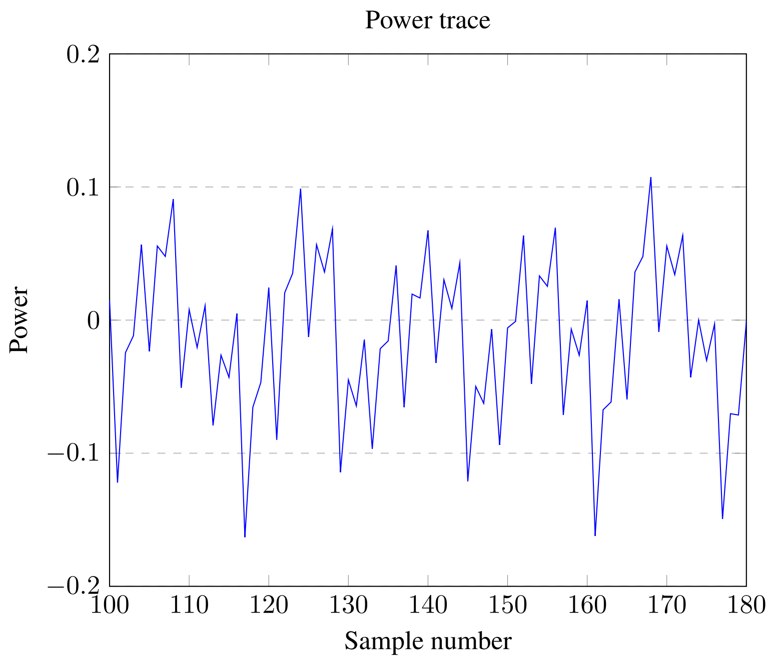


Figure 7.2: Power trace showing power consumed by STM32H7 with AES baseline implementation. Clock frequency 7.37MHz.

7.2 CPA implementation

The attack used in this thesis is the incremental CPA (Inc CPA) from the Jlsca software suite. [1]. The difference between regular CPA and incremental CPA is that the incremental CPA is slightly more efficient. The Jlsca is used because of its efficiency compared to the ChipWhisperer analysis software. Jlsca is written in the programming language Julia, a language made explicitly for parallel computation and efficacy. The authors of Jlsca have made a large effort in making the analysis become very efficient. The analysis results achieved are similar between the ChipWhisperer software and the Jlsca software, but with Jlsca having a speed-up around 30 times compared to ChipWhisperer [2]. With pre-processing the Jlsca used around 2 minutes to analyse 64000 traces of 5000 samples. The ChipWhisperer analyse software uses around one hour, before pre-processing. The pre-processing in ChipWhisperer uses several hours.

The FFT pre-processing used from the Jlsca needs to be tuned before each attack. A reference trace must be selected. This could typically be the first trace in the data set. The start and stop points of the reference window must be selected. It can be hard to know where the AES encryption starts, thus some trial and error is needed here. This will lead to multiple analyses run, before what seems to be the optimal start and stop points are found. Maximum shift must also be specified. This decides how much a trace is allowed to be displaced in order to best match the reference trace. At a last the rejection value must be selected. The rejection value specifies how good the frequency response in the trace analysed must match the frequency response in the reference trace. A high rejection value will discard traces where the frequency response found does not match the reference trace enough. This could be a good thing, forcing the attack to only use data from traces which are aligned and looks similar. However, if too many traces are discarded, the data basis for the CPA attack will too small, thus the attack suffers from a too high rejection value.

7.3 Results output from CPA

The output from CPA is a set of correlation values and their sample points for each key byte of interest. For AES-128 this gives values for the 16 subkeys. Using the Jlsca [1] CPA attack only the five highest ranking correlation values are given in the output. If the highest ranking guesses for each subkey combined does not match the full AES-128 key, there are two factors that needs to be considered when interpreting these values: The absolute value of the highest ranking, and the difference between the highest ranking value and the second and third highest value. Table 7.1 shows the output from JlSca CPA attack using the MBED TLS baseline implementation and 64000 traces. Numbers in bold font indicates correct subkey guess. Only five candidates for each subkey is included in this table. In the attack, all 256 ranks for each subkey are included.

Table 7.1: JlSca CPA output on STM32F7 using 64000 traces

	subkey 0	subkey 1	subkey 2	subkey 3	subkey 4	subkey 5	subkey 6	subkey 7
Rank 1	**0.541654**	**0.433204**	**0.183768**	**0.575257**	**0.372388**	**0.546091**	**0.395136**	**0.268970**
Rank 2	0.187777	0.192916	0.089844	0.145373	0.176642	0.126287	0.101311	0.108223
Rank 3	0.183098	0.186341	0.089717	0.128452	0.136719	0.110666	0.086402	0.104458
Rank 4	0.16161	0.183261	0.088665	0.124053	0.134910	0.108673	0.085880	0.101554
Rank 5	0.155725	0.182503	0.085534	0.119529	0.131772	0.104604	0.084925	0.100774

	subkey 8	subkey 9	subkey 10	subkey 11	subkey 12	subkey 13	subkey 14	subkey 15
Rank 1	**0.233743**	**0.636611**	**0.318354**	**0.176471**	**0.570988**	**0.366985**	**0.396251**	**0.572046**
Rank 2	0.185402	0.135774	0.106781	0.075821	0.193582	0.096643	0.125063	0.143516
Rank 3	0.181454	0.132189	0.091804	0.072507	0.193539	0.095701	0.103648	0.122472
Rank 4	0.180729	0.130820	0.089456	0.071509	0.185426	0.095645	0.098361	0.119350
Rank 5	0.179661	0.127964	0.086684	0.070669	0.163294	0.089842	0.095620	0.118004

The highest value says something about how certain one can be that the specific guess is the correct one. The higher the value is, the clearer the correlations in the data set are. When working with ChipWhisperer's tutorial series, correlation values around 0.99 were achieved for the highest ranking guesses when only 50 traces were used working with the X-MEGA128. This implies that the best guess is a really good guess. When working with STM32xx devices, more traces are needed compared to the X-MEGA128. The highest ranking correlating values are usually at minimum ten times smaller than with the X-MEGA128, and the differences between the highest ranking correlation values within one subkey are relatively smaller. This indicates that it is more difficult to extract the correct key bytes with STM32xx compared to X-MEGA128. The values in Table 7.1 shows that there are clear subkey candidates for the STM32F7. The best candidates have almost twice the value as the second best. These good results are due to a high number of traces. With less traces available, the output values from the attack will be smaller and the difference between the best values will be smaller.

The difference between the highest ranking correlation values of any guessed subkey says how much better the first guess is compared to the next best guess. The larger the difference is, the better or clearer the best guess is. Smaller differences between the different highest ranking correlation values indicates that the quality or the certainty one can have about a certain guess is lower. If the two highest ranking values are close to equal, it is not easy to tell which of the two is really better if the encryption key was not fully extracted. When working with the X-MEGA128, the second and third best subkey candidates' correlation values were often below half the value of the highest ranking subkey candidate. This gives a delta correlation above 50 percent. For the STM32xx devices this is in general not the case. Here the difference between the two highest ranking guesses range from a few to around fifty percent when more traces are used.

If some of the subkeys are not extracted correctly it is possible to execute a brute-force attack given the hypothesis that it will be clear to the attacker when the correct key is found. The brute-force attack should be executed in such a way that the subkeys where the difference between the highest ranking correlation values are smallest will be changed to the next best guess. E.g. it could be that for one or two subkeys the difference between the highest ranking values are much smaller than for the rest of the subkeys. Starting by changing the subkey guess where the difference is smallest will hopefully output the correct subkey. This makes it possible to only iterate over the different highest ranking correlations values making the attack still much more efficient than brute force attacking without any side channel information.

7.4 Brute force attack sorting algorithm

Pan, et. al. [21] have made an algorithm which finds the next best key guess from a data set after CPA attack. In their paper they call it the *sub-optimal sorting algorithm*. This algorithm finds the globally next best subkey candidate, before substituting this candidate in to all previous full key guesses, given the subkey the candidate belongs to has not already been substituted in. This will make the number of full key guesses almost double for each new candidate being tested. The algorithm works in the following way:
Fist initialise the search.

1. Calculate the global relative correlation for all candidates within all subkeys. This is done through dividing each candidate's correlation value within a subkey by the subkey's best candidate's correlation value. The best candidates will have relative correlation values of 1, and the rest will have a value between 0 and 1.
2. Make a list, *C*. Fill this with all candidates sorted by their global relative correlation values, except for the top candidates for each subkey.
3. Make a list, *G*. Initialise *G* with the best full key guess, i.e. all the best subkey candidates combined is the best full key guess.

Now the initialisation part is over and the guess-and-check begins.

1. Make a copy of *G*, e.g. *G-copy*.
2. Pop the first element in *C*, and store this as *candidate-next*. This is always the relatively next best candidate which should likely be in the full key.

3. For all full key guesses in *G-copy*, change the subkey *candidate-next* belongs to, to *candidate-next*'s guess.
4. Test all new full keys in *G-copy*.
5. If the correct key is not found, append *G-copy* to *G* and continue from 1 again.

The code listing *Brute force sorting algorithm*, this author's implementation of the *sub-optimal sorting algorithm*, is added to the appendix. Since the testing executed in this thesis is always with a known key, the full key guesses are only checked against the real key instead of encrypting/decrypting a message to speed up the search. Here the parts of the code where the logs from the CPA output are parsed and where the relative correlation values are calculated are left out. It should be easy to understand how the relative peak is calculated. This algorithm uses very much memory when reaching a high number of guesses, thus it is forced to stop executing when a fixed number of guesses is reached.

7.5 Final result output

At the end of the sorting process, the final output of the attack will be a number stating how many guesses in addition to the CPA attack is needed in order to find the correct key. In every test 64000 traces are captured and analysis is done for 75, 125, 250, 500, 1000, 2000, 4000, 8000, 16000, 32000 and 64000 traces. The output of every test will be a plot displaying how the different hypotheses affect the different MCUs and how it affects the number of traces needed. Figure 7.3 shows how the STM32F7 with the baseline MBED TLS implementation behaves. Here it is clear that when the number of traces reaches a certain amount, few guesses are needed in order to fully extract the correct key. The dashed line is only added to show the trajectory of how the behaviour is changing. For trace numbers below 500 the test setup was not able to find the correct key before the max guesses limit was reached. In the results chapter, chapter 8, results from the different tests will be presented as shown here with plots.

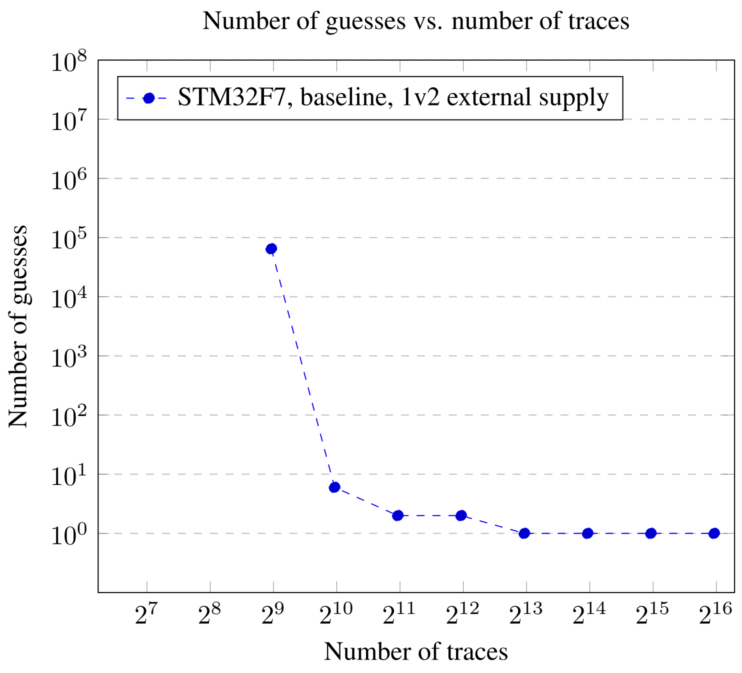


Figure 7.3: STM32F7 baseline MBED TLS

Chapter 8

Results

8.1 Measurement and analysis scheme

For every hypothesis under testing, 64000 traces are captured. CPA attack and brute force guessing is done for different subsets of the captured traces. 75, 125, 250, 500, 1000, 2000, 4000, 8000, 16000, 32000 and 64000 traces are used in analysis. The power supply, firmware, and target MCU is specified in the following sections where the different results are presented. The STM32F7 with baseline AES implementation is used as reference target, and a clock frequency of 7.37 MHz is used unless otherwise specified. The following hypotheses, which are the hypotheses tested, are presented in the listed order:

Hypotheses 7 - different process node,

Hypotheses 1 - added dummy AES rounds,

Hypotheses 5 - measuring on internal regulator decoupling pin,

Hypothesis 6 - measuring on 3v3 main input and

Hypotheses 4 - code executed from RAM,

Hypothesis 8 - different clock frequencies

Their results are presented in this chapter. The other hypotheses, such as e.g. Hypothesis 2 - added random noise, Hypothesis 9 - removing decoupling capacitors and Hypothesis 10 - dual core MCU, are not tested due this thesis' duration.

Before interpreting the results from the plots within this chapter, two things are worth mentioning. The number of guesses needed in order to find the used key is the inverse of the quality of the useful information within the traces in the trace set. The quality will increase when the number of traces increase. The number of traces needed to succeed with finding the used key is the most interesting measure. This tells how vulnerable a system is to a power analysis attack.

8.2 Results from different process nodes

Figure 8.1 shows how using a MCU with 90nm compared to a chip with 40nm process node affect the useful power leakage. In both tests an external power supply of 1v2 to the digital core is used. Measurement is done as shown in Figure 3.2, where a shunt resistor is placed between the digital core's decoupling/inut power pin and the digital core's decoupling capacitor. Both tests use the same baseline AES firmware. The results from the STM32F7 (90nm) are used as reference results throughout this thesis. The Figure 8.1 shows that only 500 traces are needed with the STM32F7 in order to fully extract the AES key used, while 4000 traces are needed for the STM32H7. The plots stop where with the minimum amount of traces where found. The dashed lines between the marks are only added to show the trajectory of how the number of guesses could be.

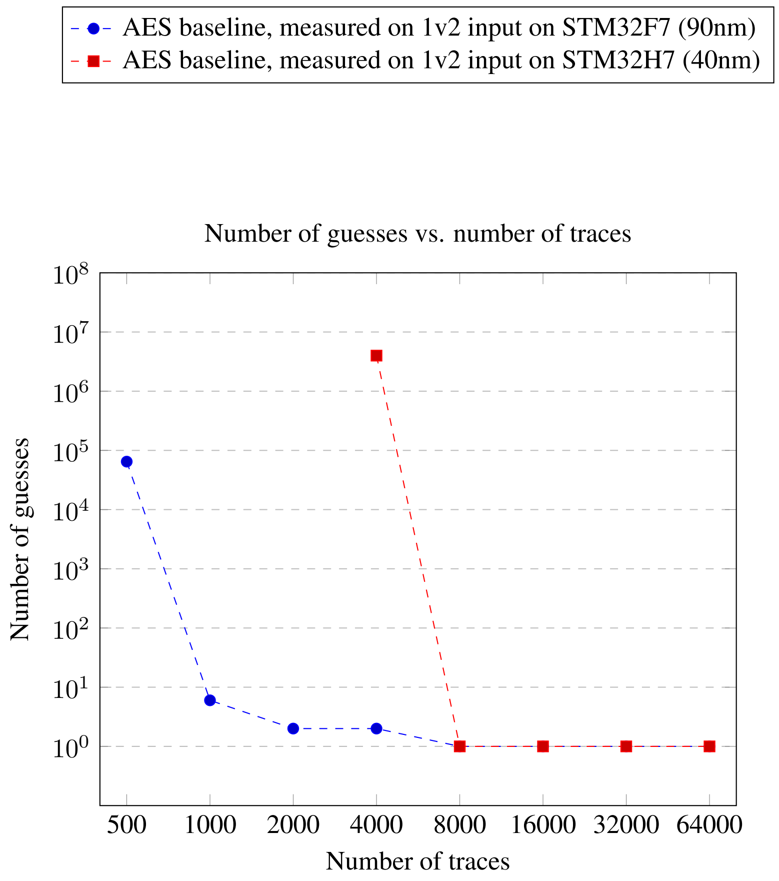


Figure 8.1: Plot showing how different process nodes impact number of guesses vs. traces for STM32H7(90nm) and STM32F7(40nm).

8.3 Results from different firmware implementations

Figure 8.2 shows how running the firmware with added dummy AES rounds affect the leakage from the STM32H7 compared to the leakage from the baseline AES implementation. Both tests are measured on 1v2 input. The reference results from STM32F7 and the results from STM32H7 baseline AES implementation measured on 1v2 input are added to give the plot more perspective. It is clear that with the dummy AES rounds added, a much higher amount of traces are needed to extract the used AES key compared to the baseline implementation.

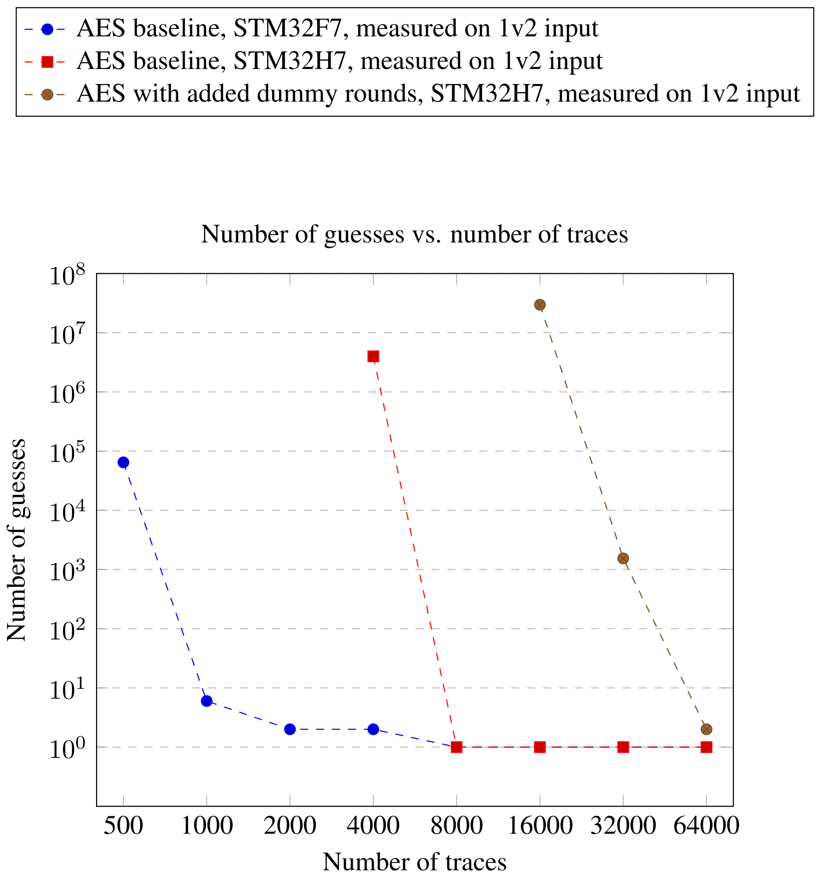


Figure 8.2: Plot showing how different firmware impact number of guesses vs. traces for STM32H7.

8.4 Results from different power supplies and measuring positions

Figure 8.3 shows how measuring on internal regulator decoupling pin and measuring on 3v3 main input affects the leakage from the STM32H7 compared to measuring on 1v2 input. The reference results from STM32F7 and the results from STM32H7 baseline AES implementation measured on 1v2 input are added to give the plot more perspective. Internal decoupling pin is measured as shown in Figure 3.3 and 3v3 main input is measured as shown in 3.1. All tests use the baseline implementation of AES. When measuring on the internal regulator decoupling, 8000 traces are needed. When measuring on the 3v3 main input 16000 traces are needed.

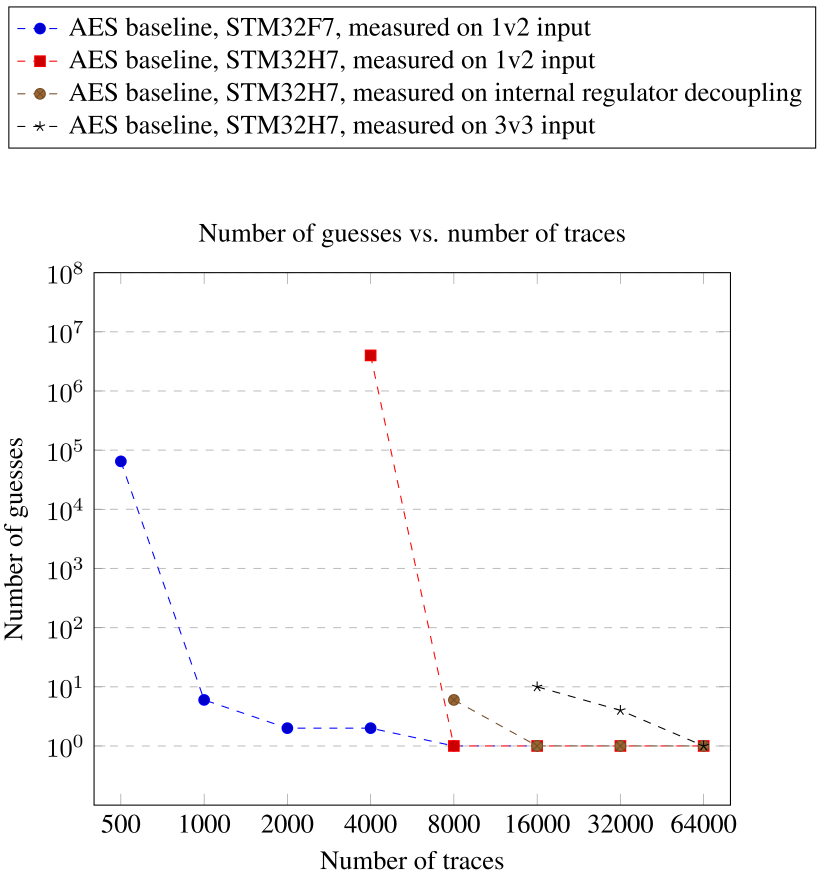


Figure 8.3: Plot showing how different power supplies and measuring positions impact number of guesses vs. traces for STM32H7.

8.5 Results from code execution from RAM

Figure 8.4 shows how execution encryption code from RAM affects the leakage from the STM32H7 compared execution from FLASH (normal). This is tested on the STM32H7 when measuring on both 1v2 input and 3v3 main input. The reference results from STM32F7 and the results from STM32H7 baseline AES implementation measured on 1v2 input and 3v3 main input are added to give the plot more perspective. The plots shows that more traces are needed when encryption code is executed from RAM. The red lines show that measuring on the STM32H7's 1v2 input, 8000 traces are needed compared to 4000 when code is executed from flash. The green lines shows that when measuring on 3v3 main input 16000 traces are needed in both cases, but with a much higher guessing number when code is executed from RAM instead of flash.

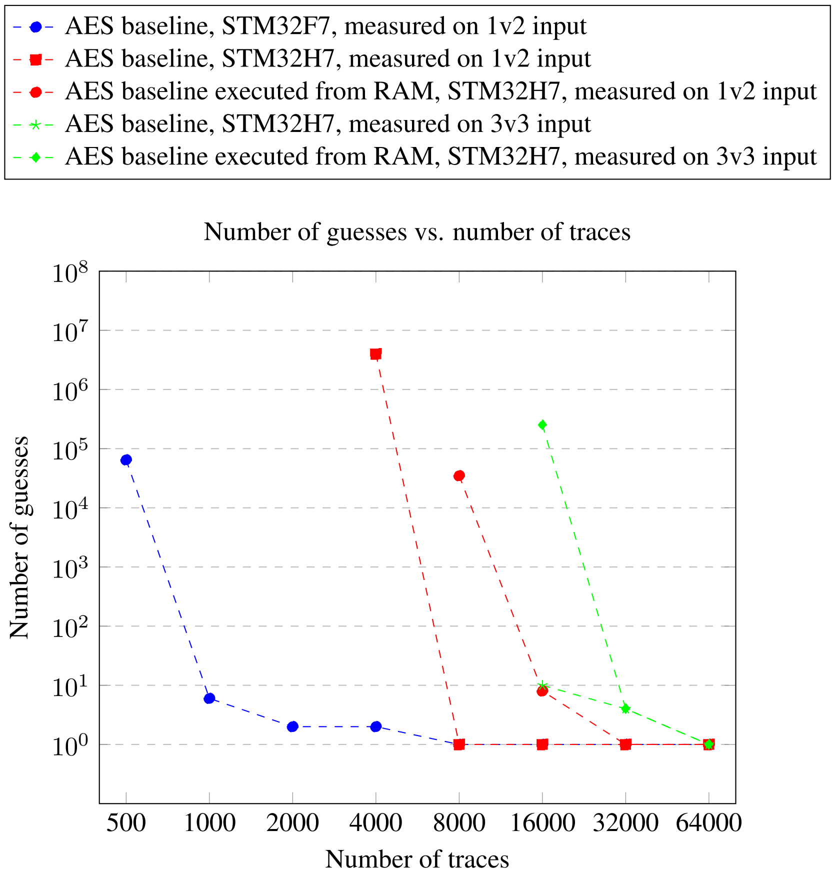


Figure 8.4: Plot showing how code execution from RAM impact number of guesses vs. traces for STM32H7.

8.6 Results from different clock frequencies

Figure 8.5 shows how different clock frequencies affect the leakage from the STM32H7 and STM32F7. On the STM32H7 this is tested and measured both on the input of the 1v2 external supply for the digital core and on the main 3v3 external supply, while for the STM32F7 only on the 1v2 input on the digital core. All tests use the baseline implementation of AES. The reference results from STM32F7 and the results from STM32H7 baseline AES implementation measured on 1v2 input are added to give the plot more perspective. The blue lines in the plot shows that with the same amount of traces are needed for the STM32F7 measured in 1v2 input independent of the clock frequency. The red lines shows that the STM32H7 measured on 1v2 input needs less traces with the higher frequency. This also applies for the green lines which shows the STM32H7 measured on 3v3 main input.

For STM32H7 measured on 1v2 only 1000 traces are needed, compared to 4000 with 7.37 MHz clock frequency.

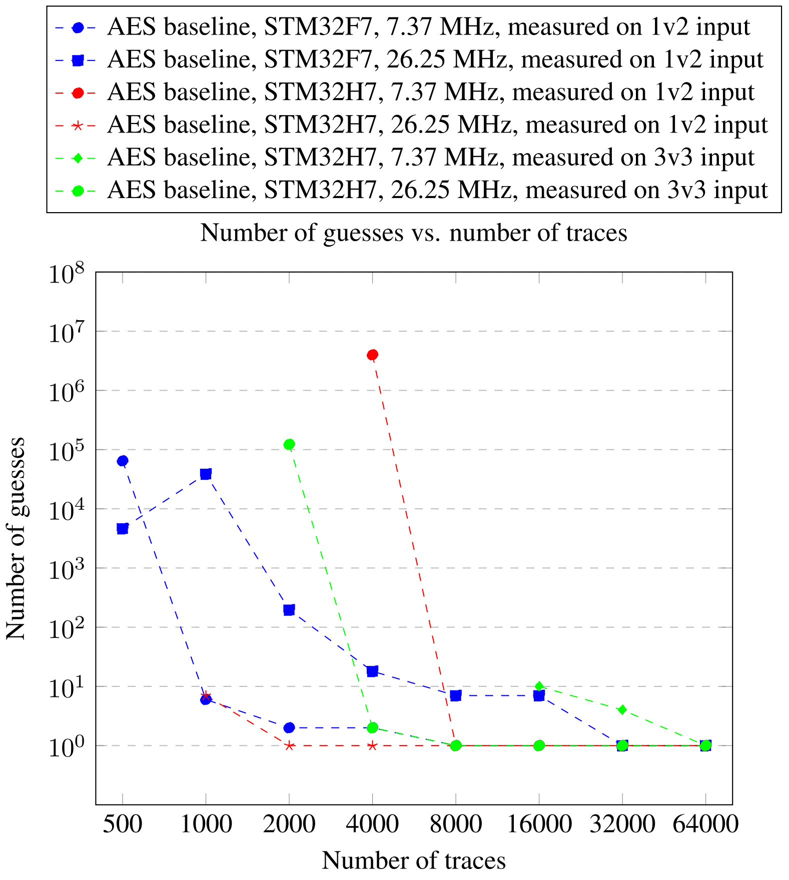


Figure 8.5: Plot showing how different clock frequencies impact number of guesses vs. traces for STM32H7.

8.7 Results summary

Table 8.1 presents the minimum number of traces needed to extract full AES key for the different tests done in this thesis. The relative improvement is compared against the AES baseline implementation running on STM32F7 measured on 1v2 input. Equation 8.1 shows how the improvement is calculated where A is the number of traces needed for STM32F7 baseline measured on 1v2 input and B is the number of traces needed for the specified test.

$$Improvement = \frac{B - A}{A} * 100 \qquad [\%] \qquad (8.1)$$

Table 8.1: Table presenting minimum number of traces needed in order to extract full AES key for different hypothesis and measurement setups. Improvement is relative to the STM32F7 AES baseline trace set measured on 1v2 input.

Target	**Hypothesis / Measurement setup**	**Number of traces**	**Improvement**
STM32F7	baseline measured on 1v2 input, 26.25 MHz clock	500	0%
STM32H7	baseline measured on 1v2 input, 26.25 MHz clock	1000	100%
STM32H7	baseline measured on 3v3 input, 26.25 MHz clock	2000	300%
STM32H7	baseline measured on 1v2 input	4000	700%
STM32H7	baseline measured on internal regulator decoupling pin	8000	1500%
STM32H7	code executed from RAM, measured on 1v2 input	8000	1500%
STM32H7	code executed from RAM, measured on 3v3 input	16000	3100%
STM32H7	added dummy AES rounds, measured on 1v2 input	16000	3100%
STM32H7	baseline measured on 3v3 input	16000	3100%

Chapter 9

Discussion

Side channel attacks are cheap, easy and effective attacks against MCUs. These attacks are used to extract sensitive information using side channels from a given application. A typical and very important application is encryption / decryption. Almost all data shared between units in insecure environments must be encrypted. When encryption keys are extracted from a system, the data from the system is compromised and the system does not fulfil is purpose. In recent years there have been a lot of effort put into understanding how different side channel attacks work, and how to achieve a better resilience against them. Today there are still many vulnerabilities when using standard encryption software on standard MCUs. The purpose of this thesis is to achieve a larger understanding of how to better protect against power analysis, a subset of side channel attacks.

9.1 Experimental work

In this thesis side channel resilience using different techniques are measured and quantified. In order to do this a methodology that enables the researcher to find and quantify different hypotheses, that could be interesting in light of side channel resilience, is developed. Much laboratory work has been done in order to test and verify different hypotheses, and to see the effects of how different counter measures affect the number of traces needed in order to fully extract AES encryption keys. The important findings of the experimental work in this study are the not the exact number of traces needed for extracting used keys, but the understanding of how different counter measures influence the number. The testing is done using modern, high end MCUs suited for IoT applications. The testing of the MCUs are done through using the ChipWhisperer measurement system. Two MCUs have been used in the testing, the STM32F7 and the STM32H7. These MCUs have very similar architecture, e.g. both use the Arm Cortex M7 core, but have different process nodes.

The testing is done through measuring the different MCUs' power consumption while encryption a message. Different, randomly generated messages are encrypted, while always using the same encryption key. The goal is to find how many traces needs to be captured in order to fully extract the used key in analysis afterwards. The results presented in chapter 8 shows that, for the tests executed in this thesis, if enough traces are captured it is possible to extract the used key. Two most interesting results are how different clock frequencies and different process nodes influence the useful leakage.

9.1.1 Different clock frequencies

For the 40nm MCU, the STM32H7, higher clock frequency gives a better measurement. Traces were captured with clock frequencies of 7.37 MHz and 26.25 MHz. The lower frequency is used because it is the default clock frequency in the ChipWhisperer setup, and the higher frequency is used because it is the highest clock frequency available for the ChipWhisperer Pro while still capturing four samples per clock cycle Only 1000 traces are needed in order to extract key used when having a clock frequency of 26.25 MHz compared to 4000 traces when having a clock frequency of 7.37 MHz. There could be several reasons to this phenomena. A simple definition of dynamic power is shown equation 9.1, where C is the total capacitance, f is the operation frequency and V is the voltage.

$$p = C * f * V^2 \tag{9.1}$$

This shows that there is a linear relationship between operation frequency and power usage. This means that a MCU will consume more power when a higher frequency is used. The increase in the total power consumption is caused by a a larger portion of dynamic power. The dynamic power consumption is the interesting power for an attacker. Thus a higher frequency will give a better SNR, as more of the total power consumption is caused by the dynamic power.

Most likely when measuring with slower frequencies, internal capacitance could be able to fully or at least partly discharge between the different samples. This makes the captured traces more saw-tooth like instead of being smooth. Figure 7.2 (7.37MHz) and Figure 7.1 (26.25MHz) have clear differences where it is possible to see that the trace for 26.25MHz is much smoother than the trace for 7.37MHz which is more saw tooth like. Now the MCU, due to the slow clock frequency, function as a filter. The MCU is filtering out the leakage information useful for power analysis attacks. Another reason could be that since each clock cycle is longer the static power consumption between each sample increases. This could influence how much the dynamic power consumption will affect the captured samples.

9.1.2 Different process nodes

The effect of moving from a MCU with 90nm process node to a MCU with 40nm process node gives a huge improvement in the number of traces needed in order to execute the full key. In the experimental work, MCUs with very similar underlying architecture are used. The improvement given the setup and test scheme in this thesis is as much as 700%, moving from 500 to 4000 traces, when using equal firmware and equal clock frequencies of 7.37 MHz. This alone is a huge improvement in power analysis resilience. When testing the MCUs with a clock frequency of 26.25 MHz, only 100 % improvement is achieved, moving from 500 to 1000 traces. It is still a significant increase in number of traces.

9.1.3 Other results

It is clear from chapter 8 that it as harder to find enough correlations to extract the key used when the SNR is worsened. It is a relationship between measuring on 1v2 input compared to 3v3 input on the STM32H7. The number of traces needed are increased from 4000 to 16000 traces. The same occurs when executing the encryption code from RAM. Due to a lower dynamic power consumption, RAM being more power efficient than flash, a less useful leakage is produced.

Dis-aligning the traces with adding dummy AES rounds is also an effective counter measure. The number of traces needed are also increased from 4000 to 16000 traces on the STM32H7. This is an all firmware based counter measure. This is easy to implement and execute, hence there is no need for e.g. a larger RAM. This will however make the encryption have a longer execution time than the standard encryption algorithm.

9.2 Limitations of this study

The result from this thesis should only be used a guideline to see how different counter measures influence the needed number of traces. The results are not to be used as a guarantee or standard to compete against. Only two MCUs are tested; the trends from the results could be specific for these two MCUs. There are no guarantees that the results are valid for other MCUs. However the hypotheses the results are based upon are general and not specific for the tested MCUs, besides the comparison of different process nodes. The results shows that when SNR is worsened the number of traces needed clearly increases. The hypotheses tested regarding SNR will therefore likely hold for other MCUs as well.

Chapter 10

Conclusion

10.1 Main results

The main result from this thesis, besides the technical details, is that the proof of concept works, even on the most recent and modern MCUs suited for IoT applications. The attack used works with different AES firmwares and with different measurement setups. The main contribution is the methodology developed in order to quantify how different counter measures affect the resilience against power analysis. This methodology enables the researcher to try out different counter measures and easily see their effects. Through the understanding of how the different counter measures tested in this thesis affect the useful leakages obtained through measuring, a greater understanding of how to increase power analysis resilience is achieved. The effects of this methodology has lead to interesting results. The two most fascinating and important results from the experimental work in this thesis are how MCU clock frequency influence the useful power leakage, and the huge improvement in number of traces needed in order to fully extract the AES encryption key when using a MCU with 40nm process node compared to a MCU 90nm process node. Otherwise the results achieved are as expected. A general conclusion could be that it is harder to extract full AES keys when there are more activity on the target MCU or when the power from the digital core is not measured isolated. A worse SNR gives a higher number of traces needed in order to extract the used key.

10.2 Future work

Future work will be to complete testing of the hypotheses list. It is especially interesting to look into how different clock frequencies impact the useful information within the traces. Different shunt resistors could have an impact, especially when measuring on internal regulator decoupling, thus testing with different shunt resistor values should be done. In this thesis, the same encryption key is used in every test. In order to improve the results more (random) encryption keys should be used to find if there are any correlations between key and number of traces needed. Maybe some keys are more resilient than others.

Bibliography

[1] Jlsca - github page. https://github.com/Riscure/Jlsca, 2018. [Online; accessed 30-May-2017].

[2] Jlsca dpa tools benchmarking - github page. https://github.com/ikizhvatov/dpa-tools-benchmarking, 2018. [Online; accessed 30-May-2017].

[3] National institute of standard and technology. https://www.nist.gov/, 2018. [Online; accessed 23-May-20178.

[4] Wikipedia - rijndael's key schedule. https://en.wikipedia.org/wiki/Rijndael_key_schedule, 2018. [Online; accessed 23-May-20178.

[5] Atmel. Datasheet atmel avr 8-bit atxmega 128d4. 2016.

[6] 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 Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[7] Chipwhisperer lite (cw1173) two-part version. http://store.newae.com/chipwhisperer-lite-cw1173-two-part-version/, 2018. [Online; accessed 21-February-2018].

[8] Chipwhisperer-pro (cw1200) complete level 3 starter kit. http://store.newae.com/chipwhisperer-pro-complete-level-3-starter-kit/, 2018. [Online; accessed 21-February-2018].

[9] Federal Information Processing Standards (FIPS). 140-2 - security requirements for cryptographic modules. 2001.

[10] Forbrukerraadet - elendig sikkerhet i gps-klokker for barn. `https://www.forbrukerradet.no/siste-nytt/elendig-sikkerhet-i-smartklokker-for-barn`, 2017. [Online; accessed 14-Decemeber-2017].

[11] IoT Security Foundation. Iot security compliance framework, release 1.0. 2016.

[12] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[13] Paul C. Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *J. Cryptographic Engineering*, 1(1):5–27, 2011.

[14] Thanh-Ha Le, Jessy Clédière, Cécile Canovas, Bruno Robisson, Christine Servière, and Jean-Louis Lacoume. A proposition for correlation power analysis enhancement. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2006.

[15] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, pages 157–171, 2005.

[16] Arm mbed tls. `https://tls.mbed.org/`, 2017. [Online; accessed 26-February-2018].

[17] Newae technology. `https://wiki.newae.com/Main_Page`, 2017. [Online; accessed 19-Decemeber-2017].

[18] Newae technology correlation power analysis. `http://wiki.newae.com/Correlation_Power_Analysis`, 2018. [Online; accessed 26-February-2018].

[19] Newae technology correlation power analysis tutorial. `https://wiki.newae.com/Tutorial_B6_Breaking_AES_(Manual_CPA_Attack)`, 2018. [Online; accessed 20-April-2018].

[20] Newae technology cw308 ufo target. `https://wiki.newae.com/CW308_UFO_Target`, 2018. [Online; accessed 9-May-2018].

[21] Jing Pan, Jasper G. J. van Woudenberg, Jerry den Hartog, and Marc F. Witteman. Improving DPA by peak distribution analysis. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*, pages 241–261. Springer, 2010.

[22] The register - iot worm can hack philips hue lightbulbs, spread across cities. https://www.theregister.co.uk/2016/11/10/iot_worm_can_hack_philips_hue_lightbulbs_spread_across_cities/, 2017. [Online; accessed 14-Decemeber-2017].

Appendix

MBED TLS, AES-128, standard implementation

Code listing 10.1 shows the MBED TLS's AES implementation.

Listing 10.1: C code which shows the MBED TLS AES implementation.

```c
int mbedtls_internal_aes_encrypt( mbedtls_aes_context *ctx,
                                  const unsigned char input[16],
                                  unsigned char output[16] )
{
    int i;
    uint32_t *RK, X0, X1, X2, X3, Y0, Y1, Y2, Y3;

    RK = ctx->rk;

    GET_UINT32_LE( X0, input,  0 ); X0 ^= *RK++;
    GET_UINT32_LE( X1, input,  4 ); X1 ^= *RK++;
    GET_UINT32_LE( X2, input,  8 ); X2 ^= *RK++;
    GET_UINT32_LE( X3, input, 12 ); X3 ^= *RK++;

    for( i = ( ctx->nr >> 1 ) - 1; i > 0; i-- )
    {
        AES_FROUND( Y0, Y1, Y2, Y3, X0, X1, X2, X3 );
        AES_FROUND( X0, X1, X2, X3, Y0, Y1, Y2, Y3 );
    }

    AES_FROUND( Y0, Y1, Y2, Y3, X0, X1, X2, X3 );

    X0 = *RK++ ^ \
            ( (uint32_t) FSb[ ( Y0       ) & 0xFF ]       ) ^
            ( (uint32_t) FSb[ ( Y1 >>  8 ) & 0xFF ] <<  8 ) ^
            ( (uint32_t) FSb[ ( Y2 >> 16 ) & 0xFF ] << 16 ) ^
            ( (uint32_t) FSb[ ( Y3 >> 24 ) & 0xFF ] << 24 );

    X1 = *RK++ ^ \
            ( (uint32_t) FSb[ ( Y1       ) & 0xFF ]       ) ^
            ( (uint32_t) FSb[ ( Y2 >>  8 ) & 0xFF ] <<  8 ) ^
            ( (uint32_t) FSb[ ( Y3 >> 16 ) & 0xFF ] << 16 ) ^
            ( (uint32_t) FSb[ ( Y0 >> 24 ) & 0xFF ] << 24 );

    X2 = *RK++ ^ \
            ( (uint32_t) FSb[ ( Y2       ) & 0xFF ]       ) ^
            ( (uint32_t) FSb[ ( Y3 >>  8 ) & 0xFF ] <<  8 ) ^
            ( (uint32_t) FSb[ ( Y0 >> 16 ) & 0xFF ] << 16 ) ^
            ( (uint32_t) FSb[ ( Y1 >> 24 ) & 0xFF ] << 24 );

    X3 = *RK++ ^ \
                ( (uint32_t) FSb[ ( Y3       ) & 0xFF ]       ) ^
                ( (uint32_t) FSb[ ( Y0 >>  8 ) & 0xFF ] <<  8 ) ^
                ( (uint32_t) FSb[ ( Y1 >> 16 ) & 0xFF ] << 16 ) ^
                ( (uint32_t) FSb[ ( Y2 >> 24 ) & 0xFF ] << 24 );

    PUT_UINT32_LE( X0, output,  0 );
    PUT_UINT32_LE( X1, output,  4 );
    PUT_UINT32_LE( X2, output,  8 );
    PUT_UINT32_LE( X3, output, 12 );

    return( 0 );
}
```

Linker script, MBED TLS, AES-128, executed from RAM

Code listing 10.2 shows relevant changes made in the linker script. The new .user section added starting in line 19 will force all functions whose name starts with aes or mbed to be place in RAM. When programming the MCU. the code will be written to flash as normal. During the MCU's start-up, it writes the specified code into to RAM.

Listing 10.2: Linker script which shows changes added to linker script to let all encryption code be executed from RAM.

```
1  /* Memories definition */
2  MEMORY
3  {
4    RAM (xrw)    : ORIGIN = 0x20000000, LENGTH = 320K
5    ROM (rx)     : ORIGIN = 0x8000000, LENGTH = 512K
6  }
7
8  /* Sections */
9  SECTIONS
10 {
11   /* The startup code into ROM memory */
12   .isr_vector :
13   {
14     . = ALIGN(4);
15     KEEP(*(.isr_vector)) /* Startup code */
16     . = ALIGN(4);
17   } >ROM
18
19   .user :
20   {
21     . = ALIGN(4);
22     *(.text.aes*)
23     *(.text.mbed*)
24   } >RAM
25
26   /* The program code and other data into ROM memory */
27   .text :
28   {
29     . = ALIGN(4);
30     *(.text)           /* .text sections (code) */
```

```
    *(.text*)           /* .text* sections (code) */
    *(.glue_7)          /* glue arm to thumb code */
    *(.glue_7t)         /* glue thumb to arm code */
    *(.eh_frame)

    KEEP (*(.init))
    KEEP (*(.fini))

    . = ALIGN(4);
    _etext = .;         /* define a global symbols at end of code */
  } >ROM


  /* Constant data into ROM memory*/
  .rodata :
  {
    . = ALIGN(4);
    *(.rodata)          /* .rodata sections (constants, strings, etc.) */
    *(.rodata*)         /* .rodata* sections (constants, strings, etc.) */
    . = ALIGN(4);
  } >RAM
```

Brute force sorting algorithm

Listing 10.3: Implementation of the sub-optimal sorting algorithm used for brute force attacks based on CPA output.

```
print "Initialise g[] with top candidates from attack"
g = []
index_list = []
g.append([])
index_list.append([])

# Initialise g with best candidates
for i in range(0, 16):
  g[0].append(dataSet[i][0].candidate)
  index_list[0].append(dataSet[i][0].rank)
guessCount = 1

if g[0] == knownKey:
  print "Found correct key, guess count: ", guessCount
  print "######################"
  exit()

for guess in range(0, 16*255): #  #subBytes x #guesses
  # Create new object
  nextBestCorrelationValue = CorrelationValue()
  # Initialise to zero
  nextBestCorrelationValue.relativePeak = 0
```

```
# Iterate over every subKey
for subByte in range(0, 16):
  for guessRank in range(1, len(dataSet[subByte])):
    # Find the next candidate with the largest relative
    # correlation which is not already used
    if dataSet[subByte][guessRank].relativePeak >  \
    nextBestCorrelationValue.relativePeak and \
    dataSet[subByte][guessRank] not in G:
      nextBestCorrelationValue = dataSet[subByte][guessRank]
print "Next Best Correlation Value: ",
printCorrelationValue(nextBestCorrelationValue)
print "Currently guesses in g[]: ", len(g)
print "Guess positions: ", index_list[0]
# Append found candidate to G
G.append(nextBestCorrelationValue)
dataSet[nextBestCorrelationValue.subKey] \
.remove(nextBestCorrelationValue)
# pop from dataSet when element is added to G

# Iterate over all previous guesses and swap the new candidate
# with its corresponding best guess
for fullKeyGuessNumber in range(len(g)):
  if g[fullKeyGuessNumber][nextBestCorrelationValue.subKey] == \
  dataSet[nextBestCorrelationValue.subKey][0].candidate:
    newGuess = copy.deepcopy(g[fullKeyGuessNumber])
    newGuess[nextBestCorrelationValue.subKey] = \
    nextBestCorrelationValue.candidate
    g.append(newGuess)
    newGuessIndex = copy.deepcopy(index_list[0])
    newGuessIndex[nextBestCorrelationValue.subKey] = \
    nextBestCorrelationValue.rank
    index_list[0] = newGuessIndex

    guessCount += 1

    if newGuess == knownKey:
      print "Found correct key, guess count: ", guessCount
      print "######################"
      exit()

    if guessCount == 100000000:
      print "Out of range, no key found"
      print "######################"
      exit()
```