Universiteit Leiden
The Netherlands

Computer Science

Exploration of the ChipWhisperer Lite ARM board for education on Side-Channel Power Analysis

Gijs Burghoorn

Supervisors:
N. Mentens & F. Hermans

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

15/06/2021

Abstract

Power analysis provides a way to break confidentiality of electronic devices by precisely inspecting their power usage. The ChipWhisperer framework provides a cleaner and more accessible way to perform power analysis, which could make it suited to teach power analysis methodology to those inexperienced in the field. An evaluation of the ChipWhisperer framework's tutorials and documentation is done to determine its capabilities and shortcomings for beginners. A short-course for Master students is created to more clearly assess the needed materials and create a centralized resource for beginners.

Contents

1 Introduction

Electronic devices have grown to become a major part of daily life and the concern for the security and privacy practices these devices utilize has grown accordingly. A major part of security and privacy is confidentiality: being able to keep certain information a secret. Many techniques which are used in breaking confidentiality attempt to construe information from the usual input or output of a device. This can be by fishing someone's password by email or by sending malicious input data to a website to expose user-data. There are also many unconventional inputs and outputs of these devices. These include the power a device uses, the sounds a device makes and the electromagnetic field a device generates. The analysis involved in breaking certain security and privacy principles using these unconventional side channels is appropriately called *side-channel analysis (SCA)*.
A form of side-channel analysis which is especially interesting and potent is *power analysis*. Power analysis studies the amount of power an electronic device uses over time, its so called *power traces*. One can imagine that an electronic device sitting idle uses less power than that same device performing calculations at the highest computational intensity. Similarly, one can measure minute differences in power usage between different calculations or input data used. This is especially interesting when measuring the power usage for computer algorithms which rely upon confidentiality. Therefore, the focus often falls on cryptography and specifically encryption algorithms. These algorithms enable a major part of the confidentiality most electronic devices rely on. Examples of where these algorithms are used are in chat applications or whilst accessing bank accounts online. Studying cryptography algorithms — to tell how well these resist power analysis or whether algorithm implementations can be more resistant to these attacks — has become a topic of interest in security and privacy research. One platform for testing algorithm implementations or for preparing more advanced attacks on electronic devices is the *ChipWhisperer framework*. This framework provides many of the tools to make power analysis simpler and more accessible. This, however, leads to the need for evaluation of how well suited this framework is for people who are inexperienced in the power analysis field. Furthermore, if there are clear gaps in the resources provided by the ChipWhisperer framework, they can be attempted to be filled in.
This paper attempts to create a short-course for Master students in the field of computer science. The course can be seen as a gateway to assess the state of the ChipWhisperer and power analysis documentation and tutorials for beginners. More specifically, this work and the course will focus on a specific electronic device provided by the ChipWhisperer framework called the ChipWhisperer Lite ARM board. Furthermore, this course should then also provide a centralized resource for inexperienced students to get started with the power analysis field. Therefore, this work questions the following *"What are the current limitations of the information surrounding the ChipWhisperer Lite ARM board"* and *"How can the information surrounding the ChipWhisperer framework be improved to create an introductory Master of Science course on power analysis in hardware security?"*.

2 Background & Related Work

Within this section, necessary background information is addressed to create a better understanding of the problem stated and the methods used. Furthermore, this chapter will also cover some related work on which this work is based. First, this chapter go into the cryptography algorithms often used as a target in power analysis attacks. Then, this chapter will cover the fundamentals of side-channel analysis and power analysis. Afterwards, the ChipWhisperer framework will be covered in more depth. Lastly, semantic waves will be covered in a section on pedagogy and education in relation to the short-course created.

2.1 Cryptography

Cryptography has been used for millennia to ensure the protection of information whilst in transit or storage [1]. The internet has produced the need for faster and more secure encryption standards. Two common standards used today are the *Advanced Encryption Standard (AES)* [2] and the encryption algorithm designed by *Ron Rivest, Adi Shamir and Len Adleman (RSA)* [3], which both represent two different strategies and requirements for encryption. Namely, these can also be categorized as symmetric and asymmetric ciphers, respectively. A schematic overview of symmetric and asymmetric encryption can be found in Figure 1.

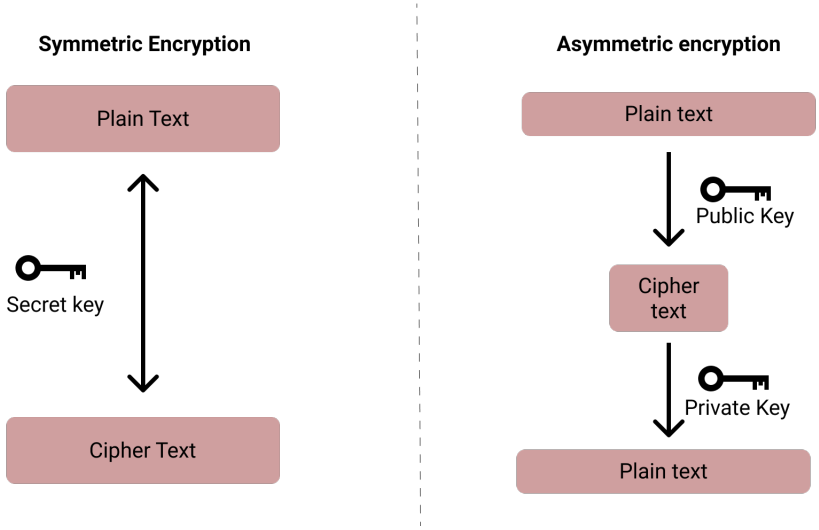


Figure 1: A schematic overview of symmetric and asymmetric encryption.

Symmetric encryption uses the same key for encryption and decryption similar to how opening and closing of a lock would function in the real world. The current standard for symmetric encryption is AES [2]. If an encrypted message needs to be sent from one device to another over a public channel, they would first both have to agree on a secret key to use. Therefore, the secret key needs to be agreed upon over a secure channel or through using asymmetric encryption over a public channel. A commonly used algorithm to agree upon a secret key over a public channel is the Diffie-Hellman key-exchange algorithm [4].
As can be seen from symmetric encryption, secretly agreeing on a secret key can cause difficulty. *Asymmetric ciphers* have a solution to this problem by having two keys: a public and a private key.

The public key is publicly available to every sender that wants to encrypt a message before sending it to a receiver. The receiver holds the corresponding private key with which the decryption of that message can be done. One commonly used standard today for asymmetric ciphers is RSA [3].
With some asymmetric encryption algorithms, the private key can also be used to verify the identity of the holder of the private key by using *digital signatures* [5]. The process can be divided into the signing and the verification part as is shown in Figure 2.
A digital signature can be signed or created in three steps. First, the data is summarized to a fixed data length. This summarization process is called hashing [5]. Then using the private key, the hash is encrypted. Finally, this encrypted hash (referred to as a signature) is combined with a digital certificate to create the digital signature. Centralized certificate authorities manage these digital certificates. A digital signature can be verified by first verifying the digital certificate and then comparing the hash of the predetermined data to the signature after it is decrypted with the public key.

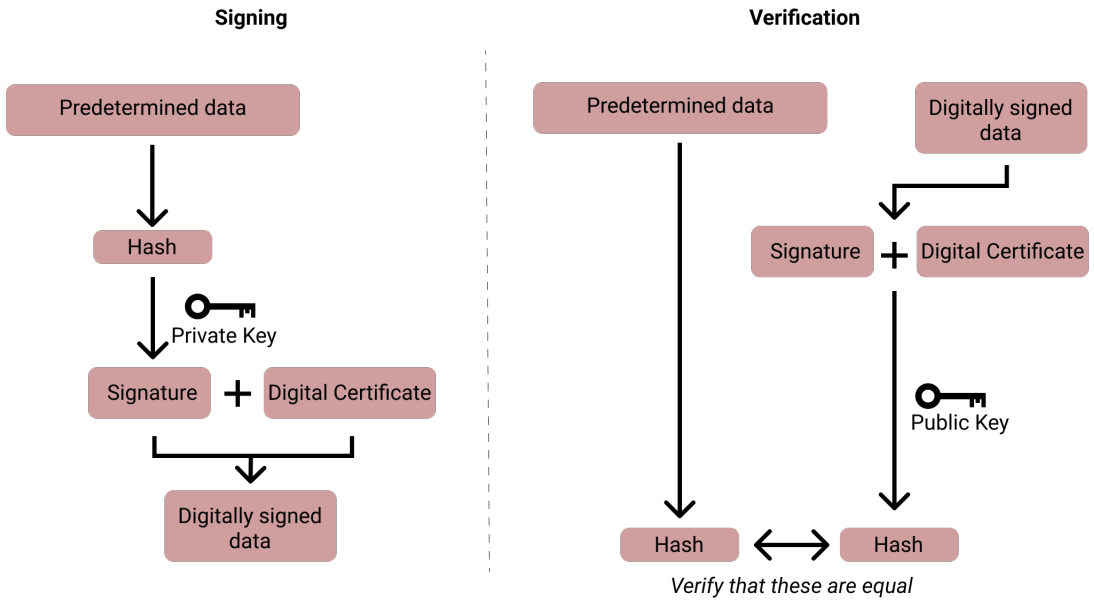


Figure 2: A schematic overview of the signing and verification process of digital signatures.

In essence, symmetric encryption is mostly used for one-to-one encryption — as may apply to chat applications — and asymmetric encryption is mostly used for many-to-one encryption — as may apply when requesting a website from a web-server. This is generally done since symmetric encryption will expose all information to all holders of the secret key. This increases the attack surface for the number of people participating and generally disallows for multiple levels of confidentiality. Furthermore, the performing numerous key-exchanges is resource intensive. Performing asymmetric encryption, on the other hand, is generally slower and requires larger keys.

2.2 Side-channel Power Analysis

For most embedded hardware there is some form of input and output. Inputs of electronics range from buttons and environment sensors to mice and touchscreens. Outputs range from lights and displays to speakers. Beyond the intended inputs and outputs, however, there are also some inadvertent interfaces. Some examples of these unintended in- and outputs are the electromagnetic field generated by currents, the sound produced by electronics or their peripherals (such as keyboards or mice), or the power used by electronics. The latter will be the main point of interest in this work.

There are many possible avenues that attacks on encryption standards can take. Since implementations of encryption standards are done on multiple different levels — including the mathematical specification, software implementation, and hardware operations [6] — there are also multiple levels to attack. There are many examples of attacks to all these individual levels in the encryption implementation stack. Cipher specifications are often tested using differential cryptanalysis [7], linear cryptanalysis [8], or frequency analysis [9]. Software implementations are often broken using misused memory pointers [10]. Within hardware, speculative execution is exploited which could lead to elevation of privileges and exposure of private keys [11].

Whilst many of these areas are often exploited individually, some issues happen on the conjunction of these different levels [6]. Exploiting an implementation's or a specification's vulnerabilities with power analysis requires combining knowledge of these three areas. Power analysis looks at the power consumed by an electronic device to make statements about what data were used or which algorithm was executed.

2.2.1 Leakage Models

Microprocessor-controlled devices store data within components such as registers, caches, or memory buses, and share data between such components [12]. Storing data in memory uses small *capacitors* which represent the value of the data by the charge on the capacitor and consequently the voltage over the capacitor. The value stored in an individual capacitor — also referred to as a bit — can be represented using the commonly used *on/off model* with the 0 for a low voltage and 1 for a high voltage. Both the exchanging and the storing of data containing multiple bits can be modeled as bigger capacitors. This is done by viewing a capacitor which holds an amount of charge equal to the sum of the charges hold by the individual capacitors representing those bits [12].

The power used by electronic devices over time can be measured using specialized equipment such as *multimeters* or *oscilloscopes*. Oscilloscopes are often used while performing analysis on such a power measurement also called a *power trace* because they allow for a more detailed waveform analysis. Most modern oscilloscopes also allow for digital storage of the measured power traces. After the traces are stored, more detailed analysis can be done on the power traces.

To properly model the power consumption of capacitors and extract secret information from power traces, there are two commonly used models: the *Hamming-Weight* and *Hamming-Distance* models [12]. These models are also called leakage models. The Hamming-Weight model states that there should be a correlation between the number of capacitors in the 1 state and the power consumption of a power trace. The Hamming-Distance model states that there should be a correlation between the number of capacitors switching between states and the power consumption within a power trace. This Hamming-Distance model is primarily of use for modeling hardware implementations [12] of encryption algorithms since hardware registers are often updated to a fixed value

between the 0 state and the 1 state before being written to [12]. Contrarily, the Hamming-Weight model is more commonly used for software implementations.

2.2.2 Simple Power Analysis

Simple Power Analysis (SPA) aims to extract secret information from a single power trace. An easy-to-understand example is SPA on the RSA algorithm. Within implementations of the asymmetric encryption algorithm RSA [3], there are common uses of a memory-efficient version of the modular exponentiation algorithm. Within this algorithm — of which Python code can be found in Figure 3 along with a visual overview in Figure 4 — every bit is inspected one-by-one and depending on whether it is in the 0 or 1 state an extra multiplication is done. Here there is a one-to-one relation (bijection) between the type of instructions which are executed and the possible private keys. Since we can often recognize the instructions done by purely looking at the power used by a microprocessor, it could be possible to infer information about the private key by looking at a power trace of decryption being performed using those private keys [6].

```
# Custom implementation of pow(x, y) % p
# With p >= 2
def custom_pow_mod(x, y, p):
    res = 1

    # Until we have reached the highest power
    while (y > 0):
        # If the last byte is a one
        if (y & 0x01):
            res *= x
            res %= p # Make sure we stay modulo p

        # Move on to the next byte
        y >>= 1
        x *= x
        x %= p # Make sure we stay modulo p

    return res
```

Figure 3: A Python implementation of the memory-efficient modular exponentiation.

For example, if we use the algorithm from Figure 3 as it is used in the decryption process of RSA. The value of y loosely corresponds with the private key. If, for the example, y is assumed to equal binary 10011101. Please note that this is not a normal value for y. Apart from y being usually much greater in value, there are other requirements for the value of y. In the power trace, there would be eight distinct spikes. Each being either a longer spike or a shorter spike. The longer spikes correspond to the extra multiplications — thus 1's — and the shorter spikes correspond to skipping this multiplication — thus 0's. The order that would be seen is: *long, short, long, long, long, short, short, long*. This can also be seen in Figure 5. As can be noticed, the power trace

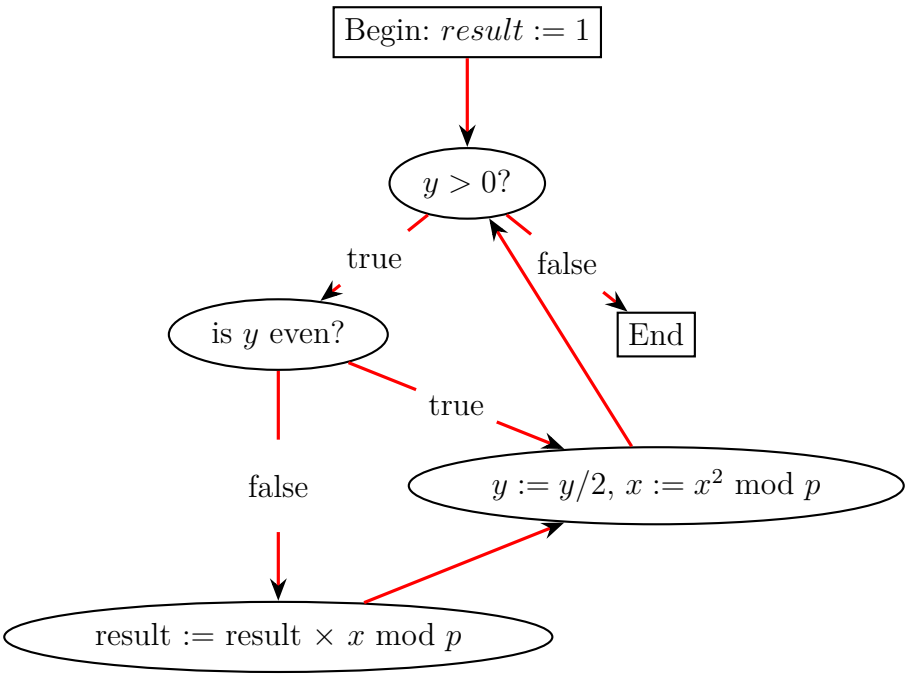


Figure 4: A visual overview of the modular exponentiation algorithm shown in Figure 3.

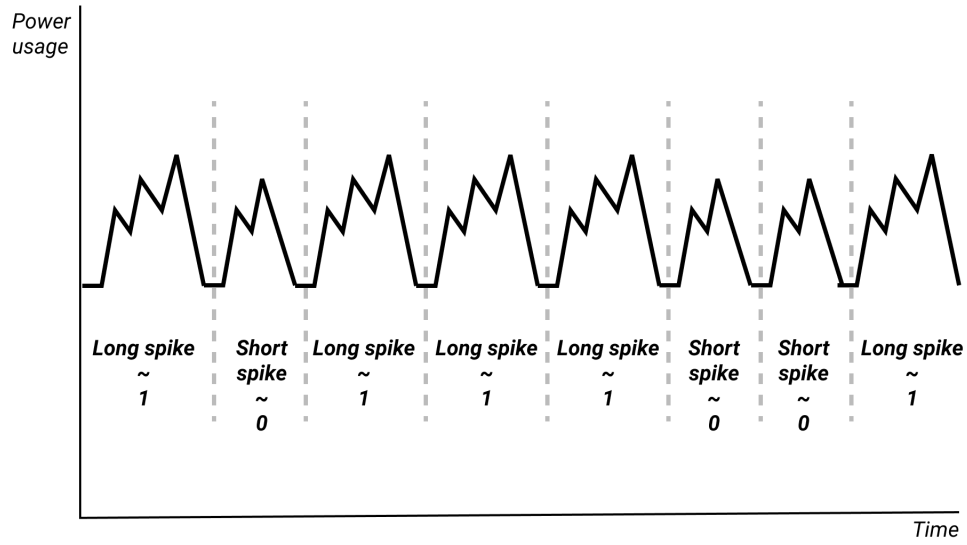


Figure 5: An idealized overview of a power trace of modular exponentiation using an exponent of binary 10011101.

exposes the value of y by its power spikes corresponding to the reverse of y's value.
SPA requires the type and/or the number of instructions executed on a microprocessor by an algorithm to be dependent upon (part of) the data used [6]. This is the case with RSA [3] as demonstrated in the previous paragraph.

2.2.3 Correlation Power Analysis

While Simple Power Analysis solely works on algorithms that have a different set of instructions depending on the input data, *Differential Power Analysis (DPA)* [6] works with the aforementioned leakage models to combine more than one power trace in order to extract secret information. Within DPA there is a statistical comparison between the hypothetical power consumption of our leakage model and the actual power consumption of the device.
One way to determine whether a model and the real-world power trace are correlated is using *Pearson Correlation Coefficients* [13]. In which case, we are talking about a subset of DPA called *Correlation Power Analysis (CPA)*. The formula for Pearson Correlation Coefficient takes two finite sets or two functions and returns a value between -1 and 1: -1 meaning a perfect inverse correlation, 0 meaning no correlation at all, and 1 meaning a perfect correlation. When comparing the leakage model to a power trace there is an indirect search for the location within each trace where the hypothesized leakage model has an (inverse) correlation with the trace at that point in time.
When trying to optimize the search for correlations — meaning the correlation to be as far from 0 as possible whilst requiring the least amount of traces — it helps to synchronize the different traces. Synchronization meaning to match up specific parts of computations with each other in power traces. Since microprocessors have some instability in the frequency of internal clocks, there may be jitters in power traces [14]. There are several techniques to resynchronize power traces including the commonly used *Sum of Absolute Difference (SAD)*.

2.3 ChipWhisperer framework

Early on in embedded security research there was a realization that comparing side-channel attacks between different platforms was difficult [15]. Therefore, there were several attempts at creating standardized targets, software, and educational resources [15]. ChipWhisperer [12] is a framework for finding vulnerabilities of embedded systems based on side-channel and fault analysis. Fault analysis is a different attack vector which is not covered this paper. ChipWhisperer creates a fully open-source [15] platform for performing Side-Channel attacks which includes all parts of such an attack: hardware and software for both target and capture devices [15].
Since its initial creation, ChipWhisperer has created more than just the initial platform based on an FPGA [15]. The framework now also contains targets based on the STM32 ARM Cortex MCUs and AVR XMega. All of these platforms are commonly used within hardware security. Furthermore, capture devices can be used in combination with other targets already available in measurement labs [15].
Apart from the hardware the ChipWhisperer framework provides, the framework also provides analysis software [15]. This analysis software is used to perform reproducible and thus shareable side-channel attacks. The software also provides ways to perform the previously mentioned synchronization.

2.4 Semantic waves

This work will involve creation of course material for students. Therefore, it is of value to discuss pedagogy and a concept called semantic waves, which is a part of the study of meta learning. In

short, semantic waves provide a structure for your language-use and explanation-methods. This provides guidelines on when to use technical or everyday language and when to use examples or talk more generally or abstractly. Semantic waves along with its sub-concepts called semantic gravity and semantic density are explained in more detail in the following paragraphs.
It impacts students positively when educators have "pedagogic practice" [16] — meaning having experience in meta learning —, even when educators from one field study meta learning from the perspective of other fields [16]. In short, when educators generally learn about meta learning, it helps students learn. This fact leads to the need to talk about education in more abstract terms. Mainly, the need for terms to describe the relationship between meaning and context in an educational setting independent of the field of study [16]. Put into simpler words: the relationship between how much technical knowledge is required and how abstract a concept can be explained.
Semantic gravity refers to how close an explanation is to the real world [16]. High semantic gravity (SG+) indicates a description being closer to common real-world phenomena and low semantic gravity (SG-) indicating more abstract in nature. In Table 1 a few examples of comparisons of SG- versus SG+ can be seen.

Semantic gravity level	Example 1	Example 2	Example 3
Low semantic gravity / SG-	$x \times y = \sum_{i=1}^{y} x$	A lack of concern for online safety can lead to your device obtaining computer viruses	Swimming is all about being calm in the water
High semantic gravity / SG+	Having 3 nets of 5 oranges. How many oranges does one have?	If you recklessly open files you downloaded from the internet, hackers may be able to look at your webcam without your permission	If one wants to swim better, they need to learn to be comfortable in the water.

Table 1: Examples illustrating the difference between low and high semantic gravity.

Semantic density refers to the amount of technical language used [16]. High semantic density (SD+) indicates a high amount of jargon used and low semantic density (SD-) implies everyday language. In Table 2 a few examples of comparisons of SD- versus SD+ can be seen.
Downward semantic shifts go from SG-/SD+ to SG+/SD-. Namely, going from technical terms in which explanation remain in the abstract to everyday terms in which explanations remain close to real world situations. This direction for a semantic shift is also called *unpacking*. Similarly, going in the reverse direction is called *repacking* [16].
When analyzing classroom practice, there are often several downward semantic shifts in succession [16]. This can be seen in Figure 6. While the "down escalator" profile [16] — involving these repeated downward shifts — is commonly used, it fails to recontextualize the *unpacked* knowledge: failing to *repack* knowledge. This can be solved by appending "upward semantic shifts" [16] to each *unpacking* curve. The approach of smoothly appending *repacking* actions after *unpacking* is

Semantic gravity level	Example 1	Example 2	Example 3
Low semantic density / SD-	When you are sleepy, you want to sleep	Obtaining higher grades leads to higher chances at getting into college	Using more examples and real-world analogies in your explanations may make children understand you better
High semantic density / SD+	When one experiences somnolence, one feels the need to suspend conciousness	Obtaining a higher GPA leads to higher chances at getting into college	Statements of lower semantic gravity may be more understandable for younger generations

Table 2: Examples illustrating the difference between low and high semantic density.

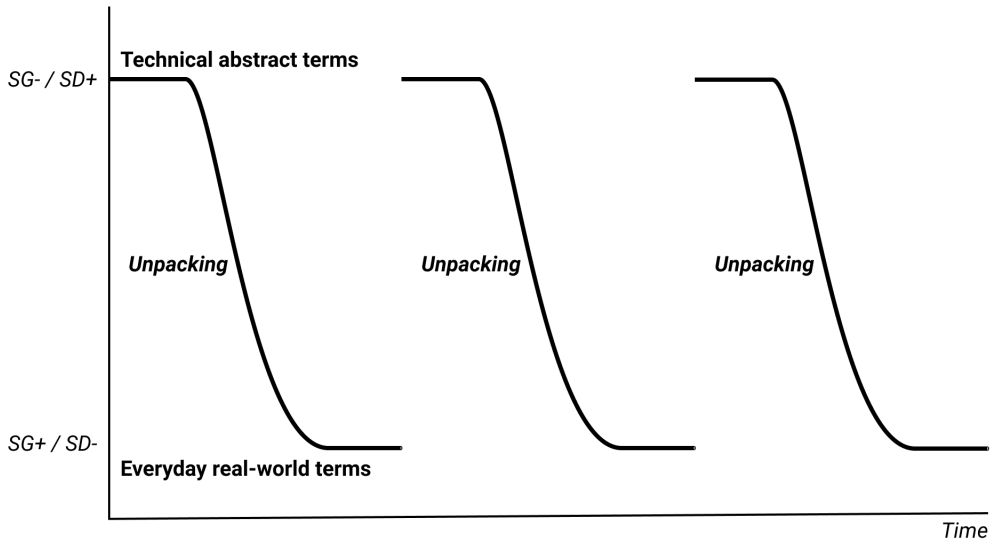


Figure 6: Several downward semantic shifts in succession.

also called the *semantic wave* [16]. A representation of a semantic wave can be seen in Figure 7. Semantic waves provide guidelines on the structure of language-use and explanation-methods within the explanation of a concept, the structure of a chapter and similarly the structure of the entire course. When following semantic waves, every unit of explanation (the explanation of a concept, a chapter or the entire course) should first state its premise abstractly. This premise should not involve examples of the real world. Then, this premise is *unpacked* into everyday terms using examples from the real world. And finally, the explanation is *repacked* using more technical language and going back to general terms.

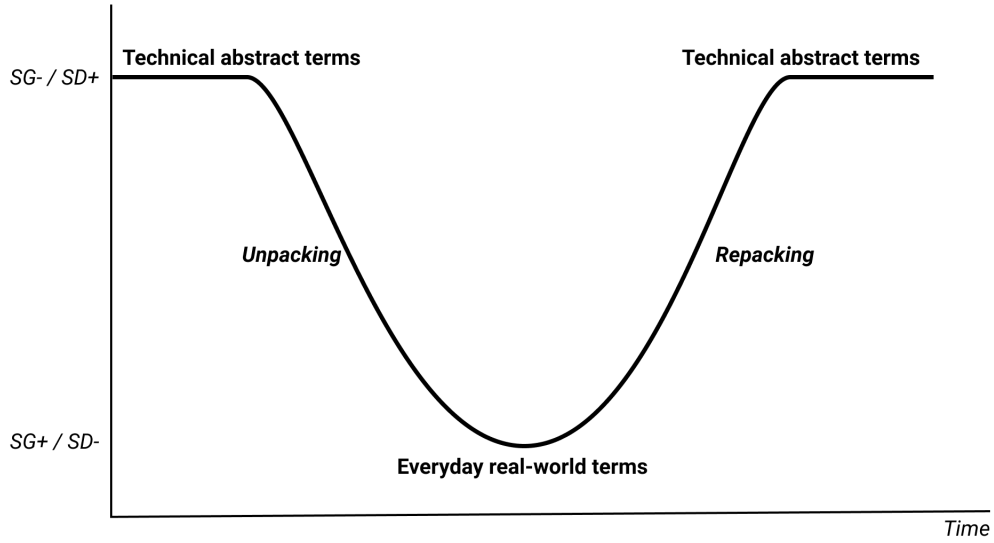


Figure 7: A graphical representation of a semantic wave.

3 Methodology

In order to create a short-course which has the required amount of functionality and has a proper delivery, this work is conducted following a specific structure. The methodology is divided over three categories.

1. Discovering and selecting of existing resources.
2. Creating and refining course material.
3. Evaluating the functionality and delivery of course material.

First a detailed description of each of the three categories will be given. Subsequently, the relationships between these different categories will be explored. Afterwards, the steps to collect material will be discussed.

3.1 Discovering and selecting of existing resources

In order to create a well-formulated and semantically sound explanation covering an aspect of power analysis (PA), background information needs to be gathered, and relevant works need to be evaluated. The combination of background information and relevant works are hereafter referred to as background resources. These background resources were evaluated on two specific grounds: the *completeness* of a background resource and the *complexity* of a background resource.
Completeness is indicated by the amount of extra external information or referenced resources needed for the target audience of the final short-course to understand the background resource. A lower amount of extra external information or referenced resources needed to understand the background resource implies a higher amount of *completeness*. In essence, *completeness* is a measurement for how well a resource provides a centralized knowledge-base for the target audience on a specific subject.
Complexity of a background resource can be seen as the amount of excess information or unneeded abstract concepts used in or exposed by that resource. This is evaluated relative to the concepts which were sought to be explained by the resource. A lower amount of excess information or unneeded abstract concepts implies that a background resource has a lower *complexity*. Fundamentally, *complexity* is a measurement for how fitting a resource is for the target audience on a specific subject.
Background resources are sought to be both high in *completeness* and low in *complexity*.

3.2 Creating and refining course material

The resulting short-course may be divided in the multiple components. The combination of these components is hereafter referred to as course material. There are four different types of components for this short-course. These are *utilities*, *documentation*, *manuals* and *exercises*. These are illustrated in Figure 8.
Utilities aim to provide a significant time gain and increased ease-of-use to users whilst performing a commonly done or an otherwise difficult to perform task. Utilities can provide a baseline from which to start. In which case, they may also be called *template utilities*. These utilities form a

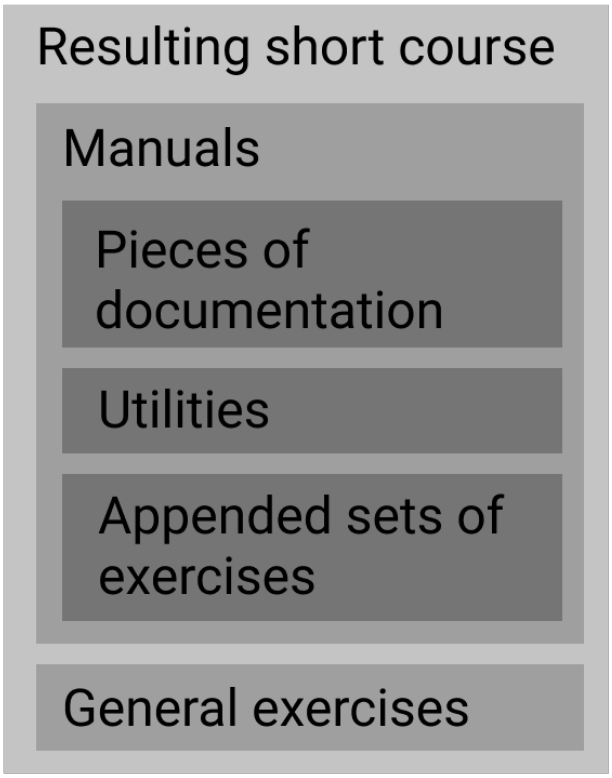


Figure 8: An overview illustrating the structure of course material.

template from which to start. Utilities can also abstract difficult concepts and tasks or mask concepts and tasks which are unrelated to the course's objective. In which case, they are often referred to as *wrapper utilities*. These utilities wrap around more abstract concepts.
Documentation is a resource to quickly understand the specific behavior of protocols, programs and utilities. In essence, documentation provides an avenue to interact which these resources without needing to concern oneself with implementations. Therefore, it may prevent unnecessary distraction and provide a possibility for black-box implementations. It can also provide context as to why certain design choices are made, extra notes or warnings for when using that item, as well as share examples of how to perform common or specific tasks using that resource.
Manuals are a collection of programs, utilities and documentation, gathered as to achieve specific goals. A manual takes a reader through performing certain tasks. They provide a close to centralized location for information on performing that task.
Exercises are problems for readers to solve, which are aimed to test and advance the depth of understanding of the reader. Exercises can be appended to manuals to test the reader on their understanding of the manual's information, or they can be more isolated in order to concern more generalized and/or applied knowledge.
The resulting short-course is a combination of manuals, some with appended exercises and a more generalized applied exercise. Course material — consisting of these manuals and the utilities, documentation and exercises from which the manuals are compiled — needs to be created and refined. The evaluation procedure for course materials will be covered in more detail in Section 3.3

and the refining process for course materials will be covered in more detail in Section 3.4.

3.3 Evaluating the functionality and delivery of course material

In order to evaluate the course material created, and determine if and on what ground to refine and improve that component of the short-course, two factors are taken into account: *functionality* of that component and *delivery* of that component.
Proper *Functionality* is satisfied completely when a component is coherent and it provides all the information and/or capabilities needed in order to perform its goal. For a utility, its capabilities and the level to which it achieved its set goals are most important. The problem such a utility intended to speed up and/or simplify relates to its goal. For documentation and manuals, the goal relates to the amount and completeness (discussed in Section 3.1) of information provided. For exercises, the goal relates to what information was supposed to be accessed and how well that information was tested.
Delivery of course material refers to the ease with which a component can be understood and/or utilized. This includes the complexity (discussed in Section 3.1), abstract language use and semantic waves (discussed in Section 2.4). Specifically for utilities, delivery refers to how simple it is for one to get started with that utility without prior knowledge of it.

3.4 Relationships between sections

The parts of the methodology were not followed chronologically, but in fact followed a network of relationships. A representation of the entire methodology can be seen in Figure 9.
Starting with no resources or documentation, one attempts to create and correctly formulate a manual covering an aspect of power analysis. If a concept — which needs to be explained and thus formulated — proves to be either impossible to reproduce or formulate using the current resources, an attempt is made to find resources which fill this conceptual gap. When a potential resource is found, it is evaluated on both complexity — including readability and semantic context — and completeness. Whenever both requirements are sufficiently met as explained in Section 3.1, the resource is accepted and implemented as course material. If either of the conditions is not sufficiently met, the resource is re-evaluated in order to determine whether the missing aspects can be resolved. Furthermore, it is also evaluated whether a different resource exists which scores better in the evaluation on complexity and completeness. If so, that resource is used instead.
If a resource is selected, it is either directly referenced or adapted into a more fitting format. This is signified by a resource having a lower complexity, a more complete explanation and/or it being better suited to the manual's semantic wave (discussed in Section 2.4). This may again require new resources to fill conceptual gaps. In that case, the process is repeated.
If no resource exists to fill a conceptual gap — and that gap creates a significant shortcoming in the information or functionality — a new resource has to be created without using background resource as a base. In this case, complexity and completeness can be taken into account whilst creating the resource.
When a manual covering an aspect of power analysis is created and formulated, the functionality is evaluated. In the case where a certain part of the required functionality is missing, that part of functionality is then created and formulated using the processes described above. If all requested functionality is present, we move on to evaluating the formulation and semantic context. There

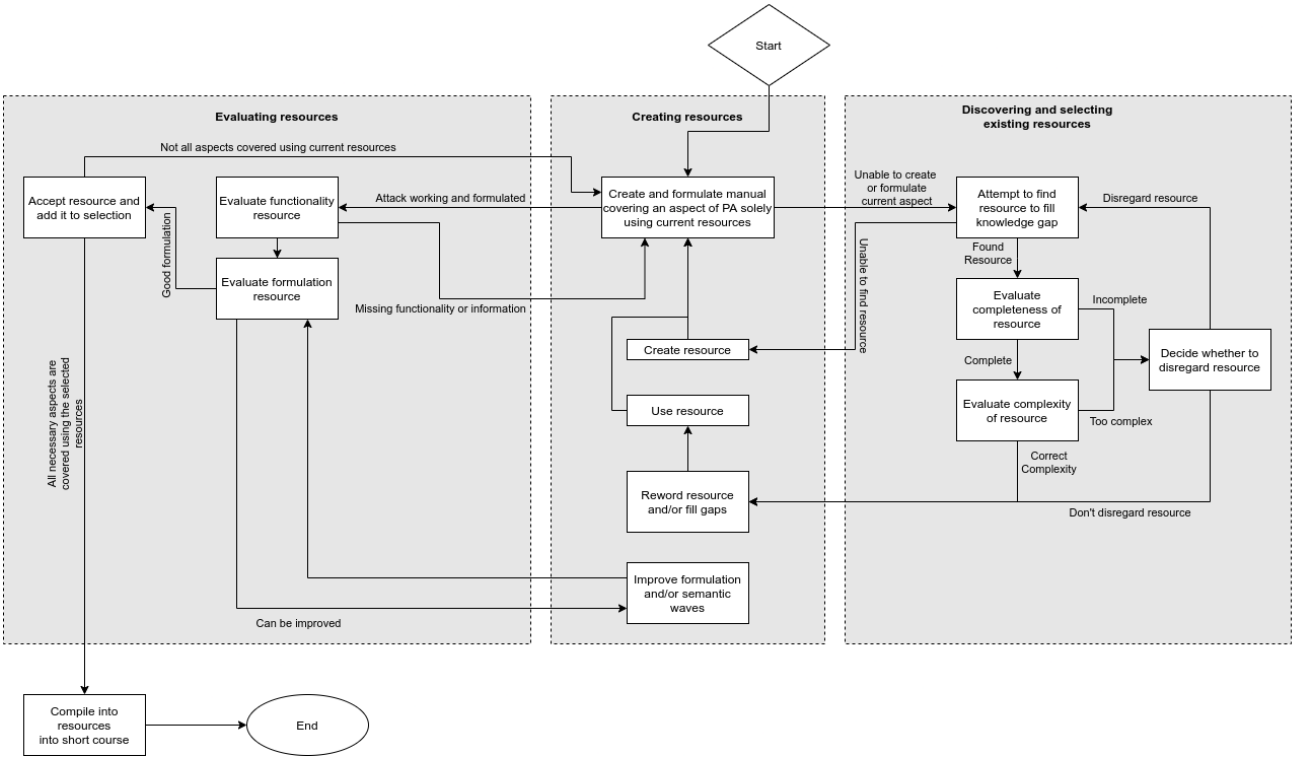


Figure 9: A diagram showing the methods followed. A enlarged version can be found in Appendix A.

may be several cycles of improvement to these aspects. If more manuals are needed — when all combined — to create a satisfactory overview of power analysis, more manuals are created by recursion of the process.

3.5 Collection into a short-course

When it is determined that the current selection of manuals paint a satisfactory overview of power analysis, all manuals are compiled into a short-course. Some alternations to the semantic structure or formulation can be made in order to create a better overall semantic structure or formulation for the short-course. This compilation aims to be a centralized resource on power analysis and in its structure should aim to provide an uninterrupted record from start to end. Furthermore, at the end of the short-course, a more generalized and gradable assessment of knowledge should be done.

4 Course material

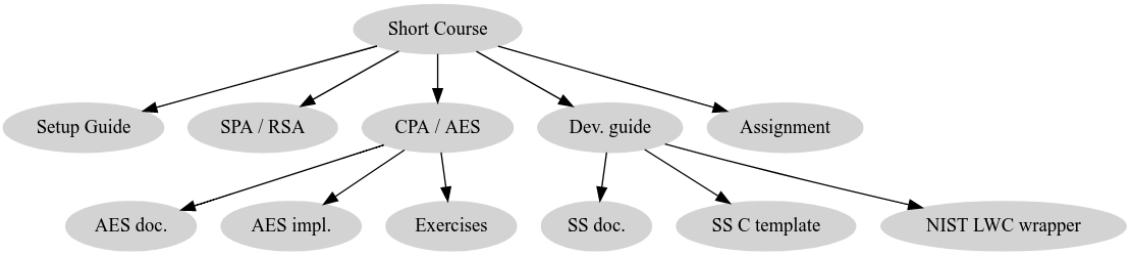


Figure 10: A diagram showing all course material created.

Resulting from the methodology explained in Section 3, many assets were created and then collected into a short-course. As also explained in the Section 3, these resources can be categorized by utilities, documentation, manuals and exercises.
This chapter will go over all the resulting materials by inspecting their origin and goals. Firstly in Section 4.1, the manual which aims to document and simplify the steps whilst setting up the reader's environment is covered. Then in Section 4.2, a manual containing the main concepts of power analysis is introduced. Thirdly in Section 4.3, a manual aimed to introduce the reader to the ChipWhisperer framework and CPA is discussed. Fourthly, Section 4.4 covers resources aimed at simplifying the experimentation and development process. Lastly, Section 4.5 discusses a more general exercise aimed at gradable evaluation.

4.1 Setup Guide

One of the first mandatory steps taken when starting with side-channel Analysis and with the ChipWhisperer framework is setting up all the prerequisites and installing the dependencies on your computer. Creating a utility which would automatically set up all necessary components was ultimately decided against, since it was thought to be limited in its cross platform capabilities and was thought not to add value when considering the target audience of the course. Therefore, largely adapted from the ChipWhisperer documentation [17] on prerequisites and installation, a created setup guide [18] aimed to summarize the extensive pieces of documentation.
This manual first provides instructions on how to install Python [19] along with PIP [20], TQDM [21], MatplotLib [22], and NumPy [23], which are all mandatory or helpful dependencies while performing analysis of power traces created by the ChipWhisperer capture devices. The manual attempts to provide an installation guide for all commonly used operating systems.
Following the installation guide of Python and other packages, there is an installation guide on how to install the ChipWhisperer Python Library along with its dependencies. Namely, LibUSB [24] and GNU make [25]. Again, this manual provides specific instructions for all commonly used operating systems.

4.2 Simple Power Analysis

Solely introducing the concepts of power analysis — isolated from any examples or implemented attacks — is missing in the open-source tutorials from the NewAE [17]. This causes the open-source

tutorials to lack the SG-/SD+ part of the semantic wave, which also creates a gap in a centralized source. Articulating, in its most basic sense, the idea that power analysis provides is the goal of the simple power analysis manual [18].
This manual first provides the idea behind SPA and then using the example of RSA attempts to give an empirical meaning to the concepts explained. The aim is to provide a visual intuition for SPA, which is then to rebuilt that into theoretical knowledge. An excerpt of the manual can be found in Appendix B.

4.3 Correlation Power Analysis

The core manual of the resulting short-course is the Correlation Power Analysis course [18]. This course means to adapt the NewAE CPA tutorial [17] into a more centralized, complete and comprehensible package. Its target being to provide adequate theoretical knowledge, and to have a proper transition from theory and abstract knowledge to implementation. The manual intends to achieve this goal by combining knowledge about AES with the knowledge about CPA. Both of these are standalone sections in the manual.
The CPA manual divides into four parts, which occur in the respective order listed below.

1. CPA's theory, which covers the theoretical basis on Correlation Power Analysis. Along with information about the Hamming-Weight and Hamming-Distance leakage models. It also covers Pearson Correlation Coefficients.

2. AES theory, which contains documentation on the AES algorithm's specification along with a section covers how CPA works with AES.

3. ChipWhisperer theory, which covers how to measure power traces using the ChipWhisperer and how to export these traces to perform analysis later.

4. Implementing a CPA attack, which includes analyzing trace data along with an extra section on optimizing the speed of our analysis.

4.3.1 AES Documentation

Detailed documentation, which covers how AES works, is missing from the mentioned NewAE tutorial [17]. This was, therefore, appended to create a more centralized knowledge base for a correlation power analysis attack. This documentation adapts the AES specification [2] into, what is meant to be, a more approachable and concise explanation. The aim is to introduce the mathematical operations used within AES. Furthermore, this documentation also includes the cryptographic added value — namely the reasoning for inclusion in the algorithm — of mathematical operations, which should provide a deeper understanding of the inner workings of AES. This broader grasp of AES should simplify reinvention of the attack method by the reader. An excerpt of this documentation can be found in Appendix C.

4.3.2 AES CPA attack implementation

The CPA attack on AES detailed in the ChipWhisperer tutorials [17] contained sufficient completeness but failed to unpack semantically dense (SD+) explanations. The concepts explained

in a semantically dense way include correlations and AES leakage models. Therefore, it was attempted to adapt the CPA attack covered in the ChipWhisperer tutorials with more attention to these semantically dense explanations. Since the provided CPA attack is functional but computationally intensive, an added section on code-optimization creates a lower computational hurdle for the reader. An excerpt of the manual covering modeling AES can be found in Appendix D.

4.3.3 Exercises

Testing the knowledge gained from the CPA manual, in a more open-ended way, is also missing from NewAE tutorials. Thus, some more open-ended broader questions had to be introduced. These questions attempted to allow the reader to engage with them for a longer time. This is done by providing room for experimentation and additional research.

4.4 Development Guide

When experimenting with SPA and CPA attacks, some development of low-level algorithm implementations has to be done. This can include both code complications steps and low-level programming. This topic formed a gap in the NewAE documentation [17]. A manual needed to be written to provide accessible and easy to understand information on developing and compiling algorithms for the ChipWhisperer. Along with this manual, extra documentation and two utilities help to simplify the development process. This manual also contains a list of useful resources when compiling algorithms for ChipWhisperer targets, which also includes all the course material mentioned.
One of the challenges faced when trying to experiment with algorithms on the ChipWhisperer is implementation; it is still difficult to implement algorithms which can be executed on the ChipWhisperer boards. Therefore, three resources were created in order to simplify this process. Firstly in Section 4.4.1, a resource is covered in which documentation for a commonly used protocol was written. Secondly, Section 4.4.2 details an attempt at a well-documented template utility. Lastly, Section 4.4.3 covers a wrapper utility which aims to simplify experimentation for a selection of algorithms.

4.4.1 SimpleSerial Documentation

A protocol often used for ChipWhisperer targets is *SimpleSerial*. This protocol allows for communication between the device capturing the power trace and the device performing the encryption calculations, and it lacked up-to-date documentation. Therefore, more detailed documentation about the SimpleSerial protocol was added to the NewAE ChipWhisperer repository [26]. This documentation includes an interface reference along with a broad protocol specification. It aims to make utility development easier using this protocol. Large parts of this documentation were written in collaboration with developers at NewAE.

4.4.2 SimpleSerial C Template

While the NewAE ChipWhisperer repository [27] provides a base template for SimpleSerial target source code, it misses things such as a Python code example for capturing power traces, references to external resources and documentation on compilation. Furthermore, the base template also

creates unnecessary complexity for new users over topics such as the version of the SimpleSerial protocol.
To address these mentioned problems, a new template repository was created [28]. This repository attempts to provide a readable and albeit comprehensible Python power trace capturing code example. Furthermore, its *README* file intends to provide comprehensible documentation on compilation as well linking to external resources for more information. One of these resources being the SimpleSerial Documentation discussed in 4.4.1.

4.4.3 NIST Lightweight Cryptography Wrapper

In addition to providing a base template for SimpleSerial targets which is discussed in Section 4.4.2, a wrapper around the NIST Lightweight Cryptography [29] contest algorithms was built [30]. This wrapper has multiple goals.

1. Universal: This wrapper repository aims to work for all NIST Lightweight Cryptography contestants.

2. Little boilerplate: The wrapper intends to require very little boilerplate code and setup when compiling algorithms or obtaining power traces.

3. Abstraction: The wrapper's design intends to abstract and thus wrap as much ChipWhisperer specifics as well as abstracting away common steps taken to implement algorithms for ChipWhisperer.

4. Accessibility: The wrapper aims to provide an extremely accessible option to provide embedded hardware encryption algorithms.

In pursuing these goals an abstraction over the SimpleSerial protocol was constructed, as well as creating a wrapping interface for the ChipWhisperer Python API [27]. Compiling NIST Lightweight Cryptography algorithms for ChipWhisperer targets using this wrapper consists of few steps. The documentation of these steps can be found in the repository's [30] *README* file. An excerpt of this file can be found in Appendix E.
Many steps taken to set up and transfer data — such as keys, input, and associated data — to and from the target are routine, error-prone and not the main concern for students studying power analysis. The wrapper, therefore, aims to abstract setting up and transferring the differing key sizes, variable sized input and associated data byte-arrays, as well as retrieving the data back.

4.5 Assignment

As a requirement set by the original goal of creating a short-course for a Master level class, it was necessary to also create an assessable exercise for students. This exercise, which was the final amendment to the short-course [18], aimed to provide an open-ended but thorough examination of the student's knowledge. The goal is pursued by assigning students with a report. This report should entail the details on some component of power analysis along with a demonstration. Furthermore, the report should at least include a "reproducible demonstration and explanation of your method" and a "description of (remaining) attack vectors through power analysis".

The report's subject choice is left up to the student whilst still providing a list of example subjects to pick from. This method of subject selection intends to give complete freedom to enthusiast students whilst not also trying not to overbear more reserved students.
Along with explanation on the subject and requirements of the assignment, a set of focus points is provided which will be graded upon. This has two objectives. Firstly, it aims to provide the students with transparency on the grading process of the assignment. Secondly, it provides the professor, student assistant or other moderators grading the assignment some guidelines on grading.

4.6 Resulting short-course

The resulting short-course compiles all these materials into one semantically waving collection. This compilation aims to have the possibility to be read from beginning to end, without need for additional resources. This is done by slightly transitioning from manual to manual, illustrated by introducing the next manual at the end of a manual, and incorporating the conclusions from a manual in the next one.

5 Discussion

When searching for the current limitations of information surrounding the ChipWhisperer Lite ARM board, the results indicate that finding a centralized source providing complete information when starting out with the board is difficult. Furthermore, the ChipWhisperer framework lacked some critical utilities and documentation surrounding important protocols. These three points were all addressed and improved upon. This was done in accordance with creating a Master course on power analysis.

The short-course created during this project [18] provides additional explanations about topics and steps of the power analysis process. These topics and steps were previously either difficult to follow (being targeted at more experienced users or not paying attention to semantic waves), had hard to locate documentation or were completely undocumented. Furthermore, the course has also centralized close to all necessary knowledge in one location. Both of which create an easier opening for new users of the ChipWhisperer framework to get started with power analysis. This leads to it being suited for an introductory Master course on the subject. Currently, however, the short-course only demonstrates one CPA implementation using the AES algorithm. This also means that the short-course does not provide reference to or content covering topics of intermediate difficulty.

The semantic waves formed within the short-course are solid. They improve readability of the explanations of the topics and steps mentioned before. Since there is a lack of objective evaluation methods for semantic waves, the semantic waves proved difficult to evaluate. Within some specific sections, the semantic waves can still be improved. Mostly, these improvements can be done in the chapter on SPA and the AES documentation section.

Arguably the biggest addition this project made is not the short-course but instead the combination of the improved SimpleSerial documentation [26], the new template repository [28], and the NIST wrapper [30].

The new template repository and the improved SimpleSerial documentation provide a much simpler, well documented and less cumbersome way to compile one's binaries for the ChipWhisperer Lite ARM board. This should allow for more experimentation using the ChipWhisperer framework, although knowledge of the SimpleSerial protocol is still required. Therefore, it is still not trivial to either create an algorithm implementation or wrap an existing algorithm implementation and compile one's binaries from those.

The NIST wrapper provides a well documented and easy to use wrapper around the NIST Lightweight Cryptography competition's algorithms. It invites for more research, as will be covered in Section 7. It also potentially — without that being part of its initial intentions — provides a simpler method to execute created algorithm implementations on the ChipWhisperer than the template repository. Although, no documentation for using the wrapper for more than just the NIST Lightweight Cryptography contestants has been written.

6 Conclusions

The research intended to identify the current limitations of the information environment surrounding the ChipWhisperer Lite ARM board. Based on creating a short-course for Master students to fill in these limitations, it can be concluded that — before the project concluded by this research — the information environment provided many resources but existing step-by-step documentation targeted at beginners had limitations. These limitations included a large amount of needed preliminary knowledge, explanations not suited for beginners, and a lack of set up and experimentation documentation.
Regarding the improving the information surrounding the ChipWhisperer framework by creation of a short-course for Master of Science students, the following conclusion can be drawn. The short-course provides a centralized resource which contains the needed knowledge for students to get started with power analysis. The semantic structure of the short-course can be further evaluated and improved. Furthermore, the created utilities and written documentation enable beginners to more easily utilize the framework.

7 Future Work

Based on the conclusions drawn in Section 6 and on the evaluation of the results done in Section 5, future studies can address the following areas.

7.1 Evaluation of semantic waves

In this research, it became apparent that, currently, objectively evaluating the semantic waves created by informational texts is difficult. This is also briefly mentioned in Section 5. The process of creating, improving and refining informational texts could be greatly improved if more objective measurements or evaluations or semantic waves were available.

7.2 Implementation in other programming languages

During this research, it was attempted to implement some utilities or documentations in other programming languages rather than the languages used by the ChipWhisperer framework. The programming language attempted to be used was Rust [31] whilst the framework exclusively uses Python [19] and C. Ultimately — due to time restraints — this implementation attempt was halted. Implementing the SimpleSerial protocol in Rust could help the usability of the ChipWhisperer framework and allow for modernization of the side-channel analysis community.

7.3 Information on steps to take next

As discussed in Section 5, the results demonstrated by this research lack the information on where to go when finished with the introductory short-course. A recommendation on these steps could be documented or appended to the short-course.

7.4 NIST Lightweight Cryptography wrapper

The NIST Lightweight Cryptography wrapper, which is one of the results of this research, shows enormous potential which was left unexplored by this research. There are two recommendations regarding this wrapper future studies could address.
Firstly, the wrapper could be adapted into a more general wrapper of the ChipWhisperer framework. More general here meaning that it would provide a simpler, more intuitive and less-boilerplate method to wrap or develop algorithms which are not part of the NIST Lightweight Cryptography competition and measure their power traces. This is also briefly mentioned in Section 5. This could provide a more beginner-friendly interface to the framework.
Secondly, the wrapper can be utilized to provide a comparison analysis between the contestants of the NIST Lightweight Cryptography contest. For each algorithm, the level to which they are possible to attack using side-channel analysis could be evaluated. Referencing the algorithm's memory usage and computational intensity, this can then optionally be equated with how lightweight the algorithm is.

References

[1] Simon Singh. The Code Book - the Secret History of Codes and Code-Breaking. 2000.

[2] Joan Daemen. The Rijndael Block Cipher. 1999.

[3] R L Rivest, A Shamir, and L Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. 1978.

[4] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6), November 1976.

[5] Muhammad Iqbal and Andysah Putera Utama Siahaan. Combination of MD5 and ElGamal in verifying file authenticity and improving data security. November 2018.

[6] Paul Kocher and Joshua Ja. Differential Power Analysis. 1999.

[7] Eli Biham and Adi Shamir. Differential Cryptanalysis of the Full 16-round DES. 1992.

[8] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In Gerhard Goos, Juris Hartmanis, and Tor Helleseth, editors, *Advances in Cryptology — EUROCRYPT '93*, volume 765. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994. Series Title: Lecture Notes in Computer Science.

[9] Harinandan Tunga and Soumen Mukherjee. A New Modified Playfair Algorithm Based On Frequency Analysis. *International Journal of Emerging Technology and Advanced Engineering*, January 2012.

[10] Hui Xu, Zhuangbin Chen, Mingshen Sun, Yangfan Zhou, and Michael Lyu. Memory-Safety Challenge Considered Solved? An In-Depth Study with All Rust CVEs. February 2021. arXiv: 2003.03296.

[11] Andrew Prout, William Arcand, David Bestor, Bill Bergeron, Chansup Byun, Vijay Gadepally, Michael Houle, Matthew Hubbell, Michael Jones, Anna Klein, Peter Michaleas, Lauren Milechin, Julie Mullen, Antonio Rosa, Siddharth Samsi, Charles Yee, Albert Reuther, and Jeremy Kepner. Measuring the Impact of Spectre and Meltdown. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, Waltham, MA, September 2018. IEEE.

[12] Alex Dewar, Jean-Pierre Thibault, and Colin O'Flynn. NAEAN0010: Power Analysis on FPGA Implementation of AES Using CW305 & ChipWhisperer. October 2020.

[13] Philip Sedgwick. Pearson's correlation coefficient. *BMJ*, 345, July 2012.

[14] Sergei Skorobogatov. Synchronization method for SCA and fault attacks. *Journal of Cryptographic Engineering*, 1(1), April 2011.

[15] Colin O'Flynn and Zhizhang Chen. ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design*, volume 8622. Springer International Publishing, Cham, 2014. Series Title: Lecture Notes in Computer Science.

[16] Karl Maton. Making semantic waves: A key to cumulative knowledge-building. *Linguistics and Education*, 24(1), April 2013.

[17] ChipWhisperer documentation. Available at https://chipwhisperer.readthedocs.io/en/latest/index.html.

[18] Gijs Burghoorn. Power analysis Introductory Walkthrough. Available at https://coastalwhite.github.io/intro-power-analysis/, 2021.

[19] python. Available at https://github.com/python/cpython, May 2021.

[20] pip. Available at https://github.com/pypa/pip, May 2021.

[21] tqdm. Available at https://github.com/tqdm/tqdm, May 2021.

[22] matplotlib. Available at https://github.com/matplotlib/matplotlib, May 2021.

[23] numpy. Available at https://github.com/numpy/numpy, May 2021.

[24] Chris Dickens, Ludovic Rousseau, Nathan Hjelm, and Ihor Dutchak. libusb. Available at https://github.com/libusb/libusb, May 2021.

[25] GNU Make. Available at https://www.gnu.org/software/make/.

[26] Chipwhisperer repository simpleserial documentation. Available at https://github.com/newaetech/chipwhisperer/blob/1eb7c2e047d4a9256b10a3fc3f603ae0fc59638d/hardware/victims/firmware/simpleserial/README.md, May 2021.

[27] Chipwhisperer repository. Available at https://github.com/newaetech/chipwhisperer, May 2021.

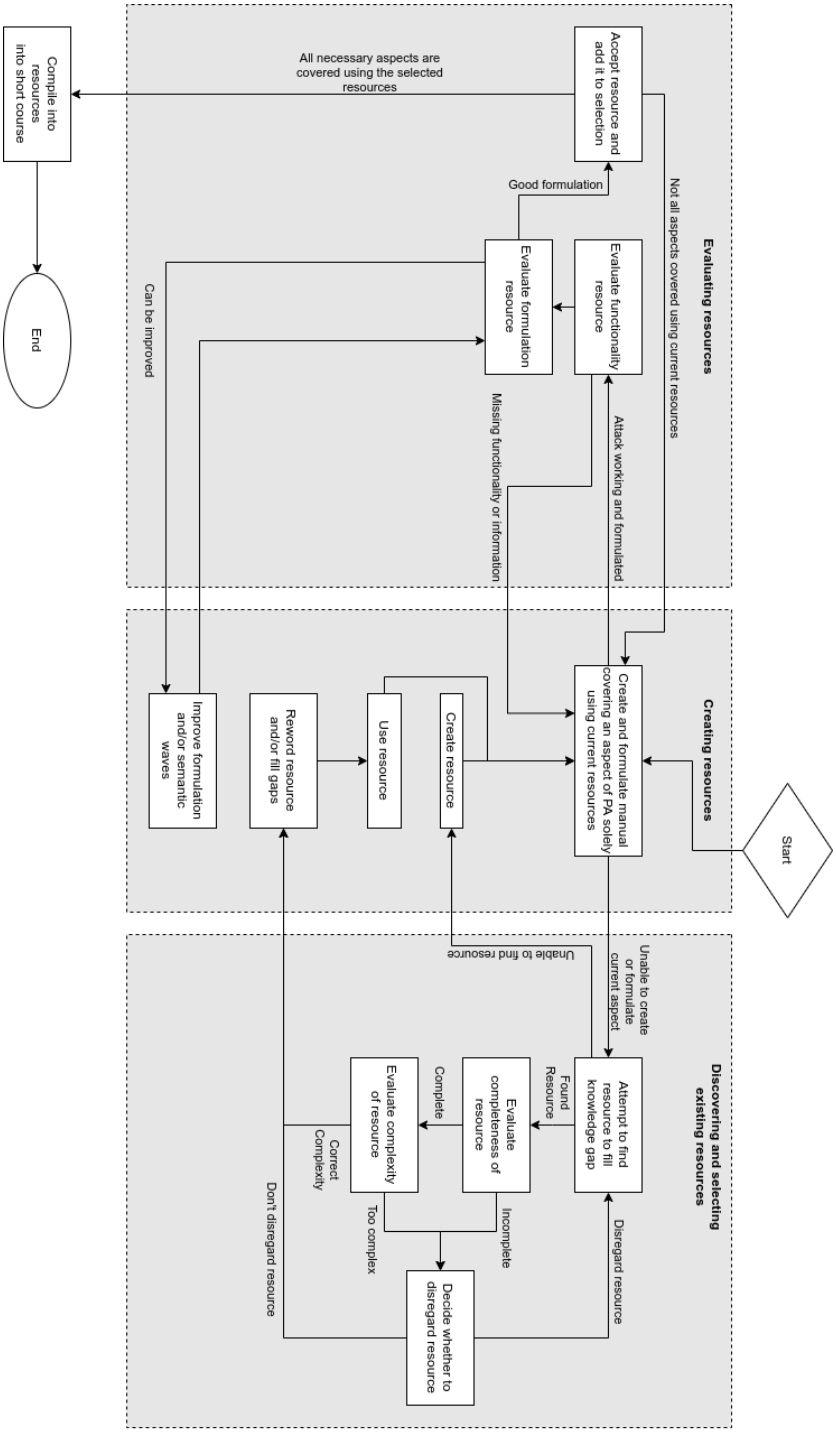
[28] Gijs Burghoorn. simpleserial-c-template. Available at https://github.com/coastalwhite/simpleserial-c-template, April 2021.

[29] Information Technology Laboratory Computer Security Division. Lightweight Cryptography | CSRC. Available at https://csrc.nist.gov/projects/lightweight-cryptography, January 2017.

[30] Gijs Burghoorn. chipwhisperer-nist-lwc. Available at https://github.com/coastalwhite/chipwhisperer-nist-lwc, May 2021.

[31] rust. Available at https://github.com/rust-lang/rust, June 2021.

A Appendix: Method-Structure



B Appendix: SPA Excerpt

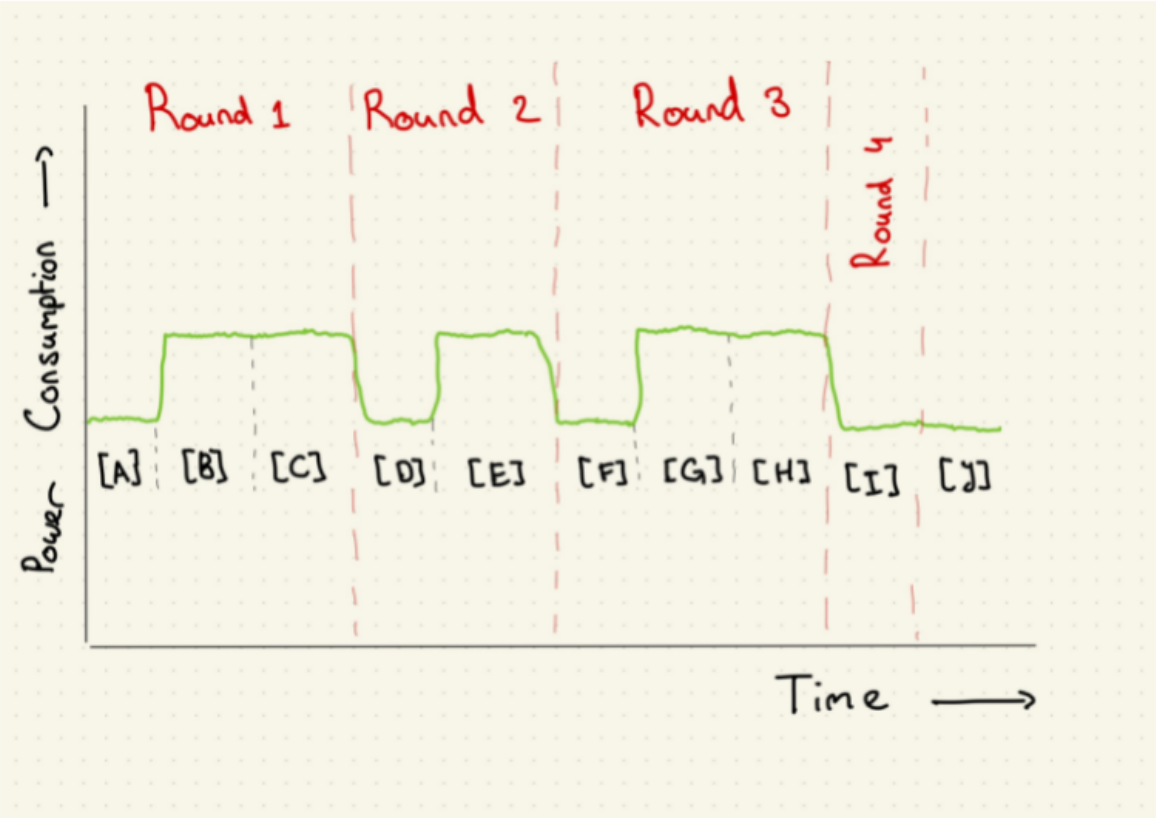


Figure 1: A projected power trace for the `custom_pow_mod(3, 5, 15)` *function call.*

> *Note:* This sketch uses the estimate that conditionals and loops (`if` and `while`) are less power consuming than normal numerical calculations (`>>` , `*` and `%`), which isn't trivially true, but for the sake of simplicity we are going to assume it is true.

You might notice that given this sketch, we can reconstruct some information about the argument `y` provided to the `custom_pow_mod` function. We start with a long spike, thus, the binary number representation of `y` starts with a `1` . This is followed by a short spike, which indicates a `0` . And lastly, we see a long spike again. Therefore, we end with an `1` . And we get the binary number `101` for our `y` . This equals 5 in decimal, which is correct.

Figure 11: An excerpt of the section on SPA in the resulting short-course. Available at https://coastalwhite.github.io/intro-power-analysis/rsa.html

C Appendix: AES Documentation Excerpt

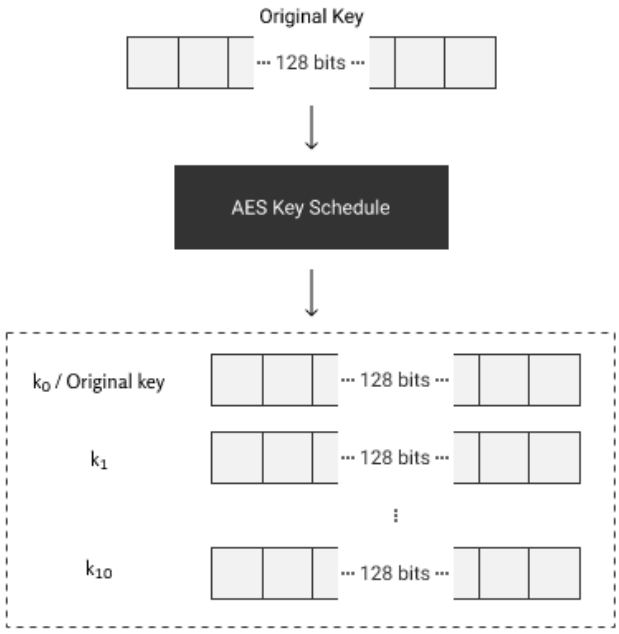


Figure 1: The AES Key Schedule

With these keys we perform a XOR on a block. The XOR operation is a notorious one way operation. This is due to the lack of information the output shares about the input. When we do a one bit XOR operation, and we receive 1 as an output, the input could have been (0,1) or (1,0). We also have two options when we get 0 as output. In the case of one bit, this is not that useful. However, when we have a lot of bits the XOR operator is impossible to instantly reverse for every output and brute forcing time is equal to trying every option divided by two. Mathematically this caused by the XOR operation being non-injective. When we have the outcome and one of the inputs however, this step is extremely easy to reverse. These two properties make it ideal for a lot of encryption algorithms.

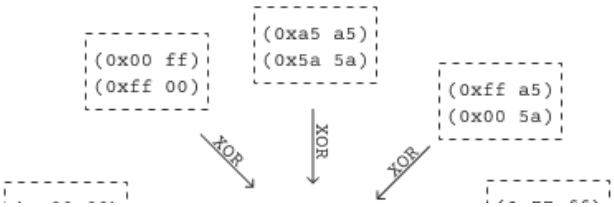


Figure 12: An excerpt of the AES documentation in the resulting short-course. Available at https://coastalwhite.github.io/intro-power-analysis/aes/workings.html

D Appendix: AES Modeling Excerpt

Finding a memory state

Suppose we have an input block *Input* and an output block *Output*, which is a reasonably common situation. If we wanted to check whether a supposed *Key* is the key used, we could run through the entire algorithm to check whether $\mathrm{AES}(Input, Key) = Output$. This is quite inefficient and is no better than brute forcing the key. Knowing what we know about the memory state of AES, we can however do two extreme optimizations.

Shortcutting calculations

The first optimization we can do has to do with identifying the first memory state where the key and the *Input* are combined. If we can identify this memory state in the power trace, we can determine a probability a certain key was used. There are two problems with this, however. Firstly, identifying the presence or the location of this memory state is non-trivial. It is very difficult to manually look at a power trace and tell something about memory states. This is mostly due to the variances in baseline power consumption but also due to noise and other factors. Some of these factors however can be nullified if we look at multiple power traces instead of one. In order to make it even easier, we also look at different input blocks. This allows us to shortcut calculation time by a lot, since we don't have to do multiple rounds. But we still have to brute force through every key option.

Limiting the amount of keys

If we have done the first optimization, there is another optimization which would lower the amount of possible keys. For the 128 (2^{128} different keys), 192 (2^{192} different keys) and 256 (2^{256} different keys) bits key variants, this leaves just 2^{12}, $(3 \cdot 2^{11})$ and 2^{13} key possibilities left, respectively. These are massive differences, which allows us to break AES in just a few seconds.

How does it work? Please read back How AES Works - Shifting for one second and see if you find what we can exploit, when have a value before this step happens. As it mentions this, the step is needed to prevent **attacking each column individually**. Since we can now produce a value before this step is done. We can attempt to break parts of the key one at the time. The parts of this key are called sub-keys, and they are 1 byte in size. This means we can take the sum of different values for the sub-keys instead of the product.

Figure 13: An excerpt of the section on AES modeling in the resulting short-course. Available at https://coastalwhite.github.io/intro-power-analysis/aes/modeling.html

E Appendix: NIST Wrapper ReadMe Excerpt

1. Adding the implementation to your targets

If you have found an implementation you are happy with, you need to add it into the *targets* folder of your cloned version of this repository. On Unix systems (Mac and Linux), I suggest just creating a symlink to the implementation you would like to use.

The implementation here is the folder containing the *api.h* and your other *C* and header files.

2. Configuring the build process

> **Note:** This building process expects you to have the proper C toolchain installed. For more information take a look over here. Under the compiler heading for your OS.

The only configuration which needs to be done in order to compile a target algorithm is in the *makefile*. You have to select which target you want to compile (setting `TARGET_DIR` to the name of the folder in your *targets*), enable which components you want to include (`DO_ENCRYPT` , `DO_DECRYPT` and/or `DO_HASH`), and optionally select which C files you want to add to your compilation (`SRC +=`).

Then we can run the following command to build our target code. Replace `[YOUR PLATFORM]` with a target of your choice. A list of possible targets can be seen here.

```
PLATFORM=[YOUR PLATFORM] make
```

This will output all files to the `outputs/[NAME OF YOUR TARGET_DIR]` folder.

3. Creating your trace capturing script

Below is an example of a trace capturing script. This uses functions from the *cwlwc.py* file in the

Figure 14: An excerpt of the *README.md* from the NIST LWC wrapper repository. Available at https://github.com/coastalwhite/chipwhisperer-nist-lwc