User Friendly SCA Tool

Rui Miguel Alexandrino Tomé
Instituto Superior Técnico, Universidade de Lisboa
Lisboa, Portugal
rui.tome@tecnico.ulisboa.pt

Abstract

Side-channel analysis exploits information leakage related to a device, with the aim of extracting the secret key. There are multiple sources of information leakages, such as power consumption, electromagnetic radiation, timing or heat. These signals need to be collected and handled by proper software in order to allow a researcher to analyze them, perform side-channel analysis algorithms, and be able to extract the secret key. This process comprises two components: capture and analysis. The capture component consists in collecting data from the device under test, whilst the analysis component tries to extract the secret key by optimizing the data in analysis and applying algorithms on it.

However, the open-source tools available today do not follow the growth in this area, missing features that would allow researchers to test cryptographic devices. In this work, an optimized and flexible tool that allows to perform power and electromagnetic analysis was developed. It supports a XY Plotter to assist the capture process of electromagnetic traces over a given cryptographic chip area, allowing to run analysis scripts on the collected traces. To optimize the analysis component, an open-source tool in side-channel analysis (ChipWhisperer Analyzer) was extended, providing the ability to convert data from/to Matlab. An interface to Matlab code and an optimized CPA algorithm were also added to the platform, providing flexibility and performance in the research work.

1 Introduction

Cryptographic algorithms are responsible to protect the communications of millions of electronic devices. They provide confidentiality, integrity and authenticity of data. A cryptographic algorithm is a mathematical function that usually takes as input two parameters, a key and a message (plaintext), and maps these parameters to an output: the ciphertext. This process is called encryption, and follows the Kerckhoffs' Principle [19] that states that all the details about this process are publicly known and only the secret key is kept secret.

Cryptographic keys and cryptographic algorithms are typically implemented and stored in electronic devices. These devices are called cryptographic devices. Some examples of cryptographic devices are smartcards, tokens or Radio-Frequency Identification (RFID) tags.

However, when implemented, there are physical properties of the implementation that may provide additional, unintended sources of information. Power consumption, electromagnetic radiation, timing or heat dissipation are some properties that can provide some information. These sources are called side-channels and their exploitation is called side-channel attacks. Since Kocher et al. presented the first attack [12] in 1998, the aim of the research community has been to improve the efficiency of these attacks and the implementation of adequate countermeasures.

Researchers also aimed to integrate the most powerful attacks into a single evaluation platform that could assess the security of cryptographic devices. However, the platforms available today are either closed and expensive tools or platforms that miss features and optimizations. This document describes the development of an evaluation tool, with features, integrations and optimizations, that can assess the efficiency of side-channel attacks countermeasures implemented on cryptographic devices.

1.1 Motivation

To perform side-channel analysis it is necessary to collect measurements of a device, apply preprocessing techniques that improve the speed and effectiveness of the method chosen, and evaluate the results by statistical analysis. In order to deploy cryptographic services on devices, it is essential for the researcher to have a platform that can perform side-channel analysis from the beginning to the end. This platform can provide the tools to collect measurements from a target device, preprocess that measurements, perform the attack, and do post-processing analysis that will allow characterizing the attack countermeasures effectiveness.

Actually, most of the side-channel analysis tools available today are commercial tools that focus their compatibility with vendor's hardware. Academically, there has not been so much progress in this area. MATLAB is also one of the most used software to evaluate the security of a cryptographic device. The lack of tools in this area leads to extensive use of MATLAB software due to their features in signal processing techniques and its popularity, but it is a closed and commercial software that is not focused on side-channel analysis.

This panorama was changed by Colin O'Flynn with the development of the ChipWhisperer [18] project. The aim of the project is to provide to new researchers in side-channel attack (SCA) the tools for their research. This includes both hardware and software platforms. Although the software developed by Colin was great, it misses features, integrations and optimizations necessary to comply with today's side-channel attacks.

During electromagnetic side-channel attack process, rigorous measurements of EM leaks must be done. This attack can be aimed by placing a EM probe above the chip surface, while a XY table scans the chip while it runs cryptographic operations. Although there are some tools (Universal GCode Sender [25], Benbox [2], etc) to control XY tables, they are

not crafted to provide features that can relieve the process of trace collecting.

1.2 Objectives and Requirements

The goal of this project is to develop a flexible and optimized tool that allows evaluating the protection of a cryptographic device against side-channel attacks. The solutions available today are either expensive commercial solutions or solutions that lack in features and optimizations. To achieve the goal of the project, the development of an open-source tool that offers the following requirements, is needed:

- Support the use of components made in other platforms, through the development of an interface that allows to call external code;
- Development of a tool to control XY Plotter's in order to support electromagnetic attacks;
- Provide compatibility with existing platforms;
- Efficient memory management, in order to support large amounts of data;
- In order to allow the optimization of the attacks, develop features that can provide the assessment of the leakage of a given implementation.

1.3 Contributions

This work resulted in several contributions for the Chipwhisperer platform, namely:

- Add support to use Correlation Power Analysis (CPA) algorithms in Matlab on Chipwhisperer Analyzer software.
- Add support to use a faster CPA implementation on Chipwhisperer Analyzer software.
- Development of a tool to convert Matlab format files to Chipwhisperer, allowing to do analysis of collected power traces in Chipwhisperer.

In the community research, this work also had the following contributions:

- Add support to XY Plotter's software to assist the process of collecting data by controlling the plotter positions.
- Integration of an analysis component in the software.

1.4 Structure of the Document

The rest of this document is organized as follows. Section 2 provides an introduction to side-channel analysis and tools currently available to work this data. Chapter 3 describes the proposed solution and in Chapter 4 the components developed for the platforms are presented. Chapter 5 presents the results of the implementation of the components implemented in ChipWhisperer. Finally, Chapter 6 concludes this document by summarizing its main points and future work.

2 Related Work

This work focuses on two variants of side-channel analysis: power analysis and electromagnetic attacks. Power analysis attacks were presented to the community by Kocher in 1998 [12], with Quisquater and Gandalfi [8, 21] having coinciding results for electromagnetic analysis. These two approaches have proven to be highly effective to extract a secret key from a cryptographic device.

2.1 Power Analysis Attacks

These attacks are based on two dependency factors: data-dependency and operation-dependency [12]. These dependencies show that different cryptographic operations have different power consumption and the switch of more transistors that depend on data will lead to higher power consumption. Analyzing the power consumption measurements of a cryptographic device will allow an attacker to make a guess on the secret key used.
A Power Analysis (PA) attack generally comprises four steps: Equipment Setup, Power consumption Measurement, Data Analysis, and Evaluation.

2.1.1 ***Simple Power Analysis.*** Simple Power Analysis (SPA) [12] attacks try to retrieve the secret key by directly interpretation of the power trace. In this technique, operations from cryptographic algorithms can be identified by analyzing a single power trace or by comparing pairs of traces to find patterns.

2.1.2 ***Differential Power Analysis (DPA).*** In this attack, an attacker performs multiple encryption operations with different sets of data and records the power traces and the ciphertexts. Then, a set of power traces is partitioned into two subsets by a selection function and the difference of the averages of these subsets is computed. If this difference will approach a non-zero value, the partitioning into subsets is correlated to the traces measurements. This is done by using the differential power trace formula presented in Kocher's original paper [12].

Although the secret key is unknown, and at first sight, it may require a considerable amount of time to compute the entire key for a modern cipher, this problem becomes minimized because the computation of the intermediate value only requires part of the key (8 bits for a 128-bit key). Thus, for a 128-bit key, the attacker only needs to try $2^8 = 256$ possibilities for each subkey, which is easily computable. The correct subkey will have the largest difference-of-mean. After the first subkey has been determined, this process is repeated until the entire key is successfully recovered.

2.1.3 ***CPA.*** This attack is an extension of DPA and exploits the correlation between data and power and it is useful when the number of traces available is limited. In CPA[4], a model of how the power consumption of the device depends on some intermediate value is required, i.e. power model or leakage model. The goal of the model is to approximate the power consumption of the target device during the encryption process. This simulation will be correlated with the actual measured power consumption using a key hypothesis, and the correct key will be the one which maximizes the correlation coefficient.

To test the linear relationship between two variables, generally the Pearson's correlation coefficient is used. This value reflects the degree of linear relationship between two variables: a +1 value indicates that there is positive linear relationship

between two variables, a -1 value indicates that there is negative linear relationship, and zero shows that there is no linear relationship. In CPA, we are looking for a correlation between the values generated by the leakage model and the measured power consumption.

For every hypothetical intermediate values, hypothetical power values are computed by a power model. If the power model and intermediate values are correct, then the hypothetical power values will have a linear relation with the measured values.

This attack is faster and more accurate than DPA, however the computation of the Pearson's formula is computationally heavy. To speed up this calculation, data parallel computing can be used [1].

2.2 Existing Plattforms

Currently, the side-channel analysis platform market can be separated into four platforms: MATLAB, ChipWhisperer, Jlsca and platforms developed by industry specifically for side-channel analysis. Although there are other industry platforms, like Smart-SIC Analyzer from Secure IC [24], Inspector SCA from Riscure is one of the most complete tools in the area, and has more public information available about the platform, so it was chosen to represent this market.

MATLAB. MATLAB is a powerful mathematical tool to do signal processing but it is not a platform focused on side-channel analysis. Through custom scripts it allows to perform analysis and process the power traces, develop new power models to be used statistically or any modifications that the researcher wishes to test at any point of the process. However it does not offer data capture options, or an interface that allows to apply models and techniques that are required for side-channel analysis in an easy way. All the process (data input, preprocessing, attack and postprocessing) must be done writing MATLAB code, which requires some learning curve, it is time consuming, and it is not intuitive. Also, MATLAB continues to be a closed, commercial and paid tool.

ChipWhisperer Plattform. ChipWhisperer [18] is an open-source project developed by Colin O'Flynn that consists in both hardware and software platforms for power analysis. The software has an interface designed to work on side-channel analysis and it is divided into two applications: ChipWhisperer Capture and ChipWhisperer Analyzer.

This separation allows researchers with existing attack code to use the capture program and apply it in another software, e.g. MATLAB. On the other hand, it also allows researchers with existing traces, analyze them without using the capture portion.

Several basic preprocessing modules are implemented on ChipWhisperer Analyzer, which operate on the data before passing through the attack. Three types of resynchronization can be found: a sum-of-errors minimizer, peak detect, and cross-correlation. A simple low pass filter is also provided. The waveform display window shows the results after the preprocessing. This is useful to check if the traces are properly resynchronized in the time domain before continuing onto the analysis.

However, ChipWhisperer Analyzer takes a considerable time to do statistical analysis, and it misses features that are required to evaluate cryptographic devices that have countermeasures implemented against power analysis attacks.

JISCA. JISCA [11] is a recently publicly available open-source toolbox written by Cees-Bart Breunesse and Ilya Kizhvatov. It consists in a toolbox to do the computational part of DPA. It is a very complete tool that has support to conditional averaging, conditional bitwise sample reduction, multiple aligment algorithms, CPA, parallezation in the computation process, split input and output samples, AES128/192/256 encryption/decryption, etc. JISCA works on two different type of Traces: InspectorTrace (from Riscure products) and SplitBinary, representing and splitting the data and samples used. On the other hand, Jlsca does not have a GUI to work with, making the learning curve to work with Julia code and with Jlsca time consuming.

Industry Tools - Inspector SCA. In SCA industry, Inspector SCA from Riscure [23] is one very well known player. The software provides features to perform data acquisition as in ChipWhisperer Capture, supports power and electromagnetic analysis, it has signal processing features like filters or trace alignment and supports SPA, DPA and CPA. Although it is a complete solution in this area, it is hard to do changes on some of the modules that are provided by the platform. Also, it is a commercial product, sold on a workstation with the software pre-installed, and it is optimized to work in hardware platforms also developed by Riscure.

2.3 XY Tables, G-Code and GRBL

XY positioning tables can be used in side-channel analysis to support the probe positioning used at Electromagnetic (EM) methodology. The device under test (DUT) is assembled on a surface under the plotter axis, while it is moved along the device to measure its characteristics. One of the most popular languages used in these machines is G-code [13]. This language specifies the parameters of a Computer numerical control (CNC) machine job: which coordinates system to use, distance to travel, motor's feed rate, as well as more advanced parameters.

The fast growth of platforms like Arduino and Raspberry Pi allowed enthusiasts to build their own projects, which leads to the development of GRBL: a widely adopted open source controller to allow this devices to support G-code.

2.3.1 ***Existing platforms***. This section compares three platforms used to control CNC Machines: Benbox, Universal GCode Sender and GRBL Panel. Benbox is supplied with the Plotter used in this research, and Universal GCode Sender and GRBL Panel are two of the most popular software to control them.

Benbox. Benbox is a software supplied with the plotter with the purpose of laser engravement. It is possible to either draw

vector graphics in the software and print them or import vector graphics previously converted to G-code. This representation is a set of instructions of how to draw the vectors. Benbox also provides options for jogging the machine, allowing to provide the number of steps, directions and motor feed rate. In laser engravement options it is possible to control the engraving speed, burn time and engraving mode. This software only works with the LX-Nano ROM, which is proprietary.

Universal Gcode Sender. Universal GCode Sender is one of the most complete solutions to control CNC machines. It is developed in Java, which allows its use in any platform. Some features of Universal GCode Sender are send G-code commands individually to the device, import files with G-code that describe a job and run it, write macros and store them in the plotter and jog the machine, as well define its properties. The software also allows to reset the plotter axis, allowing to define its zero position at the current machine position, or reset it, reverting its non-permanent properties, and canceling a job currently running. The plotter responses are also displayed at the log window.

GRBL Panel. GRBL Panel provides identical features as Universal GCode Sender. It allows to jogging the machine, by defining the distance and feed rate intended, load G-code in order to execute a job, send manual commands or write G-code macros. It's development is in Visual Basic making this software focused on Windows.

3 Proposed Solution

As noticed in the previous chapter, the tools developed to control CNC machines are designed only for that purpose and not for side-channel analysis. To support electromagnetic analysis it is proposed the development of a tool to control a XY plotter, while also having capabilities to capture traces and analyze this information. With this tool it should be possible to specify an area of interest of a cryptographic chip, collect the capture information over each point of that area, and analyze it by running a CPA algorithm with the previous data.

In order to support components made in other platforms it is proposed the expansion of ChipWhisperer Analyzer V3.2.0 with MATLAB support. MATLAB has a strong presence in academic research. The fact of ChipWhisperer have an open-source project with an user friendly interface, and have an integration of preprocessing algorithms built-in, leads to the goal of expanding this software to support MATLAB on it.

It is also proposed the implementation of a less memory consuming CPA algorithm on ChipWhisperer Analyzer to support large amounts of data.

3.1 XYPlotter

Electromagnetic analysis is a powerful technique that can be used to complement or be used when it is not possible to use power analysis. This process can be assisted with a XY table. In this case, their objective is to assist in the process of data gathering while a chip runs cryptographic operations.

It is then proposed the development of a software to control a XY table with the following requirements:

- Ability to jog the machine using buttons;
- Ability to move the plotter to a point of coordinates XY;
- Specify the steps/mm of the motors;
- Limit the distance traveled by the motors in order to not override the maximum allowed;
- Specify a region of interest to be analyzed;
- Map the region of interest in a number of points;
- Run a capture on each point of the region of interest;
- Analyze the data previously collected or at the moment of the capture;
- Save the output of capture and analysis process to a file;
- Save the plotter configurations to a configuration file;
- Load configurations from saved configuration files;
- User friendly interface.
- Automation of the trace gathering process.

The use of configuration files will allow to set and load different plotter parameters or different areas of interest. This makes easier for a researcher to analyze multiple areas without have to set the parameters every time a measure is required. The capture and analysis process is highly tied with the research in progress at INESC-ID, being the capture scripts developed for the equipment used. It is also a requirement that this interaction should need the minimum changes in the code already in use that is used for trace collecting.

3.2 Chipwhisperer Analyzer Expansion

MATLAB has a strong presence in side-channel analysis for development and to work with custom power models. Thus, it is then required the development of a module for ChipWhisperer Analyzer to support it. Supporting this module will also provide ability to do pre-processing techniques over the traces collected with ChipWhisperer, allowing to use this data further in Matlab. This is possible due to Matlab Engine API Python module provided with the Matlab installation.

In order to be possible to analyze the traces collected in XYPlotter, as well the traces previously collected before this work, a tool need to be developed in order to convert Matlab captures to a ChipWhisperer project. Additionally, it will also be possible to use the ChipWhisperer data in Matlab.

Currently ChipWhisperer has two CPA implementations: Simple and Progressive. The simple algorithm is a very simple and non-optimized version of CPA in Python. On the other hand, Progressive uses an optimized version of the algorithm, however their intent is to be used providing a detailed information about the analysis, like correlation vs traces in attack or output vs point plot to see the sample where the known correct key was retrieved. This is done by storing the correlation values at a given report interval. However, for a analysis when this information is not needed, this algorithm is time and memory-consuming. Thus, it is intended to implement a optimized CPA algorithm in order to provide a faster and less memory consuming analysis over the data collected.

4 Implementation

4.1 XYPlotter

The software's purpose is to join the control of XY tables while also having capabilities of running external capture and analysis scripts for EM analysis. It allows to map the cryptographic chip's surface under a number of points, allowing to find the locations where information leaks occurs. Given an interest area, it allows to run captures on each point of it, and run a CPA algorithm over these captures, given regions of interest (traces and samples) and selecting the bytes the researcher is interested on. XYPlotter is a tool developed in Python, it uses PyQt as GUI framework and it is composed by three main modules: UI, Control and Connection. UI is responsible for all the user interface modules, like the main application and its components, settings, send commands, and save and load dialog boxes. Control has the business logic related to application and the connection module is responsible for the serial connection with the plotter. The following sections describe in detail each of these modules.

*4.1.1 **GUI**.*

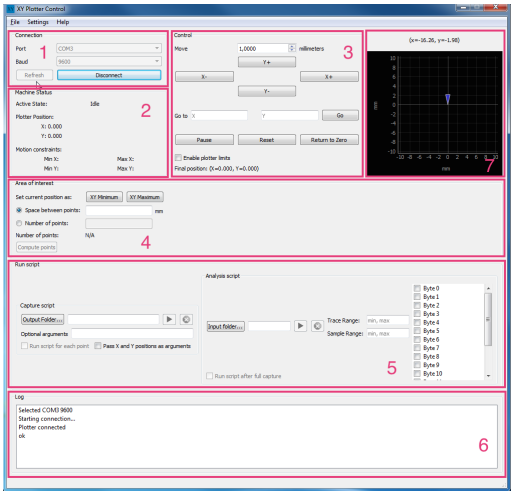

Figure 1: XYPlotter main menu.

The application is divided into the following sections as depicted by Figure 1:

(1) **Connection** - It is used to select a serial port and respective baud rate to establish the plotter communication.
(2) **Machine Status** - It shows the plotter state (idle, active, paused), its location and motion constraints defined at settings.
(3) **Control** - This section implements the plotter control. Jogging commands are displayed in this area, as well the commands to pause, reset, send the plotter to absolute positions, and enable its limits.
(4) **Area of Interest** - A region to analyze is defined in this section, by moving the plotter (using 3) to minimum and maximum limits of the interested area. Next, this region is divided into points by two available methods (distance between the points or number of points provided as input), and the points are mapped at region 7. This information may be used in the next section.
(5) **Run Script** - Allows to collect and/or run CPA over traces previously collected. It is divided into two independent areas: Capture script and analysis script. The capture script allows to collect traces over the plotter's current position or by selecting a region of interest at 4. The analysis process can run after the capture is complete by checking the "Run script after full capture" checkbox or by selecting an input folder where previous traces were collected. Once the trace, sample range and bytes are selected, the process starts and the capture saved in Matlab format is converted to Numpy format and, in order to allow the analysis of the capture in ChipWhisperer, a ChipWhisperer configuration file is generated. The Numpy files will be used later in CPA algorithm, and a log with the results is produced. A more detailed description can found at section 4.1.4.
(6) **Log** - Shows the application log and responses sent by the plotter.
(7) **Map** - Plotter position, regions of interest and physical limits are shown in this area.

4.1.2 ***Interaction between Plotter and GUI.*** The main modules of XYPlotter are UI, Control and Connection. The UI module has the logic related to the UI, such as the creation of elements in UI, its behavior, and user interaction components. This module connects with Control Module, which has most of business logic of the application in the Plotter object. By creating a Plotter instance, the UI module can now get the information needed to update its interface.

The interaction with the plotter is done by methods containing the structure of GRBL commands and sending this information to send_command. This function will use the Connection module, which is a wrapper to pyserial [15] and send the GRBL code to the controller. This interaction is depicted on figure 2.

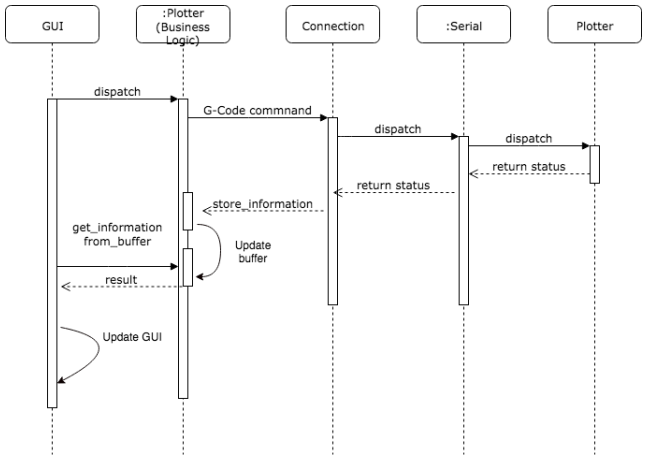

Figure 2: Sequence diagram of XY Plotter.

4.1.3 ***Points distribution.*** In order to collect information over different locations of a cryptographic chip, a distribution of points must be made over a given area. XYPlotter provides two methods to make this distribution. Both require that minimum and maximum limits should be defined, representing the area in analysis. The first method is distance between pointsin millimeters - by dividing the limits over N mm/steps provided in the GUI.

The second one is providing the number of points that the researcher wants to analyze. Given this information, we want to generate a uniform random sampling over the surface limits that is low discrepancy, i.e. a sequence where the possibility of the values being very closed (discrepancy) is low. These sequences are called quasi-random sequences [14] or low-discrepancy sequences. In this work, Halton sequences [9] for dimension 2 are used, which are a generalization of van der Corput sequences to multi-dimension.

To implement the Halton sequences in the project, the ghalton library [22] was used. By specifying the dimension and the number of points intended, the object returns a list of N points, between [0,1] with the dimension specified. Each point is then mapped to the minimum and maximum limits of the respective axis.

4.1.4 ***Capture and Analyze.*** With the capture module, a capture can be obtained for the plotter's current position or for a distribution of points previously defined. The analysis module makes it possible to run a CPA script over a given capture, and also converts the capture to the ChipWhisperer project format, allowing to be analyzed in this platform. Each module has the following behaviour:

Capture. The capture module allows to run the script analysis after the full capture is complete, by checking the respective checkbox in GUI. If the checkbox is checked, the analysis arguments (trace range, sample range, bytes) will also be used. This module handles the synchronization between the plotter movement and the trace gathering. It is also possible to define if the capture will be done for a distribution of points or a single capture, or use the plotter current position as input in the capture script, containing the oscilloscope and FPGA configurations.

When the captures for each point are complete they are stored in the respective folder. In order to write to the log file pending data, a flush is done to the standard output and the plotter returns to its zero position.

Analyze. This module uses a modified version of the Python CPA script implemented in Chipwhisperer, described later, and provides a GUI to it. In order to run a correlation power analysis script, four parameters are supplied: the traces file containing the samples gathered from the oscilloscope, the plaintext(s) file(s) used, the ciphertext(s) output and the key(s) used to cipher it. The capture script stores these files in the following format: the traces have the Matlab format, and the remaining files have txt extension. Each of these files is stored in a specific folder: trace files are stored in "traces", plaintexts in "input", ciphertexts in "output" and keys in "keys". It is our goal to generate a ChipWhisperer project from this information, and to achieve this, it is used the module described in 4.2.1. As input the referred module receives the location of the previous files. When the analysis process comes from the capture process (i.e. "Run script after full capture" is checked), the files location is obtained by getting the folders which contains the timestamp registered. Each of these folders is used as argument to a method (e.g. get_file_plaintexts_txt(path);get_file_traces_mat(path), etc) which returns the list of files of that type. These files will be used as argument to **convert_files_to_cw**, which returns the location of tracefile.npy and plaintexts.npy produced - the ChipWhisperer format.

These files, along with the trace and sample ranges and bytes checked will be used in CPA script, which will do compute CPA for the specified bytes, over the traces and sample range intended. As "Capture", this information will be stored in a log file. In order to avoid the cost of the process of file conversion if two consequent analysis are done, the last analysis is saved, registering its path, the tracefile and the plaintexts file. If this information is the same, only the CPA script will run. In case of any of these files or folder is missing, numpy files will be produced again.

4.1.5 ***Settings.*** The plotter settings (units, steps, maximum and minimum limits, and inverted positions) are attributes from Settings object, which is used to store the configurations in use.

In order to store it is used the PyYAML Framework [20] which allows to dump a Python object to a YAML [5] document and produce a Python object recurring to it. When the user saves the plotter's configuration, a YAML document is produced, describing the plotter configuration, allowing to be loaded later.

This independence allows to use files to save and load the plotter configuration, without the need of define over again the parameters stated early.

4.2 Chipwhisperer Analyzer Expansion

In the following subsections, the support of Matlab format and the implementation of a less memory-intensive CPA version are described. It is also described two modules to convert the traces from Matlab to the Chipwhisperer format and vice-versa.

4.2.1 ***Matlab to Chipwhisperer Project.*** The goal of this tool is to convert in a easy way the traces recorded in Matlab to the ChipWhisperer format, by converting the files to Numpy and generating a ChipWhisperer format configuration.

A ChipWhisperer Project has the following structure:

- A .cwp file containing the properties of the project and the location of the .cfg file.
- A .cfg file with information regarding the traces (number of traces, number of samples, and prefix).
- One single .npy file for each of the following: keylist, knownkey, textin, textout, traces.

With this module it is possible to use multiple .txt and .mat files, and produce single .npy files, along with the configuration files regarding the project.

The GUI developed is composed by 2 tabs. The first one contains information regarding the project, such as project name, project author, number of traces and number of files. The second allows the user to select the files to be used and the output folder.

The information provided about the project is used to generate Chipwhisperer configuration files describing the project structure.

The next step is to save the plaintexts, ciphertexts, keys and traces in numpy format. In traces this is done using the **loadmat** method from the scipy.io python library. For each file selected, the matlab keys are iterated, and a copy of the content of the variable with this information is appended to a list. When all Matlab files selected are traversed, the traces are stacked vertically in a numpy vstack structure, allowing to be accessed in the format traces[trace_number].

For the .txt files, the process is similar.

When each type is handled, the last step is to save the vstack array into .npy. This is done by using the numpy save method, providing the selected output folder chosen early, along with the timestamp from the .cfg file and the type of data (textin, textout, knownkey or keylist).

This tool was integrated in ChipWhisperer Analyzer, and can be found in "Tools - Matlab to Chipwhisperer Converter".

4.2.2 ***NPY to Matlab.*** This module is a simple Python script which provides a way to convert .npy files to a Matlab file. In its original version [1] the .npy files are placed inside an "input" folder. By iterating over each file, the structured is saved recurring to the scipy.io savemat native library, and using a dictionary structure where the key is the name of the .npy file, and consequently the variable of the .mat file. In ChipWhisperer this implementation is used to provide Matlab support, however, instead of using a folder to place the files to be converted, the list of files is passed as argument to npy_to_matlab, and once the process is completed, the .mat file location is returned.

4.2.3 ***Matlab Support on ChipWhisperer Analyzer.*** The Matlab Support on ChipWhisperer provides the ability to use custom power models in Matlab, with the minimum changes in ChipWhisperer's source code. This is achieved by using the Matlab Engine for Python [16], which provides an API to call Matlab functions.

The Matlab support is achieved recurring to the NPY to Matlab module (4.2.2) by writing traces, plaintexts, and keys to a .mat file, and calling the Matlab function with the file location and parameters (trace/samples range and bytes) in analysis.

Another alternative instead of using a .mat file will be to pass the information stored in memory directly to the Matlab function, however the Matlab API needs to convert the data to a Matlab format. This process is memory consuming, because two instances of the same object must remain in memory, and is extremely slow as shown in the Evaluation section.

The Matlab function receives the path of the .mat file, which has the traces, plaintexts and keys, the trace range, the sample range and the byte of interest, and computes the correlation between the Power Hypothesis and the Traces, returning this result to ChipWhisperer. Only a single byte is computed each time in order to provide feedback to ChipWhisperer (Updating the labels and progress bar) in which byte the correlation is at.

For each byte the correlation is computed, and the result is updated in ChipWhisperer stats, which will build the statistics for the Results Table.

Plaintexts, traces and keys (data) are provided from TraceSource object. If pre-processing modules are enabled, they will operate over that data. In order to convert the information to Matlab, the next step is to get the data from the TraceSource, and create numpy files with this information. The filepath from the .mat produced is stored in the TraceSource object, allowing to access this information to further purposes. Once converted, the Matlab function is called and once finished the results are returned to ChipWhisperer, which will handle the data. The .mat file produced contains data after the processing module is applied, allowing its use outside of ChipWhisperer.

4.2.4 ***A less memory-intensive version of CPA script.*** In order to take advantage of this implementation, a new attack module, named CPAMod, was added to ChipWhisperer. This module is a simple implementation of Pearson Correlation using Numpy. In this implementation the data from TraceSource is used as argument without any conversions, allowing to reduce the memory consumption and saving time in disk writings. Its purpose is to provide an analysis where detailed information is not intended, providing a faster alternative to get the keys for the trace range in analysis.

5 Evaluation

This evaluation covers the implementation of Matlab support and the new CPAMod component integrated in ChipWhisperer. Different number of samples were considered in the process, depending on the number of traces and the respective approach used. The tests were performed on a virtual machine with the following characteristics allocated: AMD Ryzen 1700 with 8 CPU's (running at 3.00Ghz), 28GB of RAM DDR4 and running Windows 7 x64. It was used an Samsung Evo 860 Solid State Drive (SSD) to store the data, with a sequential read of 550MB/s and a sequential write of 520MB/s. In both modules it was evaluated the time taken to compute the Pearson's Correlation algorithm and the memory consumed. Two approachs were used when considering the use of Matlab in Python: Pass the traces and plaintexts in memory as arguments to Matlab function's vs write data to a Matlab file and pass the respective filepath as argument. The tests were taken using the cProfile and memory_profile Python profilers recovering the first 10 bytes and the 16 Bytes of the full Advanced Encryption Standard (AES) 128 key and using Matlab R2016a. Then, four attack implementations of ChipWhisperer

[1] https://github.com/ruitome/npy_to_matlab

are compared for the same group of traces, comparing the time taken for each approach.

5.1 Call Matlab from Python - Memory vs File vs Only Python

Two approaches were considered when using Python to call a Matlab function: Pass the traces and plaintexts in memory as arguments to Matlab or write these data to a .mat file and pass the filepath as argument. In the first approach, Matlab requires that the arguments should be converted to a Matlab type, in this case, **matlab.double**. This type only accepts lists as arguments, so the traces and plaintexts must also be converted to lists before using them. After this process, it is possible to call the Matlab function. The following results were taken outside ChipWhisperer, using the Python script provided in the platform and calling the Matlab function which performs the computation. The Python approach used in analysis in XYPlotter was also added to the comparison.

50 samples of each of the following captures were analysed, and the average of the results was computed:

- 1k x 100 (578 KB)
- 100 x 1k (409 KB)
- 10 x 10k (393 KB)

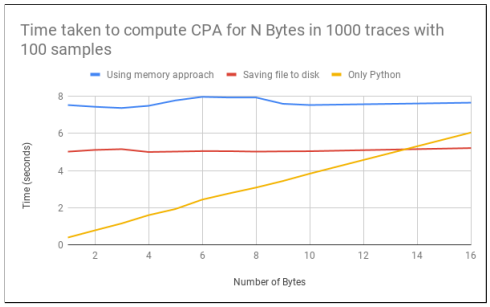

Figure 3: Time taken to compute CPA for N Bytes in 1k traces with 100 samples.

Figure 3 shows the time taken to compute CPA in this group of samples, comparing the time when downsizing the number of traces and increasing the number of samples in study. This group has a reduced number of traces/samples and can be compared to a power analysis attack on a smartcard without counter-measures implemented. As noticed, the time taken when passing the traces and plaintexts in memory from Python to Matlab is higher than other options, evidencing the conversion of the traces to the matlab.double type. On the other hand, it can be noticed that there are cases where it can be useful to choose saving a Matlab file to disk instead of Python exclusive approach. When comparing with the remaining captures, it can be noticed that using this method is ideal for captures above 1000 traces and a number of bytes $\geq$13. However, for this group of traces, this approach has a cost (in time) of initialization of the Matlab engine that is higher than the cost of converting the traces ($<$ 0.05s) and computing CPA. For the remaining captures the behaviour is identical, as the time of initilization is independent of the capture used. As expected, the approach that only uses memory to compute CPA has the highest cost in time and memory.

- 100k x 10k (3,74 GB)
- 100k x 1k (399 MB)
- 10k x 10k (383 MB)

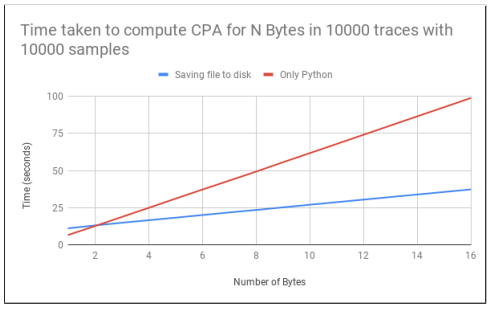

Figure 4: Time taken to compute CPA for N Bytes in 10k traces with 10k samples.

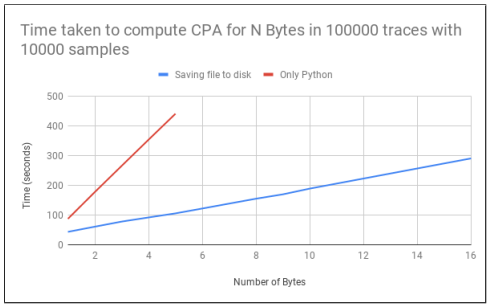

Figure 5: Time taken to compute CPA for N Bytes in 100k traces with 10k samples.

In this group of traces it is even more noticeable the time differences between the Python exclusive approach and the usage of a Matlab file to read data from. Figure 4 confirms that the usage of the Matlab file reduces significantly the time of computing CPA. It is important to refer that for this group the method of using all the data in memory was not considered because it is much slower and consumes much more resources that is out of scale for any comparison with the other approaches. Additionally, for 100k traces x 10k samples (Figure 5), the exclusive python approach has a cost much higher than the alternative approach, so only the first 5 bytes were computed. In both methods the time consumption is nearly linear, i.e. any additional byte we want to compute CPA, the cost increases the time of 1 byte only. However, as stated before, in the approach of saving file to disk there's time involved in starting the Matlab engine and converting traces (which is not noticeable for small group of traces/samples) that is nearly constant. By deducting that time we can notice linearity.

To profile the memory used during the CPA computation it was used the memory profiler module included in Python distribution and the profiler present in Matlab. In these results it is defined the term memory used as the peak memory during the process execution - i.e., the maximum amount of allocated memory during the function execution.

Figures 6 depicts the memory required to compute CPA using Python by saving the file to disk and calling Matlab to do computation. During the evaluation two analysis were made: The first one was to measure the consume of memory during the execution of the Python code. The second was measuring the memory used during the CPA computation in Matlab. Thus, the last one will give us an overview of the memory used in each function.

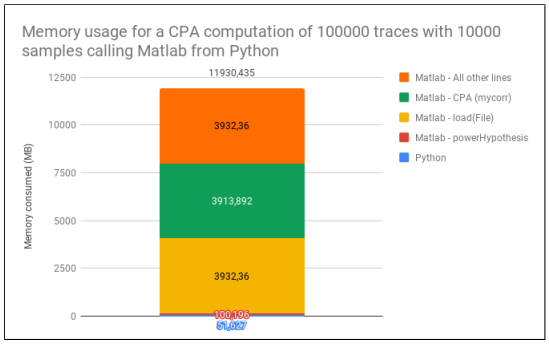

Figure 6: Memory consumed during the computation of CPA in 100k traces with 10k samples saving file to disk.

As stated before, the memory consumed corresponds to the peak memory of the process, thus, the memory in use is independent (has very little contribution) of the number of bytes in analysis. As observable, the memory usage of the Python file is constant. When the traces are converted, it is returned its location, which will be passed to the call of the Matlab code (which will perform the CPA). The conversion of the traces to the Matlab format will produce the following behaviour: load the numpy file in memory and write the same data to the Matlab format. As previously mentioned, this process is not included in the benchmarks, because the conversion occurs before any Matlab call, and it is intended to evidence the maximum amount of memory (i.e. the memory required) in use during all the process.

5.2 ChipWhisperer Integration

In this section, for the same group of traces of the previous section, it is compared the time of processing correlation power analysis in ChipWhisperer Analyzer between the methods described in 4.2.3 and the ones provided by ChipWhisperer.

As default, ChipWhisperer provides two methods to compute CPA: Simple and Progressive. The Simple method is a simple approach to compute CPA. When comparing to Progressive mode, it uses less memory because it provides feedback of the actual attack status and it stores information about the attack that can be used to view other information about the attack progression. For the evaluation, 10 attacks were made recovering the first 10 Bytes, and the 16 Bytes of the full AES-128 key, and the average of the results was computed.

The processing time using Matlab is expected to be higher than the previous results. This increase is caused by the traces conversion to Matlab file, which needs to be created every attack as the pre-processing modules may have been applied to the traces.

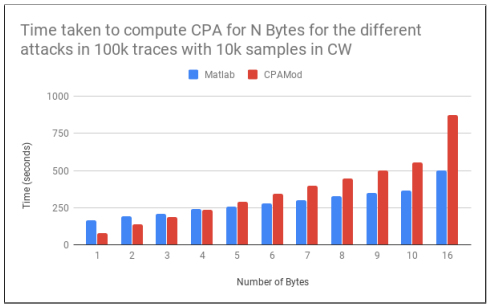

Figure 7: Time taken to compute CPA for N Bytes in 100k traces with 10k samples in ChipWhisperer.

As previously stated, the use of Matlab to compute CPA has better results where multiple bytes are in analysis, taking advantage of the initialization cost of the Matlab engine. For 100k traces with 10k samples, 100k traces with 1000 samples and 10k traces with 10k samples it is observable that CPAMod attack in ChipWhisperer has performance issues with high number of traces as depicted in figure 7. The advantage of Matlab is noticeable, where optimizations in Matlab platform are made to work with this sample data. Despite of the use of numpy Python library in CPAMod, Matlab has advantages for these cases. For these number of traces/samples, the attacks Simple and Progressive are very slow, with results out of scale, so for this reason they were omitted.

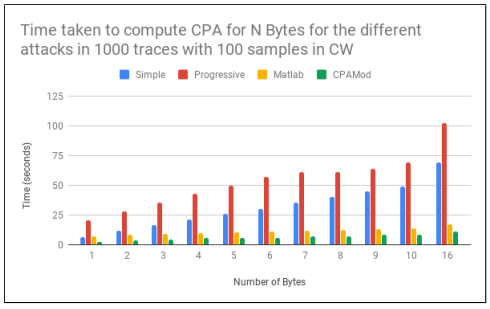

Figure 8: Time taken to compute CPA for N Bytes in 1000 traces with 100 samples in ChipWhisperer.

In both figures 8 and 9 it is noticeable that the Progressive mode is a method that is slower than the Simple method, however, when completed, it provides information about the attack progression.

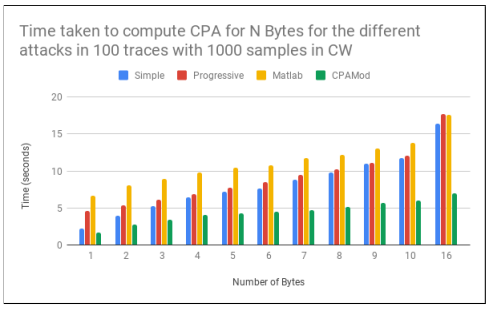

Figure 9: Time taken to compute CPA for N Bytes in 100 traces with 1000 samples in ChipWhisperer.

We conclude that both the implementations have faster results than the algorithms provided originally in ChipWhisperer Analyzer. When comparing a reduced group of traces, the Python exclusive approach is generally faster than the other algorithms implemented. When comparing to other approaches, the fact of Matlab has performance optimizations makes the CPA analysis faster when working with very large samples, improving the time taken on ChipWhisperer to compute these datasets. However, when working with Numpy traces, a conversion process must occur before working with these data, consuming time, memory and disk space. The process of the Matlab engine initialization occurs always when the Matlab algorithm is called in ChipWhisperer Analyzer, contributing to an additional time in the process and taking longer than other algorithms to compute CPA with small group of traces.

6 Conclusions

In this project, a set of tools were developed to allow efficient side channel attack evaluation of cryptographic devices. XYPlotter was developed with the purpose of integrating side channel analysis features with the control of a XY positioning table, allowing to collect electromagnetic traces over a given cryptographic chip area and also allow runs analysis algorithms on it. This analysis can be used to find where higher leaks occur, i.e., the leakage spatial distribution.

Support for both Matlab and conversion modules was added to the ChipWhisperer platform, allowing to compute CPA in a large number of traces and allowing researchers to test different correlation techniques whilst maintaining compatibility with existing environments. An optimized version of Pearson's Correlation was added to ChipWhisperer, name CPAMod, allowing it to have a faster alternative than existing algorithms when few traces are being analyzed. Timing and memory tests were performed in different Python calling Matlab code approaches, passing traces and plaintexts in memory to Matlab's function vs write the data to file and pass the respective filepath, vs a customized CPA implementation in Python, showing the best option for the trace size in analysis.

References

[1] J. Amaral, F. Regazzoni, P. Tomás, and R. Chaves. Accelerating Differential Power Analysis on Heterogeneous Systems. *Proceedings of the 9th Workshop on Embedded Systems Security*, 2014.

[2] Benbox. Benbox.

[3] E. Biham and A. Shamir. Power Analysis of the Key Scheduling of the AES Candidates. *Proceedings of the second AES Candidate Conference*, 1999.

[4] E. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model. *Cryptographic Hardware and Embedded Systems-CHES 2004. Springer Berlin Heidelberg*, 3156:16–29, 2004. ISSN 03029743. doi: 10.1007/978-3-540-28632-5_2. URL http://link.springer.com/chapter/10.1007/978-3-540-28632-5{_}2.

[5] O. B.-K. Clark Evans, Ingy döt Net. YAML. URL http://yaml.org/.

[6] J. Daemen and V. Rijmen. Resistance Against Implementation Attacks A Comparative Study of the AES Proposals. *Second AES Candidate Conference*, pages 343–348, 2000.

[7] H. Faure and C. Lemieux. Generalized Halton sequences in 2008. *ACM Transactions on Modeling and Computer Simulation*, 2009. ISSN 10493301. doi: 10.1145/1596519.1596520.

[8] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic Analysis : Concrete Results. *Cryptographic Hardware and Embedded Systems — CHES 2001. Springer Berlin Heidelberg*, pages 251–261, 2001. doi: 10.1007/3-540-44709-1_21. URL http://www.springerlink.com/content/ead10k34v7q36d3w/.

[9] J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 1960. ISSN 0029599X. doi: 10.1007/BF01386213.

[10] M. Joye, P. Paillier, and B. Schoenmakers. On second-order Differential Power Analysis. *Cryptographic Hardware and Embedded Systems–CHES 2005. Springer Berlin Heidelberg*, 3659:293–308, 2005. doi: 10.1007/11545262_22.

[11] I. Kizhvatov and C.-B. Breunesse. JuliaSCA. URL https://github.com/Riscure/Jlsca.

[12] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. *Advances in Cryptology—CRYPTO'99. Springer Berlin Heidelberg*, pages 388–397, 1999.

[13] T. R. Kramer, F. M. Proctor, and E. Messina. The NIST RS274NGC Interpreter -Version 3. Technical report, 2000.

[14] G. Levy. An introduction to quasi-random numbers. *Transport*, 2010.

[15] C. Liechti. pySerial Library.

[16] Matlab. Matlab Engine for Python. URL https://www.mathworks.com/help/matlab/matlab-engine-for-python.html.

[17] T. S. Messerges, E. A. Dabbish, R. H. Sloan, T. S. Messerges, and R. H. Sloan. Investigations of power analysis attacks on smartcards. *USENIX Workshop on Smartcard Technology*, 1999.

[18] C. O'Flynn and Z. Chen. ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. *Constructive Side-Channel Analysis and Secure Design. Springer International Publishing*, pages 243–260, 2014.

[19] F. A. P. Petitcolas. *Encyclopedia of Cryptography and Security*, chapter Kerckhoffs, page 675. Springer US, Boston, MA, 2011. ISBN 978-1-4419-5906-5. doi: 10.1007/978-1-4419-5906-5_487. URL http://dx.doi.org/10.1007/978-1-4419-5906-5{_}487.

[20] PyYAML. PyYAML Framework. URL https://pyyaml.org/.

[21] J.-J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards. *Smart Card Programming and Security*, pages 200–210, 2001. ISSN 0302-9743 (Print) 1611-3349 (Online). doi: 10.1007/3-540-45418-7_17.

[22] F.-M. D. Rainville. Generalized Halton Number Generator. URL https://pypi.org/project/ghalton/.

[23] Riscure Inc. Inspector SCA, 2016. URL https://www.riscure.com/security-tools/inspector-sca.

[24] Secure-IC. Smart-SIC Analyzer.

[25] UGS. Universal GCode Sender. URL http://winder.github.io/ugs{_}website/.