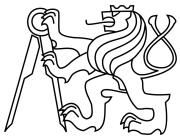
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF MASTER'S THESIS

Title:	Software toolkit for side-channel attacks
Student:	Bc. Petr Socha
Supervisor:	Ing. Vojtěch Miškovský
Study Programme:	Informatics
Study Branch:	Design and Programming of Embedded Systems
Department:	Department of Digital Design
Validity:	Until the end of summer semester 2018/19

Instructions

Implement efficient software platform for side-channel attacks (especially differential power analysis - DPA) including the measurement and the computational parts. The software should be effectivelly parallelized and executable on GPU. Use C/C++ programming language. The software should be multiplatform and distributable under open-source license.

The implementation should include:
- library for oscilloscope measurements (universal - capable of adding interfaces for various oscilloscopes),
- computational library (universal - using various power models, computational and evaluation strategies etc.),
- library for evaluation and comparison of attack complexity,
- library for graph plotting (feel free to use existing graph plotting tools),
- unified user interface for all the libraries (text, eventually also graphical).

References

Will be provided by the supervisor.

doc. Ing. Hana Kubátová, CSc.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague October 19, 2017

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Software toolkit for side-channel attacks

Bc. Petr Socha

Department of Digital Design
Supervisor: Ing. Vojtěch Miškovský

January 9, 2019

Acknowledgements

I would like to thank my family and all my friends for their support through my entire life.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on January 9, 2019 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Socha, Petr. *Software toolkit for side-channel attacks.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Kryptoanalýza postranních kanálů představuje vážnou hrozbu pro mnoho současných kryptosystémů. Útok postranním kanálem se typicky skládá z aktivní fáze, tj. sběru dat, a z analytické fáze, tj. zkoumání a vyhodnocování dat. V této práci je představen softwarový balíček, jenž obsahuje podporu pro ovládání kryptografického zařízení, měření pomocí osciloskopu, (před)zpracování dat, statistickou analýzu a pro vyhodnocení útoku. Balíček je tvořen neinteraktivními textovými aplikacemi s modulární plug-in architekturou, a je uvolněn pod svobodnou licencí.

Klíčová slova Kryptoanalýza, Útok postranním kanálem, Bezpečnost vestavných systémů, Získávání dat, Statistická analýza

Abstract

Side-channel cryptanalysis pose a serious threat to many modern cryptographic systems. Typical side-channel attack consists of an active phase, where data are acquired, and an analytical phase, where the data get examined and evaluated. A software toolkit is presented in this thesis, which includes support for cryptographic device control, oscilloscope data acquisition, data preprocessing, statistical analysis and evaluation of the attack.

The toolkit is composed of non-interactive text-based utilities with a modular plug-in architecture, and it is released under open-source licence.

Keywords Cryptanalysis, Side-channel attack, Embedded system security, Data acquisition, Statistical analysis

Contents

Introduction **1**

1 Side-Channel Security: The Theory and Related Work **3**
1.1 Measuring the Power Traces 4
1.2 Leakage Assessment: Welch's t-test 5
1.3 Correlation Power Analysis Attack 7
1.4 Multivariate Higher-Order Analysis 9
1.5 Univariate Higher-Order Analysis 9
1.6 Other Analysis Techniques 10
1.7 Existing Side-Channel Analysis Software 10

2 Toolkit Design **13**
2.1 SICAK: SIde-Channel Analysis toolKit 13
2.2 Measurement Utility 15
2.3 Preprocessing Utility 19
2.4 Statistical Analysis Utility 21
2.5 Correlation Evaluation Utility 24
2.6 Visualisation Utility 26
2.7 Toolkit-Wide Programming Support 28
2.8 Plug-ins 31

3 Implementation **43**
3.1 Command-line Utilities 43
3.2 Character Device Plug-ins 46
3.3 Oscilloscope Plug-ins 48
3.4 Measurement Scenario Plug-ins 52
3.5 Block Data Preprocessing Plug-ins 53
3.6 Statistical Analysis Computation Plug-ins 54
3.7 Correlation Matrix Evaluation Plug-ins 57
3.8 Keyguess Evaluation Plug-ins 58

3.9 Build, Release and More Information 58

Conclusion **61**

Bibliography **63**

A Acronyms **69**

B Contents of enclosed CD **71**

C Example of an Oscilloscope JSON Configuration File **73**

D Example of a Character Device JSON Configuration File **75**

E Example Usage of meas Utility **77**
E.1 Query Available Plug-ins 77
E.2 Launch Measurement 78

F Example Usage of prep Utility **81**
F.1 Query Available Plug-ins 81
F.2 Create power predictions for CPA attack on AES-128 first round S-box 81

G Example Usage of stan Utility **83**
G.1 Query Available Plug-ins 83
G.2 Create Univariate First-Order CPA context 83
G.3 Create correlation matrices from Univariate First-Order CPA contexts 84

H Example Usage of correv Utility **85**
H.1 Query Available Plug-ins 85
H.2 Evaluate correlation matrices from previous stan example . . . 85

I Example Usage of visu Utility **87**
I.1 Plot Correlation Traces 87

List of Figures

1.1 Power consumption of a device during a part of RSA computation. Image by "Audriusa", Downloaded in December 2018 from https://commons.wikimedia.org/wiki/File:Power_attack.png 3
1.2 Example of a side-channel measurement setup: Oscilloscope with a voltage probe and a shunt resistor placed in the Vdd path. . . . 5
1.3 Example of an unprocessed power trace (grey) and a t-values trace (red) based on 100,000 power traces. 6
1.4 256 correlation traces (for each key candidate), colored grey, obtained when attacking AES-128 using CPA. The right key candidate's correlation trace is colored red. 8

2.1 Example of data flow in SICAK when performing CPA attack. . . 14
2.2 Inheritance diagram for toolkit's data types. 29

I.1 Example plot containing correlation traces for all the correlation candidates, with the right key candidate's correlation trace highlighted. 87

Introduction

In the past few decades, computers and communication networks have evolved into an essential part of our everyday lives. Various computing devices are primary tool for many professionals and are also used in many security-critical applications, such as those in banking sector or government environment. Our money has reduced to a number in a computer database, with an easy access to it: an Internet banking and a debit card. When travelling abroad, our identity is verified using a passport with our biometric information inside. Nowadays, even personal cardiac pacemakers are designed as smart devices and thus may be vulnerable to a cyberattack: former Vice President of the United States, Dick Cheney, had wireless features disabled in his pacemaker in a fear of remote assassination [1]. Other IoT devices, such as personal assistants, smart televisions or lightbulbs make our lives easier, but they also provide yet another way for an attacker to get into our private space. With more Internet-connected devices being available and sold every day, and with a smartphone in every person's pocket, our sensitive information, our possessions and our secrets are more endangered than ever and keeping them safe introduces lots of new challenges [2].

Cryptography, as a way to protect private information, has been evolving for thousands of years now. It has developed from simple ideas, such as Caesar cipher, into many complex algorithms which we use to protect our privacy today. Example of such a cipher commonly used in modern security-critical applications is Rijndael, standardized by National Institute for Standards and Technology as Advanced Encryption Standard (AES), and proclaimed suitable even for confidential information [3]. Expanding market with embedded devices also calls for more lightweight alternatives, suitable e.g. for RFID keychains, such as PRESENT cipher [4]. Alongside cryptography, cryptanalysis has evolved to analyze existing cryptosystems and to attempt to reveal secret information without appropriate privileges.

While many of these cipher algorithms may be considered mathematically secure, their implementations may still be vulnerable to side-channel

attacks, which exploit implementation's weaknesses rather than the cipher's properties, and which represent a significant weak spot for many cryptosystems. This weakness may be even more insidious given the fact that many embedded cryptosystems operate in unsafe, uncontrolled environment, with attacker's direct physical access to them. Side-channel attacks include e.g. power analysis attacks, which examine leaked secret information in the power consumption of cryptographic device [5, 6, 7, 8], fault analysis attacks, which extract secret information by introducing faults to the device [9, 10], timing attacks [11, 12] and many more.

Side-channel attack may usually be split into different phases: a measurement/active attack phase and an analytical phase. To perform the attack, appropriate software support is required. While the analytical phase can be often done using a generic statistical software, the first phase usually require a custom piece of target-specific software.

A multiplatform open-source software toolkit for side-channel analysis is presented in this thesis. It offers support for various measurement scenarios for data acquisition, various data (pre)processing steps, statistical analysis, attack evaluation and a visualisation tool. Related theoretical background and state-of-the-art is presented in Chapter 1. Design of the proposed toolkit is presented in Chapter 2 and it's implementation is described in Chapter 3.

Chapter 1

Side-Channel Security: The Theory and Related Work

Implementing cryptosystems in embedded environment presents many challenges. Using platforms such as various 8-bit microcontrollers, Smart Cards, ARM processors, or even FPGAs and ASICs may be convenient in terms of space, market cost and low power consumption. But unlike universal desktop/server CPUs, these platforms usually do not provide any cryptographic primitives (such as Intel AES-NI instruction set [13]) and the responsibility for these algorithms lies on the cryptosystem designer/programmer. Also, these embedded systems may often run in an uncontrolled environment, with attacker being able to physically tamper the device, which makes these cryptosystems extremely vulnerable to side-channel cryptanalysis.

One of the side channels, through which device may leak sensitive information, is its power consumption [5]. To illustrate this, let us see how RSA algorithm can be attacked using Simple Power Analysis. Figure 1.1 depicts power consumption trace of a cryptographic device during RSA computation, where square-and-multiply algorithm takes place. By visually examining the

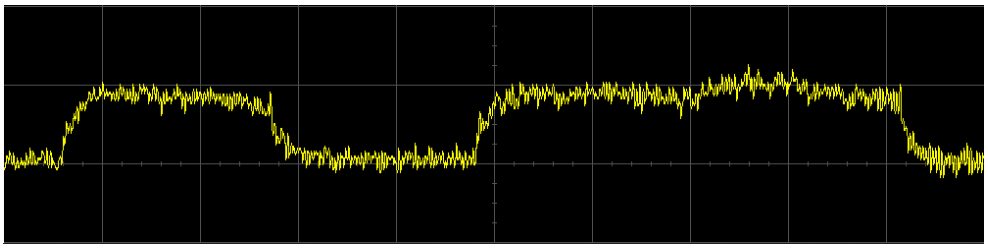


Figure 1.1: Power consumption of a device during a part of RSA computation. Image by "Audriusa", Downloaded in December 2018 from https://commons.wikimedia.org/wiki/File:Power_attack.png

power trace, different steps of the algorithm can be recognized. The short peak on the left side of Figure 1.1 suggests that only squaring happens, and thus the key bit is 0. The longer peak on the right side suggests that both square and multiply steps are being performed, and thus the following key bit is 1.

Similarly to the power consumption of device, an electromagnetic radiation [14], which is proportional to the consumption, can be used to extract secret information.

More complex attacks, which statistically evaluate subtle differences in power consumption to reveal secret information, and which are even applicable to implementations of block ciphers such as DES or AES, include e.g. Differential Power Analysis (DPA) [6] or its enhanced variant Correlation Power Analysis (CPA) [7, 8], which is described later in Section 1.3.

The process of power consumption traces measurement is briefly described in Section 1.1. Example of a leakage assessment methodology is described in Section 1.2. Correlation Power Analysis (CPA) attack is described in Section 1.3. Higher-order side-channel analysis is briefly explored in Sections 1.4 and 1.5. Section 1.6 presents other notable attacks and finally, existing side-channel analysis software is presented in Section 1.7.

1.1 Measuring the Power Traces

To obtain a power trace, such as the one in Figure 1.1, an apropriate measurement must be done. When we talk about power consumption in context of side-channel analysis, we mean the instantaneous energy consumption progression, as opposite to an absolute energy consumption over time (as in context of low-power design) [15].

Usually an oscilloscope is used to measure the power consumption of a device, with either a current probe, or with a classic voltage probe in combination with a shunt resistor. Measuring a voltage drop over the shunt resistor works thanks to the fact that $U = I \cdot R$. The shunt resistor can be placed in a ground (GND) path of the device, or in a power supply (Vdd) path [16]. When measuring in the GND path, more unwanted noise may be present, compared to the Vdd path, which may actually be separated for various chip components. Example of such a measurement setup is shown in Figure 1.2.

Since we only care for the power consumption progression and for mutually comparable power traces, and given that the oscilloscope setup does not change between measurements, we usually do not need to know the exact voltage values (oscilloscope range and offset) and we can work simply with oscilloscope's native ADC values [16].

Because the leaked information in power consumption may be very subtle, certain other conditions must be met for the side-channel attack to be successful [17, 18]. Typically, all the decoupling capacitors near the examined chip

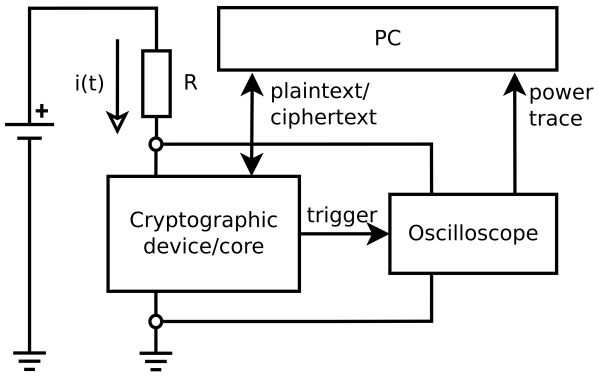


Figure 1.2: Example of a side-channel measurement setup: Oscilloscope with a voltage probe and a shunt resistor placed in the Vdd path.

need to be removed, since their primary function is to filter-out these power consumption nuances. A signal amplifier may also be necessary. To deal with excessive enviromental noise, the oscilloscope bandwidth is commonly limited to 20MHz [16]. To remove the DC shift from the signal, either an AC 1MΩ oscilloscope mode is used, or a DC blocker is employed and a DC 50Ω mode is used [16].

Excessive noise present in the power traces can be a serious obstruction while attacking either hardware or software cipher implementations [17]. Different approaches to deal with these kind of problems exist [19, 20].

Special care should also be taken when using a trigger signal to capture the power trace, since the trigger signal itself can cause a noticeable disruption in the measured power traces. This problem can easily be solved by selecting the right timing of the trigger signal, so that it won't disrupt the required sample points.

1.2 Leakage Assessment: Welch's t-test

In order to evaluate a cryptographic implementation's side-channel leakage, appropriate assessment methodology is required. Our methodology of choice is a **Non-Specific Univariate Welch's t-test** as described in [21]. Other possible choices may include e.g. χ^2-test as described in [22].

The Welch's t-test is used to examine whether two populations have equal means, i.e. to test the null hypothesis that samples in both sets were drawn from the same population (hence two-tailed test). The Welch's t-test is a generalization of Student's t-test for situations when the two populations have different variances [23]

Let μ_1 (resp. μ_2) be sample means of the two sets, let s_1^2 (resp. s_2^2) be sample variances and n_1 (resp. n_2) cardinalities of the sets. The Welch's

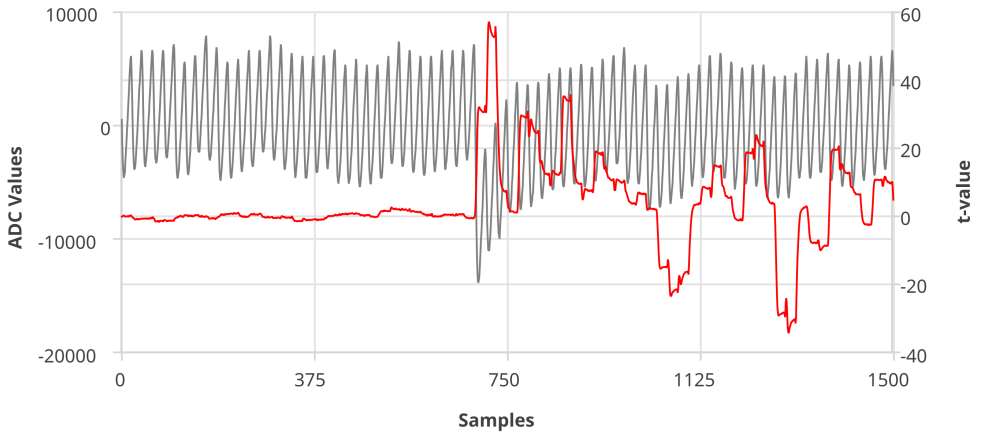


Figure 1.3: Example of an unprocessed power trace (grey) and a t-values trace (red) based on 100,000 power traces.

statistic t can be then computed as:

$$t = \frac{\mu_1 - \mu_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{1.1}$$

and the associated degrees of freedom v can be computed as:

$$v = \frac{(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2})^2}{\frac{(\frac{s_1^2}{n_1})^2}{n_1 - 1} + \frac{(\frac{s_2^2}{n_2})^2}{n_2 - 1}}. \tag{1.2}$$

The non-specific t-test is selected, as described in [21], to avoid the need for many different intermediate values as in case of a specific t-test. Non-specific test categorizes the samples into two sets, where one set contains power samples obtained while using preselected constant cipher data (plaintexts), and the other set constains power samples obtained while using random cipher data (plaintexts) – hence the name "random vs constant t-test".

A specific procedure must be followed while measuring the power traces. For every power trace measurement, the cryptographic device is randomly fed either with preselected constant data or with random data. The fact, that the device is fed either constant or random data in non-deterministic and randomly-interleaved fashion is important to avoid false positive results [21].

Given that a power trace contains N samples, the t, v statistic values need to be computed N times, for each sample point independently. Example of a t-values trace obtained from two populations (random vs constant) with total cardinality 100,000 can be seen in Figure 1.3. The higher is the t-value, the higher is the leakage.

1.3 Correlation Power Analysis Attack

Correlation Power Analysis (CPA) [8] is a side-channel attack applicable to block ciphers such as DES or AES, which aims at a part of the cipher key at a time. It's an enhanced variant of Differential Power Analysis (DPA) [6, 7].

The CPA attack depends on measuring power consumption of the cryptographic device at operation. It is based on the fact, that an intermediate value is processed in the implementation, that correlates with power consumption of the device, with the plaintext or ciphertext used, and with a part of the cipher key. If we are able to predict the intermediate value (given the plain-/ciphertext) for all possible values of the part of the key, all we need is to correlate all these power predictions to the actual power consumption. Since we measure the encryption during a whole operation (because we may not know the exact time/sample point when correlation appears), we need to correlate these power predictions to every sample point in our power traces. That leaves us with matrix $S \times K$ of Pearson correlation coefficients, with S being number of samples per trace and K being number of key candidates (possible values for the part of the key). With enough random plaintexts and power traces available, the correlation coefficient distinguishes the right key candidate. The CPA attack is repeated for every part of the key.

For example when attacking 16 bytes long cipher key of AES-128, CPA aims at a byte of the cipher at a time (this is thanks to the 8-bit S-box architecture of AES). That means we need to perform 16 CPA attacks to obtain the whole cipher key. Number of key candidates for each byte is $2^8 = 256$. The device is fed random plaintexts, power traces are captured using an oscilloscope and ciphertexts may be received back from the device. After the measurements are done, 256 power predictions are computed [24, 25] for every power trace captured, based on the plain/ciphertext used. Then the Pearson correlation coefficients are computed, between every sample and every key candidate:

$$\rho_{X,Y} = \frac{\mathrm{cov}(X, Y)}{\sigma_X \sigma_Y}, \tag{1.3}$$

where X is power consumption variable at given sample point, Y is power prediction variable for given key candidate, σ_X, σ_Y is a standard deviation of X, Y respectively, and cardinalities of both sets are same and equal to the number of measurements taken. Efficient and numerically stable approach to this computation is presented in [26].

After the correlation computation, the right key candidate is selected either manually/visually by the attacker, or algorithmically e.g. by searching for the maximum correlation coefficient or e.g. by searching for the maximum correlation trace derivative [19]. Example of correlation traces (i.e. correlation matrix) obtained while attacking AES-128 is shown in Figure 1.4.

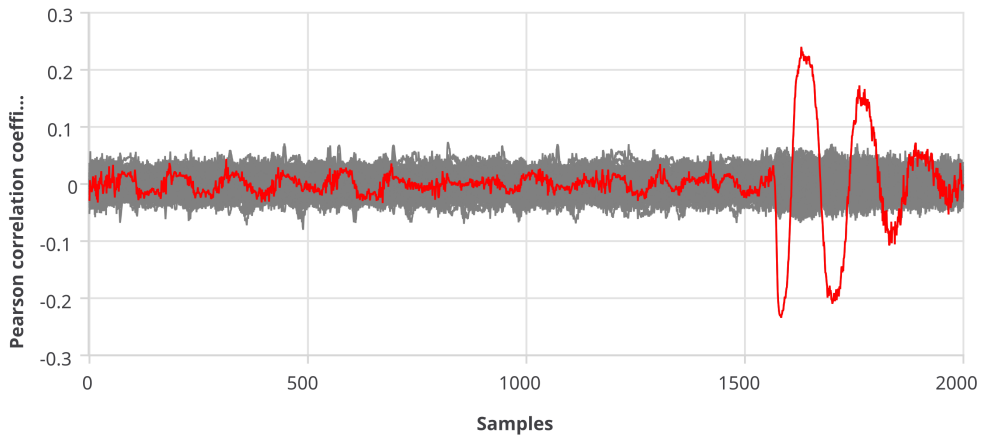


Figure 1.4: 256 correlation traces (for each key candidate), colored grey, obtained when attacking AES-128 using CPA. The right key candidate's correlation trace is colored red.

1.3.1 CPA Countermeasures

There are many different countermeasures to the CPA attack as described above. The used techniques can be categorized as either hiding or masking.

Hiding techniques try to mitigate the dependence of consumption on the data. These include e.g. usage of dual-rail precharge logic [27, 28], which attempts to balance the sensitive data paths in terms of switching activity so that as little leakage occurs as possible. Other approach is to hide the dependence in time e.g. by introducing random dummy computations [29].

Masking techniques on the other hand try to break the data dependency or make it impossible for the attacker to predict any exploitable intermediate value. A common masking countermesure is based on a random mask being xor-ed to the data and appropriately altering the algorithm to still produce valid results [30, 31]. Nowadays, threshold implementations [32], which divide the whole computation into a number of shares, seem as a very promising countermeasure.

Many existing countermeasures combine both techniques [33]. Promising countermeasures against both power analysis and fault analysis attacks are also being built using modern FPGA features, such as dynamic reconfiguration [34, 35].

Many of these countermeasures are effective against Univariate First-Order Correlation Power Analysis attack as described in Section 1.3. However, they may still fail when facing higher-order and/or multivariate statistics [36].

1.4 Multivariate Higher-Order Analysis

Higher-order analysis [36] is one of the possible ways to overcome implemented countermeasures such as masking. Higher-order DPA is defined by Kocher et al. [6] as a DPA attack that **combines one or more samples within a single power trace**. E.g. second-order attack makes use of two different samples in the power trace.

It is useful when intermediate values or their shares are processed at different time, e.g. in software implementations. Finding these time points to combine is a complex task [21].

Different combination functions can be used to combine multiple samples: the centered (normalized) product [37], the absolute difference [36] or the sum [38]. Computational aspects of the first-order as well as higher-order Correlation Power Analysis are well described in [39].

The multivariate higher-order attack/testing may be performed by preprocessing the power traces accordingly and then continuing as in the first-order case [40]. Whole operation can also be effectively parallelized and performed in a one pass through the data [41].

1.5 Univariate Higher-Order Analysis

Univariate higher-order analysis [42, 21] is yet another way of processing the power traces. Unlike multivariate analysis, this time the power traces are processed at **each sample point independently**.

It is useful e.g. when intermediate values or their shares are masked by parallel processing, i.e. they manifest themselves in the same time. Both approaches can be combined: samples in every power trace can be combined (see Multivariate Higher-Order Analysis 1.4) and then the Univariate Higher-Order Analysis can be performed upon them.

For univariate second-order attack/testing, the power traces are preprocessed by making every sample mean-free squared [21]:

$$t'_i = (t_i - \mu_t)^2, \tag{1.4}$$

where μ_t is sample mean at given sample point.

In case of d-th-order, $d > 2$, the samples are usually additionally standardized [41]:

$$t'_i = (\frac{t_i - \mu_t}{s_t})^d, \tag{1.5}$$

where μ_t is sample mean at given sample point, and s_t is standard deviation at given sample point.

Similarly as in case of Multivariate analysis, the preprocessing can be performed on the power traces, or the whole process including e.g. Welch's t-test computation or Pearson correlation coefficient computation can be combined in order to perform the whole operation effectively and parallelly [41].

1.6 Other Analysis Techniques

Other notable class of side-channel attacks are collision attacks, which use side-channel analysis to detect internal collisions in the cipher. For example, a collision attack against AES was proposed [43], where authors state that in the best case, only 40 measurements are necessary to determine the entire 128-bit key. This attack exploits key dependent collisions in the Mix Column transformation of AES.

Enhanced variant of side-channel collision attacks, applicable even to AES with masking implemented, is Correlation-enhanced power analysis collision attack [44], which is further generalized into moments-correlating DPA [45].

Different approach, based on Shannon's entropy theory, is Mutual Information Analysis [46]. The embedded device in this attack is considered a black box and the attack may be successful without any knowledge about the implementation.

It is noticeable, that these attacks usually relax the CPA requirements for a power model and power predictions, and thus may seem as an universal solution. However, when various countermeasures are implemented, higher-order analysis may provide better results.

1.7 Existing Side-Channel Analysis Software

In this Section, various software options for side-channel analysis are presented, including both commercial and free solutions.

1.7.1 ChipWhisperer

ChipWhisperer is an open-source toolchain for side-channel power analysis and glitching attacks [47], maintained by NewAE Technology Inc., written in Python. It has support for both the measurements and oscilloscopes, and for the analytical/computational part of the attack. NewAE Technology Inc. also provides hardware targets and other auxiliary side-channel analysis hardware and tools.

Given fact, that the whole toolchain is written in Python, the performance of the computational part is not really satisfactory for complex and time-demanding attacks.

1.7.2 Riscure Inspector SCA

Inspector Side Channel Analysis is a commercial tool for side-channel analysis [48] by Riscure. Official website claims that the tool "offers SPA, DPA, EMA, EMA-RF and RFA for embedded devices or Smart Cards". The tool supports both measurements and analytical/computational operations, including high-order CPA on ciphers such as 3DES, AES or RSA.

1.7.3 Jlsca

Jlsca is an open-source set of scripts for computational part of a side-channel attack [49], written in Julia. It has support for both CPA and MIA computations and a number of (pre-)processing steps, including support for attacks on AES, DES or SHA1 implementations.

1.7.4 General-purpose Statistical Software

Various general-purpose statistical/mathematical software can also be used for side-channel analysis. This includes e.g. R [50], MathWorks Matlab [51] or Wolfram Mathematica [52].

This software may often offer highly optimized implementations of statistical operations. However, when performing a side-channel attack, we usually perform these time-demanding statistical functions on a number of samples at once. Statistical processing in context of side-channel analysis thus brings another level of data parallelism, which can be only rarely exploited using these applications, resulting in an uneffective computation.

Chapter 2

Toolkit Design

A proposed side-channel analysis toolkit should provide support for following tasks:

- **data measurement** - acquisition of traces using various oscilloscopes,
- **general data (pre-)processing** - processing power traces or other data (e.g. cipher plaintexts in order to create power predictions for CPA attack),
- **statistical data processing** - moment based statistics (e.g. t-test, CPA),
- **attack evaluation**,
- **visualisation** (plot e.g. power traces, correlation coefficients,...).

With performance in mind, C/C++ language is chosen as primary programming language for the toolkit. Time-demanding operations should be parallelizable on both CPU and GPU. The toolkit should be multiplaform and open-source.

2.1 SICAK: SIde-Channel Analysis toolKit

Given the assignment, a plug-in module architecture toolkit was designed, called SICAK. Most of the toolkit utilities simply provide a non-interactive text-based UI and serve as an interface for one or more plug-in modules.

The toolkit is command-line driven in order to allow easy scripting with toolkit utilities. Input parameters can also be passed to the toolkit utilities via configuration files (e.g. oscilloscope configuration, measurement/computation settings, etc.).

It currently consists of following utilities:

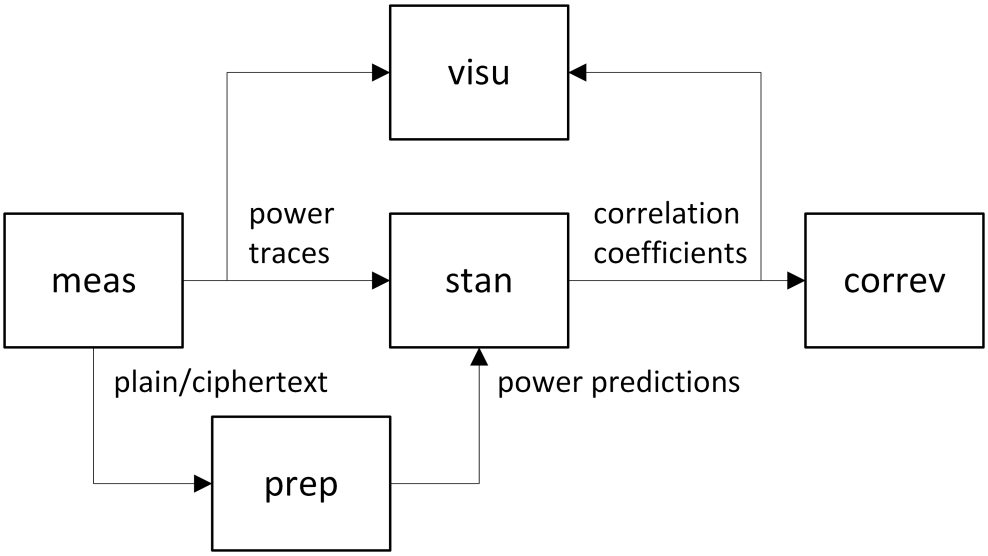


Figure 2.1: Example of data flow in SICAK when performing CPA attack.

- **meas: Measurement utility**, useful e.g. for controlling a cryptographic device and oscilloscope,
- **prep: Pre-processing utility**, useful e.g. for pre-processing power traces or e.g. for creating power predictions based on plain-/cipher-text,
- **stan: Statistical Analysis utility**, useful e.g. for correlation-based (CPA) attacks or t-tests,
- **correv: Correlation Evaluation utility**, useful for algorithmic evaluation of the CPA attack,
- **visu: Visualisation utility**, useful e.g. for plotting power/correlation traces or t-values.

These utilities are moreless interfaces for different types of plug-in modules:

- **meas**
  - **chardevice**: Character device plug-in, useful e.g. for communication over serial port or with a Smart Card
  - **oscilloscope**: Oscilloscope plug-in, useful for interfacing with an oscilloscope device, i.e. setting-up oscilloscope channels, trigger, timing, for arming the oscilloscope and downloading the sampled power traces
  - **measurement**: Measurement scenario plug-in, which receives instances of chardevice and oscilloscope modules and performs the measurement itself

- **prep**
  - **blockprocess**: Block data processing plug-in, useful e.g. for creating the power predictions when performing the CPA attack
  - **tracesprocess**: Power traces processing plug-in, useful e.g. for pre-processing the power traces when performing higher-order statistics
- **stan**
  - **cpaengine**: CPA computation engine plug-in
  - **ttestengine**: t-test computation engine plug-in
- **correv**
  - **cpacorreval**: CPA matrix evaluation plug-in, e.g. searching for maximum coefficients in a set of matrices
  - **cpakeyeval**: CPA keyguess evaluation plug-in, which takes the result of cpacorreval, i.e. array of key candidates (e.g. based on maximums), and turns it into a cipher key; e.g. reversing last round key to the cipher key

In Figure 2.1, an example of data flow between the utilities is depicted when performing a CPA attack using SICAK. Performing a t-test, only *meas*, *stan* and *visu* utilities would be required.

More informations about each utility and it's plug-in modules and interfaces can be found in following chapters.

2.2 Measurement Utility

Measurement utility is called **meas**. It runs a measurement scenario loaded from specified plug-in module. Each scenario may produce different output files, which usually include power traces or plain-/ciphertexts. A JSON file with information about created files may be produced as well.

The Measurement scenario obtains an Oscilloscope and a Character Device instances, loaded from specified plug-in modules. Both these plug-in modules require a Device ID (e.g. VISA address or COM port name) for a succesful initialization (in some cases, the Device ID can be blank). Both oscilloscope and character device can be configured using a configuration file. Usage and format of configuration files is described below.

2.2.1 Usage

Brief usage is also printed when the program is run with -h option. Example usage can be found in Appendix E.

```
./meas [options] config
```

2.2.2 Options

2.2.2.1 -I, –id {string}

The ID string will be used in output files' filenames. Default value is current datetime.

Exact filenames of output files are measurement scenario dependent, e.g. "random-traces-ID.bin". Measurement scenarios implemented in this thesis all produce a JSON configuration file "ID.json" with information about output files. All filenames and informations are, however, printed out to the standard output as well.

2.2.2.2 -Q, –query

Query available measurement, oscilloscope and chardevice plug-in modules (-M, -O, -C) and available devices (-R, -D).

2.2.2.3 -M, –measurement-module {string}

ID of measurement scenario plug-in module to use. Use -Q or –query to find available plug-ins.

2.2.2.4 -O, –oscilloscope-module {string}

ID of oscilloscope plug-in module to use. Use -Q or –query to find available plug-ins.

2.2.2.5 -R, –oscilloscope-device {string}

ID of oscilloscope device to use. Use -Q or –query to find available plug-in's devices.

2.2.2.6 -S, –oscilloscope-config {filepath}

Oscilloscope JSON configuration file. See below for config file format.

2.2.2.7 -C, –chardevice-module {string}

ID of character device plug-in module to use. Use -Q or –query to find available plug-ins.

2.2.2.8 -D, –chardevice-device {string}

ID of character device to use. Use -Q or –query to find available plug-in's devices.

2.2.2.9 -E, –chardevice-config {filepath}

Character device JSON configuration file. See below for config file format.

2.2.2.10 -n, –measurements {positive integer}

Number of measurements to make. This parameter is passed to the measurement scenario.

2.2.2.11 –param {param}

Optional measurement plug-in module parameters. Module specific option.

2.2.2.12 -h, –help

Displays help.

2.2.2.13 -v, –version

Displays version information.

2.2.3 Arguments

2.2.3.1 config

JSON configuration file(s) with Options. The JSON configuration file may contain key:string pairs, where key is a long option name and string is the value.

For example:

```
{ "id":"myMeasurement" }
```

2.2.4 Oscilloscope configuration file

Utility expects a JSON configuration file, which may contain objects with key:value pairs as described below.

- **channelN** object: settings for channel N ($N >= 1$)
  - **enabled**: {true | false}
  - **coupling**: {"AC" | "DC"}
  - **impedance**: {"50" | "1M"}

  - **rangemV**: {integer} (e.g. 1000 sets the oscilloscopes range to $-1V..+1V$)
  - **offsetmV**: {integer} (i.e. analog voltage that is added to the channel before sampling)
  - **bwLimit**: {"FULL" | "25MHz" | "20MHz"}

- **trigger** object: trigger settings
  - **enabled**: {true | false}
  - **channel**: {positive integer} (channel on which to trigger)
  - **level**: {float} (Range 0..1, where 0 is the lowest voltage in the channel's range and 1 is the highest voltage in the selected range. E.g. with channel range $-1V..1V$ and zero offset: level 0 is -1V, level 0.5 is 0V, level 0.75 is 0.5V)
  - **slope**: {"rising" | "falling" | "either"}

- **timing** object: the timebase settings
  - **preTriggerRange**: {float} (time range in seconds)
  - **postTriggerRange**: {float} (time range in seconds)
  - **samples**: {positive integer} (number of samples per power trace; this value is a best-wish: oscilloscope module may change it to the reality forced by the oscilloscope/it's driver, in order to fulfil the time ranges)
  - **captures**: {positive integer} (number of captures to be taken on one oscilloscope run/arm; this value is a best-wish as well: some oscilloscopes support only one capture per run)

When an object is defined (e.g. "channel1"), all it's settings must be set.

Example of a JSON configuration file for PicoScope 6000 can be found in Appendix C.

2.2.5 Character device configuration file

Utility expects a JSON configuration file, which may contain key:value pairs as described below.

- **baudrate**: {positive integer} (the value needs to be supported by the selected character device module)
- **parity**: {0 | 1 | 2} (0 means no parity, 1 means odd parity, 2 means even parity)
- **stopbits**: {1 | 2}

- **timeoutms**: {positive integer} (time to wait for I/O operation before throwing an error)

Example of a JSON configuration file can be found in Appendix D.

2.3 Preprocessing Utility

Data preprocessing utility is called **prep**. It loads either blocks of (char) data and processes them using Block Preprocessing Module, or it loads power traces containing (int16_t) samples and processes them using Traces Preprocessing Module. Output files (including JSON config) are generated by the plug-in modules.

This utility is useful e.g. for performing higher-order attacks (preprocessing the traces) or when performing the CPA attack (creating power predictions based on block data).

2.3.1 Usage

Brief usage is also printed when the program is run with -h option. Example usage can be found in Appendix F.

```
./prep [options] config
```

2.3.2 Options

2.3.2.1 -I, –id {string}

The ID string will be used in output files' filenames. Default value is current datetime.

Exact filenames are plug-in module dependent, similar to the *meas* utility.

2.3.2.2 -Q, –query

Query available traces and block data preprocessing plug-in modules (-T, -B).

2.3.2.3 -T, –traces-preprocess-module {string}

ID of traces preprocessing plug-in module to use. Select either -T or -B.

2.3.2.4 -B, –block-preprocess-module {string}

ID of block data preprocessing plug-in module to use. Select either -T or -B.

2.3.2.5 -t, –traces {filepath}

File containing -n traces, each of which containing -s samples (int16).

2.3.2.6 -n, –traces-count {positive integer}

Number of power traces in -t file.

2.3.2.7 -s, –samples-per-trace {positive integer}

Number of samples per trace.

2.3.2.8 -b, –blocks {filepath}

File containing -m blocks of data, each of which -k bytes long.

2.3.2.9 -m, –blocks-count {positive integer}

Number of blocks of data in -b file.

2.3.2.10 -k, –blocks-length {positive integer}

Length of data block in -b file, in bytes.

2.3.2.11 –param {param}

Optional plug-in module parameters. Module specific option.

2.3.2.12 -h, –help

Displays help.

2.3.2.13 -v, –version

Displays version information.

2.3.3 Arguments

2.3.3.1 config

JSON configuration file(s) with Options. The JSON configuration file may contain key:string pairs, where key is a long option name and string is the value.

For example:

```
{ "blocks-length":"16" }
```

2.4 Statistical Analysis Utility

Statistical processing utility is called **stan**.

The utility loads one of the two different types of plug-in modules (CPA, t-test) and runs one of the tasks:

- **CPA**
  - **create**: Creates a file with new contexts, based on power traces and power prediction sets.
  - **merge**: Merges two files with existing non-empty contexts.
  - **finalize**: Creates a file with correlation matrices, based on a contexts.
- **t-test**
  - **create**: Creates a file with new context, based on random data power traces and constant data power traces.
  - **merge**: Merges two files with existing non-empty contexts.
  - **finalize**: Creates a file with t-values and degrees of freedom, based on a context.

The reason that CPA and t-test modules are separated is that each task receives differently typed data: CPA is run upon power traces (int_16) and power predictions (uint8_t), while t-test is run upon two sets of power traces.

Different CPA/t-test computation plug-in modules may perform different tasks (univariate/multivariate, first/higher-order,...), and the tasks can be also implemented differently (e.g. on CPU and/or GPU).

2.4.1 Usage

Brief usage is also printed when the program is run with -h option. Example usage can be found in Appendix G.

```
./stan [options] config
```

2.4.2 Options

2.4.2.1 -I, –id {string}

The ID string will be used in output files' filenames. Default value is current datetime.

2.4.2.2 -Q, –query

Query available CPA and t-test plug-in modules (-C, -T), platforms (-P) and devices (-D).

2.4.2.3 -C, –cpa-module {string}

ID of a CPA plug-in module to launch. Select either -C or -T.

2.4.2.4 -T, –ttest-module {string}

ID of a t-test plug-in module to launch. Select either -C or -T.

2.4.2.5 -P, –platform {positive integer}

Platform from which to choose a device (-D). Default is 0.

2.4.2.6 -D, –device {positive integer}

Device from a platform (-P) to run computation on. Default is 0.

2.4.2.7 -F, –function {create | merge | finalize}

Select a function:

- 'create' a new context from traces/predictions,
- 'merge' existing contexts A,B
- 'finalize' existing context A.

2.4.2.8 -r, –random-traces {filepath}

File containing -n random data traces, each of which containing -s samples (int16).

2.4.2.9 -n, –random-traces-count {positive integer}

Number of random data power traces in -r file.

2.4.2.10 -c, –constant-traces {filepath}

File containing -m constant data traces, each of which containing -s samples (int16).

2.4.2.11 -m, –constant-traces-count {positive integer}

Number of constant data power traces in -c file.

2.4.2.12 -s, –samples-per-trace {positive integer}

Number of samples per trace.

2.4.2.13 -p, –predictions {filepath}

File containing -q power prediction sets, each of which containing -k power predictions (uint8) for every random trace in -r file.

2.4.2.14 -q, –prediction-sets-count, –contexts-count {positive integer}

Number of power prediction sets/number of contexts. E.g. attacking AES-128 key, this value would be 16.

2.4.2.15 -k, –prediction-candidates-count {positive integer}

Number of power predictions for each power trace in -p file. E.g. attacking AES-128 key, this value would be 256.

2.4.2.16 -a, –context-a {filepath}

Context file A, for use in Finalize or Merge functions.

2.4.2.17 -b, –context-b {filepath}

Context file B, for use in Merge function.

2.4.2.18 –param {param}

Optional plug-in module parameters. Module specific option.

2.4.2.19 -h, –help

Displays help.

2.4.2.20 -v, –version

Displays version information.

2.4.3 Arguments

2.4.3.1 config

JSON configuration file(s) with Options. The JSON configuration file may contain key:string pairs, where key is a long option name and string is the value.

For example:

```
{ "random-traces-count":"10000" }
```

2.4.4 Output

Unlike previous utilities, where plug-in modules are responsible for the output files, in this case the modules are used simply to perform the computation. The output files are created by the utility.

stan produces following files:

- **cpa create**: cpa-ID.Qctx,
- **cpa merge**: cpa-ID-merged.Qctx,
- **cpa finalize**: cpa-ID.Qcor,
- **ttest create**: ttest-ID.ctx,
- **ttest merge**: ttest-ID-merged.ctx,
- **ttest finalize**: ttest-ID.tvals,
- ID.json,

where ID is stan given parameter or default, and Q is number of power prediction sets/contexts created (when Q=1, it's omitted).

.Qcor files contain Q correlation matrices, each SxK large, where Q is number of power prediction sets, S is number of samples per trace and K is number of key candidates.

.tvals file contain Sx2 matrix, where S is number of samples per trace, where in the first row there are t-values, in the second row there are degrees of freedom.

2.5 Correlation Evaluation Utility

Correlation evaluation utility is called **correv**. It loads correlation matrices from a file and evaluates them using specified modules: Correlation matrix evaluation plug-in module finds a keyguess (i.e. a set of selected key candidates, e.g. the maximum coefficients). Keyguess evaluation plug-in module evaluates this keyguess, e.g. reverses the last round AES key.

2.5.1 Usage

Brief usage is also printed when the program is run with -h option. Example usage can be found in Appendix H.

```
./correv [options] config
```

2.5.2 Options

2.5.2.1 -Q, –query

Query available CPA correlation matrix evaluation and keyguess evaluation plug-in modules (-E, -K).

2.5.2.2 -E, –correlations-eval-module {string}

ID of a CPA correlation matrix evaluation plug-in module to use.

2.5.2.3 -K, –keyguess-eval-module {string}

ID of a CPA keyguess evaluation plug-in module to use.

2.5.2.4 -c, –correlations {filepath}

File containing -q correlation matrices, each of which -s wide and -k tall (double).

2.5.2.5 -q, –prediction-sets-count, –contexts-count {positive integer}

Number of correlation matrices. E.g. attacking AES-128 key, this value would be 16.

2.5.2.6 -k, –prediction-candidates-count {positive integer}

Number of key candidates, i.e. rows of correlation matrix. E.g. attacking AES-128 key, this value would be 256.

2.5.2.7 -s, –samples-per-trace {positive integer}

Number of samples per trace, i.e. cols of correlation matrix.

2.5.2.8 –param {param}

Optional plug-in module parameters. Module specific option.

2.5.2.9 -h, –help

Displays help.

2.5.2.10 -v, –version

Displays version information.

2.5.3 Arguments

2.5.3.1 config

JSON configuration file(s) with Options. The JSON configuration file may contain key:string pairs, where key is a long option name and string is the value.

For example:

```
{ "samples-per-trace":"2000" }
```

2.6 Visualisation Utility

Visualisation utility is called **visu**. It allows to plot power traces, correlation traces or t-values and to show the plot in graphical window or save it in raster format (jpg, png) or vector format (svg).

2.6.1 Usage

Brief usage is also printed when the program is run with -h option. Example usage can be found in Appendix I.

```
./visu [options] config series
```

2.6.2 Options

2.6.2.1 -D, –display

Display the chart in a graphical window.

2.6.2.2 -S, –save {filename}

Save the chart to a file.

The output image format is automatically selected by the filename extension (jpg, png, svg).

2.6.2.3 -W, –width {positive integer}

Width of the saved chart.

2.6.2.4 -H, –height {positive integer}

Height of the saved chart.

2.6.2.5 -T, –title {string}

Chart title

2.6.2.6 -t, –traces {filepath}

File containing -n traces, each of which containing -s samples (int16).

2.6.2.7 -n, –traces-count {positive integer}

Number of power traces in -t file.

2.6.2.8 -r, –traces-real-range {positive integer}

Maximum positive value of a power sample in mV, e.g. 2000 for range -2V to +2V.

2.6.2.9 -a, –t-values {filepath}

File containing -s t-test values (double).

2.6.2.10 -c, –correlations {filepath}

File containing -q correlation matrices, each of which -s wide and -k tall (double).

2.6.2.11 -q, –correlations-sets-count {positive integer}

Number of correlation matrices. E.g. attacking AES-128 key, this value would be 16.

2.6.2.12 -k, –correlations-candidates-count {positive integer}

Number of key candidates, i.e. rows of correlation matrix. E.g. attacking AES-128 key, this value would be 256.

2.6.2.13 -s, –samples-per-trace {positive integer}

Number of samples per trace.

2.6.2.14 -b, –samples-real-range {float number}

Time of a single power/correlation trace. Given sampling period T and -s samples, this value would be $T * (s - 1)$.

2.6.2.15 -h, –help

Displays help.

2.6.2.16 -v, –version

Displays version information.

2.6.3 Arguments

2.6.3.1 config

JSON configuration file(s) with Options. The JSON configuration file may contain key:string pairs, where key is a long option name and string is the value.

For example:

```
{ "samples-real-range":"3e-6" }
```

2.6.3.2 series

Time series to plot. For example:

- "t,25,blue" plots 26th power trace from traces file in blue
- "c,0,255" plots 255th correlation trace from the 1st correlation matrix in automatically selected color
- "c,1,all,#bbbbbb" plots all correlation traces from 2nd correlation matrix in grey
- "v,pink" plots t-values from t-values file in pink

Color is optional. When it is not set, the color is selected automatically. Hexadecimal RGB codes or Svg1.0 color names are allowed.

2.7 Toolkit-Wide Programming Support

In order for all these utilities and for the following plug-ins to cooperate, and to make future development easier, a toolkit-wide set of classes and functions was designed. These include basic data containers (Vector, Matrix), a complex data container (Two-Population Univariate Statistical Context), some functions to work with these containers (e.g. to save them to a file) and basic exceptions for the plug-ins to throw.

Inheritance diagram for designed data types is shown in Figure 2.2. The data types are briefly explained in following sections. For more detailed information, see Programmer's Guide, which is attached to the source code.

2.7.1 Array Data Types: Vector, Matrix

ArrayType and derived classes are a container class templates, which main purpose is dynamic memory management (resulting in exception safety) and logical encapsulation. Common interface includes **length()**, which returns number of elements, **size()**, which returns size of underlying memory (i.e.

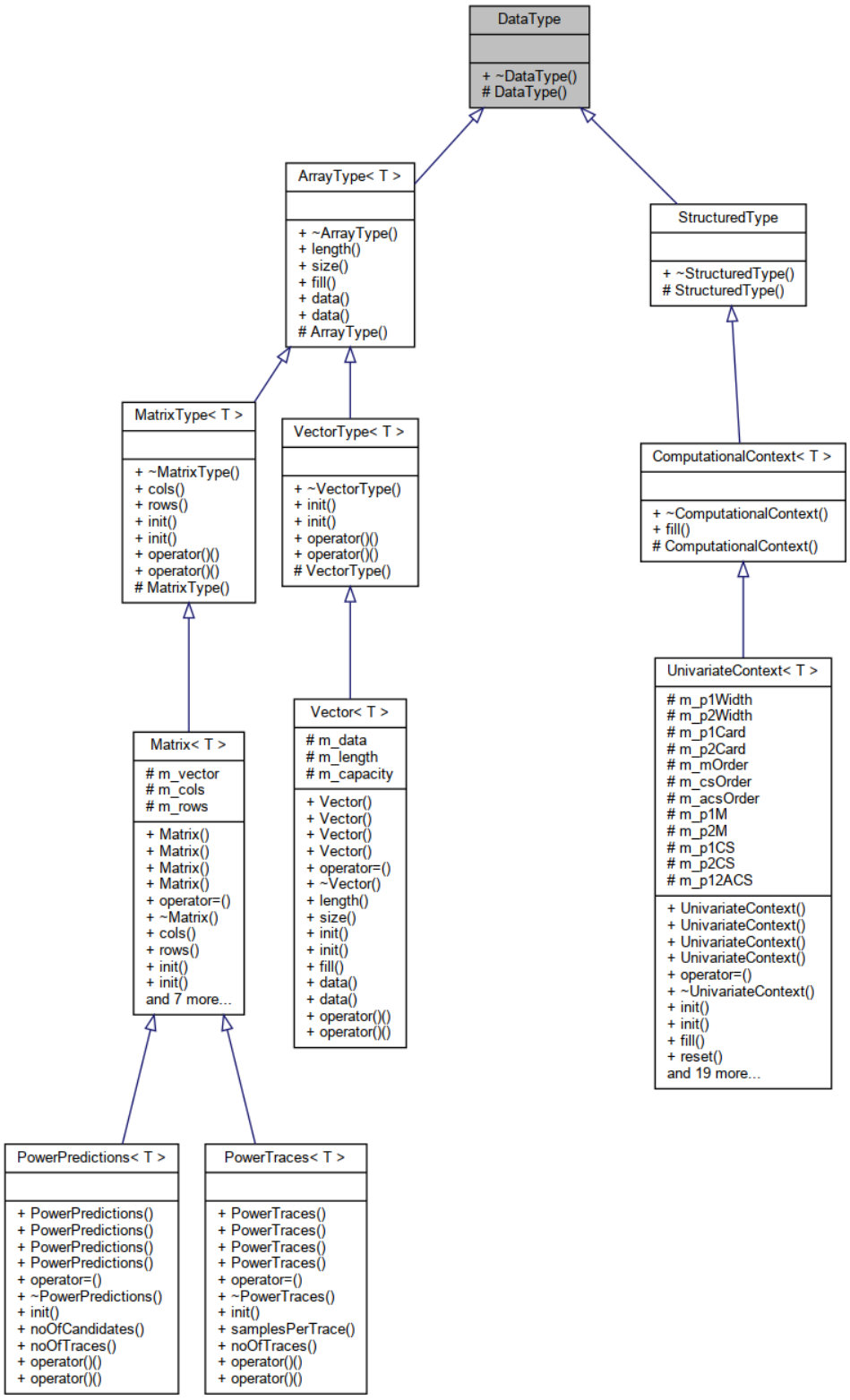


Figure 2.2: Inheritance diagram for toolkit's data types.

$length * sizeof(T)$, where T is scalar type), **data()** which returns pointer to the underlying memory, and **fill(T)**, which allows for the initialization of the array.

The array types are further distinguished as VectorType and MatrixType. **VectorType** is designed for one-dimensional arrays, which is considered by it's **init(size)** method, taking only one size parameter, and it's **operator()(size)** accessor method. **MatrixType**, on the other hand, is designed for two-dimensional arrays, and so are it's **init(cols, rows)** and **operator()(cols, rows)** methods.

Vector and **Matrix** are class templates of the basic implementations of these containers. Matrix uses Vector in terms of composition (not inheritance) and simply maps the two-dimensional indexes to the underlying C-style matrix storage provided by Vector. This approach is selected (opposite to the vector of vectors) in order to obtain maximum control over memory locality.

All the described containers have their copy constructors and copy assignment operators deleted. On the other hand, **move constructors** and **move assignment operators** are implemented and used by file handling functions.

2.7.2 Structured Data Types: Univariate Statistical Context

UnivariateContext is a container class template usable for two-population moment-based statistical analysis. It is initialized with **firstWidth**, **secondWidth**; and **mOrder**, **csOrder** and **acsOrder** parameters.

firstWidth and **secondWidth** parameters determine the widths of first and second population (e.g. number of samples per trace, and/or number of key candidates).

The context provides Raw Moment Vectors and Central Moment Sum Vectors for each population, and Adjusted Central Moment Sum Matrices for the cartesian product of the populations, for every order up to the orders specified during initialization (**mOrder**, **csOrder**, **acsOrder**). The order index starts at 1, with exception for central moment sum vectors, where index starts at 2 (first-order central moment of any variable is constant 0). Besides these vectors and matrices, the context stores cardinality of each population.

All the elements are accessible using methods **p1Card()**, **p1M(order)**, **p1CS(order)** for first population, **p2Card()**, **p2M(order)**, **p2CS(order)** for second population and **p12ACS(order)**, returning either const or non-const references. Similar to the Vector and Matrix, UnivariateContext has copy constructors and copy assignment operators deleted as well, while move constructors and move assignment operators are implemented and preferred.

2.7.3 File Handling Support

A set of function templates was designed, to provide a file handling support for class templates described above.

Function template **fillArrayFromFile** takes an ArrayType reference and file stream reference as arguments and fills the array with data from given stream. This means that the array needs to be initialized to the proper size before call. Function template **writeArrayToFile** writes specified ArrayType to the specified file stream.

Function templates **readContextFromFile** (**writeContextToFile**, resp.) read (write, resp.) the above specified statistical context. Statistical context has its own file format, where metadata are encoded first, followed by the ArrayTypes. Therefore, readContextFromFile returns the statistical context (using move semantics), instead of filling an initialized one as in case of fillArayFromFile function template (which is, however, used in implementation of this template).

All the function templates defined above depend on a correct file stream position. They do not perform any kind of position-seeking operation. This allows e.g. for loading more statistical contexts from a single file stream.

Function templates **loadPowerTraceFromFile**, **loadCorrelationTraceFromFile**, and **loadTValuesFromFile** are useful for loading a single power, correlation or t-values trace (i.e. matrix row) from filestream. These function templates receive necessary offset values (e.g. number of samples per trace) and return Vector filled with data. Unlike all the previous functions, these function templates do not depend on a correct file stream position.

2.7.4 Exceptions

To provide a safe mechanism for error handling, two types of exceptions are defined. First one is **RuntimeException**, which is preffered for situations where the user has no direct influence on the error that has occured (e.g. an error in computation or a timeout during communication). Second one is **InvalidInputException**, which is preffered when the error is caused by invalid user input (e.g. a wrong number of power traces).

Both exceptions are derived from the base class Exception, which is derived from std::exception in order to provide maximum compatibility. Both exceptions allow for setting a string error message, optionally also with an error code.

2.8 Plug-ins

In order to provide maximum flexibility, many of the designed utilities basicly serve as interfaces for various plug-in modules. The plug-in modules adhere to the abstract base classes, which define the modules' interfaces. These are described in following sections.

Common methods implemented in every plug-in module include **getPluginName()** and **getPluginInfo()**, which return basic user metadata. Some of the plug-in modules also implement the **queryDevices()** method, which

returns the Platform or/and Device IDs required for plug-in module initialization. Another methods implemented in every plug-in module are **init(...)**, which must be called before any other method on the plug-in is called (excluding getPluginName, getPluginInfo and queryDevices), and **deinit()**, which should be called when the module is no longer needed (and which should also be called by the plug-in's destructor).

Most plug-ins also receive an external **param** string, allowing for further customisation of the initialization procedure.

2.8.1 Measurement Scenario

Measurement scenario plug-in is loaded by the **meas** Measurement utility. It contains the measurement semantics itself and it is therefore target-specific.

2.8.1.1 Plug-in interface

```
1  class Measurement {
2
3  public:
4
5      virtual ~Measurement() {}
6
7      virtual QString getPluginName() = 0;
8      virtual QString getPluginInfo() = 0;
9
10     virtual void init(const char * param) = 0;
11     virtual void deInit() = 0;
12
13     virtual void run(const char * measurementId, size_t
           measurements, Oscilloscope * oscilloscope, CharDevice *
           charDevice) = 0;
14
15 };
```

2.8.1.1.1 run This is the primary method of the Measurement interface. It receives an ID (set by the utility), number of measurements requested, and either initialized Oscilloscope and Character Device plug-in modules or nullptr(s).

The method is also responsible for creating all the output files.

2.8.2 Oscilloscope

Oscilloscope plug-in is loaded by the **meas** Measurement utility. It serves as an oscilloscope interface, allowing for setting the oscilloscope channels, timing and trigger, and for capturing and downloading power traces.

2.8.2.1 Plug-in interface

```
1  class Oscilloscope {
2
3  public:
4
5      enum class Coupling {
6          AC,
7          DC
8      };
9
10     enum class Impedance {
11         R50,
12         R1M
13     };
14
15     enum class BandwidthLimiter {
16         FULL,
17         F20MHZ,
18         F25MHZ
19     };
20
21     enum class TriggerSlope {
22         RISING,
23         FALLING,
24         EITHER
25     };
26
27     virtual ~Oscilloscope() {}
28
29     virtual QString getPluginName() = 0;
30     virtual QString getPluginInfo() = 0;
31
32     virtual void init(const char * filename) = 0;
33     virtual void deInit() = 0;
34
35     virtual QString queryDevices() = 0;
36
37     virtual void setChannel(int & channel, bool & enabled,
           Coupling & coupling, Impedance & impedance, int & rangemV,
            int & offsetmV, BandwidthLimiter & bwLimit) = 0;
38
39     virtual void setTrigger(int & sourceChannel, float & level,
           TriggerSlope & slope) = 0;
40     virtual void unsetTrigger() = 0;
41
42     virtual void setTiming(float & preTriggerRange, float &
           postTriggerRange, size_t & samples, size_t & captures) =
           0;
43
44     virtual void run() = 0;
45     virtual void stop() = 0;
46
47     virtual size_t getCurrentSetup(size_t & samples, size_t &
```

```
        captures) = 0;
48
49      virtual size_t getValues(int channel, PowerTraces<int16_t> &
            traces) = 0;
50      virtual size_t getValues(int channel, int16_t * buffer, size_t
             len, size_t & samples, size_t & captures) = 0;
51
52  };
```

2.8.2.1.1 init Oscilloscope's initialization method accepts a string parameter. This is module-specific parameter (it could be e.g. oscilloscope's ID)

2.8.2.1.2 setChannel This method sets channel settings, including range or offset. When a parameter or parameters combination is invalid, the module sets parameters to the closest satisfactory values and updates the referenced variables. The **rangemV** is +- symmetric range of the channel in millivolts, i.e. rangemV=100 results in -100mV to +100mV channel range. **offsetmV** is analog offset that is added to the channel before digitalization.

2.8.2.1.3 (un)setTrigger The method (un)sets the edge trigger on defined channel. The **level** float parameter sets the trigger threshold, where 0 stands for the lowest ADC (voltage range) value, and 1 stands for the highest ADC (voltage range) value, on the selected channel.

E.g., with channel's rangemV=100 and offsetmV=-50, level=0.75 stands for threshold voltage 0 V.

2.8.2.1.4 setTiming This method accepts pre-trigger range and post-trigger range float parameters in seconds, and a number of samples per power trace that the user wishes for. When possible, the module computes and sets the nearest sampling frequency to satisfy the requested time range settings, and updates all three referenced variables appropriately. While the time range is almost always settable, the sampling frequency (thus number of samples) may be forced by the oscilloscope device. The interface also allows for setting more captures per oscilloscope run, when the device supports such thing.

2.8.2.1.5 run This method runs the oscilloscope. When triggered, the oscilloscope waits for the trigger event to occur and captures the power trace(s). When not triggered, the digitalization of channels is performed immediately.

2.8.2.1.6 stop This method stops the oscilloscope.

2.8.2.1.7 getCurrentSetup This method returns (using referenced variables) the current samples per trace and captures per run timing settings.

2.8.2.1.8 getValues This method waits for the oscilloscope acquisition to complete (as defined by previous channel, timing and trigger settings) and then returns the captured power traces, either in generic memory buffer, or in the referenced PowerTraces variable.

2.8.3 Character Device

Character Device plug-in is loaded by the **meas** Measurement utility. It serves for reading and writing to/from character device. It's interface is specifically designed for attaching to a terminal device, however, since the terminal parameters have default values specified, they can be ignored by both user and the plug-in, allowing to create a character device plug-in e.g. for a Smart Card.

2.8.3.1 Plug-in interface

```
1  class CharDevice {
2
3  public:
4
5      virtual ~CharDevice() {}
6
7      virtual QString getPluginName() = 0;
8      virtual QString getPluginInfo() = 0;
9
10     virtual void init(const char * filename, int baudrate = 9600,
           int parity = 0, int stopBits = 1) = 0;
11     virtual void deInit() = 0;
12
13     virtual QString queryDevices() = 0;
14
15     virtual void setTimeout(int ms = 5000) = 0;
16
17     virtual size_t send(const VectorType<uint8_t> & data) = 0;
18     virtual size_t receive(VectorType<uint8_t> & data) = 0;
19
20     virtual size_t send(const VectorType<uint8_t> & data, size_t
           len) = 0;
21     virtual size_t receive(VectorType<uint8_t> & data, size_t len)
           = 0;
22
23     virtual size_t send(const uint8_t * buffer, size_t len) = 0;
24     virtual size_t receive(uint8_t * buffer, size_t len) = 0;
25
26 };
```

2.8.3.1.1 init The initialization method receives a string parameter (e.g. filename or COM port) and the usual terminal parameters (baudrate, parity,

stopbits). The baudrate must be supported by the underlying hardware. Parity 0 stands for no parity, 1 stands for odd parity and 2 stands for even parity. Allowed stopbits values are 1 and 2. Since these parameters have default values, they can be ignored by both user and the module, which can be used for non-terminal devices as well.

2.8.3.1.2 setTimeout This method sets the communication timeout (time to wait for any read/write operation to happen), in milliseconds.

2.8.3.1.3 send The send methods send the data out of the referenced Vector or from a generic memory buffer.

2.8.3.1.4 receive The receive methods receive the data and saves them into the referenced Vector or into a generic memory buffer.

When the Vector referenece without length versions of the send/receive functions are used, the amout of data to send/receive is deduced from the referenced Vector length.

2.8.4 Block Data Processing

Block Data Processing plug-in is loaded by the **prep** Preprocessing utility. It allows for generic (pre-)processing of block data, e.g. for creating CPA power predictions.

2.8.4.1 Plug-in interface

```
1  class BlockProcess {
2
3  public:
4
5      virtual ~BlockProcess() {}
6
7      virtual QString getPluginName() = 0;
8      virtual QString getPluginInfo() = 0;
9
10     virtual void init(const char * param) = 0;
11     virtual void deInit() = 0;
12
13     virtual void processBlockData(MatrixType<uint8_t> & data,
           const char * id) = 0;
14
15 };
```

2.8.4.1.1 processBlockData This is the primary method of the BlockProcess interface. It receives a MatrixType<uint8_t> reference (data block per row) and processes it. The method is also responsible for any output files.

2.8.5 Power Traces Processing

Power Traces Processing plug-in is loaded by the **prep** Preprocessing utility. It allows for generic (pre-)processing of (int16_t) power traces, e.g. when performing higher-order analysis.

2.8.5.1 Plug-in interface

```
class TracesProcess {

public:

    virtual ~TracesProcess() {}

    virtual QString getPluginName() = 0;
    virtual QString getPluginInfo() = 0;

    virtual void init(const char * param) = 0;
    virtual void deInit() = 0;

    virtual void processTraces(PowerTraces<int16_t> & traces,
        const char * id) = 0;

};
```

2.8.5.1.1 processTraces This is the primary method of the TracesProcess interface. It receives a PowerTraces (i.e. MatrixType, power trace per row) reference and processes it. The method is also responsible for any output files.

2.8.6 CPA Computation Engine

CPA Computation Engine plug-in is loaded by the **stan** Statistical Analysis utility. It implements context manipulation functions (create a context, merge contexts, finalize the context) for CPA (int16_t power traces and uint8_t power predictions).

2.8.6.1 Plug-in interface

```
class CpaEngine {

public:

    virtual ~CpaEngine() {}

    virtual QString getPluginName() = 0;
    virtual QString getPluginInfo() = 0;

```

```
10      virtual void init(int platform, int device, size_t noOfTraces,
            size_t samplesPerTrace, size_t noOfCandidates, const char
            * param) = 0;
11      virtual void deInit() = 0;
12
13      virtual QString queryDevices() = 0;
14
15      virtual void setConstTraces(bool constTraces = false) = 0;
16
17      virtual UnivariateContext<double> createContext(const
            PowerTraces<int16_t> & powerTraces, const PowerPredictions
            <uint8_t> & powerPredictions) = 0;
18
19      virtual void mergeContexts(UnivariateContext<double> &
            firstAndOut, const UnivariateContext<double> & second) =
            0;
20
21      virtual Matrix<double> finalizeContext(const UnivariateContext
            <double> & context) = 0;
22
23  };
```

2.8.6.1.1 init The initialization function takes the ID of platform and device to perform the computation on, and dimensions of the input data (number of power traces, number of samples per trace and number of key candidates).

2.8.6.1.2 setConstTraces The CPA create operation is often performed number of times with different power predictions, but the same power traces. Setting constant traces using this method signals that the power traces won't change during multiple calls to createContext method, thus allowing for further optimization of the computation.

2.8.6.1.3 createContext This method returns CPA UnivariateContext created from given power traces and power predictions.

2.8.6.1.4 mergeContexts This method merges two given CPA UnivariateContexts and stores the result in the first operand.

2.8.6.1.5 finalizeContext This method evaluates given UnivariateContext and returns CPA correlation matrix.

2.8.7 t-test Computation Engine

t-test Computation Engine plug-in is loaded by the **stan** Statistical Analysis utility. It implements context manipulation functions (create a context, merge contexts, finalize the context) for t-test leakage analysis (int16_t power traces).

2.8.7.1 Plug-in interface

```
1  class TTestEngine {
2
3  public:
4
5      virtual ~TTestEngine() {}
6
7      virtual QString getPluginName() = 0;
8      virtual QString getPluginInfo() = 0;
9
10     virtual void init(int platform, int device, size_t
           noOfTracesRandom, size_t noOfTracesConst, size_t
           samplesPerTrace, const char * param) = 0;
11     virtual void deInit() = 0;
12
13     virtual QString queryDevices() = 0;
14
15     virtual UnivariateContext<double> createContext(const
           PowerTraces<int16_t> & randTraces, const PowerTraces<
           int16_t> & constTraces) = 0;
16
17     virtual void mergeContexts(UnivariateContext<double> &
           firstAndOut, const UnivariateContext<double> & second) =
           0;
18
19     virtual Matrix<double> finalizeContext(const UnivariateContext
           <double> & context) = 0;
20 };
```

2.8.7.1.1 init Similar to the CPA engine, this initialization function takes the ID of platform and device to perform the computation on, and dimensions of the input data (number of random/constant power traces, number of samples per trace).

2.8.7.1.2 createContext This method returns t-test UnivariateContext created from given power traces.

2.8.7.1.3 mergeContexts This method merges two given t-test UnivariateContexts and stores the result in the first operand.

2.8.7.1.4 finalizeContext This method evaluates given UnivariateContext and returns matrix with t-values in the first row, and degrees of freedom in the second row.

2.8.8 CPA Matrix Evaluation

CPA Matrix Evaluation plug-in is loaded by the **correv** Correlation Evaluation utility. It selects a key candidate and sample based on implemented metrics (e.g. maximum coefficient or maximum edge)

2.8.8.1 Plug-in interface

```
1  class CpaCorrEval {
2
3  public:
4
5      virtual ~CpaCorrEval() {}
6
7      virtual QString getPluginName() = 0;
8      virtual QString getPluginInfo() = 0;
9
10     virtual void init(const char * param) = 0;
11     virtual void deInit() = 0;
12
13     virtual void evaluateCorrelations(MatrixType<double> &
           correlationMatrix, size_t & sample, size_t & keyCandidate)
           = 0;
14
15 };
```

2.8.8.1.1 evaluateCorrelations This is the primary method of the CpaCorrEval interface. It searches for an element in given correlation matrix, which maximizes given criteria, and returns this element's index: sample and key candidate.

2.8.9 CPA Keyguess Evaluation

CPA Keyguess Evaluation plug-in is loaded by the **correv** Correlation Evaluation utility. It evaluates a Vector of key candidates (i.e. a keyguess) and returns the cipher key.

2.8.9.1 Plug-in interface

```
1  class CpaKeyEval {
2
3  public:
4
5      virtual ~CpaKeyEval() {}
6
7      virtual QString getPluginName() = 0;
8      virtual QString getPluginInfo() = 0;
9
```

```
10      virtual void init(const char * param) = 0;
11      virtual void deInit() = 0;
12
13      virtual Vector<uint8_t> evaluateKeyCandidates(const VectorType
            <size_t> & keyCandidates) = 0;
14
15  };
```

2.8.9.1.1 evaluateKeyCandidates This is the primary method of the CpaKeyEval interface. It receives a Vector of key candidate indexes and evaluates it into a cipher key.

Chapter 3

Implementation

The C/C++ programming language has already been selected in the assignment of this thesis. Additionally, a multiplatform application framework Qt5 [53] was selected to provide functionality such as dynamic library plug-in support or command line arguments parsing.

3.1 Command-line Utilities

All the command-line utilities are designed in a similar fashion: first the command line arguments are parsed. If help, version or query parameter is set, the appropriate output is printed out and the application quits. When none of these parameters is set, the application looks for function-determining parameters such as selection of a plug-in module. When the requested task is determined, the application looks for other parameters required to perform this task. When all the parameters are succesfully found and set, they are saved and the required task is enqueued for Qt's event loop to process.

Common methods include, besides the command-line parsing, a method searching for available plug-in modules (in ./plugins/lowercaseinterfacename directory) and for printing out their IDs (and sometimes also available Device IDs). Another typically implemented method is for loading the specified plug-in module according to given Plug-in Module ID.

Both command-line arguments parsing and plug-in handling are implemented using Qt framework.

3.1.1 Measurement Utility

Measurement utility (**meas**) is fairly simple in the terms of implemented tasks: only one (but complex) task needs to be done, i.e. load all the specified plug-in modules, configure them and launch the loaded measurement scenario.

The function-determining parameter here is Measurement Scenario Plug-in Module. When it is not defined, the application quits with "Nothing to do."

message. When it is defined, at least a number of measurements must be set as well. Oscilloscope Plug-in Module and Character Device Plug-in Module may be specified as well, however they are not required for a generic measurement scenario. If a task is determined and all the necessary parameters are set (at least Measurement Scneario and number of measurements), the measurement procedure is enqueued (Meas::run()) for the event loop to process.

The run() method, and all the other local (Meas) methods it uses, is maximally verbose. First, the specified Measurement Scenario Plug-in Module is loaded and initialized. Then the Oscilloscope and Character Device Plug-in Modules are loaded (if they are specified), initialized and configured according to the specified JSON configuration files (if they are specified). This is done by methods initConfigOscilloscope() and initConfigChardevice(), which search for the configuration and use module's API to propagate the settings, while being maximally verbose about the required and real (after setup) parameters. After this, the Measurement Scneario is finally run. In the end, all the modules are deinitialized and the application quits.

The oscilloscope configuration (initConfigOscilloscope()) is performed in three stages, first the channel settings are applied, then the trigger settings and finally the timing settings (which may depend on channel settings).

3.1.2 Preprocessing Utility

Preprocessing utility (**prep**) may launch one of the following tasks: block data preprocessing or power traces preprocessing. These tasks differ primary in the data types that are being used.

When Power Traces Processing Plug-in Module parameter is set, the power traces file, number of traces and number of samples per trace must be set as well. When these conditions are met, the Trace Preprocessing Task is enqueued. Similarly when Block Data Processing Plug-in Module is set, the data file, number of block and block lenght must be set. Then the Block Preprocessing Task is enqueued.

Both these tasks are similar, except for the module they use, and for the datatypes they work with. First the specified Plug-in Module is loaded and initialized. Then the power traces/block data memory is allocated and input files are being read. After this, the loaded data are processed by the loaded module (which also takes care of the output). In the end, the loaded modules are deinitialized and the application quits.

3.1.3 Statistical Analysis Utility

Statistical Analysis utility (**stan**) is probably the most complicated in the terms of the tasks it may perform. These are CPA Context Create, CPA Context Merge, CPA Context Finalize, t-test Context Create, t-test Context Merge and t-test Context Finalize. The first three tasks use CPA Computation

Engine Plug-in Module, while the last three tasks use t-test Computation Engine Plug-in Module. The reason these tasks are separated the way they are is because of different input data types used for each task, and because of effort for maximum performance (thus maximum code optimization).

Only one of the Computation Modules parameters may be set: either a CPA Computation Module or a t-test Computation Module. When set, one of the functions is looked for: "create", "merge" or "finalize" as described in Toolkit Analysis. Different combinations of Module type and of a function result in different parameters required for the task to be enqueued. The CPA Create function requires random traces file, number of traces, number of samples per trace, power predictions file, number of prediction sets and number of key candidates. The t-test Create function requires random and constant traces files, number of random and of constant traces, and number of samples per trace. Merge functions require two contexts and Finalize functions require one context, in case of CPA both functions also require number of contexts in a single file. When the function is determined and all the required parameters are set, one of the six tasks is enqueued.

All the tasks first load specified computation module and initialize it. Then the input files are opened, memory is allocated and the data are loaded. After all the required data are ready, one of the createContext, mergeContexts or finalizeContext functions is called from computation module's API. When creating CPA contexts (one for each predictions set), the power prediction sets are loaded successively before each createContext call. In the end, the output is written to the output files, computation module is deinitialized and the application quits.

3.1.4 Correlation Evaluation Utility

Correlation Evaluation utility (**correv**) has only one task to perform: evaluate the correlation matrices (the result of a CPA attack) and then evaluate resulting key candidates to a valid cipher key.

Both Matrix Evaluation and Keyguess Evaluation Module parameters must be set. Correlation matrices file, number of key candidates, number of samples per trace and number of correlation matrices in file must be specified as well.

First, both modules are loaded and initialized. Then the Matrix Evaluation's evaluateCorrelations function is called as many times as is the number of correlation matrices, and the obtained key candidates are stored in a Vector, forming a raw keyguess. This Vector is then evaluated using Keyguess Evaluation module, which returns uint8_t Vector containing cipher key. This key is printed to the standard output in hexadecimal ascii format. In the end, both modules are deinitialized and the application quits.

3.1.5 Visualisation Utility

Visualisation utility (**visu**) is the only utility in this toolkit which doesn't use any custom plug-in modules. It's task is to generate the specified plot containing either power traces, correlation traces or t-values (or generally double values array). This plot is either displayed in a graphical window, or saved to a file. To implement this functionality, Qt Charts module is used, along with Qt Widgets (which provide graphical toolkit and raster output functionality) and Qt Svg (which provides vector output functinality).

To determine a function, at least one of these parameters needs to be set: display, save. When save is set, the width and height of image must be set too. Then the application allows to set a power traces file (application then also requires number of traces and number of samples per trace), a correlation matrices file (application then also requires number of matrices, number of samples per trace and number of key candidates) or a t-values file (number of samples per trace is required then). It is also possible to set real data ranges: power traces voltage range and time range; and the chart's title. Then the time series to plot are parsed and being stored in a list. If no error occurs (e.g. plotting power traces with no power traces file defined), the createChart task is enqueued.

When creating a chart, the lists created during parameter parsing are iterated. Required traces are loaded from defined files, they may get processed (if real ranges are set) and Qt's QLineSeries are constructed from them. At most two vertical axes are created when needed: ADC values (or Voltage) axis, and a double axis (for correlation coefficients or t-values). Horizontal axis is shared among all plotted traces.

After the chart is done, it may be saved to a specified file and/or displayed in a graphical window. When saving a chart, either QPixmap or QSvgGenerator objects are used to produce a compressed or a vector image. The supported raster formats include all formats supported by platform's Qt library, which usually include at least PNG and JPEG.

3.2 Character Device Plug-ins

Character Device Plug-ins are used by the **meas** Measurement utility. They are useful e.g. for communication with a cryptographic device

A character device plug-in is implemented in this thesis: Serial Port communication Plug-in.

3.2.1 SerialPort Character Device Plug-in

SerialPort Character Device plug-in module provides an interface for native operating system terminal device on Windows or POSIX-compatible platform.

3.2.1.1 init

For initialization of the module, a filename and terminal parameters are required. Under POSIX, the filename is terminal device's filename, e.g. "/dev/ttyUSB0". On Windows, the filename is terminal port name, e.g. "COM1" or "\\.\COM10".

Supported baudrates are:

- 110,
- 300,
- 600,
- 1200,
- 2400,
- 4800,
- 9600 (this is default value),
- 19200,
- 38400,
- 57600,
- 115200.

Futhermore, on Windows, following baudrates are available:

- 128000,
- 256000,

while on POSIX, following baudrate is also available:

- 230400.

The parity and stopbit settings are implemented as designed by Character Device interface. Flow control is disabled and no flow control (hardware, software, Xon) is supported by this module.

The method also sets the terminal in raw mode, i.e. no transformation whatsoever is performed upon sent/received data.

3.2.1.2 setTimeout

This method sets the timeout for read/write operations, in milliseconds. On POSIX system, deciseconds are used (the millisecond value gets rounded).

3.2.1.3 send/receive

Send and receive methods are implemented using operating system's native calls, POSIX read/write or Windows readFile/writeFile.

3.3 Oscilloscope Plug-ins

Oscilloscope Plug-ins are used by the **meas** Measurement Utility. They are suitable for controlling an oscilloscope and downloading the captured data.

Two different oscilloscope plug-ins are implemented in this thesis: Keysight 3000 series [54] oscilloscope and PicoScope 6000 series [55] oscilloscope.

To implement the Keysight Oscilloscope plug-in, a communication layer is required between an application and the oscilloscope. This layer is provided by a SCPI Device class, which allows communication with any compliant oscilloscope. For the PicoScope plug-in, this layer is provided by the PicoScope SDK.

3.3.1 SCPI Device

Standard Commands for Programmable Instruments [56] (SCPI) is a standard defining syntax that is used for remote control by many oscilloscopes.

In order to implement any SCPI based oscilloscope module (e.g. Keysight 3000 series Oscilloscope), a communication layer is required, that sends SCPI commands to the oscilloscope and receives data back from it. This layer is provided by ScpiDevice class, which on Windows uses Virtual Instrument Software Architecture [57] (VISA) libraries, on Linux it takes advantage of native USBTMC class [58] kernel driver.

The ScpiDevice class offers following methods: sendString, receiveString, queryString, sendIEEEBlock, receiveIEEEBlock, queryIEEEBlock and checkForInstrumentErrors. There methods are further explained in following subsections.

3.3.1.1 init

The ScpiDevice is initialized either with a VISA address (on Windows) or with a device filename (on Linux). This is done by either VISA library calls or POSIX open function.

3.3.1.2 sendString/receiveString

These methods send a string or receive a string from oscilloscope device. Either VISA calls (viBufWrite, viScanf) are used, or standard POSIX calls upon usbtmc device file (read/write) are used.

3.3.1.3 sendIEEEBlock/receiveIEEEBlock

Besides strings, binary block of data can be transferred to/from oscilloscope device. The data are sent and received using block data format described in IEEE-488.2 [59].

3.3.1.4 queryString/queryIEEEBlock

These methods send a string (the query) and then wait to receive either string response or block of binary data back from the oscilloscope.

3.3.1.5 checkForInstrumentErrors

This function sends a standard SCPI command in order to query the device for errors:

```
1 :SYSTem:ERRor?
```

and returns either 0 when no errors were found, or the error code of the first error returned.

3.3.2 Keysight 3000 Series Oscilloscope Plug-in

The Keysight (formerly Agilent) 3000 series Oscilloscope plug-in is implemented using ScpiDevice class, described earlier. This oscilloscope's supported commands (and their syntax) are described in [54].

3.3.2.1 setChannel

The oscilloscope's channels are set using a series of :CHANnel<n> SCPI commands sent to the oscilloscope. For example:

```
1 :CHANnel1:COUPling DC
2 :CHANnel1:IMPedance FIFTy
3 :CHANnel1:RANGe 100mV
4 :CHANnel1:OFFSet 0mV
5 :CHANnel1:BWLimit 0
```

After the settings is sent, the parameters are queried back to confirm real set values, and referenced input variables are updated appropriately.

3.3.2.2 setTrigger

The oscilloscope's trigger settings is set using a series of :TRIGger SCPI commands. For example:

```
1 :TRIGger:MODE EDGE
2 :TRIGger:EDGE:SOURce CHANnel1
3 :TRIGger:EDGE:SLOPe POSitive
```

3.3.2.3 setTiming

The oscilloscope's timebase is set using a series of :TIMebase SCPI commands. For example:

```
:TIMebase:MODE MAIN
:TIMebase:REFerence CENTer
:TIMebase:RANGe 1e-6
:TIMebase:POSition 0
```

On this oscilloscope, the user is not able to affect the sampling frequency. To obtain a number of samples per power trace before any measurements are done, a dummy measurement must be performed:

```
:WAVeform:POINts:MODE RAW
:WAVeform:FORMat WORD
:WAVeform:UNSigned 0
:WAVeform:BYTeorder LSBFirst
:STOP;*OPC?
:SINGle;:TRIGger:FORCe
```

After this, the driver needs to wait for the acquisition to complete. This is done by reading the Operating Status Word Register and checking the 4th least significant bit of an answer:

```
:OPERegister:CONDition?
```

When the acquisition is done, we can finally read the number of samples per power trace, which apply to the current oscilloscope settings:

```
:WAVeform:POINts?
```

3.3.2.4 run, getValues

Arming the oscilloscope (run) with trigger set is done by a SCPI command:

```
:SINGle
```

When the oscilloscope module is set as untriggered, trigger is forced right after the Single command is sent:

```
:SINGle;:TRIGger:FORCe
```

The getValues call downloads captured power trace from the oscilloscope. First, the method waits for the acquisition to complete by reading the Operating Status Word Register and checking the 4th least significant bit of an answer:

```
:OPERegister:CONDition?
```

When the acquisition is done, the power trace is downloaded by first setting the source channel and then querying the data. For example:

```
1 :WAVeform:SOURce CHANnel1
2 :WAVeform:DATA?
```

The Keysight 3000 series oscilloscope only supports one capture per run.

3.3.3 PicoScope 6000 Series Oscilloscope Plug-in

The PicoScope 6000 series Oscilloscope plug-in is implemented using PicoScope SDK [55] API.

3.3.3.1 init

This module is initialized using a serial number of the oscilloscope. When the filename/Device ID is left blank, the driver opens first PicoScope oscilloscope found.

3.3.3.2 setChannel

Channel settings is done by calls to the PicoScope SDK functions. Unlike the Keysight oscilloscope described earlier, the PicoScope offers quite limited channel range capabilities (9 different ranges between 50mV and 20V). The closest equal or bigger available range is selected by this module and all the input variable references are updated accordingly.

3.3.3.3 setTrigger

The trigger is simply set by a call to the PicoScope SDK, no tricks here.

3.3.3.4 setTiming

On the other hand, unlike Keysight 3000 series oscilloscope, the PicoScope gives user a great power over sampling frequency and number of samples captured. It also supports multiple captures (on multiple trigger events) per oscilloscope run (i.e. rapid block mode).

Based on parameters given by user, a most appropriate available sampling frequency is selected and the oscilloscope mode is set (either block or rapid block mode). The input variable references are updated according to the chosen oscilloscope settings.

3.3.3.5 run, getValues

Running (arming) the oscilloscope is done, once again, simply by a call to the PicoScope SDK.

Obtaining the power trace(s) is done by waiting for the acquisition to complete and downloading the power trace(s) from pre-set oscilloscope buffers.

As mentioned earlier, the PicoScope 6000 series oscilloscope supports multiple captures (at multiple trigger events) per oscilloscope run.

3.4 Measurement Scenario Plug-ins

Two different measurement scenarios are implemented in this thesis, both targeting an AES-128 implementation. Both scenarios require both Oscilloscope and Character Device modules and when run without them, an exception is thrown.

Both scenarios also take advantage of oscilloscope possibly being set for more captures per run. This approach (downloading a larger number of power traces at a time) may increase the measurement time performance. Along with output files, both scenarios also generate a JSON file with metadata.

3.4.1 AES-128 CPA scenario

First, when the scenario is run, the oscilloscope setup is obtained (number of samples per trace and number of captures per oscilloscope run), appropriate memory (PowerTraces Matrix, plain text Matrix and cipher text Matrix) is allocated and output files are opened.

Then the "set cipher key" command (0x01) is sent to the cryptographic device via Character Device module, followed by a preselected 16-byte key.

After this, the oscilloscope is run. Then the "encrypt" command (0x02) is sent to the device, followed by 16 bytes of random data. The 16-byte ciphertext is received back from the device. The encryption is performed as many times, as is the number of captures per oscilloscope run. If requested number of measurements has not been yet satisfied, the oscilloscope is run again and the whole procedure is repeated as many times as necessary. After each oscilloscope run is finished, the power traces are downloaded from the oscilloscope.

Given this, the number of measurements requested must be divisible by the number of captures per oscilloscope run.

After the measurements are done, the obtained data are saved to output files.

3.4.2 AES-128 t-test scenario

The key difference between CPA measurement scenario and t-test measurement scenario is in the data the device is fed. Before each encryption, it is randomly selected whether to encrypt fresh random data or preselected constant data.

The random data power traces and constant data power traces are saved into different output files. For the random measurements, the plaintexts and ciphertexts are saved as well. Therefore, the random traces can be used for

CPA attack, and both random and constant power traces can be used for t-test analysis.

3.5 Block Data Preprocessing Plug-ins

Two block data preprocessing plug-in modules are implemented in this thesis, both for generating power predictions for CPA attack on AES-128 implementations.

The 128-bit AES encryption consists of the Key expansion and the initial (zero) round, followed by 10 rounds. In the Key expansion operation, the cipher key gets expanded into 11 round keys (first one being the cipher key). In the initial round, the plaintext gets xored with the cipher key (i.e. first round key). In the next ten rounds, this value gets further altered using four consecutive operations: SubBytes (i.e. a non-linear 8-bit substitution, i.e. S-Box), ShiftRows (i.e. a circular shift), MixColumns (i.e. a linear transformation) and AddRoundKey (with appropriate round key). In the last round, the MixColumns operation is skipped. [3, 17]

3.5.1 AES-128 First Round Hamming Weight Power Model

This power model exploits the knowledge of the plaintext used during encryption [24]. When the plaintext is known, we (the attacker) can easily perform the initial round, i.e. AddRoundKey operation, for a chosen byte of the plaintext. With all possible key candidates, this gives us 256 possible values after the initial round. This value gets further processed using SubBytes operation, which is usually implemented as a memory look-up. The Hamming weight of this look-up result is used as a power prediction for software implementations of AES. [17]

Using this approach, 256 power predictions are created for every measurement done. The whole process is repeated 16 times for different cipherkey bytes, resulting in 16 power prediction sets, that are saved to a file.

3.5.2 AES-128 Last Round Hamming Distance Power Model

Attacking hardware implementations of AES-128 requires different approach. The power consumption of the CMOS circuits depends rather on transitions made (0 to 1 or 1 to 0). This power model exploits the knowledge of the ciphertext [25] and focuses on the register holding the cipher's working state. Given that the cipher text is the value that was stored in the register after last round, we can perform inverse ShiftRows and inverse SubBytes operation on a chosen byte, and given 256 possible key candidates, perform AddRoundKey operation as well, resulting in 256 different values that were possibly stored in the working register during previous round. Hamming distance between last and 256 last but one possible values in working register gives us our power

predictions. These are based on a Hamming distance power model based on transitions on cipher's working register. [17]

This power model, however, attacks the last round key. This round key can be easily reversed to the cipher key, e.g. using a Keyguess Evaluation plug-in from this thesis.

Using this approach, 256 power predictions are created for every measurement done. The whole process is repeated 16 times for different cipherkey bytes, resulting in 16 power prediction sets, that are saved to a file.

3.6 Statistical Analysis Computation Plug-ins

Statistical Analysis plug-in modules provide a computation engine for **stan** utility, including functions to create, merge or finalize computational contexts. While different plug-ins may implement different functionality, they may also provide the computation on different devices, e.g. using GPU acceleration, distributed computation, etc.

Three plug-ins are implemented in this thesis: Univariate First-Order CPA accelerated on CPU using OpenMP, Univariate First-Order CPA accelerated using OpenCL and Univariate First-Order Welch's t-test.

3.6.1 Univariate First-Order CPA

Note that in this thesis, $\overline{x}$ stands for mean of X, $M_{2,X}$ stands for second-order central moment sum of X and $C_{2,S}$ stands for first-order adjusted central moment sum (covariance sum), as labeled in [60, 26], in contrast to the labeling used in [41]. n stands for the cardinality of the original set (in interative formulas), n_1 stands for cardinality of the first set (in merging formulas).

3.6.1.1 create

The creation of a new computational context in this module is implemented using incremental approach exactly as described in [26]. For both power traces and power predictions, means and second central moments, as well as shared adjusted central moments, are computed using incremental formulas, which provide both good time performance and numerical stability [60, 26]:

$$\overline{x'} = \overline{x} + \frac{x_{n+1} - \overline{x}}{n+1}, \tag{3.1}$$

$$M_{2,X'} = M_{2,X} + (x_{x+1} - \overline{x})(x_{x+1} - \overline{x'}), \tag{3.2}$$

$$C_{2,S'} = C_{2,S} + \frac{n}{n+1}(x_{n+1} - \overline{x})(y_{n+1} - \overline{y}). \tag{3.3}$$

The creation of CPA context is by far the most computationally challenging operation. Above described computations are optimized in the means of

memory access and parallelized over different power samples using OpenMP in order to provide maximum time performance. Comparison with different computational approaches can be found in [26].

3.6.1.2 merge

A CPA context holds means, second central moments and adjusted central moments and cardinalities of two populations. To merge two contexts means to create a context holding means and moments of data sets, as if it was created from both sets that were used for the creation of the two merged contexts combined.

Formulas provided in [41] are used for the computation of new moments:

$$\overline{x_{12}} = \frac{n_1\overline{x_1} + n_2\overline{x_2}}{n_1 + n_2}, \tag{3.4}$$

$$M_{2,X_{12}} = M_{2,X_1} + M_{2,X_2} + n_1 n_2 (\frac{\overline{x_2} - \overline{x_1}}{n_1 + n_2})^2 (n_1 + n_2), \tag{3.5}$$

$$C_{2,S_{12}} = C_{2,S_1} + C_{2,S_2} + \frac{n_2(n_1)^2 + n_1(n_2)^2}{(n_1 + n_2)^2} (\overline{x_{2,X}} - \overline{x_{1,X}})(\overline{x_{2,Y}} - \overline{x_{1,Y}}). \tag{3.6}$$

First the adjusted central moment sums are merged, then the central moment sums and means for both variables are merged.

3.6.1.3 finalize

To finalize the CPA contexts means to create a correlation matrix. Pearson correlation coefficients are easily computed from moments available in the context:

$$r_{X,Y} = \frac{C_{2,S}}{\sqrt{M_{2,X}}\sqrt{M_{2,Y}}}. \tag{3.7}$$

Both merge and finalize operations are typically computationally insignificant in comparison with the create operation, since they are performed on the statistical contexts only.

3.6.2 Univariate First-Order Welch's t-test

3.6.2.1 create and merge

These two operations are practically identical to the CPA computation, except that the adjusted central sums matrix doesn't need to be computed. I.e. only means and second-order central sums are computed.

Since the covariance (first-order adjusted central moment) sum matrix computation is by far the greatest bottleneck of the CPA computation, the t-test takes significantly less time than the CPA attack.

3.6.2.2 finalize

To finalize a t-test context means to compute t-values and degrees of freedom from available statistical moments. This is easily done using equations 1.1 and 1.2. The unbiased variance estimator s^2 is obtained from the second-order central moment sum:

$$s^2 = \frac{1}{n-1} M_2 \tag{3.8}$$

The result is a two-row Matrix, where the first row contains t-values and the second row constains the degrees of freedom, for each sample point.

3.6.3 OpenCL Accelerated CPA

The most computationally demanding CPA operation is context creation, which typically needs to process large amounts of data (power traces, power predictions). Therefore we have decided to accelerate this operation using GPUs (given it's parallel nature, where the computation is performed on each sample point separately). For this purpose, the OpenCL [61] standard for parallel programming was selected. The version 1.2 was selected in order to provide maximum compatibility among different vendors (AMD, nVidia, Intel,...).

In a nutshell, the OpenCL provides support for memory operations between a host (CPU) and a device (e.g. GPU) and for launching kernels (which are programs to be launched paralelly on the device). Before the computation is launched, the kernel code gets compiled during runtime, into the code for the dynamically selected platform. This provides a general and effective multiplatform approach to the computation.

Class template **OclEngine** provides a basic OpenCL layer for further implementations of various algorithms. It is capable of querying all the devices available (queryDevices) and when initialized with Device and Platform ID, it creates and holds the OpenCL Context and OpenCL Command Queue useful for controlling the device.

A derived class template **OclCpaEngine** provides API to perform the CPA context creation operation using OpenCL. It is responsible for allocation of memory buffers on the device, memory transfers and compiling and launching the computational kernels.

3.6.3.1 loadPredictionsToDevice

This method loads the power predictions to the device buffer. Since we usually want to perform more CPA attacks using same power traces, but different power predictions, this method is usually called before each computation is launched.

3.6.3.2 loadTracesToDevice

This method loads power traces to the device buffer.

3.6.3.3 compute

This method launches appropriate kernels in order to create a CPA context from the loaded power traces and power predictions. The method accepts a "sliceSize" parameter, which tells how many measurements should be processed at once. This is because when GPU, that is also being used by the OS for screen rendering, is not responding to the OS for a long time, it is being resetted by the OS. To avoid this, the long-time running kernels are divided into slices, so that the GPU becomes available in the meantime.

After the computation is done, the method downloads the data from the device and saves the CPA context in referenced UnivariateContext.

3.7 Correlation Matrix Evaluation Plug-ins

Correlation Matrix Evaluation plug-ins are used by the **correv** Correlation Evaluation utility. They are useful for distinguishing the right key candidate in a correlation matrix obtained usually as a result of CPA attack. Four matrix evaluation plug-ins are implemented in this thesis.

3.7.1 Maximum/minimum (absolute) coefficient

Three simple key candidate disinguisher plug-ins based on extreme searching are implemented, searching either for **maximum**, **minimum** or **maximum absolute** correlation coefficient value, and returning the key candidate for which this correlation occurs.

3.7.2 Maximum edge on the correlation trace

Maximum Edge distinguisher searches for the maximum edge on the correlation trace, i.e. largest absolute derivative of the correlation traces. This is done by first performing a discrete convolution upon all correlation traces (i.e. rows of the correlation matrix) [19]. The convolutional kernel is an approximation of the Gaussian derivative [62, 19]:

$$G(x,\sigma)' \propto \frac{x}{\sigma^2} \cdot \exp(-\frac{x^2}{\sigma^2}), \tag{3.9}$$

for $x \in \{-\lfloor \frac{d}{2} \rfloor, ...0, ..., \lceil \frac{d}{2} \rceil - 1\}$, where d is the diameter of the kernel (user given parameter) and σ is the Gaussian parameter (user given parameter).

This key distinguisher may perform better than classical Maximum/minimum searching methods, especially in a noisy environment with power switching supplies [19].

3.8 Keyguess Evaluation Plug-ins

Correlation Matrix Evaluation plug-ins are used by the **correv** Correlation Evaluation utility. They are useful for evaluation of the keyguess, i.e. a Vector of key candidates obtained by evaluation of the CPA correlation matrices.

3.8.1 Plain char

This generic plug-in is suitable for attacks aiming at a byte of the cipher key at a time. The key candidate's indexes are mapped directly to the cipher key bytes and the resulting key is returned with no transformation.

3.8.2 Last Round AES-128

This plug-in is suitable when attacking last round key of AES-128. Key candidates are threated as bytes of the cipher key. The last round key gets reversed to the first round key (e.g. the cipher key), which gets returned.

This is done by performing a series of S-Box substitutions and xor operations (also using rcons constants) [3] upon the last round key.

3.9 Build, Release and More Information

The whole toolkit takes advantage of Qt's build system: qmake [63]. The qmake is a multiplatform tool, which parses .pro files containing project specific settings and creates a Makefile for the specified platform (current platform, by default).

The toolkit contains plug-ins that require third-party system libraries to compile against. Path to these libraries is specified in the *config.pri* file in project's root directory.

To build the project under Linux using GNU toolchain, navigate to the project's root directory (containing *sicak.pro* file), and run qmake and make:

```
$ cd .../sicak
$ qmake
$ make
```

Similarly, to build the project under Windows using Microsoft Visual C++ compiler, open Qt developer console, e.g. *Qt 5.12.0 64-bit for Desktop (MSVC 2017)*, setup the environment and continue as in Linux case with the exception of using nmake instead of make:

```
> cd ...\Microsoft Visual Studio\...\VC\Auxiliary\Build
> vcvarsall.bat x86_amd64
> cd ...\sicak
> qmake
> nmake
```

To move all the applications and their plug-ins into an appropriate directory structure (so that applications are able to find the plug-ins), make the *install* target:

```
$ make install
```

```
> nmake install
```

and check the INSTALL directory in the project's root directory. For Windows users to be able to use the compiled utilities, all the Qt's dependencies may need to be deployed in the application's folder (i.e. the INSTALL folder). If this is the case, see the *windeployqt* [64] utility.

Similar, unofficial, but very interesting tool exists for Linux, see the *linuxdeployqt* [65] utility, which is capable of packaging the Linux binaries into an AppImage format, which claims to be sort of a cross-distribution package.

The source codes, along with the User's Guide, the Programmer's Guide and the binary release are available on the enclosed CD or freely on GitHub: `https://petrsocha.github.io/sicak`.

Conclusion

This Master's thesis dealt with a software support for side-channel analysis, which include a wide range of both measurement and analytical tasks. As a result, C/C++ toolkit was created, consisting of five utilities with modular plug-in architecture allowing for plenty various use-cases.

Measurement Utility *mesu* was created, allowing to run a **Measurement Scenario** plug-in. Two scenarios were implemented: CPA attack and t-test leakage assesment, both for AES-128. This utility uses an **Oscilloscope** plug-in allowing for data acquisition; two oscilloscope plug-ins were implemented in this thesis: Keysight 3000 series oscilloscope and PicoScope 6000 oscilloscope. Another plug-in used by the Measurement utility is **Character Device** plug-in, which allows for communication with the device (the serial port plug-in is implemented).

To support generic preprocessing of the measured data, **Preprocessing Utility** *prep* was created, which loads either **Block Data Processing** plug-in or **Power Trace Processing** plug-in. Two plug-ins suitable for creating CPA attack power predictions using either Hamming weight or Hamming distance were implemented.

Statistical Analysis Utility *stan* is useful for moment-based statistical tasks including e.g. correlation-based attacks (CPA) or leakage analysis (t-test). **Computational** plug-ins performing First-Order Univariate CPA attack and Non-Specific Welch's t-test were implemented. The CPA attack is additionally implemented using OpenCL framework, allowing for the computation to be accelerated using GPUs.

To evaluate the results of the CPA attack, **Correlation Evaluation Utility** *correv* was created. This utility uses **Correlation Matrix Evaluation** plug-in, which selects a key candidate based on a specified criteria (plug-ins searching for maximum/minimum (absolute) coefficient and for maximum correlation trace edge were implemented), and **Keyguess Evaluation** plug-in, which computes the cipher key (a basic byte-attack plug-in and plug-in performing AES round key inversion were implemented).

Finally, the **Visualisation Utility** *visu* is useful for plotting the power traces, the correlation traces or generic double values (e.g. t-values) and either displaying the plot or saving it using raster or vector format.

All the utilities are non-interactive and text-based. Their parameters can be set either directly or using JSON configuration files. Configuration files are also generated alongside the output files of presented utilities, allowing for simple and user-friendly chain usage and scripting.

A comparison of Pearson correlation coefficient computation approaches produced while working on this thesis was published in [26]. A novel approach to the correlation matrix evaluation based on edge detection was published in [19]. All the work presented in this thesis is released under a copyleft open-source licence and is available online including source codes.

Bibliography

[1] Vaas, L. Doctors disabled wireless in Dick Cheney's pacemaker to thwart hacking. *Naked Security*, Oct 2013. Available from: `https://nakedsecurity.sophos.com/2013/10/22/doctors-disabled-wireless-in-dick-cheneys-pacemaker-to-thwart-hacking/`

[2] Sicari, S.; Rizzardi, A.; et al. Security, privacy and trust in Internet of Things: The road ahead. *Computer networks*, volume 76, 2015: pp. 146–164.

[3] Daemen, J.; Rijmen, V. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[4] Bogdanov, A.; Knudsen, L. R.; et al. PRESENT: An ultra-lightweight block cipher. In *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2007, pp. 450–466.

[5] Messerges, T. S.; Dabbish, E. A.; et al. Investigations of Power Analysis Attacks on Smartcards. *Smartcard*, volume 99, 1999: pp. 151–161.

[6] Kocher, P.; Jaffe, J.; et al. *Differential Power Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, ISBN 978-3-540-48405-9, pp. 388–397, doi:10.1007/3-540-48405-1_25.

[7] den Boer, B.; Lemke, K.; et al. A DPA attack against the modular reduction within a CRT implementation of RSA. In *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2002, pp. 228–243.

[8] Brier, E.; Clavier, C.; et al. Correlation power analysis with a leakage model. In *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2004, pp. 16–29.

[9] Biham, E.; Shamir, A. Differential fault analysis of secret key cryptosystems. In *Annual international cryptology conference*, Springer, 1997, pp. 513–525.

[10] Tunstall, M.; Mukhopadhyay, D.; et al. Differential fault analysis of the advanced encryption standard using a single fault. In *IFIP international workshop on information security theory and practices*, Springer, 2011, pp. 224–233.

[11] Koeune, F.; Quisquater, J.-J.; et al. A timing attack against Rijndael. 1999.

[12] Osvik, D. A.; Shamir, A.; et al. Cache attacks and countermeasures: the case of AES. In *Cryptographers' Track at the RSA Conference*, Springer, 2006, pp. 1–20.

[13] Rott, J. Intel® Advanced Encryption Standard Instructions (AES-NI). Feb 2012. Available from: `https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni`

[14] Quisquater, J.-J.; Samyde, D. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *Smart Card Programming and Security*, Springer, 2001, pp. 200–210.

[15] Mangard, S.; Oswald, E.; et al. *Power analysis attacks: Revealing the secrets of smart cards*, volume 31. Springer Science & Business Media, 2008.

[16] Moradi, A. Advances in side-channel security. *Habilitation, Ruhr-Universität Bochum*, 2015.

[17] Socha, P.; Brejník, J.; et al. Attacking AES implementations using correlation power analysis on ZYBO Zynq-7000 SoC board. In *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, IEEE, 2018, pp. 1–4.

[18] Mazur, L.; Novotný, M. Differential power analysis on FPGA board: Boundaries of success. In *Embedded Computing (MECO), 2017 6th Mediterranean Conference on*, IEEE, 2017, pp. 1–4.

[19] Socha, P.; Miškovský, V.; et al. Correlation Power Analysis Distinguisher Based on the Correlation Trace Derivative. In *2018 21st Euromicro Conference on Digital System Design (DSD)*, IEEE, 2018, pp. 565–568.

[20] Liu, W.; Wu, L.; et al. Wavelet-Based Noise Reduction in Power Analysis Attack. In *Computational Intelligence and Security (CIS), 2014 Tenth International Conference on*, IEEE, 2014, pp. 405–409.

[21] Schneider, T.; Moradi, A. Leakage assessment methodology. *Journal of Cryptographic Engineering*, volume 6, no. 2, 2016: pp. 85–99.

[22] Moradi, A.; Richter, B.; et al. Leakage Detection with the x2-Test. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, volume 2018, no. 1, 2018: pp. 209–237.

[23] Welch, B. L. The generalization of student's' problem when several different population variances are involved. *Biometrika*, volume 34, no. 1/2, 1947: pp. 28–35.

[24] Schuster, A.; Oswald, E. Differential power analysis of an AES implementation. *Institute for Applied Information Processing and Communications, Graz University of Technology, Tech. Rep. IAIK-TR*, volume 6, 2004: p. 25.

[25] Alioto, M.; Poli, M.; et al. A general power model of differential power analysis attacks to static logic circuits. *IEEE transactions on very large scale integration (VLSI) systems*, volume 18, no. 5, 2010: pp. 711–724.

[26] Socha, P.; Miškovský, V.; et al. Optimization of Pearson correlation coefficient calculation for DPA and comparison of different approaches. In *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2017 IEEE 20th International Symposium on*, IEEE, 2017, pp. 184–189.

[27] Sokolov, D.; Murphy, J.; et al. Design and analysis of dual-rail circuits for security applications. *IEEE Transactions on Computers*, volume 54, no. 4, 2005: pp. 449–460.

[28] Popp, T.; Mangard, S. Masked dual-rail pre-charge logic: DPA-resistance without routing constraints. In *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2005, pp. 172–186.

[29] Jerábek, S.; Schmidt, J.; et al. Dummy Rounds as a DPA countermeasure in hardware. In *2018 21st Euromicro Conference on Digital System Design (DSD)*, IEEE, 2018, pp. 523–528.

[30] Blömer, J.; Guajardo, J.; et al. Provably secure masking of AES. In *International Workshop on Selected Areas in Cryptography*, Springer, 2004, pp. 69–83.

[31] Rivain, M.; Prouff, E. Provably secure higher-order masking of AES. In *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2010, pp. 413–427.

[32] Moradi, A.; Poschmann, A.; et al. Pushing the limits: a very compact and a threshold implementation of AES. In *Annual International Conference*

on the Theory and Applications of Cryptographic Techniques, Springer, 2011, pp. 69–88.

[33] Yu, W.; Köse, S. A lightweight masked AES implementation for securing IoT against CPA attacks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, volume 64, no. 11, 2017: pp. 2934–2944.

[34] Mentens, N.; Gierlichs, B.; et al. Power and fault analysis resistance in hardware through dynamic reconfiguration. In *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2008, pp. 346–362.

[35] Sasdrich, P.; Moradi, A.; et al. Achieving side-channel protection with dynamic logic reconfiguration on modern FPGAs. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2015, pp. 130–136.

[36] Messerges, T. S. Using second-order power analysis to attack DPA resistant software. In *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2000, pp. 238–251.

[37] Chari, S.; Jutla, C. S.; et al. Towards sound approaches to counteract power-analysis attacks. In *Annual International Cryptology Conference*, Springer, 1999, pp. 398–412.

[38] Standaert, F.-X.; Veyrat-Charvillon, N.; et al. The world is not enough: Another look on second-order DPA. In *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, 2010, pp. 112–129.

[39] Bottinelli, P.; Bos, J. W. Computational aspects of correlation power analysis. *Journal of Cryptographic Engineering*, volume 7, no. 3, 2017: pp. 167–181.

[40] Gilbert Goodwill, B. J.; Jaffe, J.; et al. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, 2011, pp. 115–136.

[41] Schneider, T.; Moradi, A.; et al. Robust and one-pass parallel computation of correlation-based attacks at arbitrary order. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, Springer, 2016, pp. 199–217.

[42] Moradi, A.; Mischke, O. How far should theory be from practice? In *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2012, pp. 92–106.

[43] Schramm, K.; Leander, G.; et al. A collision-attack on AES. In *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2004, pp. 163–175.

[44] Moradi, A.; Mischke, O.; et al. Correlation-enhanced power analysis collision attack. In *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2010, pp. 125–139.

[45] Moradi, A.; Standaert, F.-X. Moments-correlating DPA. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, ACM, 2016, pp. 5–15.

[46] Gierlichs, B.; Batina, L.; et al. Mutual information analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2008, pp. 426–442.

[47] ChipWhisperer® by NewAE Technology Inc. Available from: `https://wiki.newae.com/Main_Page`

[48] Riscure Inspector Side Channel Analysis. Available from: `https://www.riscure.com/security-tools/inspector-sca/`

[49] Cees-Bart Breunesse, I. K. Jlsca Side-channel toolkit in Julia. Available from: `ttps://github.com/Riscure/Jlsca`

[50] The R Project for Statistical Computing. Available from: `https://www.r-project.org/`

[51] MathWorks MATLAB. Available from: `https://www.mathworks.com/products/matlab.html`

[52] Wolfram Mathematica: Modern Technical Computing. Available from: `http://www.wolfram.com/mathematica/`

[53] The Qt Company. *Qt Documentation: Qt 5.12*. Available from: `http://doc.qt.io/qt-5/index.html`

[54] Keysight Technologies, Inc. *Keysight InfiniiVision 3000T X-Series Oscilloscopes Programmer's Guide*. Available from: `https://www.keysight.com/upload/cmc_upload/All/3000_series_prog_guide.pdf`

[55] Pico Technology Ltd. *PicoScope 6000 Series Programmer's Guide*. Available from: `https://www.picotech.com/download/manuals/picoscope-6000-series-programmers-guide.pdf`

[56] SCPI Consortium, IVI Foundation. *SCPI-1999 Specification*. Available from: `http://www.ivifoundation.org/docs/scpi-99.pdf`

[57] VXIplug&play Systems Alliance, IVI Foundation. *VISA Specifications*. Available from: `http://www.ivifoundation.org/specifications/default.aspx`

[58] USB Implementers Forum, Inc. *Universal Serial Bus Test and Measurement Class, Subclass USB488 Specification (USBTMC-USB488)*.

[59] *IEEE Standard Codes, Formats, Protocols, and Common Commands for Use with IEEE Std 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation*. IEEE Engineering Management Society, 1992.

[60] Pébay, P. Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments. *Sandia Report SAND2008-6212, Sandia National Laboratories*, volume 94, 2008.

[61] The Khronos Group Inc. *OpenCL 1.2 Reference Pages*. Available from: `https://www.khronos.org/registry/OpenCL/sdk/1.2/docs/man/xhtml/`

[62] Canny, J. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, , no. 6, 1986: pp. 679–698.

[63] The Qt Company. *qmake Manual*. Available from: `http://doc.qt.io/qt-5/qmake-manual.html`

[64] The Qt Company. *Qt for Windows - Deployment*. Available from: `http://doc.qt.io/qt-5/windows-deployment.htm`

[65] probonopd. *linuxdeployqt*. Available from: `https://github.com/probonopd/linuxdeployqt`

Appendix A

Acronyms

AES Advanced Encryption Standard

ARM Acorn RISC Machine

ASIC Application-Specific Integrated Circuit

CPA Correlation Power Analysis

CPU Central Processing Unit

CRT Chinese Remainder Theorem

DPA Differential Power Analysis

FPGA Field-programmable Gate Array

GNU GNU's Not Unix

GPU Graphical Processing Unit

IVI Interchangeable Virtual Instrumentation

JPEG Joint Photographic Experts Group

PNG Portable Network Graphics

RFID Radio-Frequency Identification

RISC Reduced Instruction Set Computer

RSA Rivest–Shamir–Adleman

SCA Side-Channel Analysis

SCPI Standard Commands for Programmable Instruments

SPA Simple Power Analysis

SVG Scalable Vector Graphic

USB Univeral Serial Bus

VISA Virtual Instrument Software Architecture

VME Versa Module Europa

VXI VME eXtensions for Instrumentation

Appendix B

Contents of enclosed CD

- `readme.txt` ..... CD contents description
- `release` ..... release executables
- `src` ..... source codes
  - `sicak` ..... toolkit source codes and documentation
  - `thesis` ..... LaTeX source codes of the thesis
- `thesis` ..... thesis text
  - `DP_Socha_Petr_2019.pdf` ..... thesis text in PDF format

Appendix C

Example of an Oscilloscope JSON Configuration File

```
{

    "channel1": {
        "enabled": true,
        "coupling": "DC",
        "impedance": "50",
        "rangemV": 50,
        "offsetmV": 0,
        "bwLimit": "25MHz"
    },

    "channel2": {
        "enabled": false,
        "coupling": "DC",
        "impedance": "1M",
        "rangemV": 50,
        "offsetmV": 0,
        "bwLimit": "FULL"
    },

    "channel3": {
        "enabled": true,
        "coupling": "DC",
        "impedance": "1M",
        "rangemV": 1000,
        "offsetmV": 0,
        "bwLimit": "FULL"
    },

    "channel4": {
        "enabled": false,
        "coupling": "DC",
        "impedance": "1M",
        "rangemV": 50,
```

```
35          "offsetmV": 0,
36          "bwLimit": "FULL"
37      },
38
39      "trigger": {
40          "enabled":true,
41          "channel": 3,
42          "level": 0.7,
43          "slope": "rising"
44      },
45
46      "timing": {
47          "preTriggerRange": 0.0,
48          "postTriggerRange": 3.2e-6,
49          "samples": 2000,
50          "captures": 1
51      }
52
53  }
```

Appendix D

Example of a Character Device JSON Configuration File

```
{
    "baudrate": 115200,
    "parity": 0,
    "stopbits": 1
}
```

Appendix E

Example Usage of meas Utility

E.1 Query Available Plug-ins

```
$ ./meas -Q

Found following measurement scenario plug-ins:

* Plug-in ID: 'random128co', name: 'AES-128 random (command oriented)'
    Description: 'Sends 0x01 followed by cipher key, then N times {0x02 followed by 128 bits of random data}, receives back every cipher text, and captures the power consumption.'

* Plug-in ID: 'ttest128co', name: 'AES-128 t-test (command oriented)'
    Description: 'Sends 0x01 followed by cipher key, then N times {0x02 followed by 128 bits of either random or constant data}, receives back every cipher text, and captures the power consumption'


Found following oscilloscope plug-ins:

* Plug-in ID: 'keysight3000', name: 'Keysight 3000 series oscilloscope (formerly Agilent)'
    * Device ID: 'FILEPATH', where FILEPATH is path to a usbtmc device, e.g. "/dev/usbtmc0"
     Make sure you have permissions to access the file, and the usbtmc module loaded.

* Plug-in ID: 'ps6000', name: 'PicoScope 6000 series oscilloscope'
    * Device ID: 'SERIALNO', where SERIALNO is a serial number of the oscilloscope. Leave empty to let driver automatically select first device found.
     On Linux, make sure you have permissions to access the device (/dev/usb/...).

```

```
22
23 Found following character device plug-ins:
24
25 * Plug-in ID: 'serialport', name: 'Win32/POSIX Serial Port'
26     * Device ID: 'FILEPATH', where FILEPATH is path to a terminal
          device, e.g. "/dev/ttyUSB0"
```

E.2 Launch Measurement

This command launches the ttest128co measurement scenario using keysight3000 oscilloscope and serialport character device modules.

```
1  $ ./meas -I id -M ttest128co -O keysight3000 -R /dev/usbtmc0 -S ./
       conf.json -C serialport -D /dev/ttyUSB0 -E ./conf.json -n 1000
2  SICAK MEASurements 1.0
3  * 20.12.2018 18:25:01 Starting...
4  * Measurement scenario module loaded: 'AES-128 t-test (command
       oriented)'
5  * Oscilloscope module loaded: 'Keysight 3000 series oscilloscope (
       formerly Agilent)'
6  * Oscilloscope successfully opened: '/dev/usbtmc0'
7  * Oscilloscope configuration file found: './conf.json'
8      * Requesting oscilloscope channel settings:
9          * Channel: '1'
10         * Enabled: 'true'
11         * Coupling: 'DC'
12         * Impedance: '50'
13         * Range: -+'50mV'
14         * Offset: '0mV'
15         * Bandwidth Limit: '25MHz'
16     * Real oscilloscope channel settings (after setup):
17         * Channel: '1'
18         * Enabled: 'true'
19         * Coupling: 'DC'
20         * Impedance: '50'
21         * Range: -+'50mV'
22         * Offset: '0mV'
23         * Bandwidth Limit: '25MHz'
24     * Requesting oscilloscope channel settings:
25         * Channel: '3'
26         * Enabled: 'true'
27         * Coupling: 'DC'
28         * Impedance: '1M'
29         * Range: -+'1000mV'
30         * Offset: '0mV'
31         * Bandwidth Limit: 'FULL'
32     * Real oscilloscope channel settings (after setup):
33         * Channel: '3'
34         * Enabled: 'true'
35         * Coupling: 'DC'
36         * Impedance: '1M'
```

```
        * Range: -+'1000mV'
        * Offset: '0mV'
        * Bandwidth Limit: 'FULL'
    * Requesting oscilloscope trigger settings:
        * Enabled: 'true'
        * Source channel: '3'
        * Trigger level: '0.7' for channel range 0..1
        * Edge slope: 'either'
    * Real oscilloscope trigger settings (after setup):
        * Enabled: 'true'
        * Source channel: '3'
        * Trigger level: '0.7' for channel range 0..1
        * Edge slope: 'either'
    * Requesting oscilloscope timing settings:
        * Pre-trigger time range: '0s'
        * Post-trigger time range: '3.2e-06s'
        * Samples: '2000'
        * Captures per run: '100'
    * Real oscilloscope timing settings (after setup):
        * Pre-trigger time range: '0s'
        * Post-trigger time range: '3.2e-06s'
        * Samples: '30000'
        * Captures per run: '1'
* Character device module loaded: 'Win32/POSIX Serial Port'
* Character device configuration file found: ./conf.json
* Character device successfully opened: '/dev/ttyUSB0'
    * Using following settings:
        * Baudrate: '115200'
        * Parity: 'no parity'
        * Stop bits: 'one'
* Character device timeout set: '5000ms'
* Launching 1000 measurements...
0% done... remaining time not yet available
```

E.2.1 conf.json: (input)

```
{
    "channel1": {
        "enabled": true,
        "coupling": "DC",
        "impedance": "50",
        "rangemV": 50,
        "offsetmV": 0,
        "bwLimit": "25MHz"
        },

    "channel3": {
        "enabled": true,
        "coupling": "DC",
        "impedance": "1M",
        "rangemV": 1000,
        "offsetmV": 0,
```

```
        "bwLimit": "FULL"
        },

    "trigger": {
        "enabled":true,
        "channel": 3,
        "level": 0.7,
        "slope": "either"
        },

    "timing": {
        "preTriggerRange": 0.0,
        "postTriggerRange": 3.2e-6,
        "samples": 2000,
        "captures": 100
        },

    "baudrate": 115200,
    "parity": 0,
    "stopbits": 1
}
```

E.2.2 id.json: (output)

```
{
    "blocks-count": "643",
    "blocks-length": "16",
    "constant-traces": "constant-traces-id.bin",
    "constant-traces-count": "357",
    "random-traces": "random-traces-id.bin",
    "random-traces-count": "643",
    "samples-per-trace": "30000"
}
```

Appendix F

Example Usage of prep Utility

F.1 Query Available Plug-ins

```
$ ./prep -Q

Found following traces preprocessing plug-ins:

* No traces preprocessing plug-in found!

Found following block data preprocessing plug-ins:

* Plug-in ID: 'predictaes128back', name: 'Create AES-128 byte
    power predictions using last round working register Hamming
    distance'
* Plug-in ID: 'predictaes128front', name: 'Create AES-128 byte
    power predictions using first round S-Box Hamming weight'
```

F.2 Create power predictions for CPA attack on AES-128 first round S-box

```
$ ./prep -B predictaes128front -b ciphertext-id.bin -m 100000 -k
    16 -I id
SICAK PREProcessing 1.0
Preprocessing block data...
100% done... 3s elapsed.
Created 16 power prediction sets, each containing 256 power
    predictions for each of 100000 data blocks,
and saved to 'aes128front-id.16prd'.
```

F.2.1 id.json: (output)

```
{
    "prediction-candidates-count": "256",
    "prediction-sets-count": "16",
    "predictions": "aes128front-id.16prd",
    "random-traces-count": "100000"
}
```

Appendix G

Example Usage of stan Utility

G.1 Query Available Plug-ins

```
$ ./stan -Q

Found following CPA plug-ins, platforms and devices:

* Plug-in ID: 'cpa', name: 'First Order Univariate CPA'
    * Platform ID: '0', name: 'localcpu'
        * Device ID: '0', name: 'localcpu'

* Plug-in ID: 'oclcpa', name: 'OpenCL accelerated First Order Univariate CPA'
    * Platform ID: '0', name: 'Intel(R) OpenCL' (OpenCL 1.2 LINUX)
        * Device ID: '0', name: 'Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz'


Found following t-test plug-ins, platforms and devices:

* Plug-in ID: 'ttest', name: 'First Order Non-Specific Univariate Welch's t-test'
    * Platform ID: '0', name: 'localcpu'
        * Device ID: '0', name: 'localcpu'
```

G.2 Create Univariate First-Order CPA context

```
$ ./stan -I ug -C cpa -F create -r random-traces-id.bin -n 10000 -s 2000 -p aes128back-10k.16prd -q 16 -k 256
SICAK STatistical ANalysis 1.0
Creating new CPA contexts...
100% done... 1m, 19s elapsed.
Created 16 new CPA contexts using
* 10000 power traces with 2000 samples per trace, from 'random-traces-id.bin',
```

```
* 16 prediction sets containing 256 power predictions for each of
    these power traces, from 'aes128back-10k.16prd'
and saved to 'cpa-ug.16ctx'.
```

G.2.1 ug.json: (output)

```
{
    "context-a": "cpa-ug.16ctx",
    "contexts-count": "16",
    "prediction-sets-count": "16"
}
```

G.3 Create correlation matrices from Univariate First-Order CPA contexts

```
$ ./stan -I ugc -C cpa -F finalize ug.json
SICAK STatistical ANalysis 1.0
Finalizing CPA context...
100% done... 1s elapsed.
Created 16 correlation matrices (2000x256) using
* 16 contexts based on 10000 from 'cpa-ug.16ctx'
and saved to 'cpa-ugc.16cor'.
```

G.3.1 ugc.json: (output)

```
{
    "contexts-count": "16",
    "correlations": "cpa-ugc.16cor",
    "correlations-candidates-count": "256",
    "correlations-sets-count": "16",
    "prediction-candidates-count": "256",
    "prediction-sets-count": "16",
    "samples-per-trace": "2000"
}
```

Appendix H

Example Usage of correv Utility

H.1 Query Available Plug-ins

```
$ ./correv -Q

Found following CPA correlation matrix evaluation plug-ins:

* Plug-in ID: 'maxabscoef', name: 'Maximum absolute value correlation coefficient'
* Plug-in ID: 'maxcoef', name: 'Maximum correlation coefficient'
* Plug-in ID: 'maxedge', name: 'Maximum correlation trace derivative (param="d;sigma", e.g. param="23;8.0")'
* Plug-in ID: 'mincoef', name: 'Minimum correlation coefficient'

Found following CPA keyguess evaluation plug-ins:

* Plug-in ID: 'aes128back', name: 'AES-128 last round CPA key evaluation: last round key gets reversed to the cipher key'
* Plug-in ID: 'plainchar', name: 'Simple key evaluation for byte-based CPA: no transformation after correlation evaluation (e.g. AES first round)'
```

H.2 Evaluate correlation matrices from previous stan example

```
$ ./correv -E maxcoef -K plainchar ugc.json
SICAK CORRelations EValuation 1.0
Evaluating CPA correlation matrices...
100% done... <1s elapsed.
Obtained key (hex): '36d024461d84b8375fc0f9c04cbab6bb'
```

```
$ ./correv -E maxcoef -K aes128back ugc.json
```

```
2 SICAK CORRelations EValuation 1.0
3 Evaluating CPA correlation matrices...
4 100% done... <1s elapsed.
5 Obtained key (hex): '00112233445566778899aabbccddeeff'
```

Appendix I

Example Usage of visu Utility

I.1 Plot Correlation Traces

```
$ ./visu -c cpa-ugc.16cor -q 16 -k 256 -s 2000 c,0,all,grey c,0,54
    -S plot.png -W 600 -H 400
SICAK VISUalisation 1.0
File successfully saved.
```

I.1.1 plot.png:

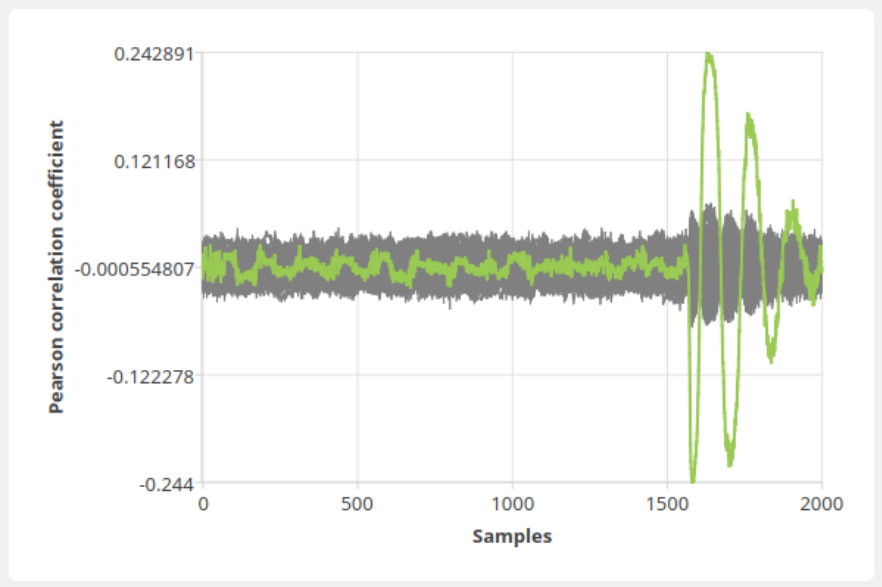


Figure I.1: Example plot containing correlation traces for all the correlation candidates, with the right key candidate's correlation trace highlighted.