Side Channel Leakage Exploitation, Mitigation and Detection of Emerging Cryptosystems

by

Cong Chen

A Dissertation

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in

Electrical and Computer Engineering

by

12 Mar 2018

APPROVED:

Professor Thomas Eisenbarth
Dissertation Advisor
ECE Department

Professor Berk Sunar
Dissertation Committee
ECE Department

Professor Yunsi Fei
Dissertation Committee
Northeastern University

Dr. Jonathan Petit
Dissertation Committee
OnBoard Security, Inc.

Abstract

With the emerging computing technologies and applications in the past decades, cryptography is facing tremendous challenges in its position of guarding our digital world.

The advent of quantum computers is potentially going to cease the dominance of RSA and other public key algorithms based on hard problems of factorization and discrete logarithm. In order to protect the Internet at post-quantum era, great efforts have been dedicated to the design of RSA substitutions. One of them is code-based McEliece public key schemes which are immune to quantum attacks.

Meanwhile, new infrastructures like Internet of Things are bringing the world enormous benefits but, due to the resource-constrained nature, require compact and still reliable cryptographic solutions. Motivated by this, many lightweight cryptographic algorithms are introduced.

Nevertheless, side channel attack is still a practical threat for implementations of these new algorithms if no countermeasures are employed. In the past decades two major categories of side channel countermeasures, namely masking and hiding, have been studied to mitigate the threat of such attacks. As a masking countermeasure, Threshold Implementation becomes popular in recent years. It is sound in providing provable side channel resistance for hardware-based cryptosystems but meanwhile it also incurs significant overheads which need further optimization for constrained applications. Masking, especially for higher order masking schemes, requires low signal-to-noise ratio to be effective which can be achieved by applying hiding countermeasures.

In order to evaluate side channel resistance of countermeasures, several tools have been introduced. Due to its simplicity, TVLA is being accepted by academy and industry as a one-size-fit-all leakage detection methodolgy that can be used by non-experts. However, its effectiveness can be negatively impacted by environmental factors such as temperature variations. Thus, a robust and simple evaluation method is desired.

In this dissertation, we first show how differential power analysis can efficiently exploit the power consumption of a McEliece implementation to recover the private key. Then, we apply Threshold Implementation scheme in order to protect from the proposed attack. This is, to the best of our knowledge, the first time of applying Threshold Implementation in a public key cryptosystem.

Next, we investigate the reduction of shares in Threshold Implementation so as to bring down its overhead for constrained applications. Our study shows that Threshold Implementation using only two shares reduces the overheads while still provides reliable first-order resistance but in the meantime it also leaks a strong second-order leakage.

We also propose a hiding countermeasure, namely balanced encoding scheme based on the idea of Dual-Rail Pre-charge logic style in hardwares. We show that it is effective to mitigate the leakage and can be combined with masking schemes to achieve better resistance.

Finally, we study paired t-test versus Welch's t-test in the original TVLA and show its robustness against environmental noises. We also found that using moving average in computing t statistics can detect higher-order leakage faster.

Acknowledgement

This dissertation was supported by the National Science Foundation (NSF), under grants CNS-1261399 and CNS-1314770.

First, I would like to thank my advisor Professor Thomas Eisenbarth for his continuous support throughout my PhD study. The advice and guidance he gave me not only help me to complete my dissertation, but also equip me best research skills for my future career. I am sincerely grateful for what I have learned from him in the past five years.

I would also like to thank my dissertation committee members Professor Berk Sunar, Professor Yunsi Fei and Dr. Jonathan Petit. They have been making careful review and advice on my research and dissertation. I am grateful for their guidance.

Special thanks are also given to Ingo von Maurich and Dr. Rainer Steinwandt who are the coauthors of our publications[CEvMS15a, CEvMS16, CEvMS15b] as discussed in Chapter 3. Materials of Chapter 4 have been published in [CFE16] and [CESY14] thanks to the contribution of Mohammad Farmani and Xin Ye respectively. I also learn a lot from Dr. Adam Ding during the collaboration in work of [DCE16] discussed in Chapter 5 and thanks for his intelligent support.

I am also grateful for the life experience with my lab peers. Thanks to your support and help during the last five years. It would be a great memory to have you around in AK212.

My parents have been supportive all the time. They respect my decision to come the states in pursuit of my graduate study and always encourage me when I feel stressed.

In the end, I would express my deepest gratitude to my wife for her sacrifice and contribution to the family. This dissertation cannot be finished with her unselfish support. It is my fortune to have her in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The past decades has witnessed the surge of many new cryptosystems to handle the challenge brought by the advancement of computing technologies. Two of the remarkable emerging cryptosystems are post-quantum cryptography and lightweight cryptography.

There has been a substantial amount of research on quantum computers – machines that exploit quantum mechanical phenomena to solve mathematical problems that are intractable for conventional computers. Up to present, IBM has been able to build a quantum computer that handles 50 quantum bits, or qubits. The rapid progress in this area poses a serious threat to traditional public key cryptography like RSA and DSA using Shor's algorithm [Sho97]. In order to protect the confidentiality and integrity of digital communications on the Internet, post-quantum cryptography has been developed to secure against both quantum and classical computers. Actually, NIST has initiated a process to solicit and standardize one or more quantum-resistant public key cryptographic algorithms. Among many submissions, McEliece cryptosystem is a promising candidate and many of its variants have been introduced since its proposal 40 years ago. However, one drawback of McEliece is that it often require large key size to achieve sufficient security. Thus, many efforts have been dedicated to shortening the key size. Recently, QC-MDPC variant of McEliece has received considerable attention since it features reasonable key sizes.

In the meantime, another advancing computing technology – Internet of Things – becomes an emerging topic. Internet of Things connects billions of resource-constrained devices including RFID tags, sensors, contactless smart cards to the Internet. In order to enjoy this new technology, security mechanisms of these devices must be carefully reviewed and proper cryptographic algorithms must be implemented to guarantee the data confidentiality, integrity and authenticity. However, it is not easy to implement conventional cryptographic functions on constrained devices due to the limitation of their resources. Therefore, many so called lightweight cryptographic algorithms have been proposed and tailored for implementation in constrained environments, such as Present [BKL+07], Prince [BCG+12], Simon [BSS+13] and so on.

It is no surprise that more and more algorithms will be introduced to provide security service in the future applications. However, security has to be understood and scrutinized under certain assumptions or adversary models. In a *black box* model, these new algorithms have passed the test or is being tested by the community. However, the discovery of side channel attacks (SCA) 20 years ago showed that the physical implementations of cryptographic algorithms can leak sensitive information via the side channel. The adversary can exploit this information to recover the internal secrets that cannot be unveiled in the black box model. Therefore, we need evaluate the algorithms in a revised model considering the side channel leakages.

In this dissertation, we investigate the side channel security of post-quantum cryptography and lightweight cryptography with case-study on the above-mentioned algorithms. Our study spans over all three important domains of side channel analysis, namely side channel *leakage exploitation, leakage mitigation* and *leakage detection.* Our goal is to answer the following questions that raised when placing these algorithms in a side channel adversary model.

- **Leakage Exploitation** Can an adversary perform differential power analysis (DPA) on post-quantum McEliece cryptosystem? There have been reports on simple power anaysis (SPA) on McEliece software implementations. However, SPA can be easily counteracted and does not work well in hardwares due to the smaller leakage. In contrast, DPA is more powerful in extracting secrets from the measurements. Thus, DPA is of particular interest for an adversary. On the other hand, from a perspective of circuit designer, a countermeasure that has a formally proved security can be implemented easily without error is desired. Therefore, another interesting question is how to protect McEliece with a sound security proof.

- **Leakage Mitigation** Threshold Implementation is a masking scheme with a security proof that has been recently well studied and applied in many conventional ciphers. However, for lightweight ciphers running on constrained devices, the overhead incurred is significant and will negatively affect its deployment. Therefore, an intriguing question is how to reduce the cost of Threshold Implementation while keep its sound security guarantee. On the other hand, effective masking requires low signal-to-noise ratio (SNR) in the side channel leakage otherwise it may fail to work. Hiding countermeasures are often used to reduce the SNR and one of important works on hiding is Dual-Rail Precharge Logic circuit style which is used to achieve constant leakage in hardwares. Can we apply the same idea in software to mitigate the leakage for masking scheme? Can we achieve constant leakage not only for internal states but also for state transitions? The answer to these questions will facilitate the protection of modern embedded processors where other hiding schemes are not easily implemented.

- **Leakage Detection** Any instantiation of countermeasures need to be thoroughly evaluated before large-scale deployment. T-test based leakage detection has become popular since it is a one-size-fits-most leakage detection test that is usable by non-experts. However, environmental effects can influence the t-test in a negative way, i.e., will decrease its sensitivity. In the worst case, this means that a leaky device may pass the test only because the environmental noise was strong enough. Can we improve the current t-test detection tool and develop a more robust and reliable leakage detection methodology? This is critical for a valid evaluation of any countermeasure.

Our Contribution This dissertation attempts to answer the challenges posed to the emerging cryptosystems as described before. The summary of our contributions are as follows.

- **Leakage Exploitation** We propose the first differential power analysis of a state-of-the-art McEliece implementation based on quasi-cyclic MDPC codes. The analysis exploits the leakages of a key rotation operation and recovers the majority of the private key bits. An innovative algebraic step to exploit the key structure results in the full key recovery. We also implement the first Threshold Implementation of McEliece cryptosystem that is leakage resistant with a formal proof and concrete attack proof.

- **Leakage Mitigation** We present the first practical threshold implementations using only two shares. We explain how using two shares can actually yield smaller cipher implementations that need less randomness and still show perfect first-order resistance indicating its promising application in lightweight ciphers for constrained environment. we also propose a balanced encoding countermeasure for software and perform the first practical evaluation. It is an effective hiding countermeasure to bring down the SNR and can be combined with masking schemes for better leakage resistance.

- **Leakage Detection** We propose a paired t-test to improve the standard methodology for leakage detection. The resulting matched-pairs design removes the environmental noise effect in leakage detection. Furthermore, we showed that moving averages increase the robustness against environmental noise for higher-order or multivariate analysis, while not showing any negative impact in the absence of noise.

The following parts of the dissertation thesis starts with the brief background introduction in Chapter 2. Next, we describe the attack on McEliece in Chapter 3. Chapter 4 introduces the two-share Threshold Implementation and balanced encoding scheme. Chapter 5 presents the study of paired t-test as a leakage detection tool. The dissertation is concluded in Chapter 6.

Chapter 2

Background

This chapter revisits the discovery of side channel analysis, followed by the introduction of two classic categories of countermeasures, namely *hiding* and *masking*. Then, we describe two leakage evaluation methods used in the dissertation. In the end, we present the emerging cryptosystems that we investigate.

2.1 Side Channel Attacks

Classic cryptanalysis treats cryptographic algorithms as *black box* and attempts to breach the cryptosystems without access to its internal execution. However, when the real-world cryptographic devices perform key-related operations, the physical observations obtained from side channels could carry information about the secret keys and thus be exploited by an adversary to recover those keys. As a consequence, the security goal established in the black box model does not hold any more in the real-world settings.

Since the discovery of timing and power channels [Koc96, KJJ99] by Kocher about 20 years ago, many other side channels such as electromagnetic emanation [GMO01], cache accesses [IIES14] and acoustic [GST14] have been investigated and demonstrated practically effective. In this dissertation, we consider mostly power attacks not only for its efficacy and efficiency but also for its widespread usage in literatures for leakage exploitation, countermeasures and evaluation.

A typical adversary model of power attacks assumes that the attacker has access to the crypto device and is even able to control the inputs to it. Next, she can measure the power consumptions of key related crypto operations on those inputs via a sampling equipment such as an oscilloscope. Let X be the inputs, k denotes the secret key and $Y = f_k(X)$ is the target crypto operation that outputs secret intermediate Y. Side channel leakage of this operation can be represented as $L = \phi(Y) + \epsilon$ where $\phi()$ is an unknown leakage function that captures the accurate power characteristics of the device and noise term ϵ is assumed to follow a Gaussian distribution. The attacker further predicts a leakage model $\tilde{\phi}(Y)$ based on her knowledge of the crypto device or profiling steps and then she will compare the leakage model and actual side channel measurements to distinguish the correct key.

Now, we briefly recall a couple of power attacks that utilize different types of distinguisher.

2.1.1 Simple Power Analysis

Simple Power Analysis (SPA) is the simplest form of SCA that attempts to recover the secret key with a small set of power measurements or even one single measurement in the most extreme cases. It often involves a visual inspection of the measurement to search for key dependent leakage patterns either in the amplitude dimension or in the time dimension. The patterns can be used to distinguish the instructions within one single measurement and the sequence of instructions can possibly reveal the keys.

Take a square-and-multiply RSA implementation for example. Within one single measurement, the adversary can possibly discern the power leakage patterns between a squaring operation and a multiplication (by inspecting the amplitude dimension) and then obtain the sequence of operations which can reveal the processed key bit.

SPA usually requires detailed knowledge of implementations of the crypto algorithms and a high SNR measurement is also necessary for a successful key recovery. Thus it is not directly used as a mean of practical attack. Instead, it can be used to derive information about the crypto implementations and assist other advanced attacks.

2.1.2 Differential Power Analysis

Differential Power Analysis (DPA) is one of most popular power analysis attacks. Unlike SPA, DPA does not require detailed knowledge of the crypto devices. In fact, it is usually sufficient to know what crypto primitive is being executed by the device. DPA can still recover the secret key even if the power measurements have fairly low SNR. To achieve this, DPA needs to collect a large set of power measurements for operation $Y = f_k(X)$ on many inputs X under an unknown key k. Then the adversary makes key guesses g and predicts the intermediate $Y = f_g(X)$. Then, she uses a bit value as a leakage model, e.g., the most significant bit of Y. Based on this power model, the measurements are divided into two groups, Then the adversary makes key guesses and predict the intermediate bit value (0 or 1) under her choices of plaintext and key guess. Based on the prediction, she can divide the power measurements into two groups corresponding to either bit value 0 or bit value 1. A difference of mean (DoM) distinguisher is then applied to the two groups and distinguish the correct key with the strongest DoM. This is because the correct key guess leads to the correct prediction of the intermediate value.

2.1.3 Correlation Power Analysis

Correlation Power Analysis (CPA) was originally proposed by Brier [BCO04]. CPA attempts to find the dependency between side channel leakage and internal secret of a cryptographic execution. To this end, CPA requires a leakage model, e.g., Hamming weight model of S-box output in a block cipher. Let Y be an S-Box output of n bits, $\mathrm{wt}(Y)$ be the Hamming weight of Y and L be the observed side channel leakage. Then the dependency between Y and L can be captured by their correlation coefficient calculated as:

$$corr(\mathrm{wt}(Y), L) = \frac{\Sigma(L - \bar{L})\big(\mathrm{wt}(Y) - \overline{\mathrm{wt}(Y)}\big)}{\sqrt{\Sigma(L - \bar{L})^2 \times \Sigma\big(\mathrm{wt}(Y) - \overline{\mathrm{wt}(Y)}\big)^2}} \tag{2.1}$$

Just as in DPA, attacker takes key guesses and then predicts corresponding intermediates (now it is n-bit state instead of a single bit) as well as the leakage model. By calculating the correlation coefficients between real leakage and predicted model, attacker can distinguish the correct key corresponding the highest correlation coefficient.

2.2 Side Channel Countermeasures

Ever since the introduction of side channel analysis, many works have studied and developed countermeasures to protect cryptosystems from such attacks. Two major lines of countermeasures are *masking*, which aims to randomize the intermediate values processed by the crypto devices, and *hiding* which attempts to break the link between the power consumption and processed data.

2.2.1 Masking Countermeasures

Masking countermeasure can be seen as a secret sharing scheme where each key dependent intermediate value Y is split into d random shares $Y_1, Y_2, ..., Y_d$. Any combination of less than d shares cannot reveal knowledge of Y. Thus even if attacker can obtain non-decaying leakage for each share, she has to integrate the leakage for all d shares, namely $L_1, L_2, ..., L_d$ in a multivariate manner or explore the d-th order statistical moments of the measurements to perform a successful key recovery which is getting more complicated as d grows larger.

Threshold Implementation is one flavor of masking countermeasures that has drawn attentions in recent years. Unlike the ad hoc masking schemes, it is a generic technique that can be conveniently applied to a wide range of algorithms. Moreover it features a security proof against glitch leakages.

Threshold Implementation Threshold Implementation (TI) was proposed by Nikova et al [NRR06] as a side-channel countermeasure to address the common problem of *glitches* that resulted in leakage for many other theoretically sound countermeasure techniques when applied to hardware.

TI combines a set of three requirements with a constructive description of how to convert an algorithm into a side-channel resistant implementation in the presence of glitches. Sensitive states are converted into a shared representation by applying an additive Boolean masking, i.e., adding randomness. Functions $F(\cdot)$ are converted meeting the requirements of correctness, uniformity, and non-completeness.

- **Uniformity** requires all intermediate states (shares) to be uniformly distributed. Uniformity is intended to ensures the mean leakages to be state-independent, a key requirement to thwart first-order DPA. To ensure uniformity in a circuit it suffices to ensure uniformity for the output share of each function, as well as for the inputs of the circuit.

- **Non-Completeness** requires subfunctions f_i of a shared function F to be independent of at least one input share for first-order SCA resistance. That is, a function $F(x)$ shall be split into subfunctions $f_i(x_{j \neq i})$. This requirement was updated in [BGN$^+$14b] to require any d subfunctions to be independent of at least one input share to achieve d-th order SCA resistance. Non-completeness ensures that the final circuit is not affected by glitches. Since glitches can only occur in subfunctions f_i, and each subfunction has insufficient knowledge to reconstruct a secret state (since it has no knowledge of at least one share x_i), no leakage can be caused by glitches.

- **Correctness** simply states that applying the subfunctions to a valid shared input must always yield a valid sharing of the correct output.

2.2.2 Hiding Countermeasures

The goal of hiding is to make the side channel leakage independent of processed intermediate values. In order to achieve this, two different approaches can be adopted. One is to randomize the leakage at each clock cycle during the crypto operations such as inserting dummy operations, shuffling (in time dimension) and decreasing signal-to-noise ratio (SNR) in amplitude dimension. The other approach aims to build a crypto device that shows constant activities regardless of operations and data being processed such that the leakage, e.g., power consumption is equally dissipated and shows no correlation with the intermediate values. *Dual-Rail Precharge Logic (DPL)* is a classic technique that achieves constant leakages for hardware implementations.

Dual-Rail Precharge Logic DPL aims to achieve constant activity at the gate level. In order to achieve this, DPL adopts the concepts of both dual-rail logic and precharge logic.

Unlike a single-rail logic where each signal is carried in one wire, dual-rail logic uses two complementary wires to represent a signal. One of the two wires will carry non-inverted signal A while the other wire carries inverted signal $\bar{A}$. That is, the valid logic signal always consists of complementary values $(A, \bar{A})$. For example, $(0, 1)$ are viewed as representation of signal 0 in a single-rail logic while $(1, 0)$ denotes 1. For a dual-rail logic gate, all the input wires and output wires are such complementary wires.

As for the precharge logic, it means that all the wires in the circuit are set to a so-called precharge value (0 for example). Then, in the evaluation phase, the signals are set to their current logic values. When combined with dual-rail logic where all signals are represented as two complementary values, there are always constant transition activities at each clock cycle since for each signal it is either transit from $(0, 0)$ to $(0, 1)$ or change from $(0, 0)$ to $(1, 0)$. In other words, there is constantly one transition at each signal leading to constant power consumption in an ideal leakage model.

Note that the Hamming weight of the pair is always constant as 1. Besides, in order to obtain constant Hamming distance, the bit pair is precharged to $(0, 0)$ before evaluation. Hence, the gate transitions from $(0, 0)$ to either $(0, 1)$ or $(1, 0)$ leaks data independent power consumption or EM emissions. Based on this idea, many DPL style have been proposed such as WDDL [TV04], MDPL [PM05] and DRSL [CZ06].

2.3 Evaluation Metrics

2.3.1 Mutual Information based Leakage Evaluation

A popular method for evaluating the side channel resistance of an implementation is mutual information. It was proposed as a side channel leakage metric for evaluation in [SMY09] and refined for practical experiments in [DSVC14]. The goal is to evaluate the leakage L of a critical intermediate state s. The evaluated intermediate state for the Prince cipher is one nibble. The mutual information $I(S;L)$ between the leakage L and the states S is computed as $I(S;L) = H(S) - H(S|L)$ where $H(S|L)$ is the conditional entropy of S, knowing the leakage L. It is given as

$$H(S|L) = -\sum_{l \in \mathcal{L}} \left(\Pr(l) \sum_{s \in \mathcal{S}} Pr(s|l) \log \Pr(s|l) \right) \tag{2.2}$$

where l and s are specific observations of the leakage and state, respectively. Given univariate templates $\mathcal{N}(\mu_s, \sigma_s)$ for each state value s and each point of the leakage, we have the probability density for observing a leakage l at that point given as $p(l|s) = \mathcal{N}(\mu_s, \sigma_s)$. Following Bayes' Theorem, we get $p(s|l) = \frac{p(l|s)\Pr(s)}{\Pr(l)}$ and, since all observations and states are equally probable, we can derive

$$\Pr(s|l) = \frac{p(l|s)}{\sum_{s^* \in \mathcal{S}} p(l|s^*)},$$

as typically done for templates. Plugging this back into Equation (2.2), we can solve Equation (2.2) by computing and summing over all $\Pr(s|l^*)$ for each $l^* \in \mathcal{L}_T$, where $\mathcal{L}_T$ is the test (or evaluation) set.

2.3.2 T-test based Leakage Detection

In the framework of [GJJR11], the potential leakage for a device under test (DUT) can be detected by comparing two sets of measurements $\mathcal{L}_A$ and $\mathcal{L}_B$ on the DUT. A popular test for the comparison is Welch's t-test, which aims to detect the mean differences between the two sets of measurements. The null hypothesis is that the two samples come from the same population so that their population means μ_A and μ_B are the same. Let $\bar{L}_A$ and $\bar{L}_B$ denote their sample means, s_A^2 and s_B^2 denote their sample variance, n_A and n_B denote the number of measurements in each set. Then the t-test statistic and its degree of freedom are given by

$$t_u = \frac{\bar{L}_A - \bar{L}_B}{\sqrt{\frac{s_A^2}{n_A} + \frac{s_B^2}{n_B}}}, \qquad v = \frac{(\frac{s_A^2}{n_A} + \frac{s_B^2}{n_B})^2}{\frac{(\frac{s_A^2}{n_A})^2}{n_A - 1} + \frac{(\frac{s_B^2}{n_B})^2}{n_B - 1}}. \tag{2.3}$$

The p-value of the t-test is calculated as the probability, under a t-distribution with v degree of freedom, that the random variable exceeds observed statistic $|t_u|$. This is readily done in Matlab as $2 * (1 - tcdf(\cdot, v))$ and in R as $2 * (1 - qt(\cdot, df = v))$. The null hypothesis of no leakage is rejected when the p-value is smaller than a threshold, or equivalently when the t-test statistic $|t_u|$ exceeds a corresponding threshold. The rejection criterion of $|t_u| > 4.5$ is often used [SM15, GJJR11]. Since $Pr(|t_{df=v>1000}| > 4.5) < 0.00001$, this threshold leads to a confidence level > 0.99999.

For leakage detection, a *specific* t-test use two sets $\mathcal{L}_A$ and $\mathcal{L}_B$ corresponding to different values of an intermediate variable: $V = v_A$ and $V = v_B$. To avoid the dependence on the intermediate value and the power model, *non-specific* t-test often uses the *fixed versus random* setup. That is, the first set $\mathcal{L}_A$ is collected with a fixed plaintext x_A, while the second set $\mathcal{L}_B$ is collected with random plaintexts x_B drawn from the uniform distribution. Then if there is leakage through an (unspecified) intermediate variable V, then

$$L_A = V(k, x_A) + r_A \qquad\qquad L_B = V(k, x_B) + r_B, \tag{2.4}$$

where k is the secret key, r_A and r_B are random measurement noises with zero means and variance σ_A^2 and σ_B^2 respectively. The non-specific t-test can detect the leakage, with large numbers of measurements n_A and n_B, when the fixed intermediate state $V(k, x_A)$ differs from the expected value of the random intermediate state $E_{x_B}[V(k, x_B)]$ where the expectation is taken over the uniform random plaintexts x_B.

2.4 Emerging Cryptosystems

2.4.1 Post-Quantum Cryptography

With the advance of quantum computers, it is likely that in short future it would undermine currently deployed asymmetric solutions, as most of these have to assume the hardness of computational problems which are known to be feasible with large-scale quantum computers [Sho97].

Given these threats, it is worthwhile to explore alternative public key encryption schemes that rely on problems which are believed to be hard even for quantum computers. The McEliece cryptosystem [McE78] proposed 40 years ago is still one promising candidate of NIST's post-quantum cryptography standard submissions.

In the following we introduce one recent variant of McEliece based on quasi-cyclic (QC) Moderate-Density Parity-Check (MDPC) Codes.

2.4.1.1 Moderate-Density Parity-Check Codes

MDPC codes belong to the family of binary linear $[n, k]$ error-correcting codes, where n is the length, k the dimension, and $r = n - k$ the co-dimension of a code C. Binary linear error-correcting codes are equivalently described either by their generator G or by their parity-check matrix H. The rows of generator matrix $G \in \mathbb{F}_2^{k \times n}$ form a basis of C while $H \in \mathbb{F}_2^{r \times n}$ describes the code as the kernel $C = \{c \in \mathbb{F}_2^n | \, Hc^T = 0^\perp\}$ where $0^\perp$ represents an all-zero column vector. The syndrome of any vector $x \in \mathbb{F}_2^n$ is defined as $s = Hx^T \in \mathbb{F}_2^r$. Hence, the code C is comprised of all vectors $x \in \mathbb{F}_2^n$ whose syndrome is zero for a particular parity-check matrix H. MDPC codes are defined by only allowing a *moderate* Hamming weight $w = \mathrm{O}(\sqrt{n \log(n)})$ for each row of the parity-check matrix. By an (n, r, w)-MDPC code we refer to a binary linear $[n, k]$ code with such a constant row weight w.

A code C is called quasi-cyclic if for some positive integer $n_0 > 0$ the code is closed under cyclic shifts of its codewords by n_0 positions. Furthermore, it is possible to choose the generator and parity-check matrix to consist of $p \times p$ circulant blocks if $n = n_0 \cdot p$ for some positive integer p. This allows to completely describe the generator and parity-check matrices by their first row. If an (n, r, w)-MDPC code is quasi-cyclic with $n = n_0 \cdot r$, we refer to it as an (n, r, w)-QC-MDPC code.

Several t-error-correcting decoders have been proposed for (QC-)MDPC codes [BMvT78, Gal62, HvMG13, HP10, MTSB13, vMOG15]. The implementation that we base our work on implements the optimized decoder presented in [HvMG13], which in turn is an extended version of the bit-flipping decoder of [Gal62]. Decoding a ciphertext $x \in \mathbb{F}_2^n$, is achieved by:

1. Computing the syndrome $s = Hx^T$.

2. Computing the number of unsatisfied parity checks $\#_{\text{upc}}$ for every ciphertext

bit.

3. If $\#_{\text{upc}}$ exceeds a precomputed threshold b, invert the corresponding ciphertext bit and add the corresponding column of the parity-check matrix to the syndrome.

4. In case $s = 0^{\perp}$, decoding was successful, otherwise repeat Steps 2/3.

5. Abort after a defined maximum of iterations with a decoding error.

2.4.1.2 McEliece Public Key Encryption with QC-MDPC Codes

The McEliece cryptosystem was introduced using binary Goppa codes [McE78]. Instantiating McEliece with t-error-correcting (QC-)MDPC codes was proposed in [MTSB13], mainly to significantly reduce the size of the keys while still maintaining reasonable security arguments. The proposed parameters for an 80-bit security level are $n_0 = 2, n = 9602, r = 4801, w = 90, t = 84$, which results in a much more practical public key size of 4801 bit and a secret key size of 9602 bit compared to binary Goppa codes which require around 64 kByte for public keys at the same security level.

The main idea of the McEliece cryptosystem is to encode a plaintext into a codeword using the generator matrix of a code selected by the receiver and to add a randomly generated error vector of weight t to the codeword which can only be removed by the intended receiver. We summarize QC-MDPC McEliece in the following by introducing key-generation, encryption and decryption.

Key-Generation. The parity-check matrix H is the secret key in QC-MDPC McEliece. As the code is quasi-cyclic, the parity-check matrix consists of n_0 concatenated $r \times r$ blocks $H = (H_0 \,|\, \ldots \,|\, H_{n_0-1})$. We denote the first row of each of these blocks by $h_0, \ldots, h_{n_0-1} \in \mathbb{F}_2^r$. The public key in QC-MDPC McEliece is the corresponding generator matrix G, which is computed from H in standard form as $G = [I_k \,|\, Q]$ by concatenation of the identity matrix $I_k \in \mathbb{F}_2^{k \times k}$ with

$$Q = \begin{pmatrix} (H_{n_0-1}^{-1} \cdot H_0)^T \\ (H_{n_0-1}^{-1} \cdot H_1)^T \\ \cdots \\ (H_{n_0-1}^{-1} \cdot H_{n_0-2})^T \end{pmatrix}.$$

The key generation starts by randomly selecting first row candidates $h_0, \ldots, h_{n_0-1} \in_R \mathbb{F}_2^r$ such that the overall row weight (wt) sums up to $w = \sum_{i=0}^{n_0-1} \mathrm{wt}(h_i)$. Since we intend to generate a code which is quasi-cyclic, the n_0 blocks of the parity-check matrix are generated from the first rows by cyclic shifts. The resulting parity-check matrix belongs to an (n, r, w)-QC-MDPC code with $n = n_0 \cdot r$. If the last block H_{n_0-1} is non-singular, i. e., if $H_{n_0-1}^{-1}$ exists, the public key is computed as $G = [I_k \,|\, Q]$. Otherwise new candidates for h_{n_0-1} are generated until a non-singular H_{n_0-1} is found.

Encryption. A plaintext $m \in \mathbb{F}_2^k$ is encrypted by encoding it into a codeword using the recipient's public key G and by adding a random error vector $e \in \mathbb{F}_2^n$ of weight $\mathrm{wt}(e) \leq t$ to it. Hence, the ciphertext is computed as $x = (m \cdot G \oplus e) \in \mathbb{F}_2^n$.

Decryption. Given a ciphertext $x \in \mathbb{F}_2^n$, the intended recipient removes the error vector e from x using the secret code description H and a QC-MDPC decoding algorithm Ψ_H yielding mG. Since $G = [I_k \,|\, Q]$, the first k positions of mG are equal to the k-bit plaintext.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

2.4.2 Lightweight Cryptography

For many embedded applications, area and hence power or energy minimal implementations of cryptography are highly desirable. This has led to a rich literature on hardware-minimal crypto cores, which often rely on the numerous proposed "lightweight" block cipher designs, such as Prince, or Simon and Present.

2.4.2.1 Present

Present is a hardware-oriented block cipher proposed in 2007, optimized for low area footprint [BKL+07]. It is a substitution-permutation network featuring a 4×4 bit S-box and a permutation layer consisting only of bit shifts, making it low cost in hardware. It features a block size of 64 bits and a key size of 80 or 128 bits, and has 31 rounds.Present has been optimized for many application scenarios, but the area-minimal implementations with a 4-bit data-path. It has also been standardized as a lightweight cryptographic block cipher as ISO/IEC 29192-2:2012. Each round of Present cipher consists of three steps including a key-addition layer, a substitution layer which is a non-linear function, and a permutation layer. In the first step, the round key which is consisted of left most significant 64 bits of the key is xored with the 64-bit current state. In the next step, the Present S-box is used which is a non-linear 4-bit to 4-bit function shown in the following table in hexadecimal notation.

The substitution layer can be performed with 16 parallel S-box or using only one S-box 16 times which depends on the application requirement. In the last step, the permutation is applied to all the 64-bit data which is just a rewiring.

At the same time, the key is updated in the key schedule part. The key can be 80-bit or 120-bit; however we use 80-bit key in this paper. In each round the 64 left most bits of the current key, $k_{79}k_{78}k_{77}...k_{17}k_{16}$, is used in addroundkey. After using the round key, the 80-bit key register is updated by shifting, using S-box, and xoring with round-counter. More details about the specification of the Present is provided in [BKL+07].

2.4.2.2 Simon

Simon is a lightweight block cipher proposed by NSA in 2013 [BSS+13]. Simon implements a Feistel structure that accepts two n-bit words as input plaintext, with $n \in \{16, 24, 32, 48, 64\}$. For each input size $2n$, Simon has a set of allowable key sizes ranging from 64 bits to 256 bits. The number of rounds in Simon ranges from 32 to 72 rounds. Simon128/128, which can be seen as a drop-in replacement for AES-128, accepts 128 bits of plaintext at a word size of 64 bits and 128 bits of key. It generates a ciphertext after 68 rounds. The Simon128/128 parameter set will be used throughout this work, though the implementation strategies apply to other parameter sets in a natural way..

We denote the input words of round i as l_i and r_i. Then the output words are given as:

$$\begin{aligned} r_{i+1} =& l_i \\ l_{i+1} =& r_i + l_i^2 + (l_i^1 * l_i^8) + k_i \end{aligned} \tag{2.5}$$

The upper index in l_i^s indicates left circular shift by s bits. The addition and the multiplication are in GF(2) and equivalent to bitwise XOR and AND operations, respectively. Given the initial key words k_0 and k_1 (and possibly k_2 and k_3, depending on the key size), which are also used as first round keys, the subsequent round

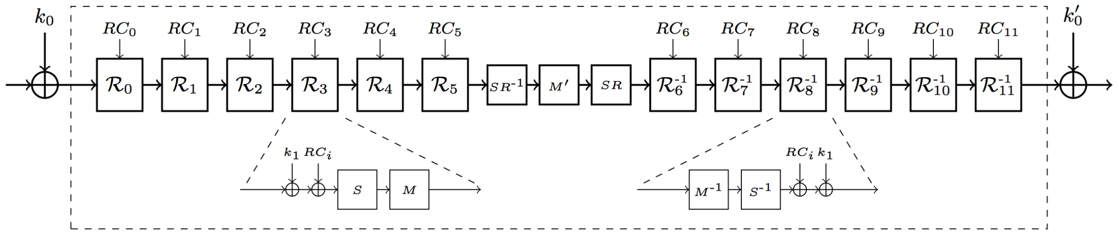


Figure 2.1: The structure of Prince cipher.

keys are computed as:

$$\begin{aligned} k_{i+2} &= k_i + k_{i+1}^{-3} + k_{i+1}^{-4} + c_i \quad \text{Two and Three Words} \\ k_{i+4} &= k_i + k_{i+1} + k_{i+1}^{-1} + k_{i+3}^{-3} + k_{i+3}^{-4} + c_i \quad \text{Four Words} \end{aligned} \tag{2.6}$$

where c_i is a round constant.

2.4.2.3 Prince

The Prince block cipher is a lightweight cipher, featuring a 64-bit block size and a 128-bit key size [BCG+12]. Prince has been optimized for low latency and a small hardware footprint. Its round function has several similarities to the AES: it features KeyAddition, S-box, ShiftRows and MixColumns operations as shown in Firgure 2.1. However, these operations are performed on a 4-by-4 array of 4-bit nibbles. This 4-bit oriented design makes Prince—unlike AES—a suitable candidate for a constant Hamming weight encoding on 8-bit microcontrollers. Prince has 12 rounds, and the last six apply the inverse operations of the first six. The 64-bit round key remains constant in all rounds, but is augmented with a 64-bit round constant to ensure variation between rounds. The remaining 64 key bits are used for pre- and post-whitening of the state. A feature of Prince is that encryption and decryption only differ in the round key.

Chapter 3

Leakage Exploitation

Side channel attacks, in particular, power attacks, have been demonstrated as serious threats to the cryptographic devices in the past decades. Unsurprisingly, without consideration of such threats, any new cryptosystem designed and deployed in the future will still be vulnerable in a hostile environment and fail to work properly in the end. One of such new cryptosystems is post-quantum cryptography which is designed to answer the challenge of the advent of quantum computers. However, even in a post-quantum era, instantiations of post-quantum cryptography would leak secret-related information via side channels. Therefore, they still require hardening against side channel attacks for practical applications.

In this chapter, we first describe briefly the challenge of power attacks on post-quantum cryptography and the motivation for protection against such attacks. Then, we thoroughly introduce the first differential power attack on a McEliece Cryptosystem. In the end, we propose the application of TI-based masking scheme to protect McEliece against DPA. To our best knowledge, this is the first attempt of application of TI on a public key cryptosystem.

3.1 Introduction and Motivation

Prominent services provided by public key cryptography include signatures and key encapsulation, and their security is vital for various applications. In addition to classical cryptanalysis, quantum computers pose a potential threat to currently deployed asymmetric solutions, as most of these have to assume the hardness of computational problems which are known to be feasible with large-scale quantum computers [Sho97]. Given these threats, it is worthwhile to explore alternative public key encryption schemes that rely on problems which are believed to be hard even for quantum computers, which might become reality sooner than the sensitivity of currently encrypted data expires [BCNS15]. The McEliece cryptosystem [McE78] is among the promising candidates, as it has withstood more than 40 years of cryptanalysis. To that end, efficient and secure implementations of McEliece should be available even nowadays. The QC-MDPC variant of the McEliece scheme proposed in [MTSB13] is a promising efficient alternative to prevailing schemes, while maintaining reasonable key sizes. The first implementations of QC-MDPC McEliece were presented in [HvMG13], and an efficient and small hardware engine of the scheme was presented in [vMG14a].

However, even in a post-quantum world, i. e., when scalable quantum computers are available, implementation-specific information leakage via side channel will remain a serious issue. Side channel leakages of McEliece have first been studied in [STM$^+$08]. This work, as well as two follow-up studies focused on analyzing timing behavior of different parts of PC implementations of McEliece [SSMS10, Str10]. Subsequently, [AHPT11] improved over prior results, presented countermeasures and pointed out leakages in the preprocessing steps of McEliece encryption. Heyse et al. [HMP10] performed power analysis on software implementations of classic

McEliece implementations. Their work relies on simple power analysis approaches, which usually do not translate well to hardware implementations, due to the increased parallel processing of data and the much smaller side-channel leakage. They also show that side channel analysis is impeded by the large key sizes of McEliece. In a recent work, AVR/ARM microcontroller implementations of QC-MDPC McEliece were shown to be susceptible to SPA attacks [vMG14b]. The found weaknesses rely on secret dependent branches, which allow to recover the encrypted message as well as to recover the secret key. However, no differential power analysis has been documented on implementations of McEliece before our work. In this dissertation we demonstrate that DPA can be a realistic threat for a state-of-the-art FPGA implementation of McEliece. We first show that significant parts of the private key can be recovered by DPA on the decoding algorithm of , then we show that knowledge of the public key can be utilized to recover missing key information or to correct remaining errors in hypothesized key bits with an algebraic calculation step. In CHES 2017, Rossi *et al* presented a similar attack [RHHM17] on QcBits a constant-time CCA-secure C implementation of QC-MDPC and they claimed their attack is the first one on a CCA-secure constant-time version of QC-MDPC. Even though the details of their attack is different, the main idea of using DPA to recover partial keys followed a full key recovery with an algebraic calculation is the same as ours. Our attack of course works for CCA-secure constant-time implementation because it exploits the leakage during the private key rotation which is regardless of what CCA protocol is used and how constant-time is achieved. Also, our attack works more efficiently mainly because we utilize the DPA results on both secret keys, i.e., h_0 and h_1 and greatly reduce the efforts for recovering the full key making it suitable for even larger keys.

In order to counteract the proposed differential power attacks, we further present a masked hardware implementation of QC-MDPC McEliece based on the lightweight design presented in [vMG14a]. In fact, the masking scheme adopted in this work is a state-of-the-art technique, namely Threshold Implementation [NRR06] which has been applied in many symmetric cryptosystems [BDN$^+$14, BGN$^+$14a, BGN$^+$14b, MPL$^+$11, STE15] to prevent first-order and higher-order side channel leakage. More importantly, most of these works have performed thorough leakage analysis and have shown that TI actually prevents the promised order leakage (if carefully implemented). However, we notice that TI has not been investigated in asymmetric cryptosystems before this work, particularly in emerging post-quantum cryptography. Therefore, it is intriguing to study TI in post-quantum publick-key cryptography. We choose TI for McEliece because TI is fairly straightforward to apply and to implement, yet it is effective. Furthermore, large parts of McEliece are linear, and hence cheap to mask using TI. The decoder part, while not linear, is also fairly efficient to mask using TI. At the same time, our implementation avoids several of the disadvantages of TI: Unlike [SMG15], we convert our addition to arithmetic masking once the values get larger, yielding a much more efficient addition engine than one solely relying on TI. By including the pseudorandom mask generation in the crypto core, we significantly cut both the required memory space usually unavoidably introduced by TI as well as the required overhead of random bits consumed by TI engines.

3.2 DPA of a McEliece Cryptosystem

This section presents the first differential power analysis of an implementation of the McEliece cryptosystem. Target of this side-channel attack is a state-of-the-art FPGA implementation of the efficient QC-MDPC McEliece decryption operation.

The presented cryptanalysis succeeds to recover the complete secret key after a few observed decryptions. It consists of a combination of a differential leakage analysis during the syndrome computation followed by an algebraic step that exploits the relation between the public and private key.

3.2.1 Target Implementation

The target under investigation is a lightweight implementation of QC-MDPC McEliece for reconfigurable devices by [vMG14a]. The resource requirements are 64 slices and 1 block RAM (BRAM) to implement encryption and 159 slices and 3 BRAMs to implement decryption on a Xilinx Spartan-6 XC6SLX4 FPGA. This lightweight implementation is possible mainly for two reasons. First, QC-MDPC codes allow smaller keys compared to (optimized) binary Goppa codes. Second, the implementation stores inputs, outputs and most intermediate values during encryption and decryption in block memories. Since our attack focuses on secret-key recovery, we limit the description of the details of the implementation to the decryption, especially to the part in which the syndrome is computed.

Decryption uses three BRAMs, one BRAM stores the $2 \cdot 4801$-bit secret key, one BRAM stores the $2 \cdot 4801$-bit ciphertext, and one BRAM stores the 4801-bit syndrome. Each BRAM is dual-ported, offers 18/36 kBit, and allows to read/write two 32-bit values at different addresses in one clock cycle. To compute the syndrome, set bits in the ciphertext select rows of the parity-check matrix blocks that are accumulated. Since only one row of each block is stored in the BRAM, they need to be rotated by one bit to generate the next rows. To generate all rows of H, the rotation is repeated 4801 times.

Rotating the two parts of the secret key is implemented in parallel, which means that the 4801-bit rows of the first and the second part of the parity-check matrix are rotated at the same time. Efficient rotation is realized using the READ_FIRST mode of Xilinx's BRAMs which allows to read the content of a 32-bit memory cell and then to overwrite it with a new value, all within one clock cycle.

The key rotation is implemented as follows: in the first clock cycle, the least significant bit (LSB) is loaded from the last memory cell. The first 32-bit of the row to be rotated are loaded next. In all following clock cycles, the succeeding 32-bit blocks of the row are read and overwritten by the rotated preceding 32-bit block. The LSB of each 32-bit block is delayed by a flip-flop and becomes the most significant bit (MSB) of the following block. An abstraction of this implementation is depicted in Figure 3.1. In addition to a rotation of the rows, this introduces a rotation of the memory cells. After one 4801-bit rotation, the most significant 32 bits of a parity-check matrix row do not reside in memory cell 0 but in memory cell 1.

The syndrome s is computed by processing the ciphertext x in a bitwise fashion. If the j-th bit is set, i. e., $x_j = 1$, then the j-th row of H is added to the syndrome s. The implementation adds two 32-bit words in parallel: one word of the rotated h_0 and one word of h_1 are processed in each clock cycle.

3.2.2 Attack Description

Usually DPA attacks exploit an intermediate state $y = f(x, k)$ that is a function of both a known data item x and a subkey k. The subkey space $\mathcal{K}$ should be small enough so that a hypothesis y can be checked for all candidates $k \in \mathcal{K}$. Some works that elaborate on this model are [KJJR11, KJJR11, WOS14]. McEliece does not offer itself for this approach, as also noted in [HMP10]. One would expect the syndrome s to serve as a potential predictable intermediate state y. However, the bits in the ciphertext x only determine which rows of the parity check matrix H

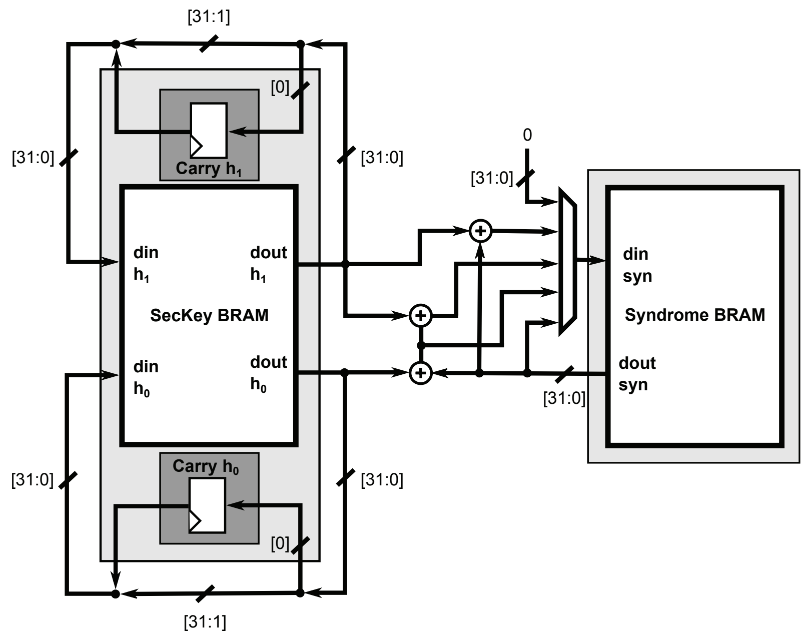


Figure 3.1: Block diagram of the syndrome computation as implemented in [vMG14a].

are added to s, where H is the secret key to be recovered. Predicting (parts of) the syndrome s requires an additional key bit hypothesis for each variation of each bit of s, i. e., each bit of s depends on l key bits after l variations, supporting the infeasibility claim of [HMP10]. One of the strengths of QC-MDPC, its small private key size, comes from the fact that secret information is highly redundant: each row of H contains the same information—namely $\langle h_0 \ggg z || h_1 \ggg z \rangle$—only rotated by one bit per row, $z \in \{0, 4800\}$. This redundancy allows for an efficient recovery of key information. More important, it enables a *differential* analysis approach which greatly enhances the visibility of even faint leakages.

We exploit this leakage of the key rotation operation during syndrome computation. Our analysis recovers a static key leakage that is completely independent of the known or chosen ciphertext input x. Since the exploited leakage occurs several times during one syndrome computation, our attack combines these leakage events, as commonly done in horizontal side channel attacks.

3.2.2.1 Leakage Behavior of the Target Implementation

The described attack recovers the key during the syndrome computation step of the decryption algorithm. The key for QC-MDPC consists of a single line of the parity check matrix H, namely $h_0||h_1$. As described in Section 3.2.1, only this line of H, or one of its rotated versions $\langle h_0 \ggg z||h_1 \ggg z\rangle$, is stored in BRAM. The key has some noteworthy features that influence the derived DPA attacks. First, the private key is of *low weight*: both parts of the secret key h_0 and h_1 are of low Hamming weight such that, $\mathrm{wt}(h_0||h_1) = w$. For the target implementation, $w = 90$ and $\mathrm{wt}(h_i) = 45$, i. e. both h_0 and h_1 have exactly 45 bits set. This means, each key bit $h_{i,j} \in \{0, 1\}$ where $i \in \{0, 1\}$ and $j \in \{0, 4800\}$ is set with probability $\Pr(h_{i,j} = 1) = w/(n_0 r) = 45/4801 \approx .94\%$. This implies *low-weight leakages*: Syndrome and key parts h_i are stored in BRAMs and are processed as 151 32-bit words. The chance of a 32-bit key word to be all-0 is still 74%, about 22% contain a single one bit, leaving the chance of having more than one bit set in a word below 5%.

The critical parts of the target implementation that feature exploitable key leakage are depicted in Figure 3.1. There are two operations that contribute to the leakage during syndrome computation. One operation is the key rotation, which is always performed. The second operation is the syndrome computation. Our analysis focuses on the key rotation operation, which is independent of the ciphertext input x. The stored key row $\langle h_0 \ggg z||h_1 \ggg z\rangle$ is constantly rotated during the syndrome generation. In fact, it is rotated by a single bit 4801 times, where each rotation takes 151 clock cycles (plus two additional clock cycles for preprocessing and a data read-write delay, resulting in the 153 clock cycles mentioned in [vMG14a]). The implementation features a separate register which stores the carry bit during rotations. In each of these clock cycles, one bit $h_{i,j}$—the LSB of the last accessed word—is written to the carry register, causing leakage $\lambda_{\text{carry}}(i,j)$. In the following clock cycle, that bit is overwritten with the LSB of the next word, $h_{i,j+32}$. Assuming a Hamming distance leakage function, this register leaks first

$$\lambda_{\text{carry}}(i,j) = w_1 \cdot \text{wt}(h_{i,j-32} \oplus h_{i,j}), \tag{3.1}$$

then, in the subsequent clock cycle, leaks $\lambda_{\text{carry}}(i,j{+}32) = w_1 {\cdot} \text{wt}(h_{i,j} {\oplus} h_{i,j+32})$, where $w_1 \in \mathbb{R}$ is an appropriate weight. Assuming that $h_{i,j} = 1$ and further $h_{i,j\pm 32} = 0$, $\lambda_{\text{carry}}(i,j)$ gives a clearly distinguishable leakage from the case where $h_{i,j} = 0$. This leakage is the target of the described attack.

In addition to the leakage of the carry register $\lambda_{\text{carry}}(i,j)$ described in Equation (3.1), there are related leakages happening in the same clock cycles. In fact, when $h_{i,j}$ is written to the carry register, the implementation also reads the word $\langle h_{i,j+1} \ldots h_{i,j+32} \rangle$ from the block memory at one address and then stores the word $\langle h_{i,j-32} \ldots h_{i,j-1} \rangle$ into the block memory at the same address. Both reading and storing operations will cause leakages at different levels. Assuming a Hamming weight leakage function here, reading data and storing data words leaks as

$$\begin{aligned} \lambda_{\text{read}}(i,j) &= w_2 \cdot \text{wt}(\langle h_{i,j+1} \ldots h_{i,j+32} \rangle) \text{ and} \\ \lambda_{\text{store}}(i,j) &= w_3 \cdot \text{wt}(\langle h_{i,j-32} \ldots h_{i,j-1} \rangle), \end{aligned}$$

respectively. Here, $w_2 \in \mathbb{R}$ and $w_3 \in \mathbb{R}$ are appropriate weights for the different types of operations. The overall observed leakage is approximated as:

$$\mathcal{L}_i(j) = \lambda_{\text{carry}}(i,j) + \lambda_{\text{read}}(i,j) + \lambda_{\text{store}}(i,j) + \mathcal{N}$$

where $\mathcal{L}_i$ is the overall leakage at the clock cycle where $h_{i,j}$ is written into the carry register and $\mathcal{N}$ is noise, which is assumed to be Gaussian. Please note that the target implementation processes h_0 and h_1 in parallel. This means that the leakage functions $\mathcal{L}_0$ and $\mathcal{L}_1$ for h_0 and h_1 overlap. There are two carry registers (cf. Figure 3.1), one stores $h_{0,j}$ when the other stores $h_{1,j}$. While these leakages slightly differ, we will not attempt to distinguish them. Instead we recover the combined leakages. That is, we predict the combined leakage $h_\Sigma = h_0 + h_1$, which is still sparse. Note that the addition here is *not* in $\mathbb{F}_2$, i. e., we can distinguish the case where $h_{0,j} = h_{1,j} = 1$ from the case $h_{0,j} = h_{1,j} = 0$, although this case is very rare (and will be ignored in the further description). While the model is not perfect, it describes the observed leakages well enough to base a decent key recovery on it.

As in the classical DPA by Kocher et al. [KJJ99], we can now hypothesize the value of each key bit $h_{i,j}$ separately. We further know at which clock cycle the leakage of the carry registers (for the key rotation) occurs. Based on this knowledge, one can build the following attack.

3.2.2.2 DPA of Key Rotation

As mentioned above, we do not distinguish $h_{0,j}$ and $h_{1,j}$. Instead, we predict the combined leakage $h_{\Sigma,j} = h_{0,j} + h_{1,j}$. Our key recovery works well for this combined leakage, as explained in Section 3.2.4. Note that we know for each key bit $h_{i,j}$ at which clock cycle it is processed (if not, several hypotheses can be checked in parallel by analyzing neighboring clock cycles). In fact, knowing the implementation, it is predictable which key bit $h_{i,j}$ enters the carry register in which clock cycle for the key rotation. We use this information to build a differential power analysis attack. In spite of the independence of the input x we claim the analysis method to be differential leakage analysis, since differential leakage traces can be computed—similar to the approach originally proposed in [KJJ99].

Our algorithm identifies all clock cycles where $h_{i,j}$ is written to or overwritten in the carry register in each trace L and extracts that leakage from L. Per processed ciphertext bit, only 150 words are rotated. The additional bit is stored in the carry register. Hence, all rotations together result in a total of $4801 \cdot 150$ carry register overwrites for each h_i. Since there are 4801 bits in h_i, each bit is written to the carry register 150 times. The corresponding clock cycles l are then identified and their corresponding leakage $\mathcal{L}_i(j,l)$ is combined, as done in horizontal SCA. The result is a differential leakage trace Δ_{carry} with only one bin per key bit. In other words, the *difference* between a key bit being zero and a key bit being one can be observed by comparing points of the leakage trace Δ_{carry} horizontally. Since the key is sparse, there are only very few bins that correspond to a bit $h_{i,j} = 1$, while most bins correspond to a bit $h_{i,j} = 0$. The implicit assumption of all bits leaking the same way is perfectly justified: each bit $h_{i,j}$ takes each column position exactly once, in a specific row. That means due to the rotation, each key bit leaks in every position exactly once, averaging out any position-specific leakages.

In order to detect whether a key bit is set, i. e., $h_{i,j} = 1$, we average over all clock cycles where $h_{i,j}$ is written to the carry register.

$$\begin{aligned}
\Delta_{\text{carry}}(j) &= \frac{1}{150}\sum_{l=1}^{150}\left(\mathcal{L}_0(j,l) + \mathcal{L}_1(j,l)\right) \\
&= \operatorname{avg}\left(\lambda_{\text{carry}}(0,j) + \lambda_{\text{read}}(0,j) + \lambda_{\text{store}}(0,j)\right. \\
&\qquad \left. + \lambda_{\text{carry}}(1,j) + \lambda_{\text{read}}(1,j) + \lambda_{\text{store}}(1,j)\right)
\end{aligned}$$

Since $h_{i,j-32} = 0$ with very high probability, $\Delta_{\text{carry}}(j)$ depends directly on the key bit. Further, $h_{i,j} = 1$ has an even stronger influence on $\Delta_{\text{carry}}(j \pm 32)$, since it leaks through $\lambda_{\text{carry}}(i,j)$ and either $\lambda_{\text{read}}(i,j)$ or $\lambda_{\text{store}}(i,j)$. The dependence of $\Delta_{\text{carry}}(j)$ on neighboring key bits $h_{i,j\pm\delta}$, with $\delta \leq 32$, implies that each set key bit not only results in an increased leakage signal for its own position (i. e., index j), but also in the neighboring positions. Note that due to the differing weights, each set key bit imprints a characteristic shape onto the leakage trace. These shapes can (and actually will) overlap if several key bits in the same region are set. Figure 3.2 shows

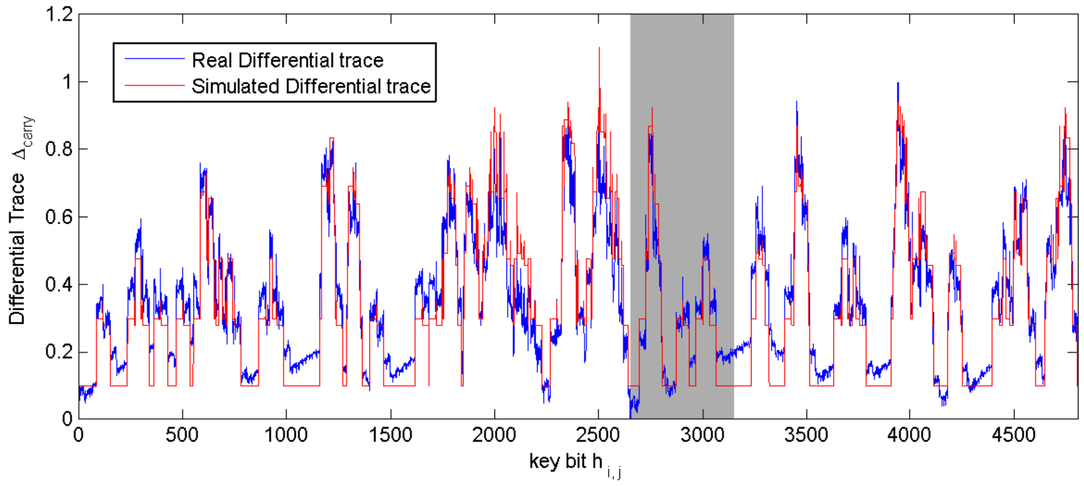


Figure 3.2: Differential leakage trace for key rotation. The plot shows the normalized leakage (vertical axis) of both key parts $h_{\Sigma,j} = h_0 + h_1$ over the key bit index (horizontal axis). The red(gray) line is the simulated leakage while the blue/black line is the observed leakage from the target implementation.

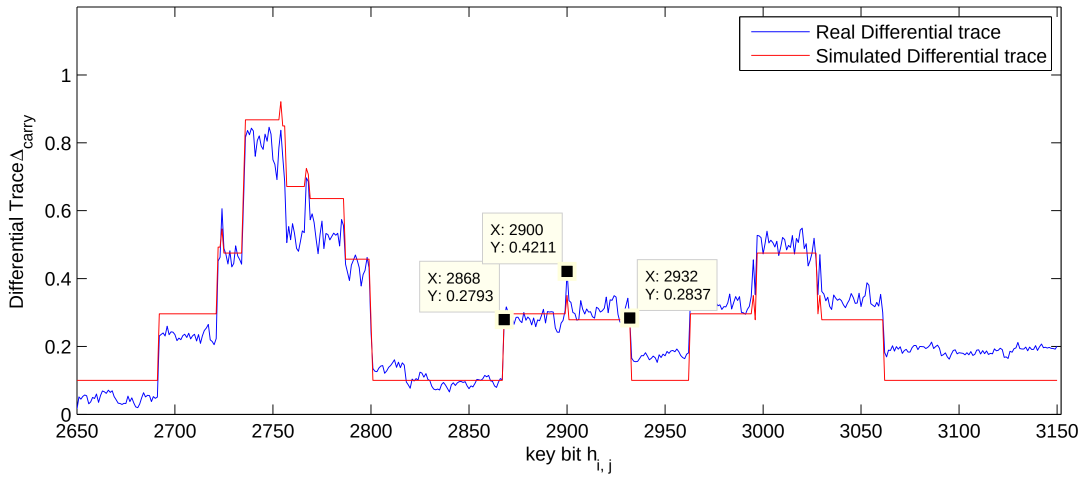


Figure 3.3: A magnified version of Figure 3.2 that highlights the characteristic shape of a single set bit (center) as well as the overlap of two (right) and three (left) “adjacent” set bits.

the comparison of the simulated leakage trace (red(gray) line) using the power model and the real leakage trace (blue/black line). The characteristic shape is highlighted in Figure 3.3, which is a magnification of a single set bit of the key, surrounded by zeroes.

In summary, the key rotation analysis allows us to detect joint leakages of h_0 and h_1. This is due to the target implementation that processes both in parallel. The key rotation leakage features a characteristic shape with easily detectable bounds. This allows for a precise location of set key bits. Furthermore, the analysis of the key rotation is mostly input-independent, as will be discussed in Section 3.2.3. More importantly, each bit features 150 leakage observations per trace L, resulting in a very strong leakage.

3.2.2.3 Key Bit Recovery

The key rotation causes leakages which can be analyzed in the presented differential leakage traces where characteristic shapes caused by set key bits can be detected and used to recover the set key bits. In the same way, the traces can be used to detect key bits that are not set. Since the analyzed implementation processes h_0 and h_1 in parallel during the key rotation, resulting in an overlap of the leakages, the differential leakage trace actually recovers the key bits of $h_\Sigma = h_0 + h_1$.

In order to recover key bits, the characteristic shapes need to be detected. We propose a generic shape detection algorithm that works as follows:

1. **Shape Definition** From the differential leakage trace, one singular characteristic shape can be identified and used as a template for set bits. The template is used to generate a shape threshold as shown in Figure 3.3. The threshold is defined by the value of features in this shape such as edges, slopes and pulses.

2. **Shape Detection** For each key bit in the differential leakage trace, we check

if this key bit together with the neighboring key bits can form a characteristic shape. This is done by checking if there are features that are beyond the threshold. If more than two features exist, it is highly probable that this key bit is set. If no feature exists, then it is highly probable that this key bit is 0. Otherwise, we mark this key bit as an undetermined bit.

Note that the shapes will overlap if two set key bits are close to each other. Furthermore, the leakage traces are noisy, hence we can only recover parts of the key bits, leaving the other key bits as undetermined. By choosing the thresholds for shape detection carefully, the number of detected bits can be maximized while keeping the number of false positive errors as low as needed.

3.2.3 Measurement Setup and Results

We ported the implementation of [vMG14a] to a Xilinx Virtex-5 LX50 FPGA which is mounted on a Sasebo-GII side-channel attack evaluation board. The implementation is clocked at 3 MHz by default. Measurements were performed using a Tektronix DPO 5104 oscilloscope at a sampling rate of 100 MS/s. Since our attack focuses on the syndrome computation, only the syndrome computation was recorded. The syndrome computation takes 245 ms, resulting in long traces. For the ease of analysis, a peak extraction was performed. In each clock cycle only the point of maximum power consumption is retained. The peak extraction prevents potential alignment issues and makes data handling much faster.

3.2.3.1 DPA Results of the Key Rotation

Since the key rotation is independent of the ciphertext, the choice of the ciphertext could be arbitrary. However, key rotation and syndrome computation run in parallel, leading to a mixed leakage. To determine the influence of the syndrome computation, two different ciphertext scenarios are studied. One is the all-0 ciphertext to minimize the influence of the syndrome computation. In this scenario the syndrome remains all-0 throughout the entire computation. The other scenario assumes random ciphertexts for each decryption, where each bit in x is set with a 50% probability. For each scenario we took 256 measurements.

Next, we averaged over all considered traces in both scenarios. From the resulting average trace, $4801 \cdot 150$ peaks are extracted and used to construct the differential leakage traces Δ_{carry} as explained in Section 3.2.2.2. Note that averaging explicitly before the computation of Δ_{carry} or implicitly during the computation of Δ_{carry} does not influence the result. Figure 3.4 shows the differential leakage traces for the key rotation, showing the key bit position (horizontal axis) vs. the bit leakage (vertical axis) for all key bits. The blue (black) line indicates the result for the all-0 ciphertext scenario while the green (gray) line indicates the results for the random ciphertext. The latter one is slightly noisier, but nevertheless provides a well-exploitable leakage for a low number of observations. Figure 3.3 shows magnifications of the differential leakage trace to highlight the characteristic shapes, particularly the one generated by setting the key bit $h_{i,2900}$ as 1 and the neighboring key bits as 0.

The other shapes in Figure 3.3 result from the overlapping of characteristic shapes that occur when set key bits of h are close to each other. We noticed that set key bits for h_0 result in a slightly different shape than those of h_1. Since this difference cannot be distinguished as easily, we did not further try to exploit this information.

3.2.3.1.1 Key Extraction. To extract keys from Δ_{carry}, we used the algorithm described in Sec. 3.2.2.3. The first step is to define the characteristic shape. Distinguishable features such as the rising edge, the pulse in the center and the falling edge

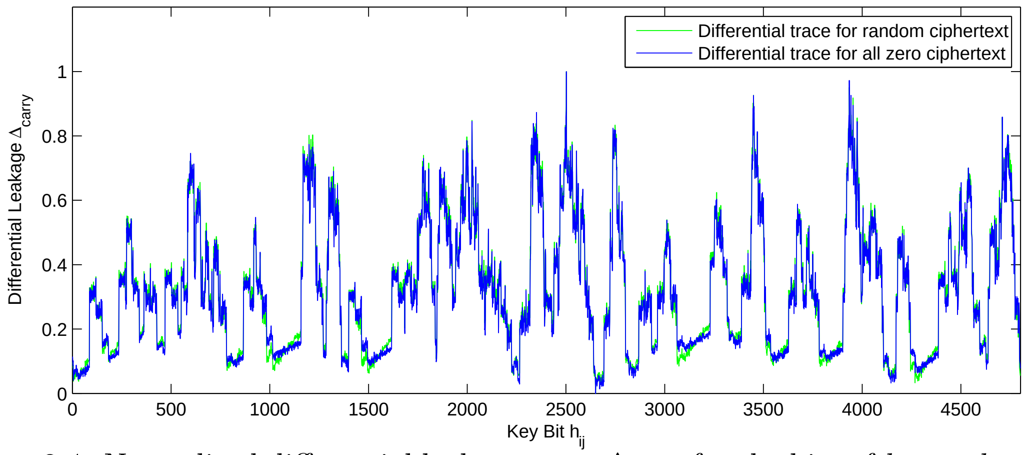
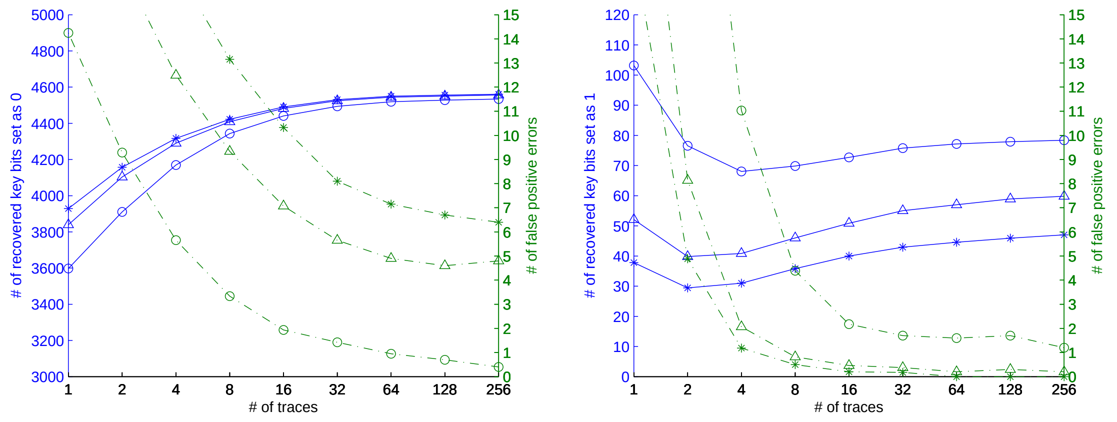


Figure 3.4: Normalized differential leakage trace Δ_{carry} for the bits of $h_{\Sigma,j} = h_0 + h_1$. Whether the ciphertext is known (green(gray) line) or all-0 (blue(black) line) has only marginal influence on the observed leakage.

are clearly visible in Figure 3.3 and are used to detect the shape. These features are quantified using a threshold vector. Then, for each key bit $h_{i,j}$ in Δ_{carry}, we check if there is a pulse at $h_{i,j}$, a rising edge at $h_{i,j-32}$ and a falling edge at $h_{i,j+32}$. If more than one feature exists for $h_{i,j}$, we take $h_{i,j}$ as 1. If no feature exists, $h_{i,j}$ is taken as 0. If only one feature exists, $h_{i,j}$ is left as undetermined key bit. Depending on the number of traces used for generating Δ_{carry}, it can be noisy and there will be false positive errors in recovered key bits. Errors can also be introduced by unfavorable overlapping of shapes.

Figure 3.5 shows how the chosen threshold affects the key recovery. Three different thresholds are used. The first one ($\circ$) is exactly the value extracted from the characteristic shape in Δ_{carry}. The other two ($\triangle$ and then $*$) are increased based on the first one. In Figure 3.5.1, as the number of traces used to generate the differential leakage trace increases, the number of recovered 0 key bits increases and the number of false positive errors decreases for all three thresholds. However, the less aggressive the threshold is, the lower is the number of false positive errors. In contrast, Figure 3.5.2 shows that with the least aggressive threshold ($\circ$), more key



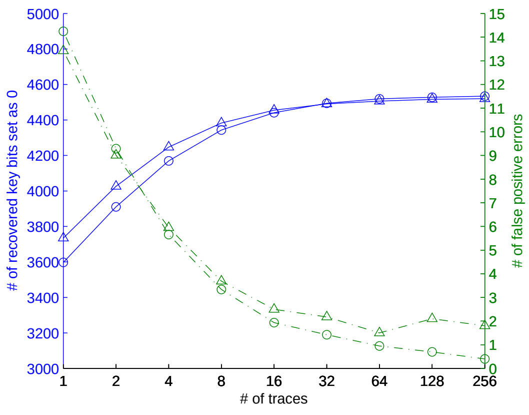
3.5.1: Recovered 0 bits vs. false positives 3.5.2: Recovered 1 bits vs. false positives

Figure 3.5: Key bit recovery rates for recovering 0 key bits and 1 key bits. Solid line indicates the number of recovered bits (out of 90 ones and 4711 zeros, scale on left), the dashed line indicates the number of false positives (scale on right). Markers $\circ$, then $\triangle$, and then $*$ indicate the increasing values for the threshold.

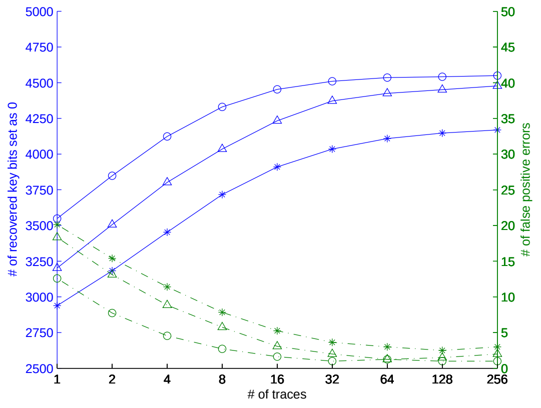
bits of 1 can be recovered with a few more false positive errors. Hence, to recover more key bits of 0 with least false positive errors, the less aggressive threshold should be used. In contrast, to recover key bits of 1 with least false positive errors, the more aggressive threshold should be used. Note that we repeated our experiments for five different randomly generated keys to ensure the result is not key dependent. The figures show the average result for those experiments.

Figure 3.6.1 shows a comparison of the number of recovered key bits and false positive errors between the all-0 ciphertext and random ciphertext. As the number of traces used to generate the differential leakage trace increases, the number of recovered key bits of 0 increases and the number of false positive errors decreases for both cases. However, with the all-0 ciphertext, there are less positive errors. In conclusion, the all-0 ciphertext is more advantageous to the DPA of key rotation. Hence, we use the traces with the all-0 ciphertext in the other experiments.

Modern electronic devices run faster than 3 MHz which is the default clock rate for the SASEBO board and widely used in power analysis experiments. In order



3.6.1: Random vs. all-0 input



3.6.2: Varying clock rates

Figure 3.6: Key bit recovery rates for recovering 0 key bits. Solid line indicates the number of recovered bits (out of 4711 zeroes, scale on left), the dashed line indicates the number of false positives (scale on right). The left figure compares *known* random ($\circ$) vs. *chosen* all-0 ($\triangle$) ciphertext inputs. The right figure compares the experiments for varying clock rates: $\circ$ for 3 MHz, $\triangle$ for 8 MHz, and $*$ for 16 MHz.

to validate our attack on faster platforms, the performance of the attack was measured for the same design clocked at 8 MHz and 16 MHz. The sampling rate was accordingly increased to to 200 MS/s and 250 MS/s, respectively. For each case, 256 traces were obtained using the all-0 ciphertext, followed by peak extraction. Figure 3.6.2 shows the degradation of the leakage over the increasing clock rate by comparing the number of recovered 0 key bits and false positive errors. In all three cases, the number of recovered 0 key bits increases and the number of false positive errors decreases, as the number of analyzed traces increases. However, the lower the clock rate is, the better the key bits extraction works. With a 3 MHz clock rate ($\circ$), almost 4500 of the 0 key bits can be recovered with about 1 false positive error when using all 256 traces while 4000 of the 0 bits are recovered with about 3 false positive errors at a clock rate of 16 MHz ($*$).

Overall, it can be seen that with as little as 10 measurements, more than half the key bits can be recovered with a remaining number of errors that is small enough to allow for efficient error correction. With 100 measurements and a careful choice of thresholds, the determined bits are entirely error-free at lower clock rates. This strong leakage is partially due to the fact that 150 leakages are extracted from each measurement, strongly amplifying the amount of leakage gained from each individual trace.

3.2.4 Full Key Recovery

Next we analyze how to recover the full key of QC-MDPC McEliece if the adversary has knowledge of several 1 bits of the key as well as several 0 bits of the key, possibly with few errors. We show that the structure of the key can be used to recover the remaining uncertain bits efficiently, or to detect remaining errors.

3.2.4.1 Connection between Secret Key and Public Key

As described before, the secret key consists of two related parts, h_0 and h_1. Due to the relation between the secret h_0, h_1 and the public matrix Q, we can express h_0 as:

$$h_0 = h_1 \cdot Q^T \tag{3.2}$$

Likewise, given h_0, one can compute h_1, since Q is invertible. This means that once the first half of the secret key is recovered, the second half can be computed using the public key. More interestingly, this relationship can be used for *error detection* for each h_i independently: since Q is of high weight (each bit has approximately a 50% chance of being 1), even a single bit error in h_i^* will result in a high weight of a consequently derived $h_{\bar{i}}^*$, i. e., $\mathrm{wt}(h_{\bar{i}}^*) \approx r/2$. A correct h_i, however, will result in an $h_{\bar{i}}$ of low weight, in our case $\mathrm{wt}(h_{\bar{i}}) = 45$. We are currently not aware how slightly faulty or noisy information of h_0 and h_1 can be combined more efficiently without a trial and error approach using the above-mentioned relationship.

If the adversary observes a combined leakage of h_0 and h_1 as described above, this is not a problem, since knowledge of $h_0 \oplus h_1$ can also enable key recovery. Adding h_1 on both sides of Equation (3.2) we obtain

$$h_0 \oplus h_1 = h_1 \cdot (Q^T \oplus I_{4801}). \tag{3.3}$$

If side-channel leakage allows us to obtain the combined leakage $h_0 \oplus h_1$ and the rank of $Q^T \oplus I_{4801}$ is high, we can solve this linear system of equations for h_1 with a computer algebra system like Magma [BCP97]—and then derive h_0 from Equation (3.2). In our experiments, the rank observed for $Q^T \oplus I_{4801}$ was 4800, resulting in two candidate solutions with only one of them having the correct Hamming weight. So in cases where all ones can be correctly identified, Equations (3.2) and (3.3) enable a practical key recovery.

Due to noise and leakage overlapping, there are probably false positive errors in the recovered bits and hence error correction would be essential to correct positions that are slightly off. Guessing error positions becomes infeasible quickly, even with small improvements over an exhaustive search of $\binom{4801}{l}$ possibilities for l errors. We did not try to devise elaborate error-correction strategies, as a different attack strategy which relies on exploiting detected zeroes turned out to be quite effective. We explain this strategy next.

3.2.4.2 Efficient Key Recovery

After having identified several 1 bits and 0 bits of the secret key correctly, we aim at an efficient way to recover remaining unknown or uncertain key bits. For this, we define B_0, B_1 and B_u as index sets indicating the locations of definite zeroes, definite ones and positions of undetermined bits in $h_0 \oplus h_1$ such that

$$B_0 \mathbin{\dot\cup} B_1 \mathbin{\dot\cup} B_u = \{0, 1, \ldots, 4800\} . \tag{3.4}$$

Positions in B_0 indicate that both h_0 and h_1 are zero in that position, while positions in B_1 will mean a one in either h_0 or h_1.[1] Hence, the uncertain positions for h_1 are $B_u^1 = B_1 \mathbin{\dot\cup} B_u$, and with Iverson's convention [Knu92] we can summarize our knowledge of $h_0 \oplus h_1$ and h_1 as $h_0 \oplus h_1 = \langle 1 \cdot [i \in B_1] + u \cdot [i \in B_u] \rangle_{0 \le i \le 4800}$ and $h_1 = \langle u \cdot [i \in B_u^1] \rangle_{0 \le i \le 4800}$, where u indicates unknown bits ("erasures"). So Equation (3.3) yields

$$\langle 1 \cdot [i \in B_1] + u \cdot [i \in B_u] \rangle_{0 \le i \le 4800} = \left\langle u \cdot [i \in B_u^1] \right\rangle_{0 \le i \le 4800} \cdot (Q^T \oplus I_{4801}).$$

As the indices in B_0 indicate definite zeros in $h_0 \oplus h_1$ and h_1, the corresponding *rows* in the matrix $Q^T \oplus I_{4801}$ will always be multiplied with a zero coefficient. We remove these $|B_0|$ rows and the corresponding known 0-entries in h_1, obtaining an updated equation system

$$\langle 1 \cdot [i \in B_1] + u \cdot [i \in B_u] \rangle_{0 \le i \le 4800} = \left\langle u \cdot [i \in B_u^1] \right\rangle_{i \notin B_0} \cdot Q'. \tag{3.5}$$

with a (smaller) matrix $Q' \in \mathbb{F}_2^{(4801 - |B_0|) \times 4801}$. There are $4801 - |B_0| - |B_1|$ unknown bits on the left- and $4801 - |B_0|$ unknown bits on the right-hand side of Equation (3.5). As we are only interested in finding h_1, we can try to eliminate unknown values in $h_0 \oplus h_1$ by dropping *columns* from Q'. One may hope that $|B_u|$ columns can be eliminated without Q' dropping in rank, so that we end up with a linear

[1]The (rare) case of h_0 and h_1 having a one in the same position is not considered here, as this situation is quite apparent from the side-channel leakage.

system of equations

$$\langle 1 \cdot [i \in B_1] \rangle_{i \notin B_u} = \left\langle u \cdot [i \in B_u^1] \right\rangle_{i \notin B_0} \cdot Q'' \tag{3.6}$$

in $4801 - |B_0|$ unknowns and a matrix $Q'' \in \mathbb{F}_2^{(4801-|B_0|)\times(4801-|B_u|)}$. If $|B_u| \leq |B_0|$ one may hope that this linear system of equations can be solved and yields a unique candidate for h_1.

To check the practical feasibility of this approach, we ran several experiments in Magma [BCP97], solving the equation system given in (3.6) for several different vectors B_0 and B_1. We were particularly interested in the situation where knowledge of 1-positions in $h_0 \oplus h_1$ is ignored (i. e., $B_1 = \emptyset$), because in our measurements the 0-detection was more reliable. With $B_1 = \emptyset$, the resulting system of equations is homogeneous and thus in addition to h_1 also has the trivial solution. From Equation (3.4) we see that the condition $|B_u| \leq |B_0|$ now implies that $|B_0| \geq \lceil 4801/2 \rceil$. Staying above this threshold, in our experiments we obtained no more than 8 candidates for h_1, and the weight condition identified the correct secret key uniquely.

For $|B_0| < 2400$, the kernel of the matrix Q'' in Equation (3.6) gets larger quickly and we obtain additional candidates for h_1, but finding the correct h_1 may still be feasible by looking at the Hamming weight of the candidates as long as the number of candidates is not overwhelming. The results in Section 3.2.3 show that for the target implementation the attacker can expect to recover more information from the side-channel than necessary for recovering the secret key. Having $|B_0|$ comfortably above the threshold of 2400, a few false positives in B_0 can be dealt with efficiently: Instead of using all of these bit positions, one can select subsets of size 2401 at random. Assuming a hypergeometric distribution, with f false positive errors among the $|B_0|$ indices, the probability of guessing 2401 error-free positions is $\binom{|B_0|-f}{2401}/\binom{|B_0|}{2401}$. E. g., with $|B_0| = 3281$ and $f = 4$, this probability is still $\approx 2^{-7.6}$. In summary, as long as more than half the bits of the key can be recovered with a low error rate, the remaining key bits can be determined using the above-described algebraic methods. Knowledge of additional bits of $h_0 \oplus h_1$ facilitates the handling of possibly remaining errors. Not being able to recover more than half the number of key bits can make the search infeasible, although—due to the highly biased key—guessing a few additional zeroes may still be an option.

3.3 Masking QC-MDPC McEliece

In this section, we show how to secure QC-MDPC McEliece by applying Threshold Implementation which aims to randomize sensitive intermediate states such that the leakage is independent of processed secrets. In QC-MDPC McEliece, the key bits and the syndrome are sensitive values that need to be protected and therefore they must be masked whenever they are manipulated. Similarly, since the decoding operation processes the sensitive syndrome, leakage of the decoder needs to be masked as well. We validate the side channel resistance of our design by practical DPA attacks and statistical tests for leakage detection.

3.3.1 Masked Syndrome Computation

As described before, the decoding algorithm begins with the syndrome computation $s = Hx^T$. Both the parity-check matrix H and the syndrome s are sensitive values and can cause side channel leakage. However, since the syndrome computation is a linear operation, masking this operation is simple and efficient. Intuitively, H can be split into two shares, H_m and M such that $H = H_m \oplus M$, by Boolean masking.

The mask matrix M is created in correspondence to H, by first generating uniformly distributed random masks for h_i, $m_0, \ldots, m_{n_0-1} \in \mathbb{F}_2^r$ of the n_0 blocks, which then comprise the first row of mask matrix M. Each bit in the m_i is uniformly set to 0 or 1. Next, the remaining rows of the mask matrix M are obtained by quasi-cyclic shifts of the first row, according to the construction of H. The masked syndrome s_m and the syndrome mask m_s can be computed independently as $s_m = H_m x^T$ and $m_s = Mx^T$. The syndrome s is available as the combination of the two shares $s = s_m \oplus m_s$.

3.3.2 Masked Decoder

After syndrome computation, the error correction decoder computes the number of unsatisfied parity check equations between the sensitive syndrome and one row of the sensitive parity check matrix. By comparing that number with a predefined threshold (usually denoted b), the decoder decides whether to flip the corresponding bit in the ciphertext. Masking the actual decoding steps is more complex, since both inputs, namely the syndrome and the parity check matrix, as well as the control flow of the decoder can leak sensitive information and thus need to be protected. Unlike the syndrome computation, the decoder performs a binary AND and a Hamming weight computation on sensitive data. Both operations are non-linear and thus need more elaborate protection than just a straightforward Boolean masking. In the following we explain how these operations can be implemented. Algorithm 1 describes the masked version of the decoder. Note that the algorithm has been formulated with a constant execution flow to better represent the intended hardware implementation. Further note that the algorithm and its FPGA implementation exhibit a constant timing behavior (except the number of decoder iterations) and that all key-related variables are masked. The number of decoder iterations can be

Algorithm 1 Masked Error Correction Decoder

Input: H_m, M_1, M_2, s_m, m_{s_1}, m_{s_2}, x, $B = b_0, ..., b_{max-1}$, max
Output: Error free codeword x or DecodingFailure

```
1:  for $i = 0$ to max$-1$ do
2:      for every ciphertext bit $x_j$ do
3:          $\#_{\text{upc}} = \mathsf{SecHW}(\mathsf{SecAND}(s_m, m_{s_1}, m_{s_2}, H_{m,j}, M_{1,j}, M_{2,j}))$
4:          $d = (\#_{\text{upc}} > b_i)$ , $d \in \{0,1\}$
5:          $x = x \oplus (d \cdot 1_j)$                                  ▷ Flip the $j$th bit of $x$
6:          $s_m = s_m \oplus (d \cdot H_{m,j} \oplus \bar{d} \cdot M_{2,j})$          ▷ Update syndrome
7:          $m_{s_1} = m_{s_1} \oplus M_{1,j}$                              ▷ Update masks
8:          $m_{s_2} = m_{s_2} \oplus M_{2,j} \oplus (\bar{d} \cdot M_{1,j})$
9:      end for
10:     if $\mathsf{SecHW}(s_m, m_{s_1}, m_{s_2}) == 0$ then         ▷ Check for remaining errors
11:         return $x$
12:     end if    ▷ For constant run time, this if-statement can be moved after the
        for-loop
13: end for
14: return DecodingFailure
```

set to maximum by simply moving the if-statement out of the loop. For the chosen 9602/4801 parameter set, max would be set to 5, increasing the average run time roughly by a factor 2 (cf. [vMOG15]).

In Algorithm 1, we make use of two special functions. Function SecAND computes the bitwise AND operation between syndrome s and secret key H in a secure way without leaking any sensitive information. The other function SecHW computes the Hamming Weight of a given vector. Both functions are explained in detail in the following. An all-zero vector with the jth bit equal to 1 is indicated by 1_j.

Secure AND Computation. One important step when decoding a QC-MDPC code is to compute the unsatisfied parity-check equations which starts with a non-linear bitwise AND operation between the syndrome and one row of the secret key matrix. Our function SecAND performs a bitwise AND operation between two bit vectors, namely $s \wedge h$. Since the AND is a non-linear operation, simple two-share Boolean masking is not applicable. Instead, we follow the concept of Threshold

Implementation as described in Section 2.2.1. We adopt the bitwise AND operation from [NRR06], which provides first-order security when applied to *three* Boolean shares. This means that the two-share representations of the two inputs, i. e., the syndrome and parity check matrix, need to be extended to a three-share representation.

To achieve a three-share representation of both syndrome and parity check matrix, the masking is expanded in the following way: After syndrome computation as explained in Section 3.3.1, the syndrome is represented as $s_m \oplus m_s$ and the secret key is represented as $H_{m,j} \oplus M_j$. Next, the syndrome representation is extended as $s_m \oplus m_{s_1} \oplus m_{s_2}$ and the key as $H_{m,j} \oplus M_{1,j} \oplus M_{2,j}$. Here, m_{s_2} and $M_{2,j}$ are two new uniformly distributed random mask vectors and m_{s_1} is derived as $m_{s_1} = m_s \oplus m_{s_2}$ and $M_{1,j} = M_j \oplus M_{2,j}$. The following equations show how to achieve a TI version of $s \wedge h$ that satisfies correctness and non-completeness, but not uniformity.

$$\begin{aligned} s \wedge h =& (s_m \oplus m_{s_1} \oplus m_{s_2}) \wedge (H_{m,j} \oplus M_{1,j} \oplus M_{2,j}) \\ =& (s_m \wedge H_{m,j}) \oplus (s_m \wedge M_{1,j}) \oplus (H_{m,j} \wedge m_{s_1}) \oplus \\ & (m_{s_1} \wedge M_{1,j}) \oplus (m_{s_1} \wedge M_{2,j}) \oplus (M_{1,j} \wedge m_{s_2}) \oplus \\ & (m_{s_2} \wedge M_{2,j}) \oplus (m_{s_2} \wedge H_{m,j}) \oplus (M_{2,j} \wedge s_m) \end{aligned} \tag{3.7}$$

As pointed out in [NRR06], in order to fulfill uniformity, one can introduce additional uniform random masks to mask each share. By introducing two more uniformly random vectors r_1 and r_2, the three output shares can be computed as follows. Let sh denote the result of the TI version of the AND operation. Using the equations above, sh can be split into three shares sh_i, which are now uniformly distributed thanks to the r_i and are given as:

$$\begin{aligned} sh_1 =&(s_m \wedge H_{m,j}) \oplus (s_m \wedge M_{1,j}) \oplus (H_{m,j} \wedge m_{s_1}) \oplus r_1 \\ sh_2 =&(m_{s_1} \wedge M_{1,j}) \oplus (m_{s_1} \wedge M_{2,j}) \oplus (M_{1,j} \wedge m_{s_2}) \oplus r_2 \\ sh_3 =&(m_{s_2} \wedge M_{2,j}) \oplus (m_{s_2} \wedge H_{m,j}) \oplus (M_{2,j} \wedge s_m) \oplus r_1 \oplus r_2 \end{aligned} \tag{3.8}$$

Secure Hamming Weight Computation. In the unprotected FPGA implementation of [vMG14a], the Hamming weight computation of sh is performed by looking up the weight of small chunks of sh from a precomputed table and then accumulating those weights to get the Hamming weight of sh. However, the weight of a chunk is always present in plain and the computation of it can result in side channel leakage that will lead to the recovery of the Hamming weight. Even though the knowledge of the weight does not necessarily recover the chunk value, it still yields information about sh and thus the secret key h.

For a side-channel secure implementation, both the input and the output of a Hamming weight computation for each chunk must be masked. Since the weight of all chunks needs to be accumulated, it is preferable to use Arithmetic masking instead of Boolean masking. For example, the Hamming weight of sh can be calculated using the following equation:

$$\mathrm{wt}(sh) = \sum_{i=1}^{|sh|} sh_{1,i} \oplus sh_{2,i} \oplus sh_{3,i} \tag{3.9}$$

where subscript i refers to the i-th bit of each share and $|sh|$ is the length of sh in bits. Using a secure conversion function from Boolean masking to Arithmetic masking [CGV14], each Boolean mask tuple $(sh_{1,i}, sh_{2,i}, sh_{3,i})$ can be converted to an Arithmetic mask pair $(A_{1,i}, A_{2,i})$ such that $sh_{1,i} \oplus sh_{2,i} \oplus sh_{3,i} = A_{1,i} + A_{2,i}$. Then, the Hamming weight of sh can be computed as:

$$\mathrm{wt}(sh) = \sum_{i=1}^{|sh|} A_{1,i} + A_{2,i} = \sum_{i=1}^{|sh|} A_{1,i} + \sum_{i=1}^{|sh|} A_{2,i} \tag{3.10}$$

According to Equation (3.10), we only accumulate $A_1 = \sum_{i=1}^{|sh|} A_{1,i}$ and $A_2 = \sum_{i=1}^{|sh|} A_{2,i}$, respectively, and sum them up in the end to obtain the total Hamming weight $\mathrm{wt}(sh) = A_1 + A_2$.

Secure Syndrome Checking. In order to test whether decoding of the input vector was successful, the syndrome has to be tested for zero. If the Hamming weight of the syndrome is zero, then all bits of the syndrome must be zero. Otherwise, there must be some bits set as 1 and the number of set bits equals the Hamming weight of the syndrome. Note that we perform SecHW operation over the three shares of syndrome s in order to prevent the leakage.

3.3.3 Implementing a Masked QC-MDPC McEliece

This section presents more details of the masked FPGA implementation of QC-MDPC McEliece decryption based on the unprotected one in [vMG14a]. We follow the structure of the original design, including the same security parameters, but replace vulnerable logic circuits with masked circuits.

3.3.3.1 Overview of the Masked Implementation

Each time before the decryption is started, both the ciphertext and the masked secret keys h_{0m}, h_{1m} are written into the BRAMs of the decryption engine. As shown in Figure 3.7, one BRAM stores the $2 \cdot 4801$-bit ciphertext, the second BRAM stores the $2 \cdot 4801$-bit masked secret key and third BRAM stores the 4801-bit masked syndrome and the 4801-bit syndrome mask. Note that the secret keys are masked

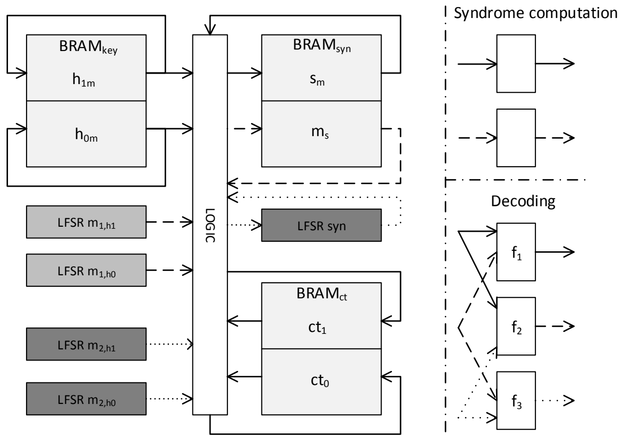


Figure 3.7: Abstract block diagram of the masked QC-MDPC McEliece decryption implementation.

before being transferred to the crypto core. The seeds for the internal PRG are transferred with the masked key. Each BRAM is dual-ported, offers 18/36 kBit, and allows to read/write two 32-bit values at different addresses in one clock cycle.

Computations are performed in the same order as in [vMG14a]: To compute the masked syndrome s_m, set bits in the ciphertext x select rows of the masked parity-check matrix blocks that are accumulated. In parallel, the syndrome mask m_s is computed in the same manner. Rotating the two parts of the secret key is implemented in parallel, as in the unprotected implementation. Efficient rotation is realized using the READ_FIRST mode of Xilinx's BRAMs which allows to read the content of a 32-bit memory cell and then to overwrite it with a new value, all within one clock cycle.

An abstraction of this implementation is depicted in Figure 3.7. The three block RAMs are used to store the masked keys (h_{0m} and h_{1m}), the shared syndrome (s_m and m_s) and the ciphertext (ct_0 and ct_1). The LFSR blocks are used to generate the missing masks on-the-fly. The logic blocks for the two phases of the McEliece decryption are shown on the left side of Figure 3.7.

3.3.3.2 Masking Syndrome Computation

The syndrome computation is a linear operation and requires only two shares for sensitive variables. Once the decryption starts, 32-bit blocks of the masked secret keys h_{0m}, h_{1m} are read from the secret key BRAM at each clock cycle and are XORed with the 32-bit block of s_m read from the syndrome BRAM depending on whether the corresponding ciphertext bits are 1. Then the result will be written back into the syndrome BRAM at the next clock cycle and at the same time the rotated 32-bit blocks of the masked keys will be written back into the secret key BRAM. Meanwhile, we need to keep track of the syndrome mask m_s. Since syndrome computation is a linear operation, we can similarly add up the secret key masks synchronously to generate the syndrome mask. In our secure engine, we use two 32-bit leap forward LFSRs to generate random 32-bit secret key masks each clock cycle which are XORed with the 32-bit block of m_s read from the syndrome BRAM depending on the ciphertext.

Cyclic Rotating LFSRs. Our 32-bit leap forward LFSRs not only generate a 32-bit random mask at each clock cycle but also rotate synchronously with the key. For example, the LFSR for h_{0m} first needs to generate the 4801-bit mask m_{h_0} in the following sequence: $m_{h_0}[0 : 31], m_{h_0}[32 : 63], \ldots, m_{h_0}[4767 : 4799], m_{h_0}[4800]$. This is done in 150 clock cycles. In the next round, the secret key is rotated by one bit as $h_{0m} \ggg 1$ and hence the mask sequence should be: $m_{h_0}[4800 : 30], m_{h_0}[31 : 62], \ldots, m_{h_0}[4766 : 4798], m_{h_0}[4799]$. After 4801 rounds of rotation, the LFSR ends up with its initial state. In order to construct a cyclic rotating PRG with a period of 4801 bits, we combine a common 32-bit leap forward LFSR with additional memory and circuits, based on the observation that the next state of the LFSR either completely relies on the current state or actually sews two ends of the sequence together, e. g., $m_{h_0}[4800 : 30]$. As shown in Figure 3.8, five 32-bit registers are employed instead of just one. The combinational logic circuit computes the next 32-bit random mask given the input stored in IntStateReg. The following steps describe the functionality of our LFSR:

1. Initially, the 32-bit seed $seed[0:31]$ of the sequence is stored in register IvReg and the first 32 bits of the sequence, e. g., $m_{h_0}[0:31]$ are stored in the other registers.

2. During the rotation, the combinational logic circuits output the new 32-bit result and feed it back. If the new result is part of the 4801-bit sequence, then it will go through the Mux, overwriting the current state registers IntStateReg and ExtStateReg at the next clock cycle.

3. If the new result contains bits that are not part of the sequence, then those bits will be replaced. For example, when $m_{h_0}[4767:4799]$ is in IntStateReg, the new result will be $m_{h_0}[4800:4831]$ in which only bit $m_{h_0}[4800]$ is in the mask sequence and $m_{h_0}[4801:4831]$ will be dropped. The Mux gate will only let $m_{h_0}[4800]$ go through together with $m_{h_0}[0:30]$ stored in ExtBit$_{0\_31}$ and the concatenation $m_{h_0}[4800:30]$ will overwrite register ExtStateReg.

4. $m_{h_0}[4800:30]$ will not be written into register IntStateReg because given $m_{h_0}[4800:30]$ as input, the combinational logic circuit will not output the next valid state $m_{h_0}[31:62]$. Therefore, we concatenate part of the seed in IvReg and part of the first 32-bits in IntBit$_{0\_31}$, e. g., $\{seed[31], m_{h_0}[0:30]\}$ and overwrite IntStateReg. Then, the new output will be $m_{h_0}[31:62]$. The concatenation is implemented as a cyclic bit rotation as shown in Figure 3.8. After 32 rotations, the seed is rotated to IntBit$_{0\_31}$ and the first 32-bit $m_{h_0}[0:31]$ is rotated to IvReg. Hence, they will be swapped back in the next clock cycle.

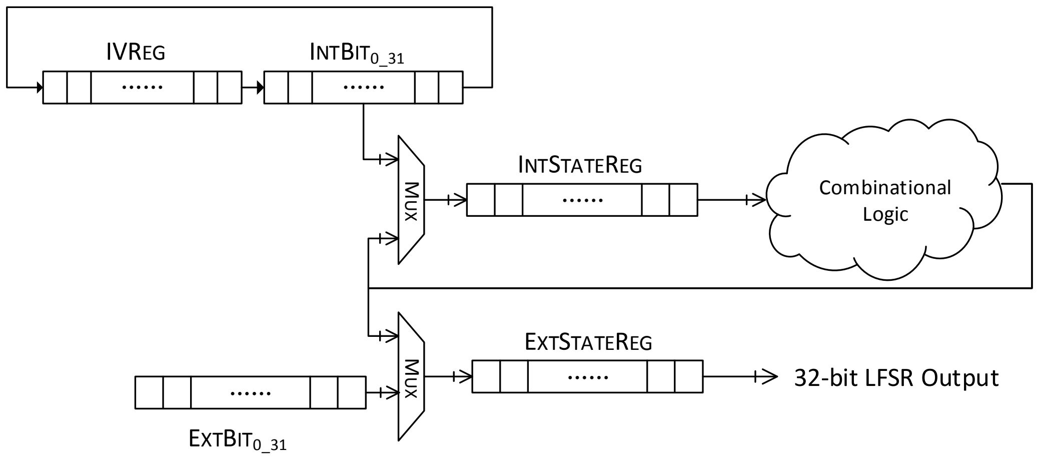


Figure 3.8: The structure of the cyclic rotating LFSR that is used to generate the masks on-the-fly.

To sum up, ExtStateReg always contains the valid 32-bit mask while IntStateReg always contains 32-bit input that results in the next valid state. The rotated secret key is generated in 150 clock cycles. After 4801×150 clock cycles, the LFSR returns to its initial state and idles.

3.3.3.3 Masking the Decoder

As mentioned previously, the masked secret keys and the syndrome are extended to three shares. Hence, more LFSRs are instantiated to generate the additional shares as shown in Figure 3.7. Two LFSRs generate the third shares of h_0 and h_1, another LFSR generates the third share of the syndrome.

We use h_0 as example to describe the decoder, since h_1 is processed in parallel using identical logic circuits. We split h_0 into three shares: h_{0m} stored in the BRAM and m_{1,h_0} and m_{2,h_0} generated by two LFSRs. The syndrome is split into s_m and m_{s_1} which are stored in BRAM and m_{s_2} which is generated by an LFSR. After decoding is started, each 32-bit share is read or generated at each clock cycle and

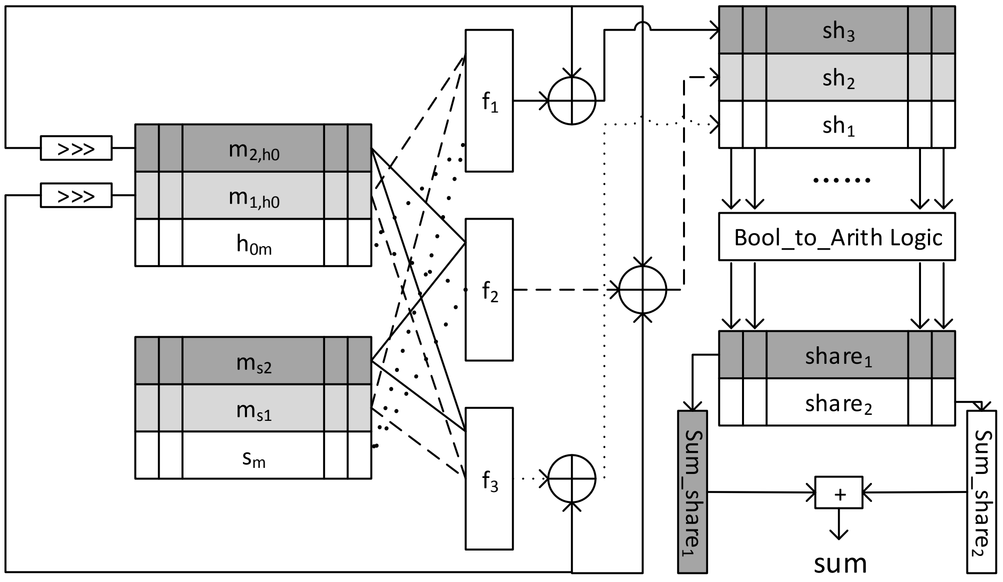


Figure 3.9: Layout of our pipelined QC-MDPC McEliece decoder for the first part of the secret key, h_0.

then SecAND and SecHW are performed. This is implemented using a pipelined approach as shown in Figure 3.9.

The left part of Figure 3.9 illustrates the bitwise SecAND operation using Equation (3.8). The 32-bit shares are fed into shared functions f_1, f_2, f_3, and the outputs are three 32-bit shares of the result. As mentioned before, two additional random vectors r_1, r_2 are required to mask the outputs in order to achieve uniformity. Our design uses only two fresh random bits b_1, b_2 together with the shifted input shares as the random vectors because the neighboring bits are independent of each other. That is $r_1 = \{b_1, m_{1,h_0}[0 : 30]\}$ and $r_2 = \{b_2, m_{2,h_0}[0 : 30]\}$. Both $m_{1,h_0}[31]$ and $m_{2,h_0}[31]$ are shifted out and are used as b_1 and b_2 in the next clock cycle. The right part shows the structure of SecHW. To compute the Hamming weight of the unmasked result $sh_1 \oplus sh_2 \oplus sh_3$ without leaking side channel information, a parallel counting algorithm is applied to accumulate the weight of each bit position of the word. We use 32×2 6-bit Arithmetic masked counters[2] and each bit in the word $sh_1 \oplus sh_2 \oplus sh_3$ will be added into the corresponding counter during each clock cycle. More specifically, the three shares of each bit of sh are converted and added into the two Arithmetic masked counters. After 150 clock cycles, we sum the overall Arithmetic masked Hamming weight. To convert and accumulate the masked weights, we employ the secure conversion method developed in [CGV14].

3.3.4 Implementation Results

The masked design is implemented in VHDL and is synthesized for Xilinx Virtex-5 XC5VLX50 FPGA which holds the crypto engine in the side channel evaluation board SASEBO-GII. The implementation results are listed in Table 3.1 in comparison with the unprotected implementation of [vMG14a]. In terms of Flip-Flops (FFs) and Look-Up Tables (LUTs), the masked implementation uses 8 times as many resources as the unprotected implementation. The increase is mainly due to the masked Hamming weight computation which requires many registers to store the Hamming weights of small chunks. Moreover, the leap forward LFSR also utilizes many Flip-Flops and has to be instantiated five times in our design. The number of occupied BRAMs remains constant, only the occupied memory within the syndrome BRAM increases by a factor of 2 in the masked implementation because the syndrome masks are also stored in this BRAM. The performance of the masked design is compromised for security and the maximum clock frequency is reduced by a factor of 4.3. This is mainly because the addition of 32 6-bit weight registers in SecHW is done in one clock cycle resulting a long critical path and in turn a low clock frequency.

[2]Note that the Hamming weight of $s \wedge H$ is bounded to the weight of h_i, i. e., $\mathrm{wt}(s \wedge h_i) \leq w/2 = 45$, i. e., 6-bit registers are always sufficient.

Table 3.1: Resources usage comparison between the unprotected and masked implementations on Xilinx Virtex-5 XC5VLX50 FPGAs.

Implementation	**FFs**	**LUTs**	**Slices**	**BRAMs**	**Frequency**
Unprotected [vMG14a]	412	568	148	3	318 MHz
Masked	3045	4672	1549	3	73 MHz
Overhead Factor	7.4x	8.2x	10.5x	1x	4.3x

Shortening the critical path can be an interesting goal in future work. Note that the number of clock cycles remains the same as for the unprotected implementation, unless the early termination of the decoder is disabled, in which case the average run time doubles compared to [vMG14a] (assuming that the maximum number of iterations is set to 5 similarly to [vMG14b], with early termination enabled the decoder requires 2.4 iterations on average as was shown in [HvMG13, vMOG15]). The resulting mean overhead of our implementation is 4, which is in line with other masked implementations. The TI AES engine in [MPL+11] introduces an area overhead of a factor 4 as well, but that implementation does not include the pseudorandom generators needed to generate the 48 bits of randomness consumed per cycle, while ours does.

3.3.5 Leakage Analysis

Now we analyze the implementation for remaining leakage. We first apply the DPA as explained in Section 3.2 on the masked implementation. Next we use the t-test leakage detection methodology to detect any other potentially exploitable leakages. The evaluated implementation is placed on the Xilinx Virtex-5 XC5VLX50 FPGA of the SASEBO-GII board. The power measurements are acquired using a Tektronix DSO 5104 oscilloscope. The board was clocked at 3 MHz and the sampling rate was set to 100 M samples per second. In order to quantify the resilience of our masked

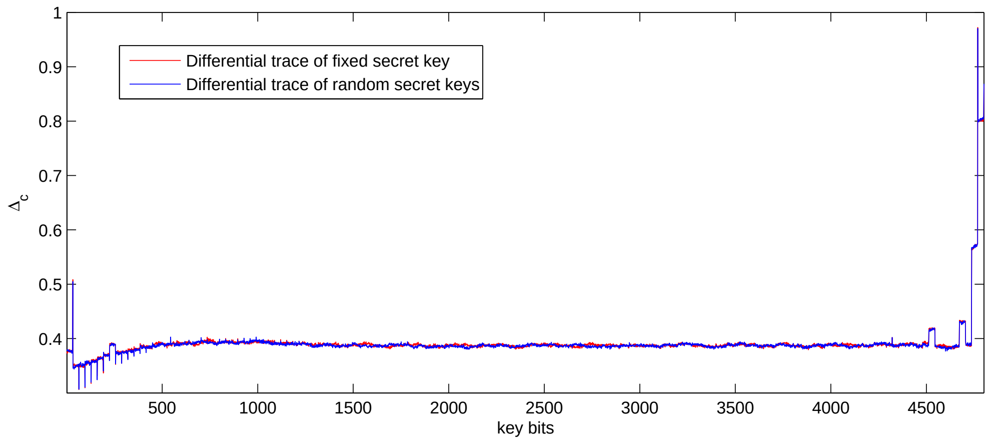


Figure 3.10: Comparison between two differential traces of two sets of secret keys.

implementation to power analysis attacks, we collected $10,000$ measurements using the same ciphertext but two different sets of secret keys. The first set is actually $5,000$ repetitions of a fixed key while the second set contains $5,000$ random keys. The two sets of keys are fed into the decryption engine alternatively.

3.3.5.1 Differential Power Analysis

As shown in Section 3.2, DPA on the target FPGA implementation requires construction of differential traces. Therefore, we generate such differential traces out of the power traces from masked implementation. In contrast to the unprotected implementation, no features are present in the differential trace of the fixed secret key (red line) even with 500 times more traces, as shown in Figure 3.10. Hence, the key bit value cannot be recovered. The peaks in the trace are not the features caused by set key bits because in the differential trace of the random secret keys where the key bits are randomly set as 1 the same peaks appear. Thus, they cannot be used as features to recover secret key bits as done in Section 3.2. The two differential traces almost overlap, showing that the leakage is indistinguishable between fixed key and random key when using a masked implementation.

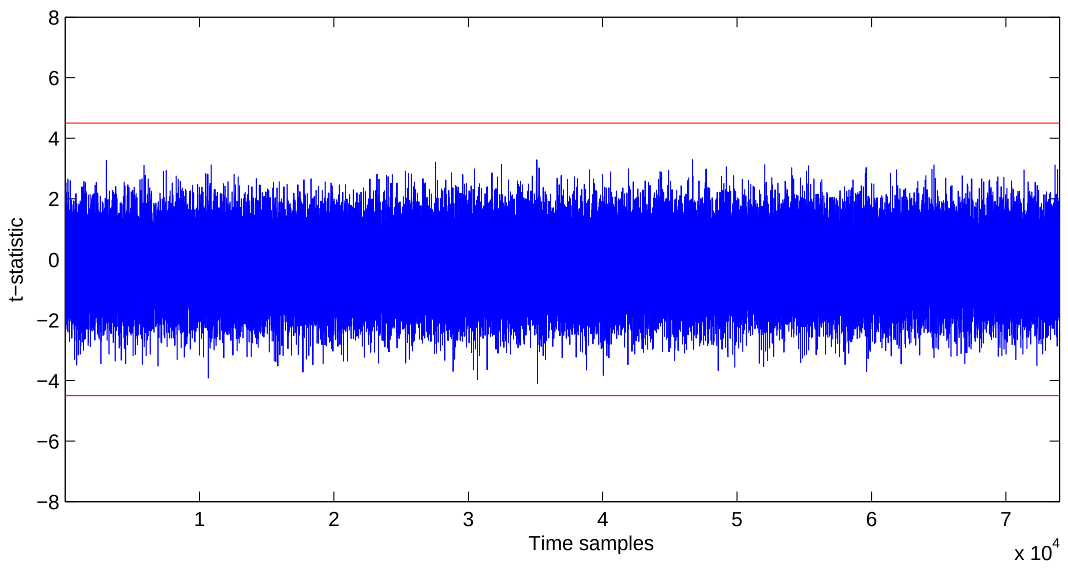
3.3.5.2 Leakage Detection

Next, we apply the t-test leakage detection methodology to find any remaining leakages. In our case, we obtained two groups of leakage samples, one for the fixed key set and the other for the random key set. Each group has $5,000$ power traces as well as $5,000$ derived differential traces. We first performed the t-test using the original power traces and Figure 3.11.1 shows the t-statistics along the whole decryption. The t-statistics are within the range of [-4.5, 4.5] which implies a confidence of more than 99.999% for the null hypothesis showing that the two sample groups are indistinguishable.

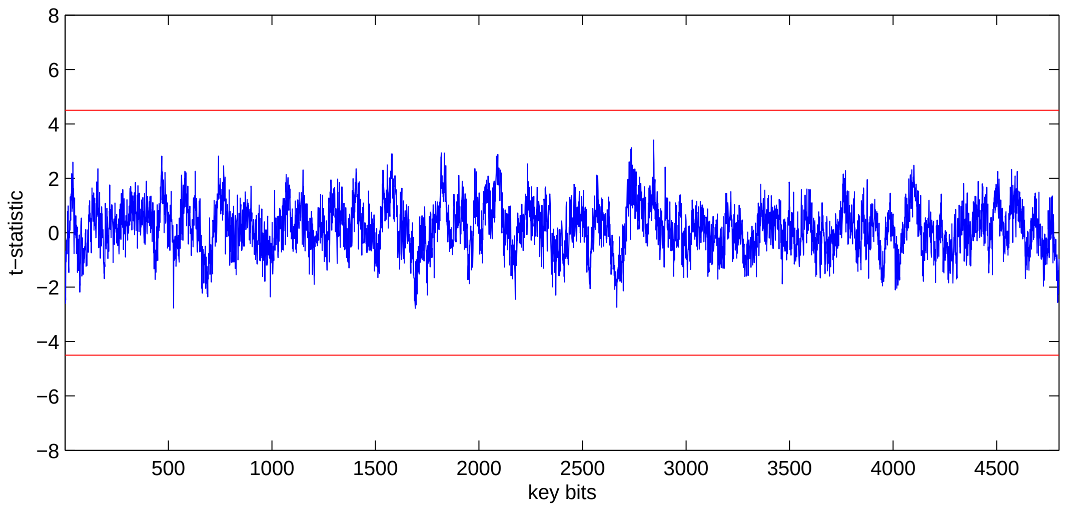
To assess the vulnerability to first-order horizontal attacks, we also performed a t-test on the derived differential traces. The results are shown in Figure 3.11.2. Similarly, the t-statistics are also within the predefined range and it validates the indistinguishability between the two sets of secret keys. Hence, it can be concluded that the design does not contain any remaining first-order leakage of the key.

3.4 Conclusion

This chapter presents the first successful differential power analysis of a state-of-the-art McEliece implementation based on quasi-cyclic MDPC codes. The analysis exploits the leakages of a key rotation operation which occurs during the syndrome computation step of the decryption and recovers a combined leakage of h_0 and h_1. The leakage model provides precise and strong leakage. The resulting attack is independent of the ciphertext and succeeds with tens of traces. A significant part of the key recovery stems from the relation between the private key and public key, which can be exploited to ease key recovery. In fact, recovering only half the bits of the (highly biased) secret key with a low error rate is sufficient for full key recovery.



3.11.1: Results of original traces



3.11.2: Results of differential traces

Figure 3.11: T-test between the two groups of *original* power traces (3.11.1) and *differential* power traces (3.11.2) corresponding to the two sets of secret keys. Both cases indicate the absence of leakage for the given number of traces.

Then, we investigate a formal way to counteract the attack by applying state-of-the-art Threshold Implementation masking scheme. While masking the syndrome computation is straightforward and comes at a low overhead, the decoding algorithm requires more involved masking techniques. Through on-the-fly mask generation, the area overhead is limited to a factor of approximately 4. While the maximum clock frequency of the engine decreases, the number of clock cycles for the syndrome computation and each decoder run is unaffected by the countermeasures. The effectiveness of the applied masking has been analyzed by leakage detection methods and by showing that previous attacks do not succeed anymore.

Chapter 4

Leakage Mitigation

Since the discovery of side channel analysis, numerous efforts have been dedicated to the study of countermeasures to protect cryptosystems against such attacks. In general, the countermeasures fall into two categories , i.e. *masking* and *hiding*. Masking works by randomizing the intermediate values processed by the crypto devices and hiding attempts to break the link between the power consumption and processed data.While they provide leakage resistance at different levels, the instantiations of both countermeasures often incur significant overhead in terms of area, performance and power consumption, hindering their deployment in resource-constrained applications like Internet of Things.

In this chapter, we present our contributions in the study of both countermeasures aiming for more efficient schemes. In terms of masking, we investigate state-of-the-art Threshold Implementation and explore the possibilities for practical TI with only two shares in order for a smaller design; whereas for hiding, we develop a scheme that has constant leakage in common linear leakage models. Constant leakage is achieved not only for internal state values, but also for their transitions.

4.1 Masking: Two-share TI

Protecting cryptographic hardware against side channel analysis is a difficult task and usually incurs significant area overheads. Especially masking schemes aimed at hardware have been found to be flawed or prone to implementation errors that leave the countermeasure at least partially insecure [CPR07, KP09, MM12].

Threshold Implementation (TI) has become a popular masking scheme for hardware implementations in the recent years, due to several advantages over competing schemes. Unlike secure logic styles [KP09], it does not require a change of the design flow. TI is fairly simple to apply to a wide range of ciphers, and its implementation is not very error-prone, if a known set of requirements and best practices is followed. Another advantage is that TI actually keeps the promise of reliable first-order side-channel resistance. It also provides good protection against higher-order attacks [MPL+11, BGN+14a].

However, like most other masking schemes, TI incurs large area and time overheads, and often consumes huge amounts of randomness for remasking, which can make practical application cumbersome. So far the best results have an area overhead of approximately three while consuming at least two times the combined plaintext and key size of randomness per encryption. Such overheads—the significant increase in area as well as the need for a high-performance random number generator—make TI an expensive choice, too expensive for a broad range of practical applications. Reparaz et al. [RBN+15] generalized TI to provide protection against higher-order attacks. The work mentioned the feasibility of reducing the number of shares to $d + 1$, where d is the desired protection order, suggesting that two shares are sufficient for first-order side channel protection. One proof of concept of using $d + 1$ shares for AES was proposed by De Cnudde *et al* in [DCRB+16] at the cost of $(d+1)^2$ fresh randomness. Later, so-called domain-oriented masking using $d+1$ shares was introduced by Gross *et al* in [GMK16] with less randomness.

However, our practical results of using $d+1$ shares in fact come earlier and show its advantages as well as its limitations. In this work we explore the practical implications of reducing the number of shares of threshold implementations to only two shares (2-TI). Such a reduction of shares enables implementations that only incur an area overhead of two and at the same time can also reduce the need of minimally required randomness by a factor of two, making the incurred cost more bearable and thus allowing side channel protection for a much wider range of applications.

Our study shows that two-share TI is first-order secure and also reduces the size of the sequential logic in hardware implementations. The 2-TI-conversion of nonlinear functions is more cumbersome and usually requires at least one additional pipeline stage, with negative impact on implementation size and/or performance. However, we also expose a strong second-order leakage in both of the designs and argue that this is inherent to two-share TI implementations. We show that these leakages exist both in the theoretical model and can also be quickly exposed by leakage detection tests. We validate the exploitability of the observed leakages by side channel key recovery attacks.

4.1.1 Threshold Implementation with Two Shares

While the constructive approach by Nikova et al. allows to implement any d-th order algebraic functions in a straightforward way, actual implementations requiring to share functions of degree greater than 2 have put significant effort into keeping the number of shares as close as possible to three, which is perceived as the minimum possible to implement nonlinear functions, until [RBN$^+$15]. In particular, [KNPW13] discussed the efficient implementation of 4-bit S-boxes with three shares. Similarly, the current TIs of AES utilize the algebraic structure of the AES S-box and four [MPL+11] or variable with up to five shares [BGN+14a] to implement the S-box on a small area.

A natural question is: *Why to stop at three shares?* If small area is desirable, using similar techniques as the ones used by the above papers could enable TIs with just two shares, further reducing the area footprint as well as the need for randomness. This approach was already discussed in [RBN+15]. The approach is straightforward for the linear operations of an implementation, and has already been widely used in several TIs for those parts [BGN+14a, CEvMS15b, BGN+15]. The simplest nonlinear operation is a simple two-input and: $c = ab$ which can be processed with two shares as

$$c_0 = a_0b_0 \qquad c_1 = a_1b_1 \qquad c_2 = a_0b_1 \qquad c_3 = a_1b_0 \tag{4.1}$$

This equation is in violation of the common interpretation of the non-completeness requirement, since c_2 and c_3 mix inputs from shares with different indices. However, non-completeness is not violated as long as a and b are statistically independent.

Equation (4.1) suggests a 4-share output, which is undesirable for a minimal implementation. To keep the number of shares low, the four shares c_i can be recombined in the next cycle, e.g. $c'_0 = c_0 + c_2$ and $c'_1 = c_1 + c_3$. However, since the recombination would violate non-completeness, it must happen after a register-stage in the next clock cycle. In other words, a pipelining stage becomes necessary, increasing the register count and the delay of the output. The share proliferation gets worse for higher-degree algebraic functions, as stated in [RBN+15]. However, hardware-minimal implementations break higher-order algebraic functions into degree-minimal building blocks anyway, making share proliferation a theoretical concern only.

To also ensure uniformity and thus gain an implementable basic nonlinear building block, we implement $z = ab + c$ in two pipeline stages as

$$z'_0 = a_0b_0 + c_0 \qquad z'_1 = a_1b_1 + c_1 \qquad z_0 = z'_0 + a_0b_1 \qquad z_1 = z'_1 + a_1b_0 \tag{4.2}$$

Note that z'_i and z_i are computed in separate cycles. Conveniently, the z'_i and z_i are uniform. Furthermore, this computation order only needs to store 2 intermediate states (unlike eq. (4.1)). However, this assumes that the inputs are available in two subsequent clock cycles, which is a valid assumption in many serialized implementations. Either way, the resulting pipelining of the nonlinear function increases area overhead of that function, and also introduces a latency according to the number of pipeline stages needed. Most of this latency can be hidden if the data path of the implementation is small enough.

4.1.2 Potential Pitfalls

The observed higher-order leakage can be explained by the significant dependence of the variance on the value of the share x. For a simple example we compare a 2-sharing S_2 and a 3-sharing S_3 of a bit x into $S_2(x) = \langle x_0, x_1 \rangle$ and $S_3(x) = \langle x_0, x_1, x_2 \rangle$ respectively. We further assume a Hamming weight ($wt(\cdot)$) leakage on the shares. Table 4.1 lists the possible states and the resulting means and variances for both sharings.

As proper TI sharings of x, the mean leakage $\mu(S_i)$ is independent of the value of x. However, the variance of S_2 depends on x, in particular $\mathrm{var}(S_2(x = 0)) = 2 \neq 0 = \mathrm{var}(S_2(x = 1))$. This is not true for the 3-sharing S_3, where the variances in both cases are identical as well. This is a strong indication why 2-sharings may have a strong second-order leakage. This was also observed for partial 2-share implemen-

Table 4.1: Comparison of leakage for a 2-sharing (S_2) and 3-sharing (S_3) of a bit x in a Hamming weight model. The 2-sharing (S_2) shows a leakage in the variance $\sigma(S_2)$.

x	$S_2(x)$	$S_3(x)$	$wt(S_2)$	$wt(S_3)$	$\mu(S_2)$	$\mu(S_3)$	$\sigma(S_2)$	$\sigma(S_3)$
0	$\{00, 11\}$	$\{000, 011, 101, 110\}$	$\{0, 2\}$	$\{0, 2, 2, 2\}$	1	$3/2$	**2**	1
1	$\{01, 10\}$	$\{001, 010, 100, 111\}$	$\{1, 1\}$	$\{1, 1, 1, 3\}$	1	$3/2$	**0**	1

tations in [BGN+15] and will be demonstrated for full 2-share implementations in the analysis of our reference implementations in Section 4.1.6.

4.1.3 Application to Simon

Threshold Implementations of Simon with three shares have been proposed in [STE15] to counteract first-order side channel attacks. Moreover, their bit-serialized implementation only consumes 87 slices on Spartan-3 xc3s50 FPGA which renders it the smallest threshold implementation of a block cipher. The authors also discussed how the requirement of *non-completeness* shuts the door on a two-share hardware implementation of Simon but not on software implementations.

In this section, we at first apply serialization technique in order to realize a two-share TI Simon on hardware. The leakage detection analysis and implementation results will be presented in Sections 4.1.5 and 4.1.6.

4.1.3.1 Simon with Two Shares

We follow the notation used in [STE15] to describe the cipher. The input plaintext is initially split into two shares as:

$$\begin{aligned} r[a]_0 =& m[p][1] \\ l[a]_0 =& m[p][2] \\ r[b]_0 =& m[p][1] + r_0 \\ l[b]_0 =& m[p][2] + l_0 \end{aligned} \tag{4.3}$$

Where r and l represents the two input words, a and b denote two shares of the variables and subscript i indicates the round of encryption. $m[p][1]$ and $m[p][2]$ are two fresh random values that mask the plaintext in the very beginning of the algorithm and no more random numbers are needed for the rest operations. Then, the round function is denoted as:

$$\begin{aligned} r[a]_{i+1} =& l[a]_i \\ l[a]_{i+1} =& r[a]_i + l[a]_i^2 + l[a]_i^1 * l[a]_i^8 + l[a]_i^1 * l[b]_i^8 + k[a]_i \\ r[b]_{i+1} =& l[b]_i \\ l[b]_{i+1} =& r[b]_i + l[b]_i^2 + l[b]_i^1 * l[b]_i^8 + l[b]_i^1 * l[a]_i^8 + k[b]_i \end{aligned} \tag{4.4}$$

Where the superscripts $1, 2, 8$ on $l[*]_i$ represent left circular shift by corresponding numbers of bits. (Notice that both addition and multiplication are in GF(2)). Obviously, the computations of $l[a]_{i+1}$ and $l[b]_{i+1}$, if directly mapped into combinational circuits, are not *non-complete* since the two shares $l[a]_i^8$ and $l[b]_i^8$ are present in the same circuit and glitches may still cause leakage. We can serialize the above equations by enforcing them being executed in two steps rather than one. That is, we first compute the intermediate values $l[a]_{i+1,int}$ and $l[b]_{i+1,int}$ using only half of the terms in the equations as follows:

$$\begin{aligned} l[a]_{i+1,int} =& r[a]_i + l[a]_i^2 + l[a]_i^1 * l[a]_i^8 \\ l[b]_{i+1,int} =& r[b]_i + l[b]_i^2 + l[b]_i^1 * l[b]_i^8 \end{aligned} \tag{4.5}$$

Then, the round outputs can be further calculated as:

$$\begin{aligned} l[a]_{i+1} =& l[a]_{i+1,int} + l[a]_i^1 * l[b]_i^8 + k[a]_i \\ l[b]_{i+1} =& l[b]_{i+1,int} + l[b]_i^1 * l[a]_i^8 + k[b]_i \end{aligned} \tag{4.6}$$

The serialization not only retains both *correctness* and *uniformity* but achieves *non-completeness* as well. In Equation (4.5), the inputs $r[a]_i$, $l[a]_i^2$, $r[b]_i$ and $l[b]_i^2$ are all uniform and therefore the output intermediates are also uniform. Each function is independent of one share of every input and hence is *non-complete*. Similarly, Equation (4.6) also satisfies the three requirements. Correctness can be easily proved by substituting $l[a]_{i+1,int}$ and $l[b]_{i+1,int}$ with Equation (4.5). The uniformity of inputs $k[a]_i$ and $k[b]_i$ makes the outputs uniform too. Moreover, each function is independent of one share of every input and thus the functions are *non-complete* as well. One may argue that $l[a]_i^1$ and $l[b]_i^8$ (or $l[b]_i^1$ and $l[a]_i^8$) are two shares of l_i with different rotations and may leak information of l_i. However, the multiplication between them is in GF(2) and is equivalent with bitwise AND operation. Further, in order to ensure the *non-completeness*, "Keep Hierarchy" property of synthesize tool (ISE with XST) is enabled to separate the LUTs for AND.

4.1.3.2 Round-based Implementation

Figure 4.1 depicts the structure of a FPGA implementation which contains two copies of the same data-path which consists of two registers L_j and R_j and the

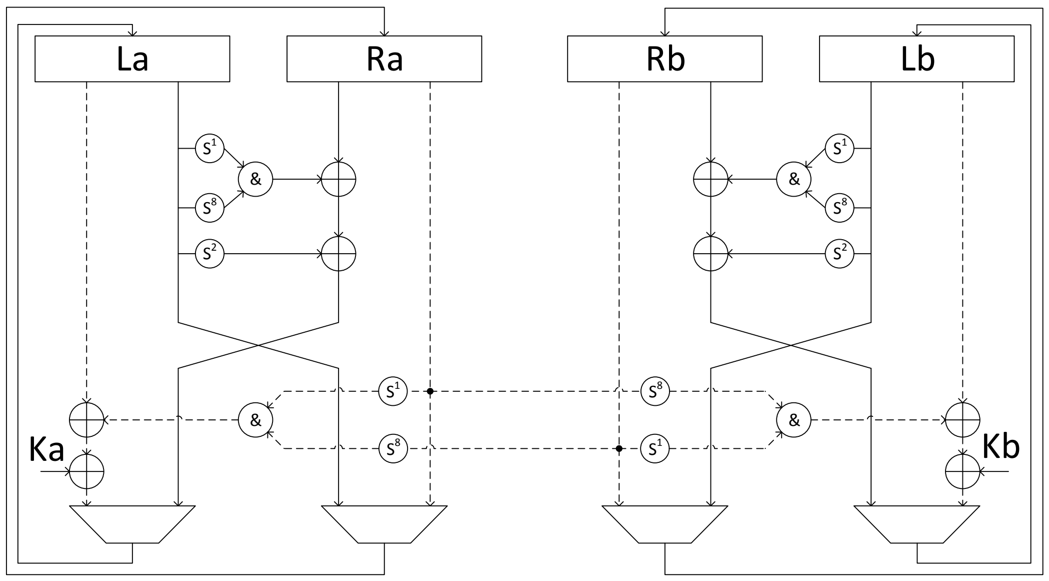


Figure 4.1: Data-path of the Simon with Two Shares. Solid line: First clock cycle; Dashed line: Second clock cycle

combinational circuits for round functions. Specifically, two clock cycles are taken to process each round operation. In the first clock cycle, the round inputs are evaluated with Equation (4.5) and then the intermediates are overwritten back into the registers as illustrated by the solid lines in the figure. Note that $r[j]_{i+1} = l[j]_i$ is stored in R_j while $l[a]_{i+1,int}$ is in L_j. Then, in the second clock cycle, Equation (4.6) is evaluated as shown by the dashed line but remember that since $l[j]_i$ is now stored in R_j and hence no extra buffer is needed for it.

The sharing of key schedule is not presented here since it consists of linear operations only and is trivial to implement.

4.1.3.3 Bit-serialized Implementation

In order to fairly compare with the bit-serialized 3-TI Simon introduced in [STE15] and achieve a even smaller size of Simon implementation, a bit-serialized 2-TI Simon is constructed as depicted in the Figure 4.2 (Only one share is shown).

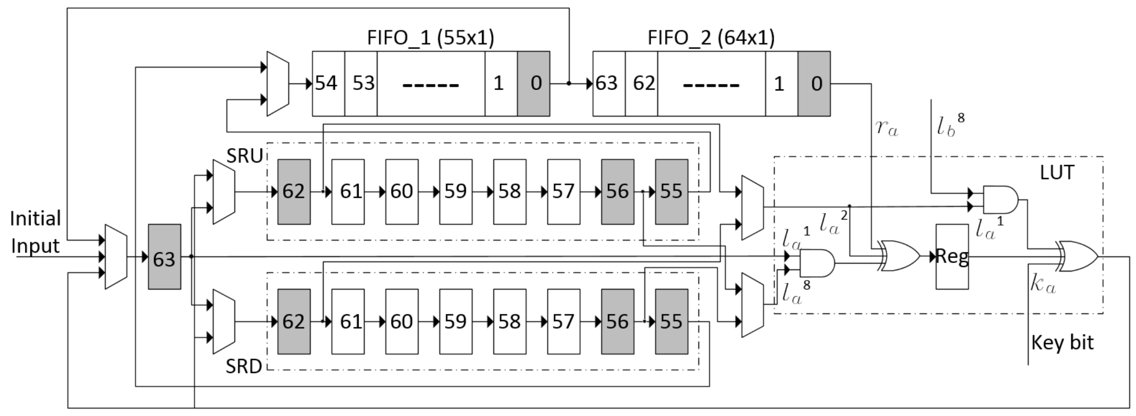


Figure 4.2: Data-path of the bit-serialized 2-TI Simon

Our design originates from the FIFO-based 3-TI bit-serialized in [STE15] but introduces new features in order for a 2-TI architecture.

First of all, the round function is adjusted according to Equation 4.5 and 4.6. (Note that both equations are evaluated in bits instead of the whole word in this case.) Therefore, as shown in the **LUT** part of Figure 4.2, a one-bit register is inserted to hold the intermediate value $l[a]_{i+1,int}$ so that $l[a]_i^8$ and $l[b]_i^8$ will not be combined to cause leakages mistakenly.

Second, due to the insertion of this register, it will take two clock cycles for **LUT** to perform round operation for each bit. However, by using pipeline technique, the overall throughput will not be scarified too much. In fact, the 2-TI architecture processes all 64 bits within 65 clock cycles which is only one more than 3-TI in [STE15]. In order to achieve this, the FIFOs and shifted registers are designed to work as following.

- Initially, the 128-bit block is stored in register #63, Shifted Registers Up (SRU) #62 to #55, FIFO_1 and FIFO_2.
- Once Encryption started, the values are right shifted and in the mean time bits in register #63, #62 and #56 as well as bit 0 in FIFO_2 are fed into **LUT**

for logic operation.

- The output will be written back to Shifted Registers Down (SRD). Note that the valid outputs are generated since the second clock cycle. And then, after 64 clock cycles, the first 63 output bits are stored in Shifted Registers Down (SRD) #62 to #55, FIFO_1 and FIFO_2. In the last (65th) clock cycle, the final output bit will be written in register #63. Therefore, the whole round operation is done within 65 clock cycles.

4.1.4 Application to Present

In this section, we apply two-share Threshold Implementation to the Present cipher. In [KNPW13], the authors presented the 3-TI Present S-box. To achieve this, they decomposed the non-linear S-box of degree 3 into the combination of two quadratic functions—G function—plus some linear functions, and then implement them with three shares. We follow their idea to use the same decomposition but then implement them with 2-TI while still retaining *uniformity*, *non-completeness*, and *correctness*. According to [KNPW13], the S-box of Present can be decomposed as:

$$S(X) = A(G(G(BX \oplus c)) \oplus d) \tag{4.7}$$

Where $G(.)$, A, B, and the constant vectors of c, d are given as follows:

$$\begin{aligned} G(x,y,z,w) =&(g_3,g_2,g_1,g_0): \\ g_3 =&x+yz+yw \\ g_2 =&w+xy \\ g_1 =&y \\ g_0 =&z+yw \end{aligned} \tag{4.8}$$

$$A=\begin{bmatrix}1&0&1&0\\0&1&0&0\\1&0&0&0\\1&0&1&1\end{bmatrix}, B=\begin{bmatrix}1&1&0&0\\0&1&1&0\\0&0&1&0\\0&1&0&1\end{bmatrix}, c=\begin{bmatrix}0&0&0&1\end{bmatrix}, d=\begin{bmatrix}0&1&0&1\end{bmatrix} \tag{4.9}$$

4.1.4.1 Present with Two Shares

A 2-sharing scheme of $G(.)$ can be expressed as follows:

$$\begin{aligned} G_0(x_0,y_0,z_0,w_0,x_1,y_1,z_1,w_1) =&(g_{03},g_{02},g_{01},g_{00}) \\ g_{03} =&x_0+y_0z_0+y_0z_1+y_0w_0+y_0w_1 \\ g_{02} =&w_0+x_0y_0+x_1y_0 \\ g_{01} =&y_0 \\ g_{00} =&z_0+y_0w_0+y_0w_1 \end{aligned} \tag{4.10}$$

$$\begin{aligned} G_1(x_0,y_0,z_0,w_0,x_1,y_1,z_1,w_1) =& (g_{13},g_{12},g_{11},g_{10}) \\ g_{13} =& x_1 + y_1z_0 + y_1z_1 + y_1w_0 + y_1w_1 \\ g_{12} =& w_1 + x_0y_1 + x_1y_1 \\ g_{11} =& y_1 \\ g_{10} =& z_1 + y_1w_0 + y_1w_1 \end{aligned} \tag{4.11}$$

The above sharing satisfies both *correctness* and *uniformity* when the input shares are uniformly distributed. However, *non-completeness* is not fulfilled since two shares of the same inputs are fed into the same functions in some of the above equations.

As before, we serialize the computations into two steps in order to achieve *non-completeness* as illustrated in the following equations.

$$\begin{aligned} G_0^1(x_0,y_0,z_0,w_0) =& (g_{03}^1,g_{02}^1,g_{01}^1,g_{00}^1) \\ g_{03}^1 =& x_0 + y_0z_0 + y_0w_0 \\ g_{02}^1 =& w_0 + x_0y_0 \\ g_{01}^1 =& y_0 \\ g_{00}^1 =& z_0 + y_0w_0 \end{aligned} \tag{4.12}$$

$$\begin{aligned}
G_0^2(x_1, y_0, z_1, w_1, g_{03}^1, g_{02}^1, g_{01}^1, g_{00}^1) =&(g_{03}^2, g_{02}^2, g_{01}^2, g_{00}^2) \\
g_{03}^2 =&g_{03}^1 + y_0 z_1 + y_0 w_1 \\
g_{02}^2 =&g_{02}^1 + x_1 y_0 \\
g_{01}^2 =&g_{01}^1 \\
g_{00}^2 =&g_{00}^1 + y_0 w_1
\end{aligned} \tag{4.13}$$

$$\begin{aligned}
G_1^1(x_1, y_1, z_1, w_1) =&(g_{13}^1, g_{12}^1, g_{11}^1, g_{10}^1) \\
g_{13}^1 =&x_1 + y_1 z_1 + y_1 w_1 \\
g_{12}^1 =&w_1 + x_1 y_1 \\
g_{11}^1 =&y_1 \\
g_{10}^1 =&z_1 + y_1 w_1
\end{aligned} \tag{4.14}$$

$$\begin{aligned}
G_1^2(x_0, y_1, z_0, w_0, g_{13}^1, g_{12}^1, g_{11}^1, g_{10}^1) =&(g_{13}^2, g_{12}^2, g_{11}^2, g_{10}^2) \\
g_{13}^2 =&g_{13}^1 + y_1 z_0 + y_1 w_0 \\
g_{12}^2 =&g_{12}^1 + x_0 y_1 \\
g_{11}^2 =&g_{11}^1 \\
g_{10}^2 =&g_{10}^1 + y_1 w_0
\end{aligned} \tag{4.15}$$

The superscript indicates the level of the circuit. Until now, we achieved a *correct*, *non-complete* and *uniform* two-share implementation of $G(.)$. the conversion of the remaining linear operations is discussed next.

4.1.4.2 Hardware Implementation

As depicted in Figure 4.3, in order to provide the *non-completeness* to the design, we use registers to separate the two parts of the G. The second part of the shares (G_0^2 and G_1^2) use not only the outputs of the first part of the shares (G_0^1 and G_1^1) but also some of their inputs as well (depicted in Figure 4.3). One 6-bit register and two 4-bit registers are used before the second part of the G module, to store the inputs x_0, x_1, z_0, z_1, w_0, and w_1; and the outputs of the first part of the G module, respectively.

In Figure 4.4, the S-box architecture is depicted which includes two G modules, and functions $BX + c_0$ and $AX + d_0$ for the first share as well as functions $BX + c_1$ and $AX + d_1$ for second share in which $c_0 + c_1 = c$ and $d_0 + d_1 = d$. Furthermore, due to *non-completeness*, we use another row of registers in between two $G(.)$ functions in the S-box. One may argue that registers should also be inserted between non-linear functions (e.g. $G(.)$) and linear functions (e.g. $AX + d_0$), since when they are merged the two shares of certain variables may be combined again which fails the *non-completeness* requirement. While this is true in general cases, our design avoids this problem as G_0^2 and G_1^2 are both independent of one share of the inputs and hence any linear combination of $g_{13}^2, g_{12}^2, g_{11}^2, g_{10}^2$ or $g_{03}^2, g_{02}^2, g_{01}^2, g_{00}^2$ still satisfies *non-completeness*.

Figure 4.5 shows the whole Present cipher with two shares. The design includes two control inputs namely `key_load` and `data_load`. If `key_load` is high, at the rising edge of the clock signal, the 80-bit input key shares-Key A and Key B- are copied to the registers Key A and Key B respectively. When the `data_load` signal is high, at the rising edge of the clock signal, 64 right-most significant bits of the input shares (`data_in A[63:0]`, `data_in B[63:0]`) are copied to state registers. It is worth mentioning that when the `data_load` is set, i.e. loading new two shares of

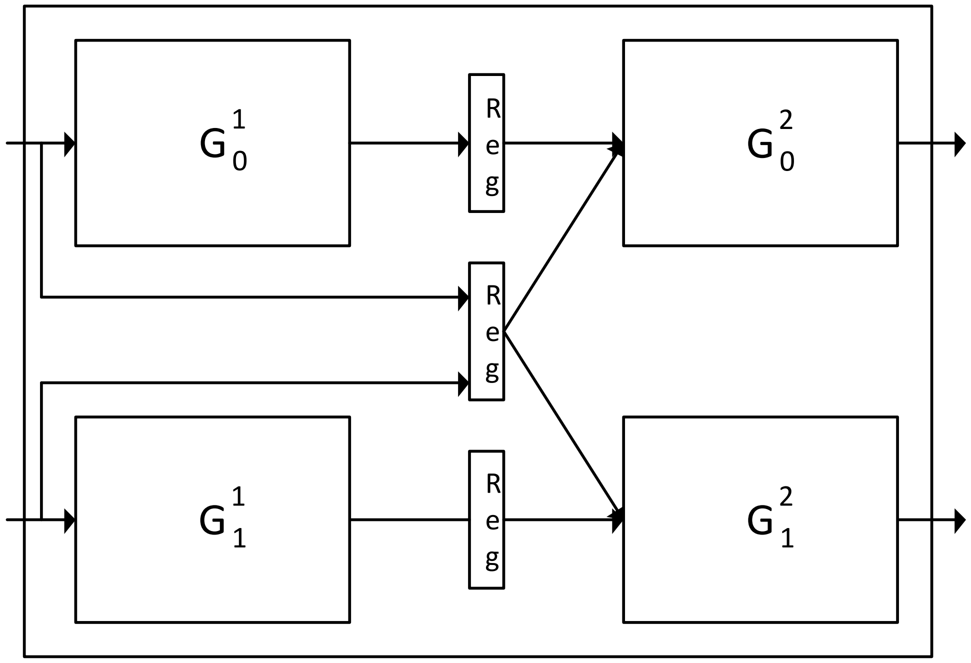


Figure 4.3: Hardware architecture of the 2-share G module

plaintext into the state registers results in a reset of the state machine. That why this design does not have a reset signal. When the two-share keys and two-share plaintexts are loaded, both `key_load` and `data_load` must be set to zero. After that, it takes 31 rounds in order to `Data_out A` and `Data_out B` have a valid ciphertexts. In each round, the S-box and permutation operations respectively operate the inputs to update the state registers for the next round. Considering the hardware design, each $G(.)$ function needs one cycle and then every S-box needs four clock cycles to compute table lookup. According to the Figure 4.5, each 64-bit input stored in the State register needs to use S-box 16 times. Hence, it needs 4 clock cycles for the first S-box due to its latency, plus 15 clock cycles for other 15 S-boxes in pipeline, also one more clock cycle for the permutation operation. Therefore, we need 20

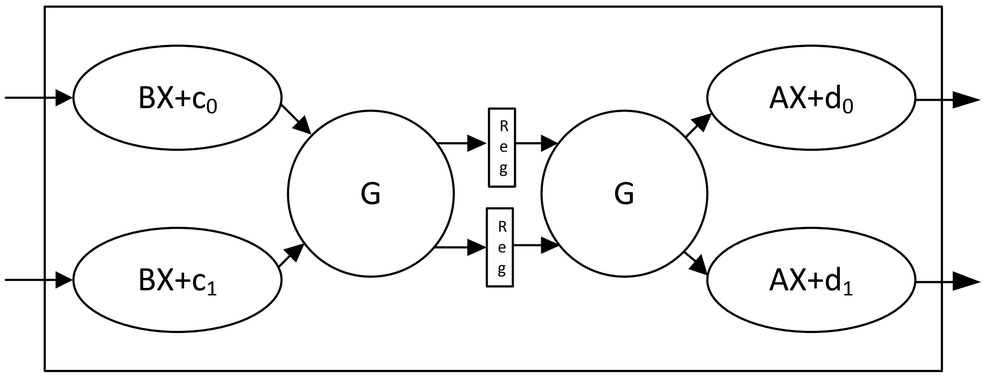


Figure 4.4: Hardware architecture of the 2-share S-box module

cycles for each round of the Present cipher. Hence, we define another control signal, 'counter', in which it updates the state registers and Key registers after each 20 cycles. After each cycle of these 20 cycles, the state registers are shifted to the right by 4 bits and the four most significant bits of the state registers are replaced by the outputs of substitution and permutation network. The Present cipher has 31 rounds, hence a full encryption of a 64-bit input takes 620 clock cycles. We also design an unprotected Present cipher to show the area overhead of the protected Present versus unprotected one as well as its impact on maximum frequency and throughput. The comparison results are shown in Table 4.2.

4.1.5 Implementation Results

Table 4.2 summarizes the overhead and performance of two-share implementations of both ciphers. Note that we only implement Simon128/128 and Present64/80 as an example to show the advantage of two-share scheme. All the designs are implemented in Verilog and synthesized for Virtex-5 (xc5vlx50) or Spartan-3 (xc3s50) using XST.

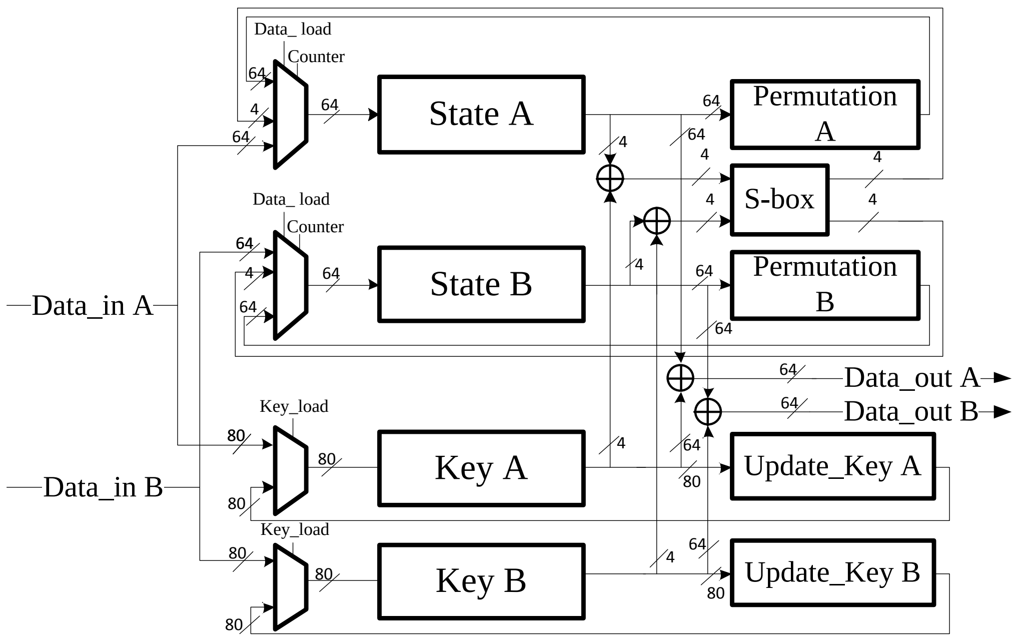


Figure 4.5: Hardware architectures of the 2-shares Present Cipher.

For round-based Simon, we have three different implementations: unprotected, 2-TI and 3-TI. In terms of slice registers used, two-share TI implementation costs twice as much as the unprotected one and one third less than the 3-TI implementation. This is not surprising since increasing by one share will consume one more copy of registers to store the new share. Similarly, number of LUTs also increases. However, each round operation in 2-TI costs double clock cycles and therefore the throughput is greatly reduced compared with the other two designs.

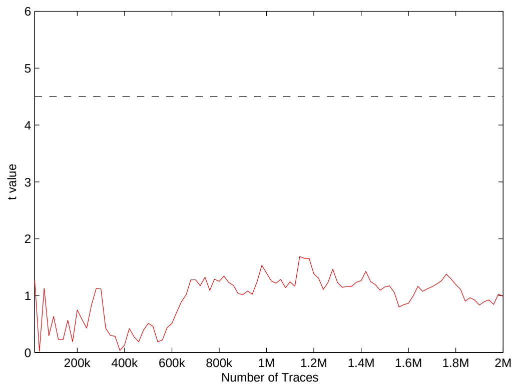
We also implement bit-serialized 2-TI Simon to compare with the currently smallest block cipher designs for FPGAs, as given in [AGS14], as well as its first-order protected 3-TI version from [STE15]. As shown in Table 4.2, our 2-TI design reduces the area overhead when compared to the 3-TI by about 13%, i.e., cannot quite reach the optimal reduction of 33% due to the pipelining overhead and the

Table 4.2: Implementation results of two-share Simon and Present.

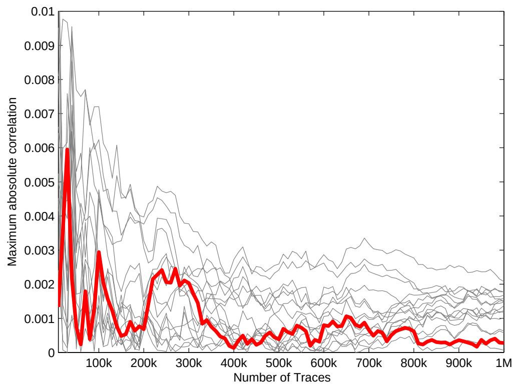
Design	**Slice** (Regs)	**Slice** (LUTs)	**Max. Frequency** (MHz)	**Throughput** (Mbps)
Present on Virtex 5				
3-TI Present	466 (3.0x)	715 (3.1x)	397.289	45.567
2-TI Present	370 (2.4x)	742 (3.2x)	490.252	50.61
Present	154 (1x)	234 (1x)	394.563	40.73
Round-based **Simon** on Virtex 5				
3-TI Simon	777 (2.8x)	1302 (2.8x)	414	779
2-TI Simon	520 (1.9x)	1169 (2.5x)	382	360
Simon	272 (1x)	473 (1x)	421	792
Bit-serialized **Simon** on Spartan 3				
3-TI Simon [STE15]	61 (2.0x)	160 (2.2x)	109.4	3.21
2-TI Simon	55 (1.8x)	135 (1.9x)	91.1	2.64
Simon [AGS14]	30 (1x)	72 (1x)	91.4	2.69

unaffected control logic. Nevertheless, this yields the smallest first-order protected block cipher design for FPGAs with the same parameters as AES-128.

With respect to Present, we have three implementations: Unprotected, Regular 3-TI, and the new 2-TI Present. In terms of slice registers used, regular 3-TI implementation used more than three times of the unprotected one. This is because we should use extra registers to guarantee the *non-completeness* of first-order resistant three-share Present cipher. Also, two-share implementation costs more than two times of unprotected Present because of the same reason mentioned before. Moreover, it is worth mentioning that the 2-TI first-order resistant implementation uses less registers than 3-TI. For example, we use extra registers in $G(.)$ function as explained in Section 4.1.4. These registers help reducing the critical path, which explains the speed-up and resulting increase in throughput for 2-TI Present.



4.6.1: 1st-order t-test



4.6.2: 1st-order CPA

Figure 4.6: First-order leakage analysis of synthetic data. Left: first-order t-test. Right: first-order CPA; Red line corresponds to the correct key guess

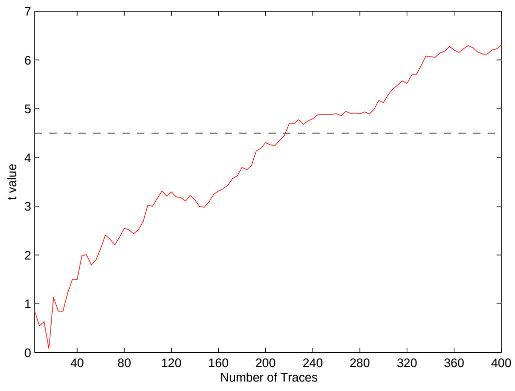
4.1.6 Leakage Analysis

In this section, we extend the discussion of a strong second-order leakage of two-share TI scheme, which was already described in Section 4.1.2, using simulation based leakage and the measurements from our reference implementations.

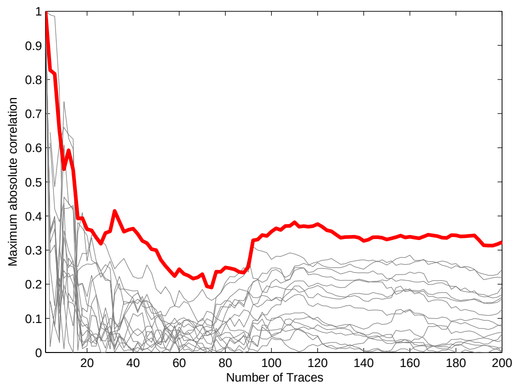
4.1.6.1 Theoretical Analysis

First we discuss the strong second-order leakage of two-share TI scheme using two-share Present S-box look-up as a target, namely the key-dependent intermediate value $y = S(x \oplus k)$ where x, y, k are 4-bit input plaintext, S-box output and sub-key receptively.

Synthetic samples and leakage model First, we generate noise free synthetic leakage samples of the 2-TI Present S-box based on Hamming weight model. As shown in Section 4.1.4, a 2-TI S-box processes two shares (4 bits for each share) in parallel and hence we use the Hamming weight of both output shares (8 bits in total) as the synthetic leakage samples. Further, in order for a second order analysis, the synthetic data should be center-and-then-squared. With respect to the leakage



4.7.1: 2nd-order t-test



4.7.2: 2nd-order CPA

Figure 4.7: Second-order leakage analysis of synthetic data. Left: second-order t-test. Right: second-order CPA; Red line corresponds to the correct key guess

model, we use the Hamming weight of the regular S-box output which equals the bitwise XOR between the two output shares in the 2-TI S-box.

First-order analysis We perform first-order *non-specific t-test* on the synthetic data and attempt to exploit any leakage using classic CPA as well. For this purpose, 1 million synthetic leakage samples for random input plaintext are generated as well as another 1 million for fixed inputs. The result of t-test using the 2 million samples is shown in Figure 4.6.1 where the t value is less than 2 as the number of traces (synthetic samples) increases to 2 million. Then, a classic first-order CPA is performed on the 1 million samples associated with the random inputs using the above-mentioned leakage model. The results in Figure 4.6.2 shows the correct key cannot be distinguished from the wrong key hypotheses with as much as 1 million samples and the attacks fail.

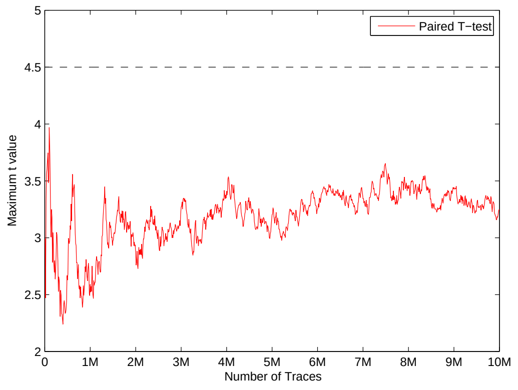
Second-order analysis Then, we proceed with second-order *non-specific t-test* and CPA. For this purpose, 200 synthetic leakage samples for random input plaintext are generated as well as another 200 for fixed inputs.Figure 4.7.1 shows that t value exceed 4.5 with only a couple of hundreds of samples while classic CPA can recover the correct key with less than a hundred samples as shown in Figure 4.7.2.

In summary, the theoretical analyses also show the first-order resistance of 2-TI scheme but reveals a strong second-order leakage. This strong second-order leakage is caused by the differing variances, as pointed out in Section 4.1.2. Note that we use perfect Hamming weight model for synthetic data without adding any noise. Hence, the CPA with a Hamming weight model can efficiently recover the key because it captures the leakage well. In fact, CPA on a perfect Hamming weight leakage is comparable to a profiled attack, in the absence of noise. But in the real world, actual leakages are more complex and CPA with Hamming weight model will not be as efficient as in this synthetic scenario. In the following we will conduct analysis on practical implementations to show this.

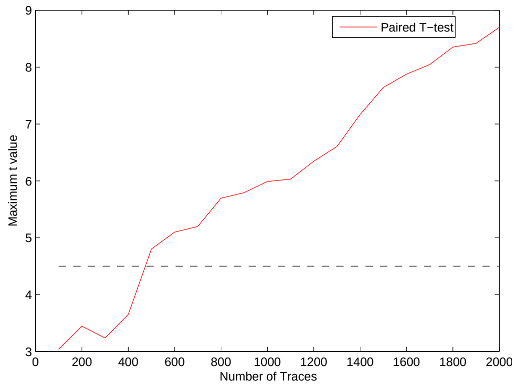
4.1.6.2 Practical Analysis

Next, we discuss the leakage analysis results for the two-share implementations of round-based Simon and Present. First, we apply the *non-specific* t-test method to detect any data-dependent leakage. Leakage detection tests are performed on round-based 3-TI Simon in order to compare with 2-TI and show the first-order leakage resistance of two-share scheme. Then, classic CPA is performed in order to exploit the second-order leakage detected by t-test and the results comply with the simulations in Section 4.1.6.1.

The analyzed implementations are ported into a Virtex-5 xc5vlx50 FPGA on the SASEBO-GII board clocked at 3 MHz. Measurements are taken using a Tektronix DPO-5104 oscilloscope which collects measurements with sample rate of 100 MS/s. The oscilloscope features a *FastFrame* functionality that can capture encryptions in bulk and thus 10 million measurements for each implementation can be taken in several hours.

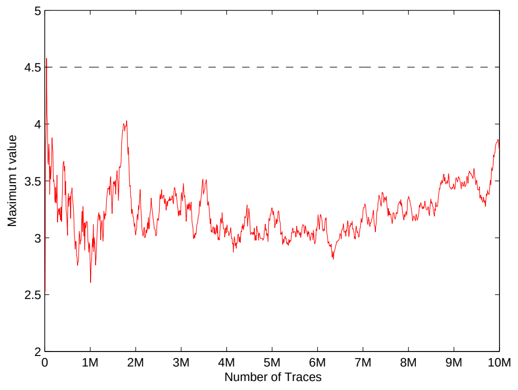


4.8.1: 1st-order t-test

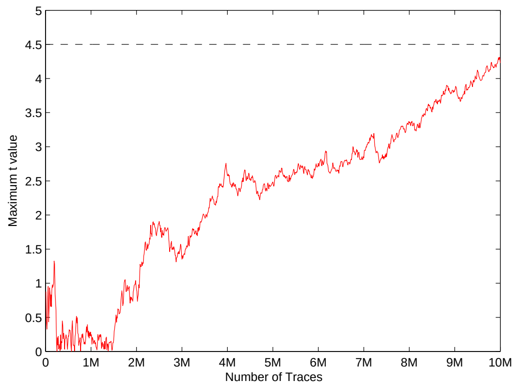


4.8.2: 2nd-order t-test

Figure 4.8: Leakage detection results for the two-share implementation of Simon for first-order (left) and second-order (right) leakage over the number of traces. Note that the dimensions change for both axes.



4.9.1: 1st-order t-test



4.9.2: 2nd-order t-test

Figure 4.9: Leakage detection results for the three-share implementation of Simon for first-order (left) and second-order (right) leakage over the number of traces.

Round-based 2-TI Simon For two-share Simon implementation, 10 million measurements are collected, yielding 5 million fixed-random pairs. Each measurement contains 5000 time samples, covering the 68 rounds of Simon. The first-order t-test is performed using $n = 5000, 10000, 15000, ...$ pairs. Figure 4.8.1 shows the first-order t-test result on the two-share Simon. The maximum absolute t value across the 5000 time samples remains below the threshold of 4.5 with 10 million traces. We conclude that the two-share Simon implementation is resistant against first-order

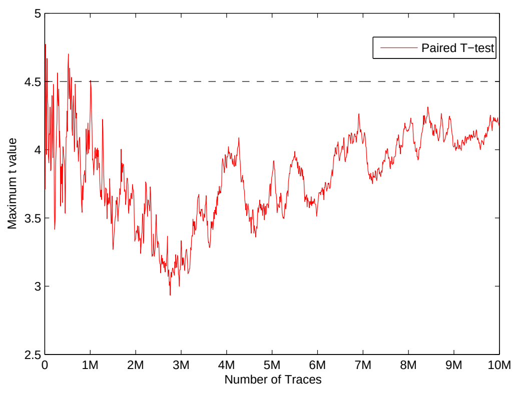
DPA and thus a validly implemented threshold implementation.

The results of the second-order t-test are shown in Figure 4.8.2. The step size is reduced to $n = 100, 200, ...$ to magnify the relevant area: The t value of the second-order analysis grows beyond 4.5 with about 500 traces. That is, a second-order leakage is detectable with just hundreds of traces.

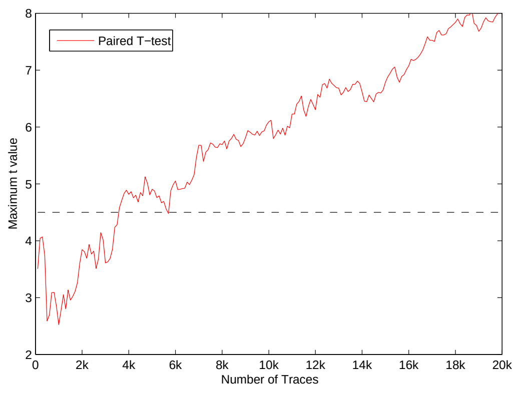
Round-based 3-TI Simon In order to practically compare the performance of 2-TI and 3-TI in resisting first-order and second-order leakage, the t-test is also applied to 10 million FRRF measurements from a round-based 3-TI Simon. Figure 4.9.1 shows similar result as in Figure 4.8.1 and the t value is below the threshold of 4.5. The comparison shows again that the first-order resistance of 2-TI is solid as a 3-TI. However, 3-TI exhibits resistance against second-order analysis as shown in Figure 4.9.2 and the t value is still below 4.5 with 10 million traces. That is, given more than 1000x as many measurements as for the 2-TI case, the leakage is just barely detectable. The results comply with the simulation analyses in Section 4.1.2 and Section 4.1.6.1 and validate the weakness of 2-TI.

2-TI Present As before, 10 million traces are captured for the two-share Present implementation, and then analyzed using t-test. The first-order t-statistic is still below 4.5 with 10 million measurements, as shown in Figure 4.10.1. The second-order t-statistics exceeds the threshold with about 6000 traces as shown in Figure 4.10.2. Again, the results suggest that two-share TI holds the promise of first-order resistance, but fares terribly on the second-order resistance.

Exploiting the Uncovered Leakages In order to practically exploit this strong second-order leakage, a classic CPA [BCO04, CJRR99, BGN$^+$14a] is performed on the measurements (center-and-then-squared) associated with the 5 million random



4.10.1: 1st-order t-test

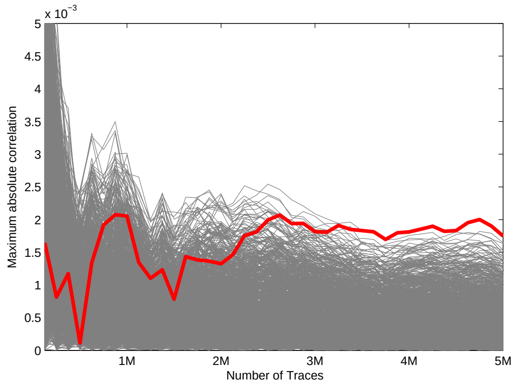


4.10.2: 2nd-order t-test

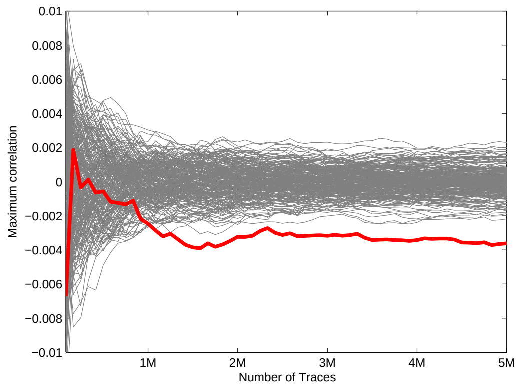
Figure 4.10: Leakage detection results for the two-share implementation of Present for first-order (left) and second-order (right) leakage over the number of traces. Note that the dimensions change for both axes.

plaintexts.

For *2-TI Simon*, the targeted operations occurred in the first clock cycle of the third round of encryption where shared values in registers L_a and L_b overwrite R_a and R_b respectively (see Figure 4.1). The leakage model used is Hamming distance between registers L and R as in a plain or unprotected implementation. The reason why third round is chosen is because of the weak non-linearity of single Simon round operation (only one AND) and attacking third round would relieve the effect of "ghost peaks" [BCO04]. Moreover, in order to reduce the computational complexity, we follow the divide-and-conquer approach and only attack the most significant four bits in L and R which are dependent on 10 bits in k_0 and 4 bits in k_1. Therefore, 2^{14} key hypotheses are required for the attack. To further reduce the complexity, we assume the knowledge of the relevant 4 bits in k_1 is known and only 10 bits in k_0 are aimed at to recover. Figure 4.11.1 shows the max correlation for each key hypothesis over the number of traces. The practical second-order attack successfully recovers the correct key with more than 3 million measurements even though ghost keys still exist. Note that these results can be significantly improved by using a



4.11.1: 2nd-order CPA of two-share Simon



4.11.2: 2nd-order CPA of two-share Present

Figure 4.11: Second-order CPA. Max correlation for each key hypothesis over the number of traces.

profiled attack, predicting more bits, and by using a pruning technique as e.g. done in [EKM+08], which is always an option for ciphers with a low algebraic depth per round. Nevertheless, the results validate the second-order leakage of two-share TI detected by the t-test and it can be practically exploited.

We also performed the same second-order CPA on 5 million random traces (center-and-then-squared) on *2-TI Present*, targeting at the S-box output to exploit the leakage. Recall our 2-TI Present in which the 64-bit state registers are right rotated by 4 bits per clock cycle so that the least significant nibble is continuously fed into the S-box look-up and output is written back to the most significant nibble after 4 clock cycles. Therefore, a Hamming distance leakage occurs between consecutive output nibbles. In this attack, we use the Hamming distance power model between the first two consecutive S-box outputs which depends on the least significant key byte and thus 2^8 key hypotheses are required. The max correlations per key hypothesis over number of traces are shown in Figure 4.11.2 and the results show that correct key can be successfully recovered with more than 1 million traces which demonstrates the practical exploitability of detected leakage.

The results from both validate our simulation analyses for the idealized case from Section 4.1.2 and Section 4.1.6.1, which suggests strong second-order leakage. The difference in sensitivity for the two implementations stems from their differing design strategies: 2-TI Simon is round based and does not use pipelining. Hence, it maximizes the leakage for the fixed-vs-random test: the entire state that is processed per cycle is constant in the fixed case and varies in the other case. For 2-TI Present, the implementation is serialized, with a 4-bit datapath, hence, a much smaller part of the implementation is updated per cycle, making the leakage less pronounced.

Moreover, unlike the theoretical analysis results in Section 4.1.6.1 where the number of traces needed for successful second-order t-test and CPA are of the same order magnitude, a lot more traces are needed for practical second-order CPA with Hamming distance model to exploit the leakage detected by t-test with only hundreds to thousands of traces. This is mainly because: 1) Practical implementation don't leak a perfect Hamming weight or Hamming distance leakage; 2) Noises also render the practical attacks inefficient.

While two-share TI shows potential in preventing first-order leakage with less overhead, its poor performance on second-order leakage resistance compared with three-sharing makes it less worthwhile.

4.2 Hiding: Balanced Encoding

With the advent of the Internet of Things, an ever-increasing number of embedded devices enters our lives and homes. These devices handle and exchange possibly sensitive information, raising the need for data security and privacy. High-end security solutions such as the processors found in passports and security smart cards come with an abundance of hardware protection to mitigate all kinds of physical and side channel attacks. However, most embedded devices are consumer-grade products that usually have to rely on unprotected off-the-shelf microprocessors. Only a limited number of methods are available to protect cryptographic software against side channel attacks on such a platform. Previous section shows that masking is an effective option. However, one of the biggest problems for getting a high level of protection of microprocessors is that masking is only effective if the processor has a low signal-to-noise ratio [CJRR99, PR13]. On modern embedded processors, this is usually not the case, requiring the combination of masking with other countermeasures that decrease the signal-to-noise ratio. Due to the fixed architecture of processors, real hiding countermeasures that achieve leakage reduction are hard to achieve. Proposed countermeasures for embedded software cryptosystems are mostly randomization countermeasures, i.e. leakage is not reduced, but rather randomized in time. Examples include shuffling [TH08, VCMKS12] or random delays [CK10].

This work explores a true hiding countermeasure in software. The idea is to ensure a constant leakage for all intermediate states. There is some limited prior work proposing constant Hamming weight (HW) encodings of intermediate states. In [HDD14], a secure assembly coding style based on the concept of Dual-Rail Precharge Logic (DPL) was proposed. The authors claim that a constant activity can be achieved using their specific data representation and programming rules. Their work is purely theoretic, no experimental results to support their idea were presented. Furthermore, the computation protocol did not completely prevent Hamming distance (HD) leakage. In [RGN13], the authors present the methods and tools to generate DPL style code automatically. In [HZL12], a similar data representation called Bitwise Balanced enCoding scheme was proposed. This scheme appears to be flawed: the XOR operation will leak information of one of the two inputs, as we explain later. They also just present simulation results that assume an idealized leakage. Hence, that work also lacks any analysis of real-world applicability.

In this work, we present new constant-leakage encodings. As prior works, we require intermediate states to be represented by constant Hamming weight encodings. We go beyond prior studies by showing that requiring constant Hamming distance transitions between states is also feasible. Unlike prior work, we actually implement the countermeasure, allowing us to realistically judge resulting implementation overheads. More importantly, we evaluate the achieved leakage reduction on a modern 8-bit microcontroller. We show how the constant leakage can be implemented not only for state representations, but also for state transitions, This allows us to apply the encodings to create a protected implementation of the Prince block cipher.

As most countermeasures, this countermeasure cannot provide perfect protection by itself. The leakage of real-world microprocessors deviates from linear and balanced models like the Hamming weight or Hamming distance model. However, forcing the side channel adversary to exploit the non-linear and imbalanced components of the leakage requires more sophisticated attacks and an increasing number of leakage observations. In other words, the countermeasure can effectively decrease the signal-to-noise ratio. The proposed countermeasure is orthogonal to masking or randomization countermeasures. Hence it can easily be combined with those to achieve an even higher overall resistance.

4.2.1 General Balanced Encoding Countermeasure

The non-balanced encoding of the algorithmic inputs and internal states usually causes side channel leakage during the execution of crypto primitives. The leakage can be exploited from classical side channel attacks such as DPA, CPA or MIA. The proposed countermeasure aims at encoding the internal states with longer bit length but resulting in constant Hamming weight of state and constant Hamming distance between two consecutive states. This trade-off sacrifices some memory and efficiency but achieves a balanced representation internal data and therefore mitigates the impact of side channel threats.

Formally, the balanced encoding requires the uniform distribution of Hamming weight for all codewords. Namely, every codeword should have the same Hamming weight, like the idea of constant-weight code or (m of n code). Clearly, the natural binary encoding is not such a candidate (e.g. $HW(0) \neq HW(1)$) since the resulting distribution of Hamming weight is binomial rather than uniform. The idea of balancing encoding can be realized only if using more than necessary bit length. A balanced encoding uses an embedding mapping τ from the natural binary encoded space $\mathcal{C} = \mathbb{F}_2^m$ for all $c \in \mathcal{C}$ into an extension $\mathbf{ext}(\mathcal{C}) = \mathbb{F}_2^n$ with $n > m$. In order to satisfy the constant Hamming weight of the new codeword, a necessary condition is that $C_n^{n/2} \geq 2^m$, where the image $\tau(\mathcal{C})$ sits entirely in the subset $S_{n/2} = \{u \in \mathbb{F}_2^n \mid HW(u) = n/2\}$.

Secondly, the newer encoding should preserve the basic bivariate operations $f(\cdot, \cdot)$ like xor and more complicated univariate operation $g(\cdot)$ such as the non-linear S-box mapping. More precisely, for any $v_1, v_2 \in \mathcal{C}$, it should hold that $\tau(f(v_1, v_2)) = \tilde{f}(\tau(v_1), \tau(v_2))$, where $\tilde{f}$ is the n-bit adjustment of the m-bit operation f. Similarly, for any $v \in \mathcal{C}$, it should hold that $\tau(g(v)) = \tilde{g}(\tau(v))$. Preserving such operations ensures the validity of the algorithmic evolution.

Thirdly, we also want such balanced encoding that achieves constant Hamming distance between any two consecutive states. This may not be easily realized with the choice of the codeword by requiring $HW(\tau(v) \oplus \tilde{g}(\tau(v)))$ being constant for any $v \in \mathcal{C}$. But it can be easily achieved with implementation tricks such as flushing registers before overwriting them with new values. That is, in order to mitigate the leakage generated from overwriting values, say for example, the state representation $\tau(v)$ which is stored in register $R1$ needs to be replaced by the univariate functional output $\tilde{g}(\tau(v))$, the procedure is first to store the output $\tilde{g}(\tau(v))$ at a different pre-cleared register $R2$, then clear register $R1$ and finally copy the register value from $R2$ back to $R1$ and free the temporary register $R2$. This approach sacrifices the efficiency of the code, but prevents Hamming distance leakage from overwriting the state. Another solution is to apply different balanced encodings to the two consecutive states to achieve not only constant Hamming weight but constant Hamming distance as well. More details of this solution will be given in the following section.

4.2.2 A Case Study Based on the Prince Cipher

In this section, we use Prince as an example to present the balanced encoding scheme. Prince is a nibble-based block cipher, as detailed in Section 2.4.2.3. Since our target platform is an 8-bit processor, a simple balanced encoding can be achieved by simply adding complementary bits, as done for dual-rail logic styles. That way, each state nibble is encoded as a 8-bit balanced encoding by inserting the complementary bits. For any nibble $b_3b_2b_1b_0$ where b_i is one bit data, the complementary nibble is $\bar{b_3}\bar{b_2}\bar{b_1}\bar{b_0}$, where $\bar{b_i}$ is the inverse of b_i. Concatenating these two nibbles forms a balanced encoding $\bar{b_3}\bar{b_2}\bar{b_1}\bar{b_0}b_3b_2b_1b_0$. An alternative is the encoding $\bar{b_3}b_3\bar{b_2}b_2\bar{b_1}b_1\bar{b_0}b_0$. Theoretically, under the Hamming weight leakage assumption, any sequence of those bits can be used as a balanced encoding because the Hamming weight is always 4. In the following we will use two different such encodings, i.e. $enc_I = \bar{b_3}b_3\bar{b_2}b_2\bar{b_1}b_1\bar{b_0}b_0$, which we refer to as *encoding I*, and $enc_{II} = b_0\bar{b_2}b_1b_3\bar{b_1}b_2\bar{b_0}\bar{b_3}$, which we refer to as *encoding II*. Both of the encodings ensure the constant Hamming weight of states. The *encoding II* is used to guarantee the constant Hamming distance between state transitions and the way this specific encoding is determined will be explained in the following section.

4.2.2.1 KeyAddition with Constant HW/HD

In the unprotected Prince implementation, the KeyAddition operation is denoted as $r_3r_2r_1r_0 = b_3b_2b_1b_0 \oplus k_3k_2k_1k_0$ where k is the subkey, b is a state nibble before the KeyAddition and r is the result of KeyAddition. For the protected Prince, we want an XOR-addition where secret inputs and outputs have a balanced encoding. However, for the initial key whitening at the input of the cipher, the plaintext input can be assumed not critical. Hence, only the output r and the key k are mapped to encoding I, i.e. $\bar{r}_3r_3\bar{r}_2r_2\bar{r}_1r_1\bar{r}_0r_0$ and $\bar{k}_3k_3\bar{k}_2k_2\bar{k}_1k_1\bar{k}_0k_0$. As in [HZL12], we can simply XOR-add k in encoding I to b encoded as $b_3b_3b_2b_2b_1b_1b_0b_0$ to realize the partially-protected XOR. This way, the Hamming weight of r is constant as well as the Hamming distance between r and b. The encoding for b does not satisfy the balanced encoding requirement, but has instead double Hamming weight leakage. Therefore, this only works for the initial KeyAddition where the plaintext is known.

After the first KeyAddition, the state becomes sensitive and need the balanced encoding. Hence, for the KeyAddition inside each round, b uses encoding I. Instead, we map k to the encoding $k_3k_3k_2k_2k_1k_1k_0k_0$, resulting in a remaining *constant* leakage for the round keys. Since the leakage is constant, it is not exploitable by CPA or DPA. Note that this leakage can also be avoided by using the XOR addition described in the following MixColumns section. It is more costly than the above described XOR variant, but all inputs and outputs have a balanced encoding and all transitions a constant Hamming distance.

4.2.2.2 Table Lookup with Constant HW/HD

The S-box operation can be described as $s_3s_2s_1s_0 = S(r_3r_2r_1r_0)$ where $S(\cdot)$ denotes the S-box, r denotes an input nibble, and s denotes the output. To protect it, a new lookup table based on the balanced encoding is designed in order to minimize the leakage. The S-box operation is denoted as $\bar{s_3}s_3\bar{s_2}s_2\bar{s_1}s_1\bar{s_0}s_0 = S'(\bar{r_3}r_3\bar{r_2}r_2\bar{r_1}r_1\bar{r_0}r_0)$ where the $S'(\cdot)$ represents the new S-box. Therefore the Hamming weight of S-box output bits is constant. Note that, unlike the regular S-box of size of 1×16, the new S-box is a 16×16 table where the only 16 positions contain the output value and all other positions are unused. The new S-box prevents the Hamming weight leakage but cannot prevent the Hamming distance leakage. One solution is to precharge the target register with zero before writing s into it. An alternative is applying encoding II to s, which is found by exhaustive search in all the possible encodings. For the Prince cipher, the S-box output in encoding II can be denoted as $s_0\bar{s_2}s_1s_3\bar{s_1}s_2\bar{s_0}\bar{s_3}$. In this way, the Hamming weight of S-box output is still constant as 4 and the Hamming distance between input in encoding I and output in encoding II becomes constant as $HD(enc_I(r), enc_{II}(s)) = 4$.

The cost of using two different encodings is an additional reordering layer which coverts encoding II back to encoding I. This is because the following operations such as MixColumns and ShiftRows are based on encoding I. A straightforward idea for reordering is the bit rotation which can be implemented using AND, LSL, LSR and OR instructions. AND instruction is used to pick out each single bit in encoding II by zeroing the other bits. Then we shift it to its position in encoding I. Finally, we combine all bits together to form encoding I. The disadvantage is that it is time consuming and it still causes side channel leakage. Instead, we can implement the reordering layer as a 16x16 lookup table R. The reordering table take the encoding II as input and output encoding $\bar{s_3}s_3\bar{s_2}s_2\bar{s_1}s_1s_0\bar{s_0} = R(enc_{II}(s))$. Note that, the output of R is a variant of encoding I by swapping the two LSBs. This is because $HD(enc_I(s), enc_{II}(s))$ is either 2 or 4 but $HD(\bar{s_3}s_3\bar{s_2}s_2\bar{s_1}s_1s_0\bar{s_0}, enc_{II}(s))$ is constant as 4. Then, the output of R is XORed with $0x03$ which swaps the two LSBs back to encoding I.

4.2.2.3 MixColumns with Constant HW/HD

The MixColumns operation can be implemented as XOR operations between the intermediate data. Unlike the XOR operation in KeyAddition, all the data involved in the MixColumns operation are sensitive and must hence be encoded in balanced encoding scheme to avoid the information leakage. Thus we need to design a new **constant XOR operation** instead of reusing the XOR from the KeyAddition. After the S-box substitution, the data in MixColumns operation are represented in encoding I. Denote the two operands of the constant XOR are as follows: x : $\bar{x_3}x_3\bar{x_2}x_2\bar{x_1}x_1\bar{x_0}x_0$ and y : $\bar{y_3}y_3\bar{y_2}y_2\bar{y_1}y_1\bar{y_0}y_0$. The XOR result is z : $\bar{z_3}z_3\bar{z_2}z_2\bar{z_1}z_1\bar{z_0}z_0$. The constant XOR can be implemented using the following steps:

Step 1: Divide the operand x into two parts and construct two new bytes as x_L : $\bar{x_3}x_3\bar{x_2}x_2\bar{x_3}x_3\bar{x_2}x_2$ and x_R : $\bar{x_1}x_1\bar{x_0}x_0\bar{x_1}x_1\bar{x_0}x_0$. In AVR microcontroller, this step can be easily done by AND, SWAP and OR instructions. For operand y, we construct y_L and y_R in the same way. The following code to the generate x_L can also be applied to the generation of x_R, y_L and y_R.

```
Input: r1 = x
Output: x_L
1: ldi r16, 0xF0
2: ldi r17, 0xF0
3: and r16, r1                    ▷ Cut off the right nibble of x
4: and r17, r1                    ▷ Cut off the right nibble of x
5: swap r17                       ▷ Swap the left nibble to the right
6: or r16, r17                    ▷ Generate x_L
```

Step 2: Do the regular XOR operation between x_L and `0xA5` to generate x'_L : $x_3x_3x_2x_2\bar{x_3}\bar{x_3}\bar{x_2}\bar{x_2} = x_L \oplus (10100101)_b$. Then $z_L = x'_L \oplus y_L = \bar{z_3}z_3\bar{z_2}z_2z_3\bar{z_3}z_2\bar{z_2}$. We also can generate z_R with the similar operations.

```
Input: r16 = $x_L$, r18 = $y_L$
Output: $z_L$
1: ldi r17, 0xA5
2: eor r16, r17                              ▷ Convert $x_L$ to $x_L$'
3: eor r16, r18                              ▷ Generate $z_L$
```

Step 3: Combine the most significant nibble of z_L and the least significant nibble of z_R to construct $z : \bar{z}_3 z_3 \bar{z}_2 z_2 \bar{z}_1 z_1 \bar{z}_0 z_0$.

```
Input: r1 = $z_L$, r2 = $z_R$
Output: $z$
1: ldi r16, 0xF0
2: ldi r17, 0x0F
3: and r16, r1          ▷ Cut off the least significant nibble of $z_L$
4: and r17, r2          ▷ Cut off the least significant nibble of $z_R$
5: or r16, r17                                   ▷ Generate $z$
```

Note that all above instructions operate on constant Hamming weight representations. Furthermore, there are no transitions that feature a non-constant Hamming distance in any operands. Hence, while costly, this XOR operation is free of Hamming weight or Hamming distance leakages in the operands.

4.2.3 Evaluation Results

To verify the balanced encoding scheme, we performed side channel evaluation on three implementations and compared the results between them.

2Prince The first implementation is the unprotected nibble-parallel Prince implementation from [SCE14], in which the 16-nibble states are stored in 8 registers. All round operations process two nibbles in parallel in order to achieve better performance. This implementation feature should result in slightly increased noise if the adversary only predicts a single nibble.

Balanced Prince The second implementation is the protected Prince using encoding I only. In this case, the precharge phase is added to the S-box lookup to achieve not only constant HW but constant HD as well.

Double-Balanced Prince The third one is also the protected Prince but using both encoding I and encoding II. This implementation differs from the second one in that the constant HD is obtained by using encoding II at the S-box output followed by a reordering layer.

We used an 8-bit AVR microcontroller to run the implementations. The performance and memory usage of the implementations are presented below. An automatic power measurement platform was established using a PC, a differential probe and an Tektronix DPO5000 series oscilloscope. A total of 100,000 power traces with random plaintext inputs were obtained for each implementation. Each implementation was analyzed using Hamming weight based CPA as a reference attack. Next, Mutual Information is used as a metric to quantify the leakage and compare the implementations. To make our numbers more reliable, we use 10-fold cross-validation on the computation of the mutual information.

4.2.3.1 Implementation Results

First we compare the performance of the three analyzed implementations. Table 4.3 compares the computation time per encrypted block and resource consumption in terms of code size and RAM usage. The code size increases significantly for the protected implementations, i.e. by a factor of 3. At the same time the performance decreases by a factor of 7. This is because each round operation costs more resources in order to obtain constant activity.

Table 4.4 shows the contribution of specific operations to the overall resource

Table 4.3: Performance comparison of three Prince implementations.

Implementation	**Encryption Time** in clock cycles	**Code Size** in Bytes	**RAM Usage** in Bytes
2Prince [SCE14]	3253	1574	24
Balanced	28214	3700	472
Double-Balanced	29498	4100	472

consumption. In particular, the code size and performance are broken down into the KeyAddition (KA), byte substitution (SB), and the mixing (M) operations of the Prince cipher. For example, the S-box of the protected implementations and the unprotected one are of the same size (256 byte, not included in the table code size calculation), but the unprotected one performs two S-box lookups in parallel. Similarly, either a precharge phase (for the Balanced implementation) or a reordering layer (for the Double-Balanced implementation) had to be added in order to gain constant Hamming distance transitions, also resulting in a significant increase in memory and clock cycles. Additionally, the conversion between normal data and balanced encoded data for the plaintext and ciphertext also adds overhead. The worst overhead is due to the M-Layer, or more precisely the constant leakage XOR, which uses 58 more clock cycles than regular XOR instruction.

4.2.4 CPA Results

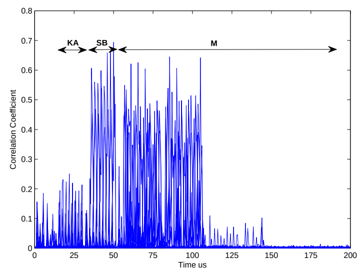
We first performed CPA on all of the three implementations. Each CPA predicts the Hamming weight of the output of a single S-box. To compare the leakage of the implementations—rather than distinguishing the correct key—we use the Hamming weight of the all 16 S-box outputs under a known key as the power model. The results are presented in Figure 4.12. The correlation between the measurements and power model is greatly reduced in the protected scenarios. For the unprotected

Table 4.4: Performance and cost comparison for the KeyAddition (KA), byte substitution (SB), and the mixing (M) layers for the three analyzed implementations.

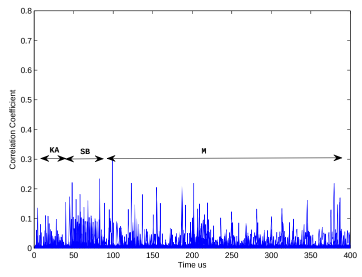
Implementation	**Operation**	**Performance** in clock cycles	**Code Size** in Bytes
2Prince [SCE14]	KA	72	80
	SB	41	36
	M	162	286
Balanced	K	57	68
	SB	90	62
	M	2156	1193
Double-Balanced	KA	57	68
	SB & RO	180	129
	M	2156	1193

implementation, the correlation coefficients range from 0.6 to 0.8 which is only about 0.1 to 0.3 in the protected implementations. Note that a few of the 16 nibbles feature a much stronger leakage than the others in the protected cases (cf. Fig. 4.12.2 and Fig. 4.12.3). This might be an implementation artifact and not due to the countermeasure itself. Similarly, the double-balanced implementation features its strongest leakage in the reordering layer. The results show that the balanced encoding scheme is effective in reducing the Hamming weight leakage. However, due to differences in the leakage of individual bits, the leakage does not completely disappear.

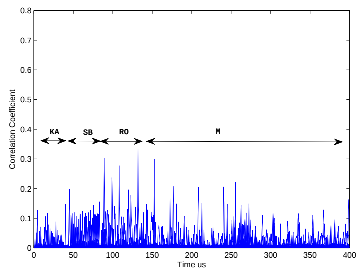
Figure 4.13 compares the trend of the correlation coefficients of the implementations (vertical axis) over the number of power traces (horizontal axis). We can observe that the correct subkey hypothesis can be easily distinguished from the wrong key guesses with as little as one hundred traces for the unprotected Prince in Figure 4.13.1. However, for both Balanced Prince and Double-Balanced Prince in



4.12.1: 2Prince

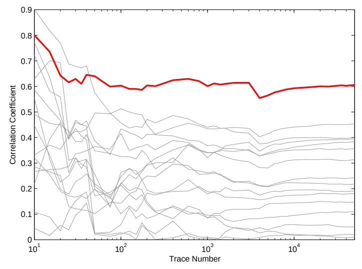


4.12.2: Balanced

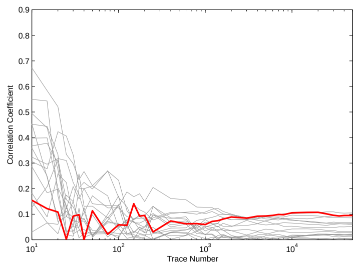


4.12.3: Double-Balanced

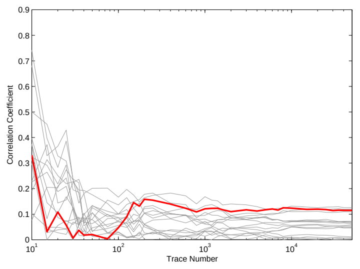
Figure 4.12: Result of CPA of three Prince implementations on the S-box output. The unprotected implementation (1) leaks significantly stronger than the two protected implementations (2) and (3). (KA: KeyAddition; SB: S-box Lookup; RO: reordering M: Mixing Layer)



4.13.1: 2Prince



4.13.2: Balanced



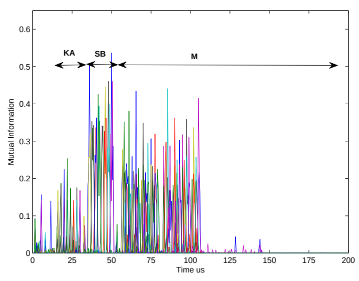
4.13.3: Double-Balanced

Figure 4.13: CPA results for the Hamming weight of the S-box output for the unprotected implementation (1) and the two protected implementations (2) and (3). The vertical axis indicates the absolute value of the correlation coefficient; The horizontal axis indicates the number of traces used. The comparison of the three plots shows the significant improvement resulting from the balanced encodings, if applied correctly. Plot (1) clearly shows the effect of the ghost peaks mentioned in Section 2.3.

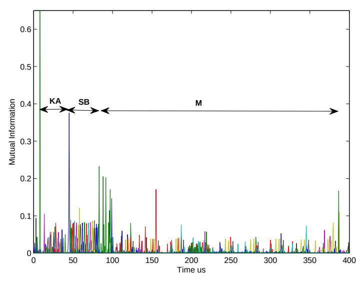
Figures 4.13.2 and 4.13.3, the correlation coefficient is significantly smaller and it is hard to distinguish the correct key hypothesis, even for as many as 50,000 observations. Note that this problem is not obvious in Figure 4.12, since that figure only contains correlations for the correct subkey hypotheses.

4.2.4.1 Mutual Information Based Leakage Analysis

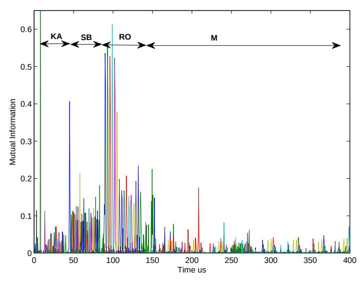
To compare the implementations in a leakage-model independent setting, we apply the mutual information based methodology introduced in Section 2.3 during the first



4.14.1: 2Prince

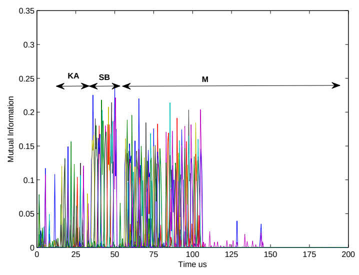


4.14.2: Balanced

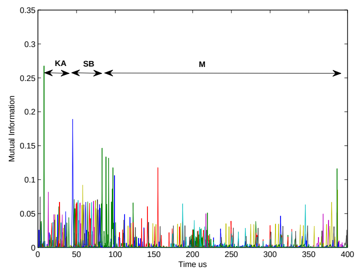


4.14.3: Double-Balanced

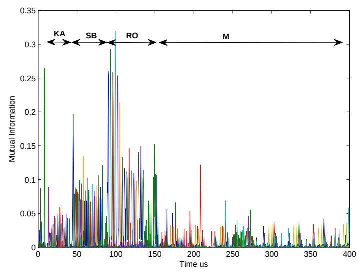
Figure 4.14: Mutual information between the state and the leakage for the unprotected (1), Balanced (2), and Double-Balanced (3) implementations during the first round.



4.15.1: 2Prince



4.15.2: Balanced



4.15.3: Double-Balanced

Figure 4.15: First order mutual information between the state and the leakage for the unprotected (1), Balanced (2), and Double-Balanced (3) implementations during the first round.

round of the Prince implementation. We apply it in two different ways: First, by using classical univariate templates with an individual mean and variance for each possible nibble state; Next, by using reduced univariate templates with an individual mean for each nibble state, but a common variance for all templates. The latter approach allows to only evaluate first-order leakages.

Figure 4.14 shows the mutual information for all 16 state nibbles for the first round, as derived from full univariate templates. Figure 4.15 shows the mutual information for all 16 state nibbles only for the first order leakage, as derived from the reduced univariate templates. Both plot families behave very similar, with the first-order MI being slightly lower in all cases. This indicated that the implementations on the AVR feature significant non-linear components in the leakage function.

The first-order MI is more appropriate to predict the resistance against first-order attacks such as DPA and CPA. More interestingly, the leakage drops significantly for the protected implementations. In fact, the mutual information goes down by as much as 50%. Especially the leakage of the S-box operation drops even more strongly, from .5 for the unprotected implementation to as low as .1 for the protected ones. That is, there is a single nibble that exhibits a huge leakage for the protected implementations. This is always the first nibble. To remove the leakage, we reordered the nibbles for the computation of the S-box. Surprisingly, whichever nibble is computed first, it exhibits this strong leakage. We claim this to be an implementation artifact. Similarly, there is a leakage right before the KeyAddition starts. Again, we do not have a good explanation for this leakage. However, unlike the S-box leakage, this one is not problematic, as information before the KeyAddition is plaintext, i.e. known to the attacker. As hinted at by the CPA results, both Figures 4.14.3 and 4.15.3 show that the Reordering layer still leaks a significant amount of information.

As a result, the Balanced implementation has a weaker leakage than the one of the Double-Balanced implementation. The stronger leakage for the second implementation occurs in the reordering layer. This was not expected, since it is implemented to have a constant Hamming weight and Hamming distance.

In summary, the balanced implementation is a better choice for devices that have a strong Hamming weight leakage and is a valuable new addition to the family of countermeasures in software. The Double-Balanced implementation is slightly less efficient, but suffers from the strong leakage of the reordering layer. A more careful implementation of the reordering layer could reduce the maximum leakage of the Double-Balanced implementation. One should be able to avoid the reordering layer completely by customizing operations in the MixColumns layer, but we did not further explore this route.

4.3 Conclusion

In this chapter, we investigate approaches to protect emerging cryptosystems, particularly for resource-constrained applications where lightweight cryptography will be deployed.

First, we present the first practical threshold implementations using only two shares. We showed that lightweight ciphers have several features making them good targets for threshold implementations. We explain how using two shares can actually yield smaller cipher implementations that need less randomness and still show perfect first-order resistance. While moving to two shares makes implementing the nonlinear functions of a cipher more cumbersome, resulting in either a loss in throughput, increase in circuit size, or even both, it allows to reduce the overhead of the sequential part of the implementation by only doubling the state and key size. Since the area of low-area crypto implementations usually depends mainly on the sequential part, significant improvements are possible. However, one need note that the results highlight that two-share implementations provide provable resistance against a “low” order of attack but also leak strong higher-order leakage.

Then, we propose a balanced encoding countermeasure for software and perform the first practical evaluation. While promising in theory, its standalone effectiveness on the modern microcontroller platform used for this study is significant, especially for CPA, but far from perfect. The countermeasure is of high relevance, as it is orthogonal to other software countermeasures such as shuffling and masking, i.e. it can be applied in addition to those. This is of high relevance for platforms that feature high signal-to-noise ratios, such as modern microcontrollers.

Chapter 5

Leakage Detection

The TVLA procedure using the t-test has become a popular leakage detection method. To protect against environmental fluctuation in laboratory measurements, we propose a paired t-test to improve the standard procedure. We take advantage of statistical matched-pairs design to remove the environmental noise effect in leakage detection. Higher-order leakage detection is further improved with a moving average method. We compare the proposed test with standard t-test on synthetic data and physical measurements. Our results show that the proposed tests are robust to environmental noise.

5.1 Motivation

More than 15 years after the proposal of DPA, standardized side channel leakage detection is still a topic of controversial discussion. While Common Criteria (CC) testing is an established process for highly security critical applications such as banking smart cards and passport ICs, the process is slow and costly. While appropriate for high-security applications, CC is too expensive and too slow to keep up with the innovation cycle of a myriad of new networked embedded products that are currently being deployed as the Internet of Things. As a result, an increasing part of the world we live in will be monitored and controlled by embedded computing platforms that, without the right requirements in place, will be vulnerable to even the most basic physical attacks such as straightforward DPA.

A one-size-fits-most leakage detection test that is usable by non-experts and can reliably distinguish reasonably-well protected cryptographic implementations from insecure ones could remedy this problem. Such a test would allow industry to self-test their solutions and hopefully result in a much broader deployment of appropriately protected embedded consumer devices. The TVLA test was proposed as such a leakage detection test in [CDG$^+$13, GJJR11]. The TVLA test checks if an application behaves differently under two differing inputs, e.g. one fixed input vs. one random input. As the original DPA, it uses averaging over a large set of observations to detect even most nimble differences in behavior, which can potentially be exploited by an attacker.

Due to its simplicity, it is applicable to a fairly wide range of cryptographic implementations. In fact, academics have started to adopt this test to provide evidence of existing leakages or their absence [BGG$^+$15, BGN$^+$14a, BGN$^+$14b, CEvMS15b, LMW14, MH15, NLD15, STE15]. With increased popularity, scrutiny of the TVLA test has also increased. Mather et al. [MOBW13] studied the statistical power and computation complexity of the t-test versus mutual information (MI) test, and found that t-test does better in the majority of cases. Schneider and Moradi [SM15] for example showed how the t-test higher-order moments can be computed in a single pass. They also discussed the tests sensitivity to the measurement setup and proposed a randomized measurement order. Durveaux and Standaert [DS16] evaluate the convenience of the TVLA test for detecting relevant points in a leakage trace. They also uncover the implications of good and bad choices of the fixed case for the fixed-vs-random version of the TVLA test and discuss the potential of a fixed-vs-fixed scenario.

However, there are other issues besides the choice of the fixed input and the measurement setup that can negatively impact the outcome for the t-test based leakage detection. Environmental effects can influence the t-test in a negative way, i.e., will decrease its sensitivity. In the worst case, this means that a leaky device may pass the test only because the environmental noise was strong enough. This is a problem for the proposed objective of the TVLA test, i.e. self-certification by non-professionals who are not required to have a broad background in side channel analysis.

In this work, we propose the adoption of the paired t-test for leakage detection, especially in cases where long measurement campaigns are performed to identify nimble leakages. We discuss several practical issues of the classic t-test used in leakage detection and show that many of them can be avoided when using the paired t-test. To reap the benefits of the locality of the individual differences of the paired t-test in the higher-order case, we further propose to replace the centered moments with a local approximation. These approximated central moments are computed over a small and local moving window, making the entire process a single-pass analysis. In summary, we show that

- the paired t-test is more robust to environmental noise such as temperature changes and drifts often observed in longer measurement campaigns, resulting in a faster and more reliable leakage detection.
- using moving averages instead of a central average results in much better performance for higher public key and multivariate leakage detection if common measurement noise between the two classes of traces is present, while intro-

ducing a vanishingly small inaccuracy if no such common noise appears. The improvement of the moving averages applies both to the paired and unpaired t-tests.

5.2 Methodology

This section introduces paired t-test and shows its superiority in a leakage model with environmental noise. The paired t-test retains its advantage of being a straightforward one-pass algorithm by making use of *moving* or local averages. By relying on the difference of matched pairs, the method is inherently numerically stable while retaining computational efficiency and parallelizability of the original t-test.

5.2.1 Paired T-Test

Welch's t-test works well when the measurement noises r_A and r_B are independent between the two sets of measurements. However, two sets of measurements can also share common variation sources during a measurement campaign. For example, power consumption and variance may change due to common environmental factors such as room temperature. While these environmental factors usually change slowly, such noise variation is more pronounced over a longer time period. With hard to detect leakages, often hundreds of thousands to millions of measurement traces are required for detection. These measurements usually take many hours and the environmental fluctuation is of concern in such situations. For example, for the DPAv2 contest, there are one million template traces collected over 3 days and 19 hours, which show a clear temporal pattern [HKSS12]. Figure 5.1 (a subgraph of Figure 2 in [HKSS12]) shows the average power consumption at 2373-th time point on the traces of DPAv2, using mean values over 100 non-overlapping subsequent

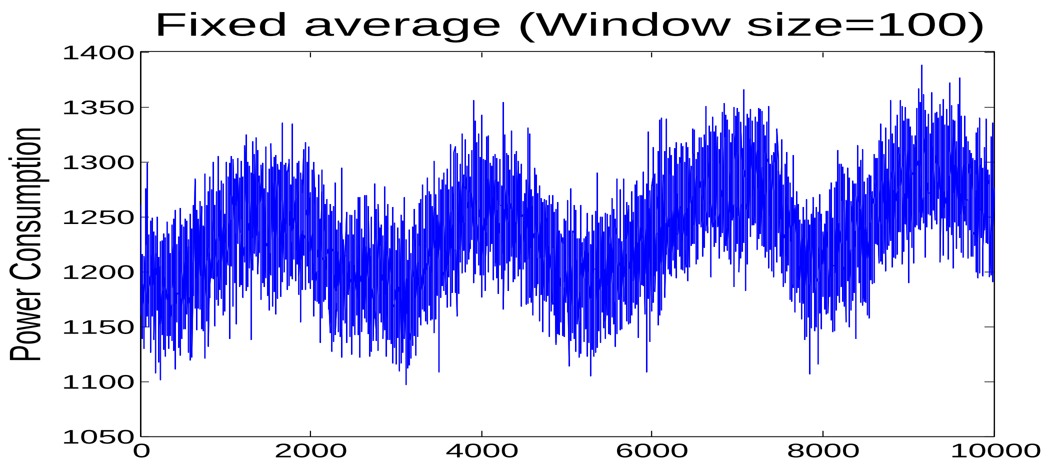


Figure 5.1: Power consumption moving averages at a key-sensitive leakage point on the DPAv2 template traces

traces.

Testing labs usually try to control the environmental factors to reduce such temporal variation. However, such effort can be expensive and there is no guarantee that all noise induced by environmental factors can be removed. Instead, we can deal with these environmental noise through statistical design. Particularly, we can adopt the *matched-pairs design* (Section 15.3 in [KNNL05]), where the measurements are taken in pairs with one each from the groups A and B. Then a *paired t-test* can be applied on such measurements, replacing the unpaired t-test (2.3). With n such pairs of measurements, we have n difference measurements $D = L_A - L_B$. The paired difference cancels the noise variation from the common source, making it easier to detect nonzero population difference. The null hypothesis of $\mu_A = \mu_B$ is equivalent to that the mean difference $\mu_D = 0$, which is tested by a paired t-test. Let $\bar{D}$ and s_D^2 denote the sample mean and sample variances of the paired differences D_1, ...,

D_n. The paired t-test statistic is

$$t_p = \frac{\bar{D}}{\sqrt{\frac{s_D^2}{n}}}, \tag{5.1}$$

with the degree of freedom $n-1$. The null hypothesis of non-leakage is rejected when $|t_p|$ exceeds the threshold of 4.5.

To quantify the difference between the two versions of t-test, we can compare the paired t-test (5.1) and the unpaired t-test (2.3) here with $n_A = n_B = n$.

First, without common variation sources under model (2.4), $Var(D) = Var(L_A) + Var(L_B) = \tilde{\sigma}_A^2 + \tilde{\sigma}_B^2$. Here $\tilde{\sigma}_A^2 = \sigma_A^2 + Var[V(k, x_A)]$ and $\tilde{\sigma}_B^2 = \sigma_B^2 + Var[V(k, x_B)]$. Notice that $\bar{D} = \bar{L}_A - \bar{L}_B$, so for large n, the paired t-test and unpaired t-test are equivalent with $t_u \approx t_p \approx (\bar{L}_A - \bar{L}_B)/\sqrt{(\tilde{\sigma}_A^2 + \tilde{\sigma}_B^2)/n}$. The paired t-test works even if the two group variances are unequal $\tilde{\sigma}_A^2 \neq \tilde{\sigma}_B^2$. The two versions of the t-test perform almost the same in this case.

However, the paired t-test detects leakage faster if there are common noise variation sources. To see this, we explicitly model the common environmental factor induced variation not covered by model (2.4).

$$L_A = V(k, x_A) + r_A + r_E \qquad\qquad L_B = V(k, x_B) + r_B + r_E, \tag{5.2}$$

where r_E is the noise caused by common environmental factors, with mean zero and variance σ_E. The r_A and r_B here denote the random measurement noises excluding common variations so that r_A and r_B are independent, with zero means and variance σ_A^2 and σ_B^2 respectively. Again we denote $\tilde{\sigma}_A^2 = \sigma_A^2 + Var[V(k, x_A)]$ and $\tilde{\sigma}_B^2 = \sigma_B^2 + Var[V(k, x_B)]$. Then $t_u \approx (\bar{L}_A - \bar{L}_B)/\sqrt{(\tilde{\sigma}_A^2 + \tilde{\sigma}_B^2 + 2\sigma_E^2)/n}$ while $t_p \approx (\bar{L}_A - \bar{L}_B)/\sqrt{(\tilde{\sigma}_A^2 + \tilde{\sigma}_B^2)/n}$. The paired t-test statistic $|t_p|$ has a bigger value than the unpaired t-test $|t_u|$, thus identifies the leakage more efficiently. The difference increases when the environmental noise σ_E increases. Hence, the paired t-test performs as well or better than the unpaired test. However, the matched-pairs design of the paired t-test cancels common noise found in both pairs, making the test more robust to suboptimal measurement setups and environmental noise.

5.2.2 Higher-Order and Multivariate Leakage Detection

The t-test can also be applied to detect higher public key leakage and multivariate leakage [GJJR11, SM15]. For d-th public key leakage at a single time point, the t-test compares sample means of $(L_A - \bar{L}_A)^d$ and $(L_B - \bar{L}_B)^d$. Under the matched-pairs design, the paired t-test would simply work on the difference

$$D = [(L_A - \bar{L}_A)^d - (L_B - \bar{L}_B)^d] \tag{5.3}$$

to yield the test statistic (5.1): $t_p = \bar{D}/\sqrt{s_D^2/n}$. Multivariate leakage combines leakage observation at multiple time points. A d-variate leakage combines leakage $L^{(1)}$, ..., $L^{(d)}$ at the d time points t_1, ..., t_d respectively. The combination is done through the centered product $CP(L^{(1)}, ..., L^{(d)}) = (L^{(1)} - \bar{L}^{(1)})(L^{(2)} - \bar{L}^{(2)}) \cdots (L^{(d)} - \bar{L}^{(d)})$. The standard d-variate leakage detection t-test compares the sample means of $CP(L_A^{(1)}, ..., L_A^{(d)})$ and $CP(L_B^{(1)}, ..., L_B^{(d)})$ with statistic (2.3). The paired t-test (5.1) uses the difference $D = [CP(L_A^{(1)}, ..., L_A^{(d)}) - CP(L_B^{(1)}, ..., L_B^{(d)})]$.

However, these tests (including the paired t-test) do not eliminate environmental noise effects on the higher public key and multivariate leakage detection. The centering terms (the subtracted $\bar{L}$) in the combination function also need adjustment due to environmental noises, which are not random noise but follow some temporal patterns. Actually, as thoroughly explained in [DCE16], we showed that it is opti-

Table 5.1: Computation Accuracy between our incremental method and Two-pass algorithm

	1st public key	2nd public key	3rd public key	4th public key	5th public key
Our method	50.0097	2.4679e+3	4.5981e+5	7.3616e+7	1.7974e+10
Two Pass	50.0097	2.4679e+3	4.5981e+5	7.3616e+7	1.7974e+10

mal to calculate the centering means $\bar{L}^{(1)}$ and $\bar{L}^{(2)}$ as moving averages from traces with a window of size n_w around the trace to be centered, rather than the average over all traces when there are strong environmental noise at the price of a very small statistical efficiency loss when no environmental noises exist. The details of proof can be found in [DCE16].

5.2.3 Computational Efficiency

The paired t-test also has computational advantages over Welch's t-test. As pointed out in [SM15], computational stability can become an issue when using raw moments for large measurement campaigns. The paired t-test computes mean $\bar{D}$ and variance s_D^2 of local differences D. In case there is no detectable leakage, L_A and L_B have the same mean. Hence, the differences D are mean-free. Even computing $\bar{D} = \frac{1}{n_i} \sum d_i$ is thus numerically stable. The sample variance s_D^2 can be computed as $s_D^2 = \overline{D^2} - (\bar{D})^2$, where only the first term $\overline{D^2}$ is not mean-free. We used the incremental equation from [Péb08, eq. (1.3)] to avoid numerical problems. Moreover, by applying the incremental equation for $\bar{D}$ as well, we were able to exploit straightforward parallelism when computing $\bar{D}$ and variance s_D^2.

The situation essentially remains the same for higher public key or multivariate analysis: The differences D are still mean-free in the no-leakage case. Through the use of local averages, the three-pass approach is not necessary, since moving averages are used instead of global averages. Computing moving averages is a local operation, as only nearby traces are considered. When processing traces in large blocks of e.g. 10k traces, all data needed for local averages is within the same file and can easily be accessed when needed, making the algorithm essentially one-pass. Similarly as in [SM15], we also give the experimental results using our method on 100 million simulated traces with $\sim \mathcal{N}(100, 25)$. Specifically, we compute the second parameters s_D^2 using the difference leakages: $D = L_A - L_B$ for first public key test while $D = [(L_A - \bar{L}_{A,n_w})^d - (L_B - \bar{L}_{B,n_w})^d]$ for d-th public key tests with moving average of window size $n_w = 100$. Table 5.1 shows our method matches the two-pass algorithm which computes the mean first and then the variance of the preprocessed traces. Note that D is not normalized using the central moment CM_2 and thus the second parameter is significantly larger than that in [SM15]. In the experiments, the same numerical stability is achieved without an extra pass, by focusing on the difference leakages.

5.3 Experimental Verification

To show the advantages of the new approach, the performances of the paired t-test (5.1) and the unpaired t-test (2.3) on synthetic data are compared.

First, we generate data for first public key leakage according to model (5.2), where the environmental noise r_E follows a sinusoidal pattern similar to Figure 5.1. The sinusoidal period is set as $200,000$ traces, and the sinusoidal magnitude is set as the pure measurement noise standard deviation $\sigma_A = \sigma_B = 50$. Hamming weight (HW) leakage is assumed in model (5.2). The first group A uses a fixed plaintext input corresponds to $HW = 5$, while the second group B uses random plaintexts. The paired t-test (5.1) and the unpaired t-test (2.3) are applied to the first $n = 30000, 60000, ..., 300000$ pairs of traces. The experiment is repeated 1000 times, and the proportions of leakage detection (rejection by each t-test) are plotted in Figure 5.2.

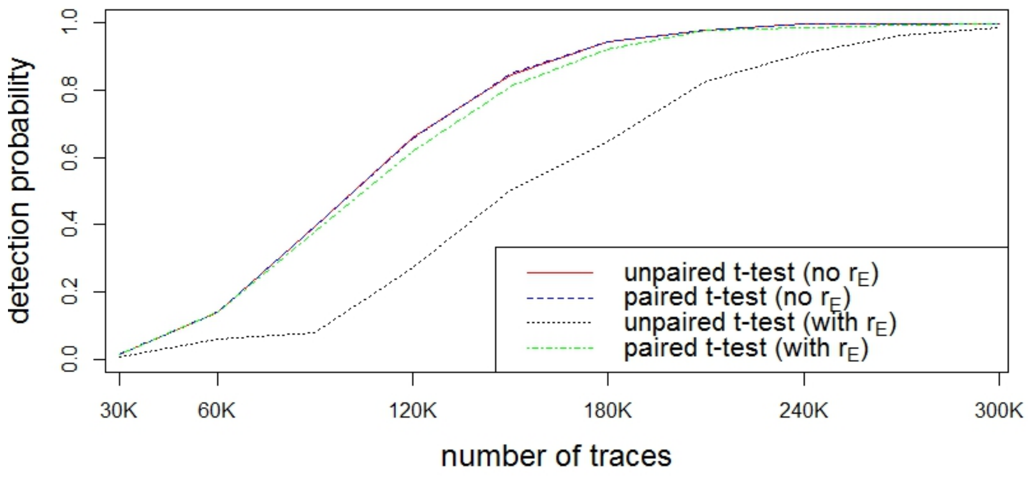


Figure 5.2: T-test comparison for 1O leakage with and without a sinusoidal drift r_E.

Without any environmental noise r_E, the paired and unpaired t-tests perform the same. Their success rate curves overlap each other. With the sinusoidal noise r_E, the unpaired t-test uses many more traces to detect the leakage, while the paired t-test does not suffer from such performance degradation.

Notice that the environmental noise r_E often changes slowly as in Figure 5.1. Hence, its effect is small for easy to detect leakage, when only a few hundreds or a few thousands of traces are needed. However, for hard to detect leakage, the effect has to be considered. We set a high noise level $\sigma_A = \sigma_B = 50$ to simulate a DUT with hard to detect first-public key leakage. This allows the observable improvement by paired t-test over the unpaired t-test.

Second, we also generate data from the 2nd-public key leakage model. The noise levels at the two leakage points, for both groups A and B, are set as $\sigma_1 = \sigma_2 = 10$ which are close to the levels in the physical implementation reported by [DZFL14].

We use the same sinusoidal environmental noise r_E as before. The first group A uses a fixed plaintext input corresponding to $HW = 1$, while the second group B uses random plaintexts. The proportions of leakage detection are plotted in Figure 5.3.

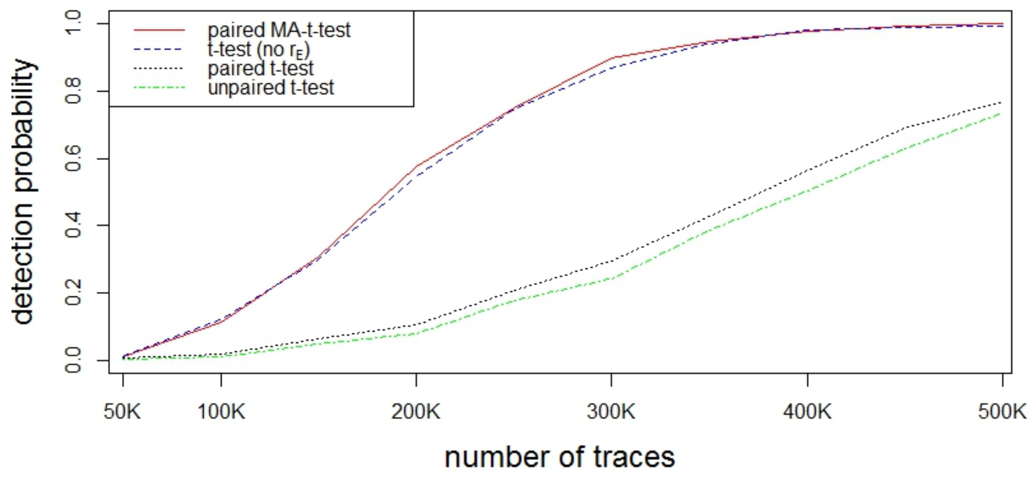


Figure 5.3: T-test comparison for 2O leakage with a sinusoidal drift r_E.

Again, we observe a serious degradation of t-test power to detect the leakage, when the environmental noise r_E is present. The paired t-test detects the leakage more often than the unpaired t-test in Figure 5.3. However, the paired t-test also degrades comparing to the case without environmental noise r_E. That is due to the incorrect centering quantity for the 2O test as discussed in Section 5.2.2. Using the proposed method of centering at the moving average with window size 100, the paired MA-t-test has a performance close to the case where all environmental noise r_E is removed.

5.4 Practical Application

To show the advantage of the paired t-test in real measurement campaigns, we compare the performances of the unpaired and paired t-tests when analyzing an unprotected and an protected hardware implementation. The analysis focuses on the non-specific fixed vs. random t-test. We apply both tests to detect the first public key leakage in the power traces acquired from an unprotected implementation of the NSA lightweight cipher Simon [BSS+13]. More specifically, a round-based implementation of Simon128/128 was used, which encrypts a 128-bit plaintext block with a 128-bit key in 68 rounds of operation. The second target is a masked engine of the same cipher. It is protected using three-share Threshold Implementation (TI) scheme, which is a round based variant of the TI Simon engine proposed in [STE15].

Both implementations are ported onto the SASEBO-GII board for power trace collection. The board is clocked at 3 MHz and a Tektronix oscilloscope samples the power consumption at 100MS/s. Since Simon128/128 has 68 rounds, one power trace has about $68 \times \frac{1}{3MHz} \times 100MS/s \approx 2300$ time samples to cover the whole encryption and hence in the following experiments 2500 samples are taken in each measurement. The measurement setup is a modern setup that features a DC block and an amplifier. Note that the DC block will already take care of slow DC drifts that can affect the sensitivity of the unpaired t-test, as shown in Section 5.3. However, the DC block does not affect variations of the peak-to-peak height within traces, which are much more relevant for DPA. As the following experiments show, the paired t-test still shows improvement in such advanced setups.

5.4.1 Solving the Test-Order Bias

In [SM15], a random selection between fixed and random is proposed to avoid effects caused by states that occur in a fixed public key, which we refer to as *test public key*. For the paired (MA-)t-test, it is preferable to have a matching number of observations for both sets. We propose a fixed input sequence which is a repetition of $ABBA$ such that all the AB or BA pairs are constructed using neighboring inputs.

For example in a sequence $ABBAABBA....ABBAABBA$, one alternately obtains AB and BA pairs with least variation. This ensures that all observations come in pairs and that the pairs are temporally close, so they share their environmental effects to a maximal possible degree. Moreover—even though the sequence is fixed and highly regular, the predecessor and successor for each measurement are perfectly balanced, corresponding to a 50% probability of being either from the A or B set. This simpler setup removes the biases observed in [SM15] as efficiently as the random selection method. Experimental data of this section has been obtained using this scheme.

Note that the paired t-test can easily be applied in a random selection test public key as well: After the trace collection, one can simply iteratively pair the leakages associated with the oldest fixed input and the oldest random input and then remove them from the sequence until no pairs can be constructed. An efficient way to do this is to separate all leakage traces into two subsets: $L_A = \{l_{A,1}, ...l_{A,n_A}\}$ and $L_B = \{l_{B,1}, ...l_{B,n_B}\}$ where $l_{A,i}$ and $l_{B,i}$ are the traces associated with i-th fixed input and i-th random input respectively in a chronological public key and thus can be straightforwardly paired. Note that the cardinality of both sets are not always the same and hence only $n = min(n_A,\ n_B)$ AB pairs can be found. This approach is of less interest because time delay between fixed data and random data in a pair varies depending on the randomness of the input sequence.

5.4.2 First-Order Analysis of an Unprotected Cipher

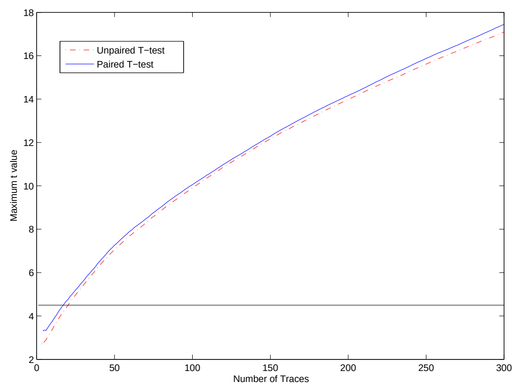
We first apply both paired and unpaired t-test to the unprotected engine which has strong first public key leakage that can be exploited by DPA with only hundreds of traces. Usually the trace collection can be done quickly enough to avoid effects of environmental fluctuation in the measurements. However, to show the benefits of the paired t-test in this scenario, a hot air blower is used to heat up the crypto FPGA in SASEBO-GII board while the encryptions are executed. We designed two conditions to take the power measurements.

1. **Normal Lab Environment**, where measurements are performed in rapid succession, making the measurement campaign finish within seconds.

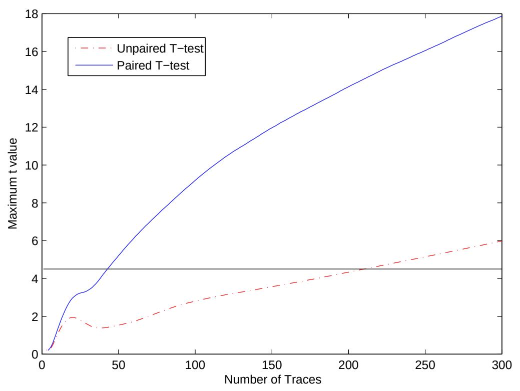
2. **Strong Environmental Fluctuation**, where a hot air blower was slowly moved towards and then away from the target FPGA to heat up and let it cool down again;

In each condition, 1000 measurements are taken alternately for the fixed plaintext and random plaintexts and later equally separated into two groups. In each group, the measurements are sorted in chronological public key such that the j-th measurements of both groups are actually taken consecutively and share common variation. As explained in Section 5.4.1, the two measurements are a *matched-pair* and there are now 500 such pairs. Then both t-tests are applied to the first $n = 5, 6, 7, ..., 500$ pairs of measurements. For each n, the t-test returns a t-statistic vector of 2500 elements corresponding to 2500 time samples in the power traces because it is a univariate t-test. Our interest is the time sample that has the maximum t-statistic and thus the following results only focus on this specific time sample.

Figure 5.4 shows the t-statistics at the strongest leakage point as n increases. In Figure 5.4.1 where there is no environmental fluctuation, both unpaired and paired t-test have the same performance as the t-statistic curves almost overlap. However, in Figure 5.4.2 where the varying temperature changed the power traces greatly, the paired t-test (blue solid line) shows robustness and requires less traces to exceed the threshold of 4.5 while the performance of the unpaired t-test is greatly reduced in the sense that more traces are needed to go beyond the threshold. Figure 5.5 shows



5.4.1: No Environment Fluctuation

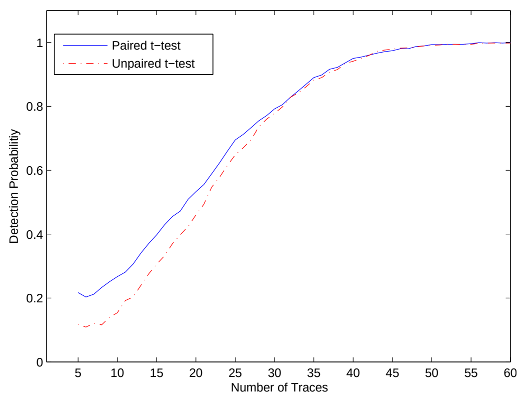


5.4.2: Environment Fluctuation

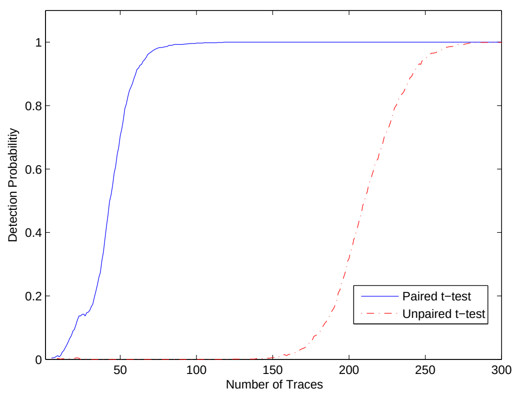
Figure 5.4: T-test comparison for 1O leakage on unprotected Simon for a single measurement campaign of up to 300 pairs of traces. The paired t-test performs as well or better in both scenarios. However, the paired t-test is more robust to environmental noise.

the detection probability of the t-tests in the same scenario. First, 1000 repetitions of the above experiment are performed and the number of experiments that result in a t-statistic above 4.5 is counted. Detection probability equals this number divided by 1000. Figure 5.5.1 shows the detection probability of two tests under normal lab condition. With more than 30 pairs, both tests can detect the first public key leakage with the same probability. With more than 60 pairs the detection probability rises to 1 for both tests which shows the efficiency of both tests on the normal traces. Figure 5.5.2 shows that paired t-test (solid line) is still robust in spite of varying environmental factors. With less than 100 pairs, the detection probability of paired t-test is already 1 while unpaired t-test requires much more traces to achieve the same probability.

In summary, the paired t-test is more robust and efficient in detecting first public key leakage when the power traces are collected in a quickly changing environment.



5.5.1: No Environment Fluctuation



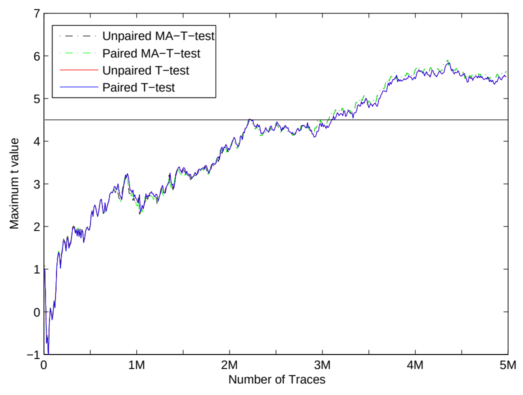
5.5.2: Environment Fluctuation

Figure 5.5: T-test detection probability for 1O leakage. Again, the paired t-test performs at least as well as the unpaired, while being much more robust in the presence of environmental noise.

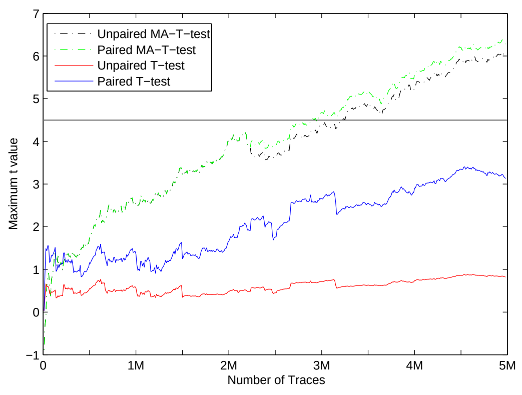
5.4.3 Second-Order Analysis on a Protected Design

In public key to validate the effectiveness of the paired t-test in a longer measurement campaign, where environmental fluctuations are very likely to occur, a first-public key-leakage-resistant Simon engine protected by a three-share Threshold Implementation scheme is used as the target. Five million power traces are collected in a room without windows and without expected fluctuations in temperature over a period 5 hours. As before, one measurement campaign is performed in a stable lab environment where the environmental conditions are kept as stable as possible. In the other scenario, we again used the hot air blower in intervals of several minutes to simulate stronger environmental noise. This is because the environmental noise might not be strong during the 5-hour collection period. However, in scenarios where hundreds of millions of measurements are needed and taken over a period of several days, then environmental fluctuation can be found, as in Figure 5.1.

As before, the 5 million traces are equally divided into two groups for fixed and random plaintext respectively. The first public key t-test does not indicate any leakage ($|t| < 3$), as expected. Figure 5.6 shows the t-statistics of the second public key



5.6.1: No Environment Fluctuation



5.6.2: Environment Fluctuation

Figure 5.6: T-test detection probability for 2O leakage

t-tests as the number of traces increases in both the stable lab environment and the simulated lab environment noise scenario. In the first experiment in a stable environment, depicted in Figure 5.6.1, we compare both tests using global average and moving average. The curve of four tests almost overlap and they perform about the same with about three million traces needed to achieve a t-statistic above 4.5. This shows that paired t-test works as well as unpaired one for constant collection environment. Also, the moving average based tests perform very similar to the global average based tests, with a minor improvement in the relevant many-traces case. Figure 5.6.2 depicts the results for the experiment with strong environmental fluctuations. The paired MA-t-test performs best and goes beyond 4.5 faster than the unpaired one. The other two tests using global average are still below the threshold with 5 million traces. The paired t-test still clearly outperforms the unpaired t-test. In sum, the paired t-test based on moving average is the most robust to fluctuation and significantly improves the performance of higher public key analysis.

5.5 Conclusion

Welch's t-test has recently received a lot of attention as standard side channel security evaluation tool. In this work we showed that noise resulting from environmental fluctuations can negatively impact the performance of Welch's t-test. The resulting increased number of observations to detect a leakage are inconvenient and can, in the worst case, result in false conclusions about a device's resistance. We proposed a paired t-test to improve the standard methodology for leakage detection. The resulting matched-pairs design removes the environmental noise effect in leakage detection. Furthermore, we showed that moving averages increase the robustness against environmental noise for higher public key or multivariate analysis, while not showing any negative impact in the absence of noise. The improvement is shown through mathematical analysis, simulation, and on practical power measurements: both paired and unpaired t-test with and without the moving averages approach are compared for first public key and second public key analysis. Our results show that the proposed (moving average based) paired t-test performed as well or better in all analyzed scenarios. The new method does not increase computational complexity and is numerically more stable than Welch's t-test.

Chapter 6

Conclusion and Future Direction

Emerging computing technologies and applications have posed a serious challenge to the conventional cryptography and many efforts have been dedicated to the design of reliable algorithms for the future. Post-quantum cryptography will protect the data communication when large-scale quantum computers are available; lightweight cryptography tailored for constrained environment will secure billions of devices in the Internet of Things. However, side channel analysis remains a threat when these new algorithms are running on physical devices.

6.1 Summary

In this dissertation, we investigate the side channel security of several emerging cryptosystem from three major aspects in the study of side channel analysis, namely leakage exploitation, leakage mitigation and leakage detection.

In terms of leakage exploitation, we propose the first differential power analysis of a QC-MDPC McEliece Implementation. We show that the leakages of a key rotation operation can be efficiently exploited to recovers the majority of the private key bits. Then, we designed an algebraic step to exploit the key structure leading to the full key recovery. In order to counteract the attack, we apply the state-of-the-art Threshold Implementation that is leakage resistant with a formal proof and concrete attack proof. This is so far as we know the first application of Threshold Implementation in public key cryptography.

With respect to leakage mitigation, we propose the first practical threshold implementations using only two shares. We show how using two shares can actually yield smaller cipher implementations with less randomness and perfect first order resistance. Applying it on lightweight ciphers will secure the Internet of Thing with low cost. We also propose a balanced encoding countermeasure for software and perform the first practical evaluation. It is an effective hiding countermeasure to achieve constant leakage so as to bring down the SNR and can be combined with masking schemes for better leakage resistance.

Eventually, we propose a paired t-test to improve the standard methodology for leakage detection. The resulting matched-pairs design removes the environmental noise effect in leakage detection. Furthermore, we showed that moving averages increase the robustness against environmental noise for higher order or multivariate analysis, while not showing any negative impact in the absence of noise.

6.2 Future direction

In addition to the topics discussed in the previous sections, we would continue our research on post-quantum cryptography and investigate other post-quantum public key algorithms from the perspective of side channel vulnerability. In fact, there exist many post-quantum candidate schemes. In the first round submission for the post-quantum cryptography standardization by NIST, there are already 69 candidates which are based on different hard problems such as Lattice-based and

Code-based algorithms of which we are particularly interested in LWE schemes. We will implement one of these schemes and attempt to exploit the power leakage for key recovery. Further, we will investigate efficient countermeasures to protect the physical implementations of such schemes.

Bibliography

[AGS14] A. Aysu, E. Gulcan, and P. Schaumont. SIMON Says: Break Area Records of Block Ciphers on FPGAs. *Embedded Systems Letters, IEEE*, 6(2):37–40, June 2014.

[AHPT11] Roberto Avanzi, Simon Hoerder, Dan Page, and Michael Tunstall. Side-channel attacks on the McEliece and Niederreiter public-key cryptosystems. *Journal of Cryptographic Engineering*, 1(4):271–281, 2011.

[BCG$^+$12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Sren S. Thomsen, and Tolga Yalçin. PRINCE – A Low-Latency Block Cipher for Pervasive Computing Applications. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology — ASIACRYPT 2012*, pages 208–225. Springer Berlin Heidelberg, 2012.

[BCNS15] Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *36th IEEE Symposium on Security and Privacy*, 2015.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems — CHES 2004*, volume 3156 of *Springer LNCS*, pages 135–152, 2004.

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *Journal of Symbolic Computation*, 24:235–265, 1997.

[BDN$^+$14] Begül Bilgin, Joan Daemen, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Gilles Van Assche. Efficient and First-Order DPA Resistant Implementations of Keccak. In Aurlien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications*, LNCS, pages 187–199. Springer, 2014.

[BGG$^+$15] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and Franois-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer International Publishing, 2015.

[BGN$^+$14a] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. A More Efficient AES Threshold Implementation. In David Pointcheval and Damien Vergnaud, editors, *Progress in Cryptology –AFRICACRYPT 2014*, volume 8469 of *LNCS*, pages 267–284. Springer, 2014.

[BGN$^+$14b] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology — ASIACRYPT 2014*, volume 8874 of *LNCS*, pages 326–343. Springer, 2014.

[BGN$^+$15] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. Trade-Offs for Threshold Implementations Illustrated on AES. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(7):1188–1200, July 2015.

[BKL$^+$07] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings*, pages 450–466, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[BMvT78] Elwyn R. Berlekamp, Robert J. McEliece, and Henk C.A. van Tilborg. On the Inherent Intractability of Certain Coding Problems (Corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, May 1978.

[BSS$^+$13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The simon and speck families of lightweight block ciphers. *IACR Cryptology ePrint Archive*, 2013:404, 2013.

[CDG$^+$13] J Cooper, E DeMulder, G Goodwill, J Jaffe, G Kenworthy, and P Rohatgi. Test vector leakage assessment (tvla) methodology in practice. In *International Cryptographic Module Conference*, 2013.

[CESY14] Cong Chen, Thomas Eisenbarth, Aria Shahverdi, and Xin Ye. Balanced encoding to mitigate power analysis: a case study. In *International Conference on Smart Card Research and Advanced Applications*, pages 49–63. Springer, 2014.

[CEvMS15a] Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. Differential power analysis of a mceliece cryptosystem. In *International Conference on Applied Cryptography and Network Security*, pages 538–556. Springer, 2015.

[CEvMS15b] Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. Masking Large Keys in Hardware: A Masked Implementation of McEliece. In *Selected Areas in Cryptography — SAC 2015*. Springer LNCS, August 2015. Preprint available at `http://eprint.iacr.org/924`.

[CEvMS16] Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. Horizontal and vertical side channel analysis of a mceliece cryptosystem. *IEEE Transactions on Information Forensics and Security*, 11(6):1093–1105, 2016.

[CFE16] Cong Chen, Mohammad Farmani, and Thomas Eisenbarth. A tale of two shares: why two-share threshold implementation seems worthwhileand why it is not. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 819–843. Springer, 2016.

[CGV14] Jean-Sébastien Coron, Johann Großschädl, and PraveenKumar Vadnala. Secure Conversion between Boolean and Arithmetic Masking of Any Order. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 188–205. Springer Berlin Heidelberg, 2014.

[CJRR99] Suresh Chari, Charanjit S Jutla, Josyula R Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *Advances in Cryptology – CRYPTO'99*, pages 398–412. Springer, 1999.

[CK10] Jean-Sebastien Coron and Ilya Kizhvatov. Analysis and improvement of the random delay countermeasure of ches 2009. In Stefan Mangard and Francois-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 95–109. Springer Berlin Heidelberg, 2010.

[CPR07] Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side Channel Cryptanalysis of a Higher Order Masking Scheme. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings*, pages 28–44, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[CZ06] Zhimin Chen and Yujie Zhou. Dual-rail random switching logic: a countermeasure to reduce side channel leakage. In *Cryptographic Hardware and Embedded Systems-CHES 2006*, pages 242–254. Springer, 2006.

[DCE16] A Adam Ding, Cong Chen, and Thomas Eisenbarth. Simpler, faster, and more robust t-test based leakage detection. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 163–183. Springer, 2016.

[DCRB+16] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with $d+1$ Shares in Hardware. In Benedikt Gierlichs and Y. Axel Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016: 18th International Conference*, pages 194–212. Springer Berlin Heidelberg, 2016.

[DS16] Franois Durvaux and Franois-Xavier Standaert. From improved leakage detection to the detection of points of interests in leakage traces. accepted at Eurocrypt 2016, 2016. preprint available at `http://ia.cr/2015/536`.

[DSVC14] F. Durvaux, F.-X. Standaert, and N. Veyrat-Charvillon. How to certify the leakage of a chip? In *to appear in the proceedings of Eurocrypt 2014*. Springer LNCS, 2014.

[DZFL14] A.Adam Ding, Liwei Zhang, Yunsi Fei, and Pei Luo. A statistical model for higher order dpa on masked devices. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 147–169. Springer Berlin Heidelberg, 2014.

[EKM+08] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T Manzuri Shalmani. On the Power of Power Analysis in the Real World: A Complete Break of the Keeloq Code Hopping Scheme. In *Advances in Cryptology–CRYPTO 2008*, pages 203–220. Springer, 2008.

[Gal62] Robert Gallager. Low-density Parity-check Codes. *Information Theory, IRE Transactions on*, 8(1):21–28, 1962.

[GJJR11] G Goodwill, B Jun, J Jaffe, and P Rohatgi. A testing methodology for side-channel resistance validation. In *NIST Non-Invasive Attack Testing Workshop*, Sept. 2011.

[GMK16] Hannes Gross, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, pages 3–3. ACM, 2016.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded SystemsCHES 2001*, pages 251–261. Springer, 2001.

[GST14] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA Key Extraction via Low-bandwidth Acoustic Cryptanalysis. In *International Cryptology Conference*, pages 444–461. Springer, 2014.

[HDD14] Philippe Hoogvorst, Guillaume Duc, and Jean-Luc Danger. Software Implementation of Dual-Rail Representation. In *2nd International Workshop on Constructive Side-Channel Analysis ande Secure Design — COSADE 2011*, February 24-25 2014.

[HKSS12] Annelie Heuser, Michael Kasper, Werner Schindler, and Marc Stöttinger. A new difference method for side-channel analysis with high-dimensional leakage models. In Orr Dunkelman, editor, *Topics in Cryptology – CT-RSA 2012*, volume 7178 of *Lecture Notes in Computer Science*, pages 365–382. Springer Berlin Heidelberg, 2012.

[HMP10] Stefan Heyse, Amir Moradi, and Christof Paar. Practical Power Analysis Attacks on Software Implementations of McEliece. In Nicolas Sendrier, editor, *Post-Quantum Cryptography – PQCrypto 2010*, volume 6061 of *Lecture Notes in Computer Science*, pages 108–125, Berlin Heidelberg, 2010. Springer.

[HP10] W. Cary Huffman and Vera Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, United Kingdom, 2010.

[HvMG13] Stefan Heyse, Ingo von Maurich, and Tim Güneysu. Smaller Keys for Code-Based Cryptography: QC-MDPC McEliece Implementations on Embedded Devices. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems – CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 273–292, Berlin Heidelberg, 2013. Springer.

[HZL12] Yang Han, Yongbin Zhou, and Jiye Liu. Securing lightweight block cipher against power analysis attacks. In Ying Zhang, editor, *Future Wireless Networks and Information Systems*, volume 143 of *Lecture Notes in Electrical Engineering*, 2012.

[IIES14] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Wait A Minute! A fast, Cross-VM Attack on AES. In *International Workshop on Recent Advances in Intrusion Detection*, pages 299–319. Springer, 2014.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397, Berlin Heidelberg, 1999. Springer.

[KJJR11] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011.

[KNNL05] Michael H Kutner, Christopher J Nachtsheim, John Neter, and William Li. Applied linear statistical models. In *Applied linear statistical models*. McGraw-Hill Irwin New York, 2005.

[KNPW13] Sebastian Kutzner, PhuongHa Nguyen, Axel Poschmann, and Huaxiong Wang. On 3-Share Threshold Implementations for 4-Bit S-boxes. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design*, volume 7864 of *Springer LNCS*, pages 99–113, 2013.

[Knu92] Donald E. Knuth. Two Notes on Notation. *The American Mathematical Monthly*, 99(5):403–422, May 1992.

[Koc96] Paul Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Annual International Cryptology Conference*, pages 104–113. Springer, 1996.

[KP09] M. Kirschbaum and T. Popp. Evaluation of a DPA-Resistant Prototype Chip. In *Computer Security Applications Conference, 2009. ACSAC '09. Annual*, pages 43–50, Dec 2009.

[LMW14] Andrew J. Leiserson, Mark E. Marson, and Megan A. Wachs. Gate-Level Masking under a Path-Based Leakage Metric. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, volume 8731 of *Springer LNCS*, pages 580–597, 2014.

[McE78] Robert J. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*, 44:114–116, January 1978.

[MH15] Amir Moradi and Gesine Hinterwälder. Side-channel security analysis of ultra-low-power fram-based mcus. In *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, pages 239–254, 2015.

[MM12] Amir Moradi and Oliver Mischke. How Far Should Theory Be from Practice? In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012: 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, pages 92–106, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[MOBW13] Luke Mather, Elisabeth Oswald, Joe Bandenburg, and Marcin Wójcik. Does my device leak information? an a priori statistical power analysis of leakage detection tests. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, volume 8269 of *Lecture Notes in Computer Science*, pages 486–505. Springer Berlin Heidelberg, 2013.

[MPL$^+$11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In Kenneth G. Paterson, editor, *Advances in Cryptology — EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 69–88. Springer, 2011.

[MTSB13] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo S. L. M. Barreto. MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes. In *Proceedings of the 2013 IEEE International Symposium on Information Theory (ISIT)*, pages 2069–2073. IEEE, 2013.

[NLD15] Erick Nascimento, Julio Lopez, and Ricardo Dahab. Efficient and secure elliptic curve cryptography for 8-bit avr microcontrollers. In Rajat Subhra Chakraborty, Peter Schwabe, and Jon Solworth, editors, *Security, Privacy, and Applied Cryptography Engineering*, volume 9354 of *Lecture Notes in Computer Science*, pages 289–309. Springer International Publishing, 2015.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng

Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer Berlin Heidelberg, 2006.

[Péb08] Philippe Pébay. Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments. *Sandia Report SAND2008-6212, Sandia National Laboratories*, 2008.

[PM05] Thomas Popp and Stefan Mangard. Masked dual-rail pre-charge logic: Dpa-resistance without routing constraints. In JosyulaR. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 172–186. Springer Berlin Heidelberg, 2005.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against Side-Channel Attacks: A Formal Security Proof. In Thomas Johansson and PhongQ. Nguyen, editors, *Advances in Cryptology — EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159. Springer Berlin Heidelberg, 2013.

[RBN$^+$15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating Masking Schemes. In *Advances in Cryptology–CRYPTO 2015*, pages 764–783. Springer LNCS, 2015.

[RGN13] Pablo Rauzy, Sylvain Guilley, and Zakaria Najm. Formally proved security of assembly code against power analysis: A case study on balanced logic. In *eprint*, 2013. https://eprint.iacr.org/2013/554.pdf.

[RHHM17] Mélissa Rossi, Mike Hamburg, Michael Hutter, and Mark E Marson. A side-channel assisted cryptanalytic attack against qcbits. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 3–23. Springer, 2017.

[SCE14] Aria Shahverdi, Cong Chen, and Thomas Eisenbarth. AVRprince - An Efficient Implementation of PRINCE for 8-bit Microprocessors. Technical report, Worcester Polytechnic Institute, 2014. `http://users.wpi.edu/~teisenbarth/pdf/avrPRINCEv01.pdf`.

[Sho97] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms On a Quantum Computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

[SM15] Tobias Schneider and Amir Moradi. Leakage assessment methodology - a clear roadmap for side-channel evaluations. In Tim Gneysu and Helena Handschuh, editors, *CHES*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2015.

[SMG15] Tobias Schneider, Amir Moradi, and Tim Gneysu. Arithmetic addition over boolean masking - towards first- and second-order resistance in hardware. International Conference on Applied Cryptography and Network Security – ACNS 2015, 2–5 June 2015.

[SMY09] François-Xavier Standaert, Tal G. Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. *Advances in Cryptology — EUROCRYPT 2009*, pages 443–461, 2009.

[SSMS10] Abdulhadi Shoufan, Falko Strenzke, H.Gregor Molter, and Marc Stöttinger. A Timing Attack against Patterson Algorithm in the McEliece PKC. In Donghoon Lee and Seokhie Hong, editors, *Information, Security and Cryptology – ICISC 2009*, volume 5984 of *Lecture Notes in Computer Science*, pages 161–175. Springer, Berlin Heidelberg, 2010.

[STE15] Aria Shahverdi, Mostafa Taha, and Thomas Eisenbarth. Silent Simon: A Threshold Implementation under 100 Slices. In *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*, pages 1–6, May 2015.

[STM$^+$08] Falko Strenzke, Erik Tews, H. Gregor Molter, Raphael Overbeck, and Abdulhadi Shoufan. Side Channels in the McEliece PKC. In Johannes Buchmann and Jintai Ding, editors, *Post-Quantum Cryptography – PQCrypto 2008*, volume 5299 of *Lecture Notes in Computer Science*, pages 216–229, Berlin Heidelberg, 2008. Springer.

[Str10] Falko Strenzke. A Timing Attack against the Secret Permutation in the McEliece PKC. In Nicolas Sendrier, editor, *Post-Quantum Cryptography – PQCrypto 2010*, volume 6061 of *Lecture Notes in Computer Science*, pages 95–107, Berlin Heidelberg, 2010. Springer.

[TH08] Stefan Tillich and Christoph Herbst. Attacking State-of-the-Art Software Countermeasures – A Case Study for AES. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 228–243. Springer, Berlin Heidelberg, 2008.

[TV04] Kris Tiri and Ingrid Verbauwhede. A logic level design methodology for a secure dpa resistant asic or fpga implementation. In *Proceedings of the conference on Design, automation and test in Europe*, page 10246. IEEE Computer Society, 2004.

[VCMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against Side-Channel Attacks: A

Comprehensive Study with Cautionary Note. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 740–757. Springer, Berlin Heidelberg, 2012.

[vMG14a] Ingo von Maurich and Tim Güneysu. Lightweight Code-based Cryptography: QC-MDPC McEliece Encryption on Reconfigurable Devices. In *Design, Automation and Test in Europe – DATE 2014*, pages 1–6. IEEE, 2014.

[vMG14b] Ingo von Maurich and Tim Güneysu. Towards Side-Channel Resistant Implementations of QC-MDPC McEliece Encryption on Constrained Devices. In Michele Mosca, editor, *Post-Quantum Cryptography*, volume 8772 of *Lecture Notes in Computer Science*, pages 266–282. Springer, 2014.

[vMOG15] Ingo von Maurich, Tobias Oder, and Tim Güneysu. Implementing QC-MDPC McEliece Encryption. *ACM Trans. Embed. Comput. Syst.*, 14(3):44:1–44:27, April 2015.

[WOS14] Carolyn Whitnall, Elisabeth Oswald, and François-Xavier Standaert. The Myth of Generic DPA...and the Magic of Learning. In Josh Benaloh, editor, *Topics in Cryptology – CT-RSA 2014*, volume 8366 of *Lecture Notes in Computer Science*, pages 183–205, International Publishing, 2014. Springer.