Technische Universität München
TUM School of Computation, Information and Technology

Advances of Side-Channel Analysis in the Areas of Post-Quantum Cryptography and Deep Learning-Based Attacks

Thomas Schamberger

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz:
Prof. Dr.-Ing. Antonia Wachter-Zeh

Prüfende der Dissertation:
1. Prof. Dr.-Ing. Georg Sigl
2. Prof. Dr.-Ing. Elif Bilge Kavun

Die Dissertation wurde am 25.06.2024 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 16.12.2024 angenommen.

I would like to sincerely thank Prof. Dr.-Ing. Georg Sigl for the opportunity to pursue this PhD and for supporting me with my research ideas. Further, I would like to thank Dr. Johanna Sepúlveda for encouraging me to take this opportunity while always providing me with full support and believing in my abilities. I'm also thankful to Dr.-Ing. Michael Pehl for the provided support in all administrative and scientific matters.

I would also like to thank my colleagues who made the years of this PhD project a period of my life on which I will certainly look back with very happy memories. I'm very thankful for the productive scientific discussions but also the numerous chats about the small and big questions of life. In particular, I would like to thank Lars Tebelmann, Michael Gruber and our newest member Tim Music for the nice atmosphere in the office, which certainly contributed to the success of this thesis. Additionally, I would like to thank Christoph Frisch and Michaela Brunner for the mutual support during the final phase of our thesis. I'm glad for all the meetings of the "post-quantum group" with Patrick Karl and Jonas Schupp and I'm happy that I was able to work with such motivated and brilliant colleagues.

I'm very thankful to all co-authors who shared their knowledge with me and contributed to my and our shared research. Without their help this thesis would not have been possible. I would especially like to thank Lars Tebelmann with whom I have co-authored many of my publications. Thanks for the proposed creative solutions, our shared work on the "Attack Framework" and the knowledge we built together. Further, I would like to thank Julian Renner, Lukas Holzbaur, and Prof. Dr.-Ing. Antonia Wachter-Zeh who shared their extensive knowledge of error-correcting codes with me and invested a lot of work on joint publications. Additionally, I would like to thank all students who worked under my supervision and whose work has contributed to my research. A special thanks goes to Maximilian Egger and Tim Kaiser for their excellent and dedicated work, which enabled parts of the research presented in this thesis.

Finally, I would like to thank my family and friends for their unconditional support. There are no words to describe my feelings of gratitude for Lisa, who joined me on this journey in the very beginning and has supported me with all her wisdom also during the challenging times. With this part of my life finally concluded, I'm very much looking forward to the upcoming joint adventures in life.

Abstract

Since the publication of the first power side-channel attack by Kocher et al. over two decades ago, attacks still threaten the information security of embedded devices. This is the case as these attacks do not exploit weaknesses in the mathematical security of an algorithm but rather exploit the inherent data-dependent power consumption of Complementary Metal-Oxide-Semiconductor (CMOS) logic in order to extract used cryptographic secrets. Countermeasures, therefore, have to be developed or adapted individually for each implementation or used cryptographic algorithm. Research in the domain of Side-Channel Analysis (SCA) is therefore crucial in order to develop new attack methodologies and to evaluate algorithms regarding possible attack vectors with the goal of developing resource-efficient countermeasures against them. This thesis provides multiple contributions in this domain.

The first part of this thesis discusses side-channel attacks and countermeasures for two post-quantum cryptography algorithms. These algorithms are based on other mathematical problems than established public-key cryptography and are considered secure even in the presence of a large-scale quantum computer. They are developed in the context of a standardization process by the National Institute of Standards and Technology (NIST) that identified that solutions have to be ready in the event of a sudden breakthrough in quantum computing as the migration of cryptographic infrastructure requires significant time. First, an evaluation of masking for NTRUEncrypt is presented, which was a promising candidate for standardization after a merger with a similar cryptosystem. Further, a comprehensive side-channel study of the HQC cryptosystem is shown. This algorithm is still a candidate for standardization and is based on error-correcting codes. The study includes the development and practical evaluation of chosen-ciphertext attacks for the initial version of HQC as well as its update with an improved error correction. Finally, the first part of this thesis is concluded with the evaluation of multiple countermeasures against the proposed attacks.

The second part of this thesis discusses the promising attack direction of using deep neural networks for profiled side-channel attacks. This field of deep learning-based side-channel analysis received considerable research interest as attacks have been shown to be able to break countermeasures while reducing the preprocessing effort of measurements significantly as, e.g., side-channel information that needs to be combined for a successful attack is identified automatically. Nevertheless, this leads to a reduced explainability of attack results, which is unfortunately required by the designer of a protected implementation to identify weaknesses and to strengthen applied countermeasures. This thesis shows contributions to explainability methods for these attacks in order to reduce this downside. First, the foundation for research on explainability methods is created with a comprehensive evaluation of the standard research datasets called ANSSI SCA Database (ASCAD) with classical side-channel analysis and a comparison of the different dataset versions. The shown results include unexpected side-channel leakage for a protected implementation and differing side-channel leakages between the target bytes of the attacked secret. Based on these observations, multiple advancements to the occlusion explainability method are presented. These include the identification of suitable occlusion parameters for ASCAD as well as a method to identify combinations of measurement regions required for a successful attack.

Zusammenfassung

Seit der Veröffentlichung des ersten strombasierten Seitenkanalangriffs durch Kocher et al. vor mehr als zwei Jahrzehnten bedrohen Angriffe immer noch die Informationssicherheit von eingebetteten Geräten. Dies ist der Fall, da diese Angriffe nicht Schwächen in der mathematischen Sicherheit eines Algorithmus nutzen, sondern vielmehr den inhärenten datenabhängigen Stromverbrauch von CMOS-Logik ausnutzen, um die verwendeten kryptografischen Schlüssel zu extrahieren. Gegenmaßnahmen müssen daher für jede Implementierung und jeden verwendeten Algorithmus individuell angepasst werden. Die Forschung auf dem Gebiet der Seitenkanalanalyse (SCA) ist daher von entscheidender Bedeutung, um neue Angriffsmethoden zu entwickeln und Algorithmen auf potenzielle Angriffsvektoren zu evaluieren, mit dem Ziel, ressourceneffiziente Gegenmaßnahmen zu entwickeln. Diese Arbeit trägt mehrfach zu diesem Bereich bei.

Der erste Teil dieser Arbeit befasst sich mit Seitenkanalangriffen und Gegenmaßnahmen für zwei Post-Quantum-Kryptografie-Algorithmen. Diese Algorithmen basieren auf anderen mathematischen Problemen als die der etablierten asymmetrischen Kryptosysteme und gelten auch in Gegenwart eines Quantencomputers von ausreichender Größe als sicher. Die Algorithmen wurden im Rahmen eines Standardisierungsverfahrens des National Institute of Standards and Technology (NIST) entwickelt, um im Falle eines plötzlichen Durchbruchs in der Quantencomputerforschung Lösungen bereitstellen zu können, da die Umstellung der kryptografischen Infrastruktur sehr zeitintensiv ist. Zunächst werden Maskierungsgegenmaßnahmen für das Verfahren NTRUEncrypt vorgestellt, das nach einer Fusion mit einem ähnlichen Kryptosystem ein vielversprechender Kandidat für die Standardisierung war. Außerdem wird eine umfassende Seitenkanalstudie des HQC Kryptosystems gezeigt. Dieses System ist ein Kandidat für die Standardisierung und basiert auf fehlerkorrigierenden Codes. Die Studie umfasst die Entwicklung und praktische Evaluierung von Chosen-Ciphertext-Angriffen für die ursprüngliche Version von HQC sowie dessen Weiterentwicklung mit einer verbesserten Fehlerkorrektur. Abschließend werden verschiedene Gegenmaßnahmen gegen die präsentierten Angriffe evaluiert.

Im zweiten Teil dieser Arbeit wird der vielversprechende Angriffsvektor der Nutzung von tiefen neuronalen Netzwerken für die Seitenkanalanalyse diskutiert. Dieser Bereich der Seitenkanalanalyse erregt großes Forschungsinteresse, da gezeigt wurde, dass Angriffe in der Lage sind Gegenmaßnahmen zu umgehen und gleichzeitig den Aufwand für die Vorverarbeitung von Messungen erheblich zu reduzieren, da z.B. Seitenkanalinformationen, die für einen erfolgreichen Angriff kombiniert werden müssen, automatisch identifiziert werden. Dies führt jedoch zu einer verminderten Erklärbarkeit von Angriffsergebnissen, was aber für die Identifikation von Schwachstellen einer geschützten Implementierung von entscheidender Bedeutung ist. Diese Arbeit zeigt Verbesserungen von Erklärungsmethoden für diese Angriffe, um diesen Nachteil abzuschwächen. Als Grundlage für die Erforschung von Erklärbarkeitsmethoden werden Ergebnisse einer umfassenden Auswertung der Standard-Forschungsdatenbank ASCAD mit klassischer Seitenkanalanalyse und ein Vergleich der unterschiedlichen Datenbankversionen gezeigt. Die Ergebnisse dieser Untersuchungen zeigen unerwartete Seitenkanalinformationen für eine geschützte Implementierung und unterschiedliche Seitenkanalinformationen zwischen den angegriffenen Teilen des verwendeten Schlüssels. Auf der Grundlage dieser Beobachtungen werden mehrere Weiterentwicklungen der Occlusion-Erklärungsmethode vorgestellt. Dazu gehören die Bestimmung geeigneter Parameter der Methode für eine Evaluation von ASCAD sowie eine Methode zur Ermittlung der für einen erfolgreichen Angriff erforderlichen Kombination an Seitenkanalinformation.

Contents

List of Figures

List of Tables

1 Introduction

With the rise of the Internet of Things (IoT) in recent years, there are more and more connected devices, including smart devices, intelligent factories in the industry 4.0, and cars. This leads to a total of 15 billion connected devices in 2023, which is estimated to almost double until 2030[1]. In order to secure the communication of these devices, cryptographic algorithms and protocols are used to achieve the desired security goals like confidentiality, authenticity, and integrity. Fortunately, the mathematical security of algorithms, i.e., their resistance against cryptanalysis, is well understood and therefore established algorithms are considered secure given a sufficient length of the used secret key. Nevertheless, as an attacker can possess or obtain physical access to a device, it has to be secured against physical attack attempts. A prominent attack vector are side-channel attacks that exploit a so-called *side-channel* of a device in order to extract information about the used secret with the goal of completely retrieving it. Examples of side-channels are the power consumption of the device, electromagnetic emanations, and timing information of the executed algorithm. Possible countermeasures against these attacks are based on either making the power consumption independent of the used secret (masking) or increasing the noise of measurements due to a randomized execution of the targeted algorithm (hiding). Although countermeasures are known, side-channel attacks are still a threat to the information security of embedded devices as advanced attacks are developed and required equipment becomes affordable. Additionally, the implementation of countermeasures is error prone such that countermeasures are implemented insufficiently (see, e.g., [RLMI21, SWUH21]). Therefore, research on Side-Channel Analysis (SCA) is crucial in order to a) develop new attack methodologies to ultimately identify countermeasures against them and b) evaluate (novel) algorithms for possible attack vectors such that resource-efficient countermeasures can be developed.

One major field of research is the SCA evaluation of algorithms in the context of the post-quantum cryptography standardization competition by the National Institute of Standards and Technology (NIST) [Nat16], which aims at standardizing novel public-key cryptosystems based on mathematical problems that are considered secure in the presence of a large-scale quantum computer. For these algorithms, a thorough side-channel evaluation is especially important as, due to the alternative mathematical problems, novel constructions of cryptosystems have been developed with building blocks that differ significantly from established systems and have therefore not seen much scrutiny by the SCA community. The competition was initiated as, with the publication of Shor's algorithm [Sho97], the underlying problems of established public key cryptography, such as the Rivest-Shamir-Adleman (RSA) cryptosystem or Elliptic-Curve Cryptography (ECC) can be solved in polynomial time given a quantum computer of sufficient size to attack typical key sizes. Although currently available quantum computers are only capable of solving small instances of, e.g., integer factorization, and therefore, substantial effort is required in order to scale a quantum computer up to a cryptographically relevant size, large public and private investments led to noticeable advancements in quantum computing [SWM$^+$24]. Besides the fact that experts consider the near-future risk of a large-scale quantum computer quite low[2] there are two main reasons why a timely migration to post-quantum cryptography is required: a) sensitive information can be stored now with the goal of decrypting it in the future when a large-scale quantum computer is available; b) adapting the cryptographic infrastructure usually takes a large amount of time. Consequently, the need for a fast development of post-quantum secure

[1] `https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/`, last accessed 17th May 2024

[2] An estimation through the extrapolation of the current quantum computing hardware progress suggests that RSA-2048 will unlikely ($< 5\,\%$ confidence) be factorized before 2039 [SR20]. See [SWM$^+$24, Section 4.2] for a more detailed discussion of this estimate.

algorithms has also been identified by NIST, which started its competition in 2016. A timeline of the competition is shown in Fig. 1.1.

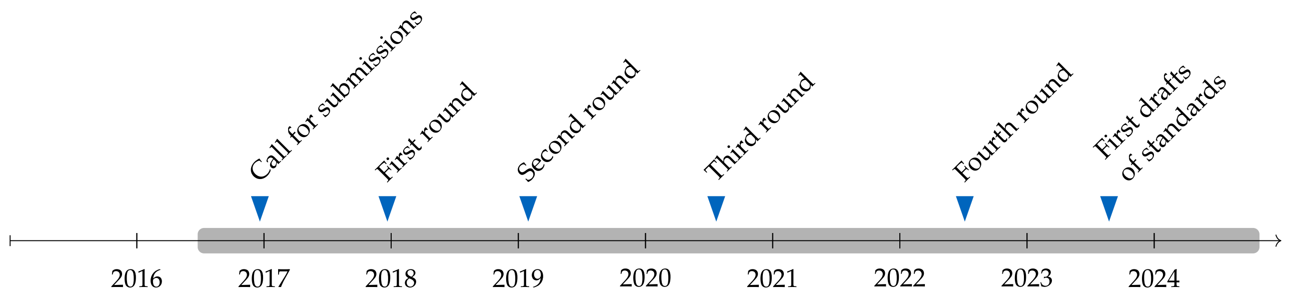


Figure 1.1 Timeline of the NIST post-quantum competition.

For the first round of the competition, a total of 69 submissions were accepted, and the selection of algorithms has been refined by NIST for each additional round based on selected criteria obtained from contributions of the scientific community. These include, e.g., the achievable performance and resource efficiency of implementations and identified side-channel attacks in combination with the required resources to achieve a side-channel secured implementation. Especially the need for research on SCA was encouraged by NIST within their status report of the second round [Nat20], which states "NIST hopes that [...] the public review period will include more work on side-channel resistant implementations [...]", as the first two rounds were mainly dominated by research on cryptanalysis and efficient implementations. After the third round, four algorithms were selected for standardization and the initial public drafts of the corresponding standards were published in August 2023. The selected algorithms are the Key Encapsulation Mechanism (KEM) CRYSTALS-Kyber [ABD+20] and the three digital signature algorithms CRYSTALS-Dilithium [BDK+20], Falcon [FHK+20] and SPHINCS+ [ABB+20b]. Most of the selected algorithms are based on hard problems over lattices, as NIST concluded that with these types of systems the best performance in combination with the smallest parameter sizes can be achieved. In order to not rely on the security of a single mathematical problem, NIST advanced four[3] additional KEMs to the fourth round with the goal of standardizing one of them as an alternative. The remaining selected algorithms are Bit Flipping Key Encapsulation (BIKE) [ABB+20a], Classic McEliece [ABC+20], and Hamming Quasi-Cyclic (HQC) [MAB+20] that are systems based on error-correcting codes.

A second field of research that received a lot of attention is the usage of machine learning methods for SCA, especially the usage of methods from deep learning (DL), i.e., deep neural networks. After the first publication of DL-based SCA in 2016 by Maghrebi et al. [MPP16], there were two events that sparked the research interest in this field. First, researchers of the French cybersecurity agency ANSSI published an opensource database called ANSSI SCA Database (ASCAD) [BPS+18] that includes measurements and the required metadata for an attack on a protected version of the Advanced Encryption Standard (AES). Within their corresponding paper [PSB+18], they perform an in-depth evaluation of different neural network architectures that can be used to successfully attack the database and provide example code of their architecture with the best attack performance in order to enable further research. Second, attention to this research field was additionally drawn with the CHES 2018 challenge called "Deep learning vs. classic profiling"[4] where multiple datasets for an attack on block ciphers as well as the RSA cryptosystem were provided. This led to a total of 183 publications until 2021, as noted by a recent systemization of knowledge paper on the field [PPM+23]. Within

[3] An example why an alternative candidate could be needed is the complete break of the isogeny-based SIKE cryptosystem [ACC+20] in [CD23], which was found after SIKE was chosen to advance as a fourth-round alternative candidate.

[4] `https://web.archive.org/web/20230103214718/https://chesctf.riscure.com/2018/news`, last accessed 19th May 2024

this paper, the authors identify the following main advantages in contrast to classical side-channel approaches. DL-based SCA is able to automatically break the implemented countermeasures of protected implementations with the benefit of requiring no or a significantly reduced amount of preprocessing of the side-channel measurements. This additionally makes these methods suitable for attacks on public-key cryptography, including post-quantum cryptography, as these algorithms show a largely increased execution time in contrast to symmetric algorithms like block ciphers. With neural networks, exploitable side-channel information can automatically be identified within the resulting large timeframes of measurements. For classical SCA, measurements usually have to be pre-processed in order to achieve a feasible computation complexity with the use of expert knowledge. This highlights the potential of DL-based SCA for an automated end-to-end analysis of the side-channel security of protected implementations. A first step in this direction is shown in [BIK+24], where a single network architecture is used to attack multiple implementations and types of cryptosystems. Nevertheless, besides the presented advantages, there is a major downside which lies in the reduced explainability of attack results. As the important side-channel information is automatically extracted from measurements, the respective points in time and, ultimately, the corresponding operations of an implementation that are used for a successful attack can hardly be identified. Unfortunately, this information is required by a designer of a cryptographic implementation in order to identify weaknesses of the implementation and to strengthen applied countermeasures.

Contribution and Outline This thesis provides multiple contributions in the field of SCA that can be grouped into two parts. In the first part, the development and evaluation of attacks and countermeasures for two submissions of the NIST post-quantum cryptography competition are presented, where the underlying publications have been considered as contributions to the NIST standardization process. The second part provides contributions in the field of explainability of DL-based SCA.

To begin, Chapter 2 gives an overview of the field of side-channel analysis by introducing methods to identify side-channel leakage in measurements, the different types of side-channel attacks, and possible countermeasures.

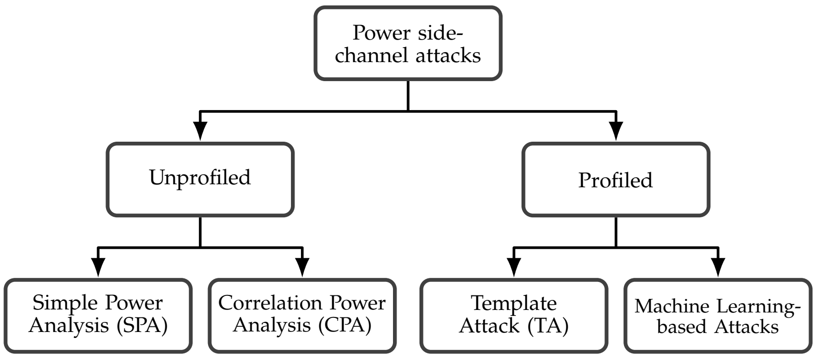
Part I begins with the evaluation of a masking countermeasure for an early version of NTRUEncrypt [CHWZ17] in Chapter 3, which has been a final candidate for the third round of the NIST competition as NTRU [CDH+19] after a merger with a similar submission. Provided practical attack results demonstrate the security of developed protected implementations, which show a low performance overhead and can additionally be combined with an efficient hiding countermeasure. This work can be considered one of the first works on countermeasures for software implementations of post-quantum cryptography. This is followed by a side-channel analysis of the code-based HQC cryptosystem in Chapter 4, which is still among the remaining three candidates that can be chosen by NIST as an alternative standard. After an introduction to error-correcting codes and the HQC algorithm, the first chosen-ciphertext power side-channel attack on the cryptosystem is presented. This attack works by observing the decoding results of the used error correction through the power side-channel for multiple ciphertexts. These are chosen by the attacker such that the decoding results depend on an individual small part of the secret key, allowing the retrieval of the complete secret with a sufficient amount of observations. Further, a novel attack strategy is shown for an updated version of the cryptosystem that employs an improved error correction chosen by the HQC authors in order to reduce parameter sizes. Finally, the chapter is concluded by an evaluation of multiple countermeasures against the proposed attacks that indicate that a complete masking scheme for HQC has to be developed in order to provide a protected implementation.

Part II begins with Chapter 5, which provides an introduction to DL-based SCA followed by an presentation of the different versions of the ASCAD database in combination with the underlying AES implementation and used neural network architectures. Chapter 6 provides a comprehensive evaluation of the ASCAD database with classical SCA as we argue that research on explainability of DL-based SCA requires a throughout understanding of the leakage characteristics of the used dataset. The presented findings show the presence of additional leakage of intermediates in the selected trace segment of the databases as well as significantly different leakage characteristics dependent on the targeted key byte. Further, attack results through classical SCA revealed flaws in the underlying masked implementation that should be considered for an interpretation of DL-based attack results. With the knowledge of the different leakage characteristics, attack results for all key bytes are shown. These indicate significant attack performance differences between key bytes and that training and attacking on different bytes poses challenges to the evaluated architectures. Finally, we show that the first version of ASCAD provides an easier attack target than the updated version of the database. In Chapter 7, we provide multiple advancements to the occlusion explainability method building on the leakage evaluation of Chapter 6. Occlusion can be considered a direct explainability method as it works by occluding parts of the used side-channel measurements and observing changes in the resulting attack result. If the attack performance decreases, it can be concluded that the used neural network has learned to extract information from the occluded part, while a comparison with a leakage evaluation allows to reason about learned intermediate values of the implementation. We propose the occlusion of multiple adjacent points in time and introduce higher-order occlusion, which allows to identify sample combinations that are used by a network in order to perform a successful attack. Additionally, suitable parameters of the developed methods for evaluating ASCAD are determined. Finally, the developed higher-order occlusion is used to show that evaluated neural network architectures actually utilize the identified different leakage characteristics of key bytes.

Chapter 8 concludes this thesis and discusses possible topics for future work.

2 Side-Channel Analysis

The security of a cryptographic algorithm is usually determined by an evaluation of the best-known attack against its underlying mathematical problem. Nevertheless, during the execution of an algorithm, an attacker is able to obtain additional information or characteristics from the device that executes the algorithm, which can be used to mount a so-called side-channel attack to derive information about internal secrets, such as the secret key of a KEM. It is essential to understand that side-channel attacks do not exploit cryptographic weaknesses in a system but are specific to a particular implementation of an algorithm. Possible side-channels include the power consumption [KJJ99], which can also be obtained through electromagnetic emanations, and the timing behavior [Koc96]. This thesis discusses power side-channel attacks that are based on measurements of the power consumption of an implementation exploiting the data-dependent power consumption of Complementary Metal-Oxide-Semiconductor (CMOS) logic [MOP07]. An overview of different power side-channel attacks is given in Fig. 2.1.



Power side-channel attacks
Unprofiled
Profiled
Simple Power Analysis (SPA)
Correlation Power Analysis (CPA)
Template Attack (TA)
Machine Learning-based Attacks

Figure 2.1 Overview of different types of side-channel attacks.

This section gives an overview of the different types of attacks as well as an introduction to possible countermeasures. In Section 2.1, different methods for leakage evaluation are presented, which are used to detect exploitable information in side-channel measurements. This is followed by an introduction to unprofiled side-channel attacks in Section 2.2, which either directly or through the use of statistical methods in combination with a generic model of the power consumption allow an attack on the used secret. In contrast, Section 2.3 introduces profiled side-channel attacks. These attacks assume an identical device with a known secret that can be used to build an exact and tailored power model of attacked intermediates. Finally, countermeasures are described in Section 2.4.

2.1 Leakage Evaluation

For practical side-channel attacks, it is often necessary to detect exploitable information (leakage) in side-channel measurements (traces). This implies to determine the points in time with the corresponding measurement samples where the highest amount of leakage can be obtained for a targeted intermediate value or operation. This process is also called point of interest (POI) detection. There are multiple reasons for performing leakage evaluation:

- It can be used as a preprocessing step prior to the actual attack to reduce the length of a trace, thereby imposing a limit on the data to be processed and reducing the required computational resources. This is especially important for profiled attacks since the amount of considered samples significantly increases the computational complexity.

- In addition to the reduced computational complexity, it allows removing samples that do not contain exploitable leakage and, therefore, act as noise for an attack.
- It can be used to infer details about a targeted implementation, e.g., samples that have to be combined in order to break protected implementations can be identified.

It can be distinguished between methods that can be used in a black-box scenario, i.e., no information about targeted intermediates is required, like the Welch's t-test [GGJR11, SM15] and methods that require knowledge about targeted intermediates[1] like the signal-to-noise ratio (SNR) [MOP07] or the Correlation Point of Interest (CPOI) [DS16].

T-Test Welch's t-test [GGJR11, SM15] can be used to statistically evaluate the presence of side-channel leakage without prior knowledge about the investigated implementation and used intermediates. It essentially evaluates whether two sets of data significantly differ from each other in the sense that their means are different. More formally, Welch's t-test gives the probability at which the null hypothesis, i.e., both sets were drawn from the same distribution, can be rejected. Given two sets $\mathcal{S}_0$ and $\mathcal{S}_1$ with their respective mean μ_0, μ_1 and variance s_0, s_1 the resulting *t-value* is calculated as

$$t = \frac{(\mu_0 - \mu_1)}{(\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}})} \, , \tag{2.1}$$

where n denotes the respective cardinality of the set. Usually, a threshold of $|t| > 4.5$ is defined, which states that there is a confidence of > 0.99999 that both sets can not be distinguished if the resulting t-value stays below this threshold. As the t-test can be individually performed for all the samples of a measurement trace, it acts as an efficient method for POI detection.

For the actual leakage evaluation, the non-specific or fixed-vs.-random t-test can be applied. It requires to measure the power consumption of multiple algorithm executions with a fixed secret while randomly choosing the input data of subsequent measurements to have a fixed or random value. Measurements are then split into $\mathcal{S}_0$ with fixed input data and $\mathcal{S}_1$ with random input data, which are then used to calculate the t-value according to Eq. (2.1). All samples that show a t-value above the threshold are identified as the resulting POIs. This shows a downside of the t-test, as an attacker does not know which POI actually corresponds to the intermediate value she would like to target, but rather all samples that exhibit side-channel leakage are revealed. Nevertheless, this makes the t-test a popular tool for the evaluation of masked implementations as resulting t-values below the threshold indicate that an implementation can be considered secure against attacks employing the used amount of traces.

Signal-to-noise Ratio (SNR) The Signal-to-Noise Ratio (SNR) as defined in the side-channel context by Mangard et al. [MOP07, Section 4.3.2] can be used as a metric for leakage evaluation. It is based on a model of the leakage in form of the power consumption of a device as

$$L = L_{data} + L_{noise} \, , \tag{2.2}$$

where L_{data} is the exploitable data dependent leakage and L_{noise} is the noise. L_{noise} includes the electronic noise, which consists of the noise of the executing device as well as noise that is induced through the measurement setup and leakage that does not correspond to the targeted intermediate.

[1]Usually the knowledge of the used secret is required in order to calculate the values of targeted intermediates.

It is modeled as a normal distribution with a zero mean and variance σ^2 as $L_{noise} \sim \mathcal{N}(0, \sigma)$ [MOP07]. The SNR is then defined as

$$SNR = \frac{Var(L_{data})}{Var(L_{noise})}, \tag{2.3}$$

which is the ratio between the variance of both parts of the leakage. In practice, side-channel measurements only contain information about L and the goal is to estimate both parts of the leakage such that the SNR can be computed.

In the following, the computation of the SNR is described according to [PGA⁺23]. First, an intermediate value v has to be selected for which the leakage is estimated using the SNR. An example is the S-box input of a block cipher like the AES, which is an 8-bit value and therefore there are 256 different values for this intermediate. Then all traces are grouped into different sets $\mathcal{L}_0, \ldots, \mathcal{L}_{255}$ according to the intermediate value, such that $\mathcal{L}_i$ contains all traces with $i = v$. For the individual sets $\mathcal{L}_i$, the corresponding estimation of the statistical mean μ_i and variance σ_i^2 can be computed. The SNR is then calculated as

$$SNR = \frac{Var(\mu_0, \ldots, \mu_{255})}{Mean(\sigma_0^2, \ldots, \sigma_{255}^2)}. \tag{2.4}$$

Correlation-based Leakage Test (CPOI) Durvaux and Standaert [DS16] propose a correlation-based leakage test that is referred to as CPOI in this thesis. It is based on a profiled instead of an abstract leakage model like the Hamming weight (HW) or Hamming distance (HD). This profiled leakage model is generated from the evaluated traces by first grouping each trace according to its value of the targeted intermediate. For each of these groups the mean for each sample of all traces in the group is computed, which leads to a mean trace vector for all possible values of the intermediate. The profiled power model consists of these individual mean trace vectors. Then, the CPOI result is computed as the correlation of the targeted intermediate for each sample. The process differs from a Correlation Power Analysis (CPA) [BCO04] in the sense that for the calculation of the correlation the corresponding mean trace, according to the value of the intermediate for each individual trace of the computation, is used as the hypothetical power consumption. Finally, this process is repeated k times, i.e., k-fold cross-validation is used, in order to obtain unbiased results. The presence of leakage and its corresponding samples is indicated by a significant correlation.

More formally, the method is described as follows. First the available traces $\mathcal{T}$ are split into k disjoint sets $\mathcal{T}^{(i)}$ of equal size for $i = 0, \ldots, k-1$. Then one of the sets is chosen as the test set $\mathcal{T}_t^{(i)}$ and the remaining $k-1$ sets are used as the profiling set $\mathcal{T}_p^{(i)} = \mathcal{T} \setminus \mathcal{T}_t^{(i)}$. The profiling set is used to estimate a separate model for all N_c possible values of the targeted intermediate value v. The different models consist of the sample mean traces $\boldsymbol{\mu}_c^{(i)}$ with $c = 0, \ldots, N_c - 1$, where only traces of $\mathcal{T}_p^{(i)}$ with a corresponding intermediate value ($c = v$) are considered for the calculation of the respective mean trace. Then the correlation of the test set $\mathcal{T}_t^{(i)}$ with the profiled power model can be calculated through the use of the Pearson correlation. For this calculation, each trace is correlated with the power model $\boldsymbol{\mu}_c^{(i)}$ of the corresponding intermediate value. The correlation coefficient for all samples of $\mathcal{T}_t^{(i)}$ is given as

$$r^{(i)} = \frac{\sum_{j=0}^{|\mathcal{T}_t^{(i)}|} (\mathcal{T}_t^{(i)}[j] - \overline{\mathcal{T}_t^{(i)}})(\boldsymbol{\mu}_{v[j]}^{(i)} - \overline{\boldsymbol{\mu}^{(i)}})}{\sqrt{\sum_{j=0}^{|\mathcal{T}_t^{(i)}|} (\mathcal{T}_t^{(i)}[j] - \overline{\mathcal{T}_t^{(i)}})^2} \sqrt{\sum_{j=0}^{|\mathcal{T}_t^{(i)}|} (\boldsymbol{\mu}_{v[j]}^{(i)} - \overline{\boldsymbol{\mu}^{(i)}})^2}}, \tag{2.5}$$

where $\overline{\mathcal{T}_t^{(i)}}$ corresponds to the mean of all traces in the test set, $\overline{\boldsymbol{\mu}^{(i)}}$ to the mean of all models, and $v[j]$ to the intermediate value corresponding to the trace $\mathcal{T}_t^{(i)}[j]$. The process is repeated for all i until all k cross-validation sets are chosen as the test set and their respective correlation is calculated. Finally, all correlation results $r^{(i)}$ for $i = 0, \ldots, k-1$ are averaged to get an unbiased result.

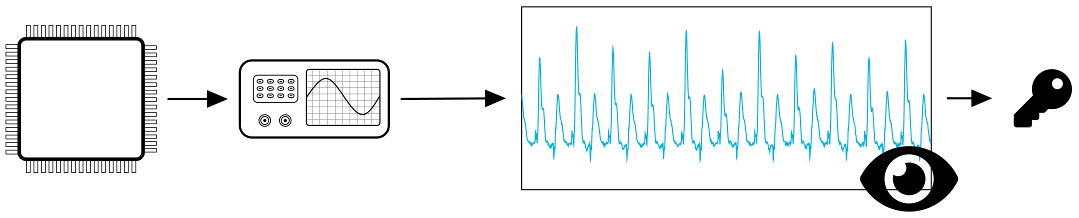


Figure 2.2 Simple Power Analysis (SPA) attack.

2.2 Unprofiled Side-Channel Attacks

The unprofiled side-channel attacks can be distinguished between Simple Power Analysis (SPA) attacks and Differential Power Analysis (DPA) attacks. In contrast to profiled attacks (c.f. Section 2.3), a hypothetical model of the side-channel leakage is used for the attack.

SPA attacks are considered *simple* as they use a single or few traces with the same input to directly infer the secret, e.g., through visual inspection [MOP07]. A prominent example is the SPA against the Square-and-Multiply algorithm used for exponentiation in the RSA cryptosystem. Using SPA on a power trace of this exponentiation allows to distinguish the characteristic power consumption of the different operations (square or multiply), which directly reveals the used secret key [KJJ99]. A visualization of a SPA attack is shown in Fig. 2.2.

In contrast, DPA attacks exploit multiple traces in order to attack a secret by comparing power measurements with a hypothetical model of the power consumption through a statistical distinguisher. The attacks use a divide-and-conquer strategy, which means that they individually target small parts of the secret and reveal the complete secret through multiple successful attacks. For the attack, an intermediate value or a function of the algorithm has to be chosen that depends on known input or output and a (small) part of the secret. Then, for each trace, a hypothetical power consumption is calculated based on all possible hypotheses for the secret using a specific power model like the Hamming weight (HW) or the Hamming distance (HD). The HW is typically used for software implementations that are running on a microcontroller, while for hardware implementations the HD is most common as it reflects the fact that the power consumption of CMOS logic is mostly dominated by the dynamic power consumption, i.e., transitions from $1 \rightarrow 0$ and $0 \rightarrow 1$ [MOP07]. Finally, all available traces are compared with their respective hypothetical power consumption through the statistical distinguisher. The hypothesis that corresponds to the correct secret is revealed as only for this comparison there exsists a statistical dependency, which is indicated by the distinguisher. In practice, due to noise in the measurements or an inaccurate power model, multiple traces have to be used until a significant dependency is revealed. The original DPA of Kocher et al. [KJJ99] uses the difference of means as a distinguisher. The most common form of DPA is the so-called CPA [BCO04] that uses the Pearson correlation coefficient in order to determine the linear correlation between actual and hypothetical power consumption. Mutual Information Analysis (MIA) [BGP$^+$10] utilizes mutual information as a generic distinguisher that is able to capture any type of dependency. The principle of a DPA attack is shown in Fig. 2.3.

2.3 Profiled Side-Channel Attacks

For profiled side-channel attacks, the attacker is able to characterize the leakage of a target device. This requires first choosing a target intermediate value v during the execution that is defined by a function $v = f(x, k)$ based on an observable input x and the secret k. There are N_c possible values for the intermediate v, which is therefore in the range $[0, N_c - 1]$. The actual attack is divided into a *profiling phase* and an *attack phase* visualized in Fig. 2.4.

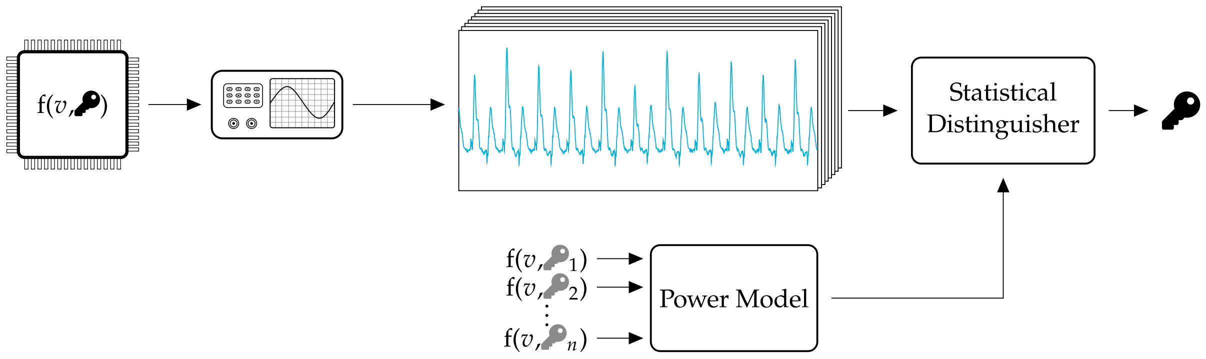


Figure 2.3 Differential Power Analysis (DPA) attack.

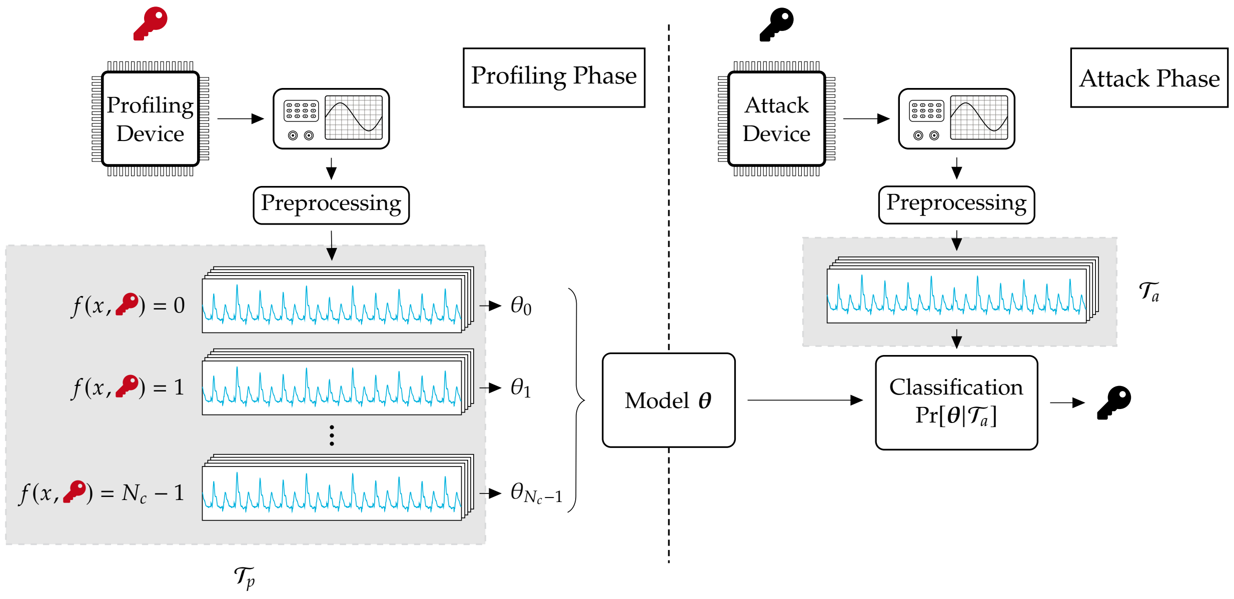


Figure 2.4 Visualization of the two stages of a profiled side-channel attack.

For the profiling phase, an attacker has complete control over a profiling device. This allows modeling the leakage of the targeted intermediate using a set of profiling traces $\mathcal{T}_p$ with known inputs and secret. Using the profiling traces, the attacker computes a model of the leakage $\boldsymbol{\theta}$ for all possible N_c values of v, resulting in the individual models θ_i with $i = 0, \ldots, N_c - 1$ for each value. In the attack phase, the attacker captures the attack traces $\mathcal{T}_a$ of an identical device with an unknown secret. Then, each trace $\boldsymbol{t} \in \mathcal{T}_a$ is compared to all models θ_i, and the resulting classification results are aggregated for all traces. This aggregation requires computing key hypotheses $\hat{k}_i$ for $i = 0, \ldots, N_c - 1$ through reverting the targeted function using the knowledge of the input x and the intermediate value v associated with the model θ_i. Then, for each trace, the corresponding classification results for the different $\hat{k}_i$ can be aggregated. Finally, the key hypothesis with the best classification result is estimated as the correct value of the targeted secret. There are multiple aspects of profiled side-channel attacks that have to be considered:

- **Preprocessing:** Captured traces can be optionally preprocessed in order to reduce computational complexity and include only samples of a trace that are considered beneficial for the classification. This can be achieved using leakage evaluation techniques introduced in Section 2.1 to identify samples with a high estimated leakage. Additionally, dimensionality reduction techniques like Principal Component Analysis (PCA) [APSQ06] or Linear Discriminant Analysis (LDA) [SA08, CK14] can be used to compress traces by transforming them into a lower-dimensional subspace.

- **Key Enumeration:** There is the possibility that the best classification result does not indicate the correct secret due to an insufficient amount of profiling or attack traces as well as low SNR measurements. Additionally, due to a limit of the available computational resources as well as a prohibitive measurement time during the profiling phase, the secret has to be split into multiple parts that are attacked individually. For example, a 128-bit AES key can be split into 16 8-bit subkeys that are individually targeted. It is, therefore, required to define a metric of the attack success on each subkey individually with the goal of estimating the final enumeration effort that is required to retrieve the complete key. Popular metrics include the *key rank* [PGA+23] that is defined as the position of the correct key in a ranked list of the final classification results, and the *guessing entropy* [SMY09], which is defined as the average key rank for multiple attacks. The enumeration effort in order to retrieve a complete key and, therefore, the remaining security after an attack can be estimated using so-called *key rank estimation* algorithms. A popular algorithm based on the convolution of histograms is presented by Glowacz et al. [GGP+15].

The classical profiled side-channel attack is the template attack by Chari et al. [CRR03]. It uses the fact that side-channel leakage can be modeled as a multivariate Gaussian distribution that is sufficiently described using a mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. Therefore, during the profiling phase, the individual models, which are also called templates, are estimated as $\theta_i = (\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ for all values of the intermediate v. Then, the probability density function of the distribution is defined as

$$p(\boldsymbol{t}|v) = \frac{1}{\sqrt{(2^{\pi})^{n_s}|\boldsymbol{\Sigma}_v|}} \cdot \exp(-\frac{1}{2}(\boldsymbol{t} - \boldsymbol{\mu}_v)^T \boldsymbol{\Sigma}_v^{-1}(\boldsymbol{t} - \boldsymbol{\mu}_v)) , \quad (2.6)$$

where n_s is the number of samples in a trace or the dimension of the trace if dimensionality reduction is used. In the attack phase, the classification of an attack trace, also called template matching, is done using Bayes' rule to compute the posterior probability $p(v|\boldsymbol{t})$. Since the attacker is only interested in the relative ranking of all classification results, the correct probabilities are not computed directly, but only a discriminant function is used to provide the correct ranking. In practice, the log-likelihood is used for this purpose [CK14]. Additionally, multiple optimizations for template attacks increase efficiency and robustness. An example is the use of a pooled covariance matrix for all templates with the assumption that only the noise characterizes the covariance, which is independent of the signal, and therefore the covariances for all templates are equal [CK14, CK18].

In contrast to the classical template attack, the classification task of profiled side-channel attacks can also be performed using machine learning techniques[2]. This includes unsupervised methods like k-means clustering [HIM+14] and supervised methods like a Support Vector Machine (SVM) [LBM11], a random forest (RF) [LBM11] and linear regression [SLP05]. The usage of deep learning (DL) methods like a Multi Layer Perceptron (MLP) or a Convolutional Neural Network (CNN) was first proposed by Maghrebi et al. [MPP16]. A detailed introduction to DL-based methods for side-channel attacks is given in Section 5.1.

2.4 Countermeasures

Countermeasures against side-channel attacks can be mainly grouped into two categories. The first category are *hiding* countermeasures that aim to reduce the amount of exploitable side-channel information on either the physical, logic, or algorithmic level. In contrast, the second category, in the form of so-called *masking* countermeasures, aims at splitting sensitive variables into multiple random shares in order to make them independent from processed secrets.

Hiding Countermeasures The goal of these countermeasures is to lower the achievable SNR of side-channel measurements. This can be done either in the amplitude domain through increasing the noise or in the time domain. Hiding countermeasures in the amplitude domain are employed on the physical or logical level. This includes shielding in order to lower electromagnetic emissions of a device [QS01], dedicated hardware that exhibits additional noise [MOP07], or the usage of dedicated logic styles that try to balance the power consumption to be independent of processed data, e.g., *dual-rail logic* [MSS09]. In the time domain, hiding countermeasures induce misalignment between measurements by inserting a random amount of *dummy operations* or by *shuffling*, which refers to the random execution of independent operations within an algorithm [VCMKS12]. The effect of hiding countermeasures can be reduced through a preprocessing of measurements, e.g., filtering of noise and integration of samples, or through re-alignment strategies like pattern matching during the measurement phase or advanced techniques like *elastic alignment* [vWWB11]. However, since hiding countermeasures only achieve a linear increase in noise, they are typically used in combination with masking.

Masking The masking countermeasure was proposed by Chari et al. [CJRR99] as well as independently by Goubin and Patarin [GP99]. It works by representing each sensitive variable of a cryptographic algorithm as a combination of multiple $(d + 1)$ shares that are uniformly distributed and, therefore, independent of the used secret. Only the knowledge of all shares allows to reconstruct the unmasked variable, e.g., at the output of an algorithm. The goal of a masked implementation is to compute a shared version of the correct output while keeping the uniformity of all shares throughout the complete execution without allowing a (partial) recombination of shares. For linear operations, this can be achieved by performing the operations of the algorithm individually for each share, but non-linear operations such as the AES S-box computation require additional care, as they typically need to be computed using multiple shares. A d^{th}-order masking scheme defines each sensitive variable x using $d + 1$ shares that statisfy the equation

$$x_1 \circ x_2 \circ \ldots \circ x_{d+1} = x \ , \tag{2.7}$$

where $\circ$ defines the group operation and usually x_i for $i = 1, \ldots, d$ is generated uniformly random and x_{d+1} is computed such that the equation holds. For *boolean* masking the bitwise XOR operation $\oplus$ is used as a group operation, and for *arithmetic* masking modular addition $+$ in the ring $\mathbb{Z}_n$ is

[2]The list of methods is not exhaustive and provided references consist of the first publication of the respective method in the side-channel context. For a detailed overview, the reader is referred to [HGG19].

used. The group operation has to be chosen for a given algorithm such that the highest amount of operations is linear under the chosen group.

In order to break a d^{th}-order masking scheme, an attacker has to estimate at least the $(d + 1)^{\text{th}}$ statistical moment, which requires to combine the leakage from all $d + 1$ shares. For univariate attacks, the computation of all shares is assumed to be performed in parallel, which is usually the case for masked hardware implementations, and therefore the higher-order statistical moments can be computed directly from the traces. Multivariate attacks, where operations on the individual shares are assumed to be performed at different points in time, require a preprocessing step where samples corresponding to leakages of the shares are combined. In [PRB09, SVCO+10], the mean-free product of samples is identified as a suitable combination function, which is additionally proven to be optimal for second-order attacks under the assumption of HW leakage. The effort of identifying corresponding samples in the measurements contributes to the security gain of a masking scheme, as the amount of sample combinations in the preprocessing step is limited by the available computational resources of an attacker. Nevertheless, the largest part of the security gain stems from the fact that the estimation of higher-order moments is significantly more difficult, i.e., significantly more traces are needed for a sufficient estimation. This is additionally enhanced by the *noise amplification* property of a masking scheme that exponentially increases the noise with the masking order d [CJRR99].

The security of a masking scheme can be formally proven under the assumption of a specific adversary model. The most prominent model is the d-probing model by Ishai et al. [ISW03], which allows an attacker to gain exact knowledge of at most d shares of an implementation. In [DDF14], it is shown that the security in the probing model can be reduced to the noisy leakage model, which better suits side-channel attack results, i.e., is more realistic, as it is based on the assumption that an adversary has access to the noisy leakage of shares. Finally, it has been shown by Barthe et al. [BBD+16] how to securely combine multiple small parts of an algorithm, which are also called gadgets, into a complete masked implementation. For hardware implementations, it is important to consider the possibility of glitches, which is reflected in the robust probing model by Faust et al. [FGMDP+18]. It can be used to prove the security of dedicated masking schemes for hardware implementations like Domain-Oriented Masking (DOM) [GMK16] or threshold implementations [NRR06].

Part I

Attacking and Securing Post-Quantum Cryptography

3 Practical Evaluation of Masking for NTRUEncrypt on ARM Cortex-M4

This chapter is based on the publication *Schamberger/Mischke/Sepulveda: Practical Evaluation of Masking for NTRUEncrypt on ARM Cortex-M4* published in *Constructive Side-Channel Analysis and Secure Design, 2019* [SMS19]. The attack on the implementation of NTRUEncrypt with ternary polynomials has been developed during my master thesis [Sch17]. In order to discuss countermeasures against this attack, it is again described in Section 3.3. Additionally, attack results of the unsecured implementation from [Sch17] are shown and discussed in Section 3.5.1. In summary, the algorithms for the index-based multiplications (Algorithms 4 and 5) as well as Figs. 3.2 and 3.3 are adapted from [Sch17].

3.1 Introduction

The lattice-based cryptosystem NTRUEncrypt was a promising candidate in the NIST post-quantum cryptography competition due to the maturity of the underlying mathematical problem, which withstood the scrutiny of the scientific community for over 20 years since its first publication in 1998. In addition, the early publication of the original system minimizes the threat of patents on the system, which would hinder its adoption after standardization by NIST. Implementations of NTRUEncrypt targeting resource-constrained devices like microcontrollers typically utilized index-based multiplication, which has been shown to be vulnerable to CPA. Although countermeasures against this attack have been proposed, an evaluation of masking was lacking at the time this research was performed. In order to fill this gap, this chapter presents a practical evaluation of masking for modern parameter sets of NTRUEncrypt on an ARM Cortex-M4 microcontroller.

Related Works In 2008, the first power side-channel attack on NTRUEncrypt was published by Atıcı et al. [ABGV08]. Lee et al. [LSCH10] showed a CPA that is able to retrieve the used private key polynomial for implementations utilizing index-based multiplication for binary polynomials. In addition to their attack, they introduce three countermeasures: blinding the result array, shuffling operations during multiplication, and masking. It has been shown how to defeat the blinding countermeasure with a second-order attack in [LSCH10] and a collision attack in [ZWW13]. Power side-channel attacks for implementations using the product scanning method for multiplications are published by Huang et al. [HCY19]. A detailed evaluation of CPAs against additional multiplication techniques, including Toom-Cook and the Number Theoretic Transform (NTT), is given in [MWK+22]. Regarding countermeasures, a resource-efficient shuffling method is proposed in [WWZ+17], and an evaluation of higher-order masking for NTRU is shown by Coron et al. [CGTZ23].

For completeness, it has to be mentioned that there is an extensive amount of work regarding chosen-ciphertext attacks on NTRU. The attacks are presented in [ZCD21, REB+21, AR21, XPOZ23] with [REB+21] giving a comprehensive overview. Additionally, there exists a single-trace attack on the polynomial sampler that allows an attack on the secret key [KAA21].

Contributions This chapter analyzes countermeasures against CPA attacks on index-based multiplication for NTRUEncrypt, a widely used multiplication method at the time of this research. The contributions can be summarized as follows:

- We use arithmetic masking as a countermeasure for index-based multiplication and show two assembly-optimized masked implementations. One implementation utilizes single instruction, multiple data (SIMD) instructions of the targeted ARM Cortex-M4 microcontroller to compute the multiplication of both shares in parallel, resulting in a small overhead of only 0.21 % in the worst case.
- Practical evaluations show that both implementations exhibit no first-order leakage for up to two million measurements. We show successful second-order attacks on both implementations with up to two hundred thousand measurements.
- When using both masked implementations in combination with a shuffling countermeasure [WWZ+17], the second-order attack is no longer successful when using up to two million measurements on our setup. The shuffling method makes use of the cyclic structure of the polynomial rings in NTRUEncrypt. It can, therefore, be used without modifications to the multiplication and results in a low overhead of at most 1.89 %.

Outline In Section 3.2, the NTRUEncrypt cryptosystem is introduced. First, its evolution from the original NTRU system to newer variants in the subsequent rounds of the NIST competition is discussed, followed by an algorithmic description as well as an introduction to the used types of polynomials and their respective index-based multiplication. In order to discuss countermeasures, the CPA attack on an unsecured version of NTRUEncrypt from [Sch17] is recalled in Section 3.3. The developed masked implementations are described in Section 3.4, and a performance evaluation is presented. An evaluation of first- and second-order attacks on the protected implementations is shown in Section 3.5. Finally, Section 3.6 discusses the applicability of the results to newer versions of the cryptosystem, which includes a discussion on the usage of index-based multiplication and its theoretical adaption.

3.2 NTRUEncrypt

The NTRU cryptosystem was first introduced by Hoffstein et al. in 1998 [HPS98]. First parameter sets of an Adaptive Chosen Ciphertext Attack (CCA-2) secure version of the system using the NAEP transformation [HGSSW03] have been shown in [HGSW05]. This system and versions that have been developed since the original publication were given the name NTRUEncrypt. Based on [HGSW05], a version of NTRUEncrypt is standardized as IEEE 1363.1-2008 [IEE09]. With some advancements in the attacks targeting the underlying NTRU problem, the authors revised the parameter sets of NTRUEncrypt in [HPS+17]. These revised parameter sets are used in the submission of NTRUEncrypt for the first round of the NIST post-quantum cryptography competition [CHWZ17]. In parallel to NTRUEncrypt, two additional systems based on the original NTRU publication [HPS98], named NTRU-HRSS-KEM [HRSS17] and NTRU Prime [BCLvV17], were also submitted. NTRU-HRSS-KEM proposes some changes to the choice of parameters that eliminate decryption failures, therefore resulting in a correct and deterministic public key encryption scheme. NTRUPrime diverges further from the original NTRU system to proactively reduce the attack surface using conservative design choices. For the second round of the NIST competition, NTRUEncrypt and NTRU-HRSS-KEM merged their submissions to NTRU [CDH+19]. For NTRU, the authors unify their systems to provide a correct and deterministic scheme, preserving the characteristic of fixed-weight polynomial sampling from NTRUEncrypt with three parameter sets. An overview of the evolution of NTRUEncrypt is shown in Fig. 3.1.

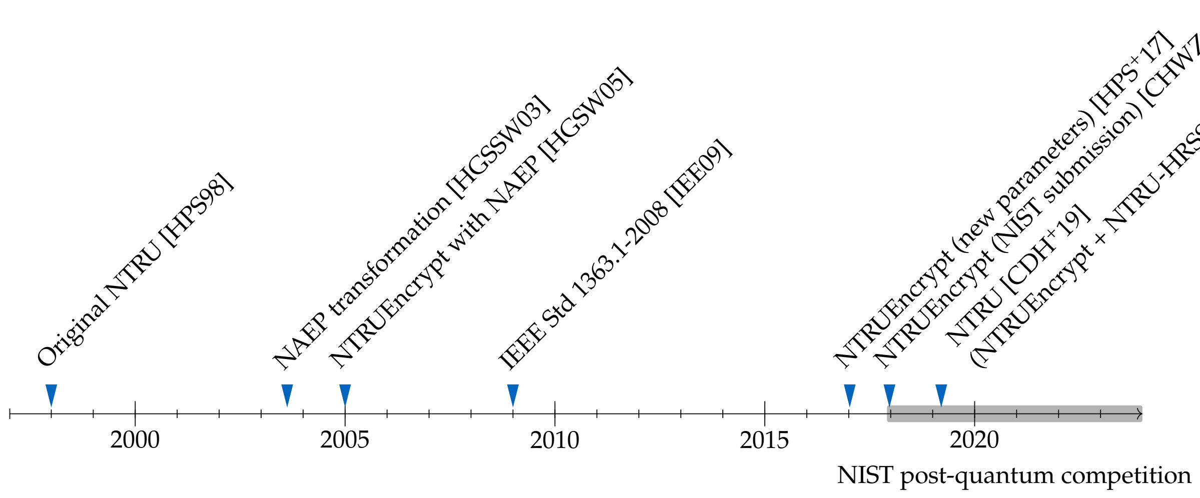


Figure 3.1 Timeline of the evolution of the NTRUEncrypt cryptosystem.

The attacks and countermeasures in this chapter are developed for NTRUEncrypt as defined in the standardized version IEEE 1363.1-2008 [IEE09] as well as the submission to the first round of the NIST competition [CHWZ17]. The following section describes the different aspects of the algorithm as defined in [CHWZ17]. Changes induced for more recent NTRU-based systems and their implications on the presented work are discussed in Section 3.6.

3.2.1 Notation and Representation of Polynomials

The main elements of NTRUEncrypt are polynomials within one of the following rings formally described as

$$\mathcal{R} = \frac{\mathbb{Z}[x]}{(x^N - 1)}, \quad \mathcal{R}_p = \frac{(\mathbb{Z}/p\mathbb{Z})[x]}{(x^N - 1)}, \quad \mathcal{R}_q = \frac{(\mathbb{Z}/q\mathbb{Z})[x]}{(x^N - 1)}. \tag{3.1}$$

In essence, this means every polynomial is at most of degree $N-1$ and has integer coefficients. For the rings $\mathcal{R}_p$ and $\mathcal{R}_q$, the coefficients of the polynomials are reduced modulo p and respectively q. This results in polynomials of the following form:

$$a(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{N-1} x^{N-1} \tag{3.2}$$

In order to ease notation, polynomials are also denoted in their vector form as $\mathbf{a} := \langle a_0, a_1, \ldots, a_{N-1} \rangle$. As the NTRUEncrypt algorithm has evolved over time, two different kinds of parameter sets were proposed. Their main difference is the choice of the modulo parameter p. This parameter defines the structure of the private key polynomial, and therefore, it is crucial to formalize a name for the different types of polynomials. In [BCE+01] the different types of a polynomial $a(x)$ are defined as:

- Binary polynomial ($p = 2$):

$$\mathcal{B}(d) : \begin{cases} a(x) \text{ has } d \text{ coefficients equal to } 1 \\ a(x) \text{ has all other coefficients equal to } 0 \end{cases} \tag{3.3}$$

- Ternary polynomial ($p = 3$):

$$\mathcal{T}(d+1, d) : \begin{cases} a(x) \text{ has } d+1 \text{ coefficients equal to } 1 \\ a(x) \text{ has } d \text{ coefficients equal to -1} \\ a(x) \text{ with other coefficients equal to } 0 \end{cases} \tag{3.4}$$

Polynomials in $\mathcal{T}$ and $\mathcal{B}$ are sampled uniformly. Only for fixed weight polynomials, as defined in Eqs. (3.3) and (3.4), the amount of non-zero elements is additionally specified. It has to be noted that earlier parameter sets [HGSW05] propose the use of binary polynomials. In contrast, more recent publications [HPS+17, CHWZ17] and the standardized version [IEE09] only use ternary polynomials.

3.2.2 Algorithm Description

This chapter describes the public key encryption variant of NTRUEncrypt according to the supporting document of the NIST submission in [CHWZ17], where it is referred to as `ntru-pke`. The KEM version of the system is omitted as it relies on calls to `ntru-pke` functions and, therefore, has no influence on the side-channel discussion.

An instance of NTRUEncrypt is described by the parameter set $\{N, p, q\}$, which defines the used polynomial rings, as well as the parameter d, which describes the amount of non-zero coefficients in the used binary or ternary polynomials. An overview of the proposed parameter sets is given in Table 3.1. The additional parameter set NTRU-1024 is omitted as it implies additional changes to the algorithm, like a different choice of the ring and the polynomial sampling method.

Parameter set	N	q	p	d	classical security [bit]
NTRU-443	443	2048	3	143	128
NTRU-743	743	2048	3	247	256

Table 3.1 Parameter sets of NTRUEncrypt as defined in [CHWZ17].

Based on a specific parameter set, the `ntru-pke` scheme consists of the three algorithms `ntru-pke`.KeyGen (Algorithm 1), `ntru-pke`.Encrypt (Algorithm 2), and `ntru-pke`.Decrypt (Algorithm 3). With `ntru-pke`.KeyGen (Algorithm 1), the private key polynomial $\mathbf{f}$ and the corresponding public key $\mathbf{h}$ can be constructed. The encryption function, `ntru-pke`.Encrypt (Algorithm 2), uses the public key polynomial $\mathbf{h}$ to encrypt the message $\mathbf{m}$ generating the ciphertext $\mathbf{e}$[1]. The decryption function, `ntru-pke`.Decrypt (Algorithm 3), takes the ciphertext $\mathbf{e}$ and the private key $\mathbf{f}$ as input in order to retrieve the message $\mathbf{m}$. The retrieval of $\mathbf{m}'$ in Line 1 of Algorithm 3 works as the parameters of NTRUEncrypt are chosen such that after the multiplication with the private key polynomial all coefficients are smaller than $q/2$ [Sch18]. This leads to the retrieval of the exact result in $\mathcal{R}$ and therefore an additional reduction modulo p recovers $\mathbf{m}'$.

The algorithms rely on a polynomial sampler SAMPLER that takes as input the desired type of polynomial and a *seed* that is used to initialize the sampler. For `ntru-pke`.KeyGen this seed is random, while for `ntru-pke`.Encrypt and `ntru-pke`.Decrypt a hash function HASH is used to deterministically derive the seed from a given input. For NTRUEncrypt, the hash function SHA-512 is used to instantiate HASH. For simplicity reasons, additional padding operations for the message $\mathbf{m}$ are not shown. It is essential for later discussions that the private key $\mathbf{f}$ is a fixed weight polynomial in either $\mathcal{T}(d+1, d)$ or $\mathcal{B}(d)$, while the ciphertext $\mathbf{e}$ is an element of $\mathcal{R}_q$. The secret key is used in the form $p \cdot \mathbf{f} + 1$ as this eliminates one multiplication step during decryption [HS00].

3.2.3 Operations on Polynomials

As discussed in Section 3.2.1 all variables of the algorithm are polynomials in their respective ring. The main property of the used rings is that elements can be at most of degree $N-1$, and an additional modulo operation, depending on the respective ring, has to be performed on each coefficient of the resulting polynomial. All operations in NTRUEncrypt are performed in $\mathcal{R}_q$, except operations that

[1]Note that related work on side-channel attacks and NTRUEncrypt descriptions prior to the NIST submission defines the ciphertext polynomial as $\mathbf{e}$, while the NIST submission defines it as $\mathbf{c}$. In order to be compatible with related work, we stick to the convention of naming it $\mathbf{e}$.

Algorithm 1 `ntru-pke`.KeyGen

```
Input: Parameter set = $\{N, p, q, d\}$, seed.
1: $\mathbf{f} \leftarrow$ SAMPLER($\mathcal{T}(d+1, d)$, seed)        ▷ For $p = 2$: SAMPLER($\mathcal{B}(d)$, seed)
2: if $\mathbf{f}$ is not invertible mod $q$ then
3:     go to step 1
4: $\mathbf{g} \leftarrow$ SAMPLER($\mathcal{T}(d+1, d)$, seed)        ▷ For $p = 2$: SAMPLER($\mathcal{B}(d)$, seed)
5: $\mathbf{h} = \mathbf{g}/(p\mathbf{f} + 1) \bmod q$
Output: Public key $\mathbf{h}$, secret key $\mathbf{f}$
```

Algorithm 2 `ntru-pke`.Encrypt

```
Input: Public key $\mathbf{h}$, message $\mathbf{m}$
1: $\mathbf{r} \leftarrow$ SAMPLER($\mathcal{T}(d+1, d)$, HASH($\mathbf{m}|\mathbf{h}$))        ▷ For $p = 2$: SAMPLER($\mathcal{B}(d)$, HASH($\mathbf{m}|\mathbf{h}$))
2: $\mathbf{t} = \mathbf{r} * \mathbf{h}$
3: $\mathbf{m}_{mask} \leftarrow$ SAMPLER($\mathcal{T}$, HASH($\mathbf{t}$))        ▷ For $p = 2$: SAMPLER($\mathcal{B}$, HASH($\mathbf{t}$))
4: $\mathbf{m}' = \mathbf{m} - \mathbf{m}_{mask} \bmod p$
5: $\mathbf{e} = \mathbf{t} + \mathbf{m}'$
Output: Ciphertext $\mathbf{e}$
```

Algorithm 3 `ntru-pke`.Decrypt

```
Input: Private key $\mathbf{f}$, public key $\mathbf{h}$, ciphertext $\mathbf{e}$
1: $\mathbf{m}' \leftarrow (p \cdot \mathbf{f} + 1) * \mathbf{e} \bmod p$
2: $\mathbf{t} \leftarrow \mathbf{e} - \mathbf{m}'$
3: $\mathbf{m}_{mask} \leftarrow$ SAMPLER($\mathcal{T}$, HASH($\mathbf{t}$))        ▷ For $p = 2$: SAMPLER($\mathcal{B}$, HASH($\mathbf{t}$))
4: $\mathbf{m} = \mathbf{m}' + \mathbf{m}_{mask} \bmod p$
5: $\mathbf{r} \leftarrow$ SAMPLER($\mathcal{T}(d+1, d)$, HASH($\mathbf{m}|\mathbf{h}$))        ▷ For $p = 2$: SAMPLER($\mathcal{B}(d)$, HASH($\mathbf{m}|\mathbf{h}$))
6: if $p \cdot \mathbf{r} * \mathbf{h} = \mathbf{t}$ then
7:     result $\leftarrow \mathbf{m}$
8: else
9:     result $\leftarrow \perp$
Output: result
```

specify an additional modulo p operation in Algorithms 1 to 3. The choice of the ring quotient $x^N - 1$ defines the multiplication of polynomials as cyclic convolution, which is defined as

$$a(x) * b(x) = \sum_{k=0}^{N-1} \left(\sum_{i+j \equiv k \pmod{N}} a_i b_j \right) x^k \tag{3.5}$$

in [HPS14]. In other words, Eq. (3.5) can be seen as the multiplication of two polynomials with an additional reduction of the result by $(x^N - 1)$ through polynomial long division. The convolution product is denoted with the symbol $(*)$, while a multiplication with a factor is marked as $(\cdot)$.

The bottleneck operation of NTRUEncrypt is the multiplication of two polynomials through this convolution product. There are several publications on optimized implementations of this multiplication. A general discussion of multiplication methods for microcontrollers with a focus on an ARM Cortex-M4 is given in [KRS19]. Another popular type of implementation for resource-constrained devices utilizes the sparse structure of binary or ternary polynomials. In [BCE+01], the authors propose Algorithm 4 for the multiplication of a polynomial in $\mathcal{R}_q$ and a binary polynomial $\mathcal{B}(d)$. With this algorithm, which is called *index-based multiplication* in this thesis, the authors substitute the multiplication of coefficients with additions based on the index of ones in the binary polynomial. As a binary polynomial is built to be sparse, the coefficients with the value zero can be skipped, resulting in a lower number of additions to execute and, therefore, a faster multiplication. This algorithm can be used for NTRUEncrypt as all convolution products of the algorithm have one sparse polynomial as an operand. This is also true for the first line of Algorithm 3 as $(p \cdot \mathbf{f} + 1) * \mathbf{e}$ can be rewritten to $(p \cdot \mathbf{f} * \mathbf{e} + \mathbf{e})$. As described in Section 3.2.1 recent parameter sets make use of ternary polynomials. These polynomials also have a sparse nature and contain only ones and minus ones, which allows abstracting the multiplication again by either addition or subtraction based on the index of nonzero coefficients. An adaption of Algorithm 4 for ternary polynomials is given in Algorithm 5.

It has to be noted that index-based multiplication has been developed for resource-constrained devices like microcontrollers, which typically do not contain a cache. Therefore, the secret-dependent array accesses can be considered as constant time on these devices. On devices that contain a cache, this multiplication method induces a timing side-channel that has to be considered. For a detailed discussion, see Section 3.6.

Algorithm 4 Index-based binary multiplication [Sch17]

```
Input: $a(x) \in \mathcal{B}(d)$ (stored as an array a[d] with indexes $a_i$); $b(x) \in R_q$
1: Initialize a temporary array $t$ of size $2N$
2: for $0 \leq j < 2N$ do                                  ▷ Initialize $t(x)$ with zero
3:     $t_j \leftarrow 0$
4: for $0 \leq j < d$ do
5:     for $0 \leq k < N$ do
6:         $t_{k+a[j]} \leftarrow t_{k+a[j]} + b_k$        ▷ Add polynomial $b(x)$ at position a[j]
7: for $0 \leq j < N$ do
8:     $c_j \leftarrow (t_j + t_{j+N}) \bmod q$            ▷ Reduction by $(x^N - 1)$ modulo q
Output: $c(x) \in R_q = a(x) * b(x)$
```

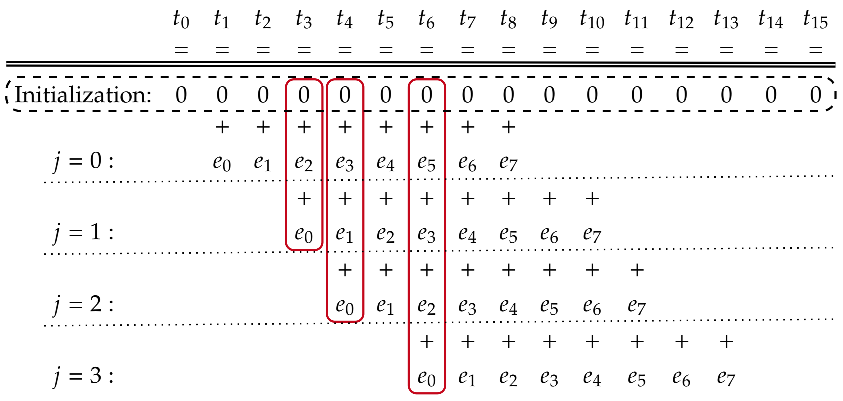

t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14} t_{15}
Initialization: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
$j = 0$: e_0 e_1 e_2 e_3 e_4 e_5 e_6 e_7
$j = 1$: e_0 e_1 e_2 e_3 e_4 e_5 e_6 e_7
$j = 2$: e_0 e_1 e_2 e_3 e_4 e_5 e_6 e_7
$j = 3$: e_0 e_1 e_2 e_3 e_4 e_5 e_6 e_7

Figure 3.2 Index-based multiplication with binary polynomials according to Algorithm 4 (lines 1 to 6) for the parameters N = 8 and $f \in \mathcal{B}(4) = [1, 3, 4, 6]$ from [Sch17]. All targeted additions with the coefficient e_0 are marked.

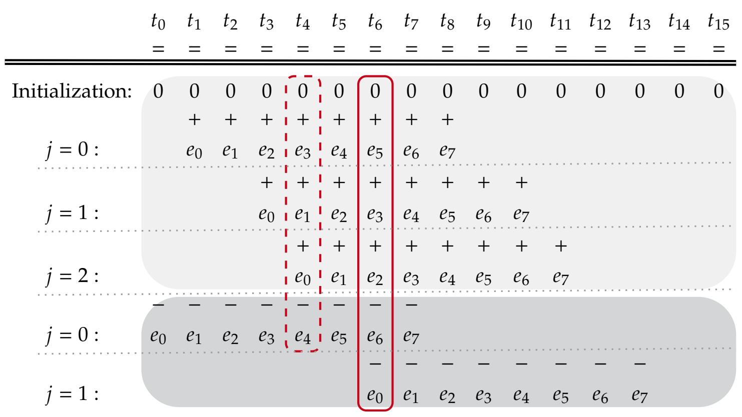
Algorithm 5 Index-based ternary multiplication [Sch17]

```
Input: $a(x) \in \mathcal{T}(d+1, d)$ (stored as arrays $a_{ones}[d+1]$ and $a_{mones}[d]$); $b(x) \in R_q$
 1: Initialize a temporary array $t(x)$ of size $2N$
 2: for $0 \leq j < 2N$ do                                  ▷ Initialize $t(x)$ to zero
 3:     $t_j \leftarrow 0$
 4: for $0 \leq j < d+1$ do
 5:     for $0 \leq k < N$ do
 6:         $t_{k+a_{ones}[j]} \leftarrow t_{k+a_{ones}[j]} + b_k$        ▷ Add $b(x)$ at position $a_{ones}[j]$
 7: for $0 \leq j < d$ do
 8:     for $0 \leq k < N$ do
 9:         $t_{k+a_{mones}[j]} \leftarrow t_{k+a_{mones}[j]} - b_k$      ▷ Subtract $b(x)$ at position $a_{mones}[j]$
10: for $0 \leq j < N$ do
11:     $c_j \leftarrow (t_j + t_{j+N}) \bmod q$                ▷ Reduction by $(x^N - 1)$ modulo q
Output: $c(x) \in R_q = a(x) * b(x)$
```

3.3 CPA Attack on NTRUEncrypt

The main power analysis attack on NTRUEncrypt consists of a CPA published in [LSCH10] based on parameter sets with binary polynomials. With this attack, the authors target the multiplication of the private key **f** with the ciphertext **e** (c.f. Line 1 of Algorithm 3), as this is the only operation on the private key with an attacker-controllable input. If this multiplication is performed with index-based multiplication, as described in Algorithm 4, an attacker can exploit the fact that the addition of the first ciphertext coefficient e_0 is determined by the indexes of ones in the private key **f**. An example of this multiplication is visualized in Fig. 3.2.

For all d rounds of additions, the ciphertext coefficients e_i are added with the content of the corresponding temporary result array element t_i in a sequential manner, starting from e_0 to e_{N-1}. Based on the additions with e_0 (marked in Fig. 3.2), the difference between the key indexes of each round of additions can be found. The attack of [LSCH10] performs a separate CPA for all rounds $j \geq 1$ with an attack on the Hamming distance of the addition of t_i with e_0 (HD($t_i, t_i + e_0$)). The corresponding values of t_i can be calculated based on a hypothesis for the difference in the associated key indexes. After the individual CPAs successfully retrieved all index differences w_i, the location of the first key index can be found by exhaustive search. We will denote the difference between the



t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14} t_{15}
Initialization: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
$j = 0:$ e_0 e_1 e_2 e_3 e_4 e_5 e_6 e_7
$j = 1:$ e_0 e_1 e_2 e_3 e_4 e_5 e_6 e_7
$j = 2:$ e_0 e_1 e_2 e_3 e_4 e_5 e_6 e_7
$j = 0:$ e_0 e_1 e_2 e_3 e_4 e_5 e_6 e_7
$j = 1:$ e_0 e_1 e_2 e_3 e_4 e_5 e_6 e_7

Figure 3.3 Example of a index-based multiplication with ternary polynomials according to Algorithm 5 (lines 1 to 9) for $N = 8$ and $f \in \mathcal{T}(3, 2) = [1, 3, 4], [0, 6]$ from [Sch17].

indexes $f[i]$ and $f[i+1]$ with w_i for the rest of the paper. For example, the difference between the first index $f[0]$ and the second index $f[1]$ will be called w_0.

Adaption of the attack to ternary polynomials In [Sch17] the adaption of the CPA attack from [LSCH10] for a convolution product with ternary polynomials as implemented with an index-based multiplication given by Algorithm 5 was performed. An example of a ternary multiplication is shown in Fig. 3.3. It can be seen that the first part of the multiplication (light grey background) is performed exactly as in the binary case. Therefore, the differences between the ones in $\mathbf{f} \in \mathcal{T}(d+1, d)$, called w_i^1, can be found by the attack methodology for binary polynomials.

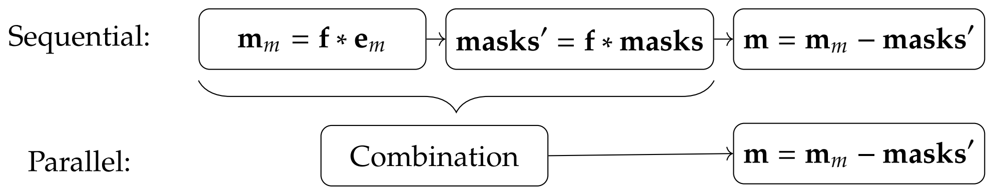
The adaption works by first attacking the difference between the last index of ones $f_{ones}[d+1]$ and the first index of minus ones $f_{mones}[0]$ during the first round of subtractions ($j = 0$). This difference will be called w^0. The correct w^0 can be retrieved by finding the index of e_i that is subtracted from the t_i corresponding to the last addition with e_0 (marked with a dotted line in Fig. 3.3). Based on the different hypotheses for w_0, the Hamming weight of the corresponding intermediate values can be calculated and attacked through CPA. It has to be noted that there is the unlikely possibility of no subtraction from t_i corresponding to the last addition of e_0 for some constructions of the private key. This can also be defeated by successively evaluating different points of subtraction, for example, the value of t_i corresponding to the last addition of e_1. With the correct w^0 the remaining differences between the indexes of minus ones w_i^{-1} can be found. Similar to the binary case, different hypothetical intermediate results for the subtraction of e_0 during the rounds $j \geq 1$ can be constructed (see the attacked subtraction result marked with a solid line in Fig. 3.3). The correct hypothesis can be found through CPA, which reveals the corresponding w_i^{-1}.

The final private key $\mathbf{f}$ can be found through an exhaustive search for $f_{ones}[0]$, as all relative index differences are known at this stage. The complexity of this search can be seen as negligible as an attacker has to try at most $N - (2d+1)$ different combinations. This leads to at most 248 combinations for the parameter set NTRU-743.

3.4 Masking Countermeasure

In addition to their attack, the authors of [LSCH10] propose three different countermeasures to secure NTRUEncrypt utilizing index-based multiplication. The different countermeasures are:

1. **Random initialization of t:** The temporary result array $\mathbf{t}$ is initialized with different random values r_i.



Sequential:
$\mathbf{m}_m = \mathbf{f} * \mathbf{e}_m$
$\mathbf{masks}' = \mathbf{f} * \mathbf{masks}$
$\mathbf{m} = \mathbf{m}_m - \mathbf{masks}'$
Parallel:
Combination
$\mathbf{m} = \mathbf{m}_m - \mathbf{masks}'$

Figure 3.4 Comparison of the developed sequential and parallel masked implementations. It can be seen that the parallel implementation combines the first two steps into one.

2. **Shuffling:** The sequence of all addition/subtraction rounds can be shuffled randomly, as the order has no impact on the final result. In theory, shuffling countermeasures can be defeated with an increased amount of traces. Therefore, the authors propose this countermeasure only in combination with masking.

3. **Masking of ciphertext e:** With this countermeasure, each coefficient e_i is masked with a random value through modular addition (arithmetic masking). A detailed evaluation of this countermeasure is given in this work.

Successful attacks have been shown against the random initialization countermeasure with a second-order CPA [LSCH10] and a first-order collision attack [ZWW13]. The second-order CPA still targets the $\mathrm{HD}(t_i, t_i + e_0)$, which is changed by the countermeasure to $\mathrm{HW}(t_i + r_i, t_i + r_i + e_0)$. The authors show that the subtraction of the power consumption of $\mathrm{HD}(r_i, t_i + r_i)$ from $\mathrm{HD}(t_i + r_i, t_i + r_i + e_0)$ can be used as a preprocessing function to mount an attack on the unmasked values with the hypothetical power consumption of $\mathrm{HW}(t_i) - \mathrm{HW}(e_0)$. In order to perform the first-order collision attack, an attacker has to observe the power consumption during the initialization phase of t with the different masks r_i. The highest correlation with the power consumption during the addition of the last ciphertext coefficient e_{N-1} allows the calculation of the corresponding private key index. This can be done for all rounds of addition.

Since possible attacks for the random initialization countermeasure of [LSCH10] have already been shown, and other approaches can be seen as hiding countermeasures, we focus on evaluating the masking of the ciphertext. Only a masking countermeasure is able to reliably provide a first-order secured implementation, as it makes the processed variables independent from the known input, in this case, the ciphertext. We use arithmetic masking with different masks on all coefficients of the ciphertext polynomial **e**. Arithmetic masking, in our case, can be defined as a modular addition of **e** with a polynomial **masks**, containing the different masks for each coefficient, as

$$\mathbf{e}_m = \mathbf{e} + \mathbf{masks} \bmod 2^n \,. \tag{3.6}$$

For our implementations, the modulus is set to 2^{16} as the elements of the temporary result array **t** are stored with 16-bit values. Arithmetic masking is more suitable for index-based multiplication as it only performs arithmetic operations, and therefore, changes to the mask are linear. In this case, changes of a mask can be computed by performing the operations on the mask itself. The masking countermeasure is implemented for the ternary index-based multiplication described in Algorithm 5. We provide two different masked implementations with the use of ARM assembly code. The first implementation sequentially performs the multiplication on the masked values and the mask itself. This approach has the downside of an increased execution time as the multiplication algorithm is executed twice in order to compute the mask changes. We eliminate this disadvantage in our second implementation, as it computes the multiplication of the masked value and the mask in parallel. To achieve this, we utilize special SIMD instructions of the ARM Cortex-M4 architecture. The idea of the different implementations is visualized in Fig. 3.4.

3.4.1 Sequential Implementation

The sequential implementation first performs a multiplication of the masked ciphertext $\mathbf{e}_m$ with the private key $\mathbf{f}$ to get the masked result $\mathbf{m}_m$:

$$\mathbf{m}_m = \mathbf{f} * \mathbf{e}_m = \mathbf{f} * (\mathbf{e} + \mathbf{masks}) \tag{3.7}$$

In a second step, the changes to the masks are computed through a multiplication of $\mathbf{f}$ with the values of the masks as:

$$\mathbf{masks}' = \mathbf{f} * \mathbf{masks} \tag{3.8}$$

In order to retrieve the unmasked multiplication result $\mathbf{m}$, all coefficients of $\mathbf{masks}'$ are subtracted from $\mathbf{m}_m$ with the result reduced modulo q.

3.4.2 Parallel Implementation

The parallel implementation of the masking countermeasure makes use of SIMD instructions of the DSP extension of an ARM Cortex-M4 architecture. With these instructions, a 32-bit word is split into smaller parts (two 16-bit or four 8-bit values) on which the corresponding arithmetic operation can be performed in parallel. For example, the SADD16 and SSUB16 operations perform an addition or subtraction on the higher and lower 16-bit parts of the operand, taking care of suppressing a potential carry overflow between the two parts. An example of an addition with SADD16 is given in Fig. 3.5.

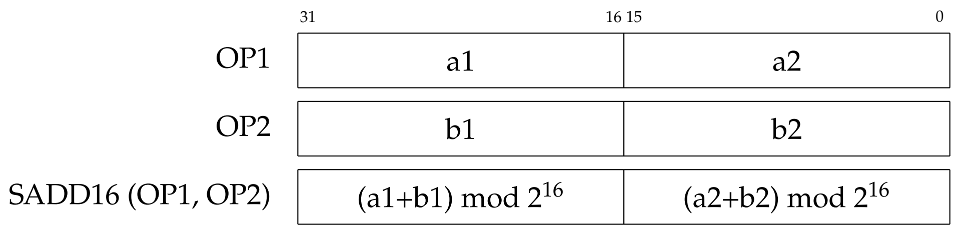


Figure 3.5 Visualization of the SADD16 instruction of an ARM Cortex-M4 microcontroller. Two 16-bit additions of $(a1 + b1)$ and $(a2 + b2)$ are performed in parallel. The subtraction with SSUB16 works accordingly.

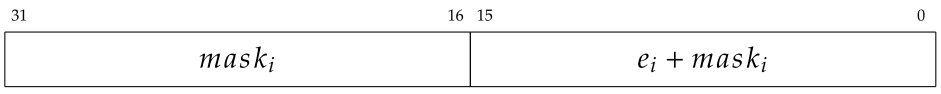


Figure 3.6 Description of the input to Algorithm 5 as a combination of the masked ciphertext and the mask itself. The individual coefficients of the ciphertext $\mathbf{e}$ are constructed in this way.

In order to benefit from these operations, we construct the individual ciphertext coefficients e_i and corresponding mask $mask_i$ as a 32-bit word as shown in Fig. 3.6. By using this construction as an input for the ciphertext coefficients e_i in Algorithm 5, all additions and subtractions can be implemented with the corresponding SIMD instruction. In this case, the mask update is computed in parallel, which is twice as fast as the sequential implementation. The implicit reduction modulo 2^{16} does not change the result of the multiplication as all coefficients of the result are reduced modulo 2^{11}, with $q = 2048$ for modern parameter sets.

This type of masking can be attacked with a univariate second-order attack, which does not require finding correct sample combinations of the mask and the masked value. Nevertheless, the implementation still benefits from the noise amplification property (c.f. Section 2.4). Additionally, it has to be noted that this type of masking is prone to unintended recombination of both shares due to effects caused by the microarchitecture of the executing device (see [MPW21] for a study of this effect). Although the experimental results in Fig. 3.10 show no first-order leakage for up to two million traces – thus demonstrating that this type of masking is indeed possible on the evaluated target – its usage has to be carefully evaluated.

3.4.3 Optional Shuffling Countermeasure

The cyclic structure of the rings in NTRUEncrypt allows for an efficient shuffling of the multiplication as described in Algorithm 6. This approach was first sketched in [ZWW13] and further analyzed in [WWZ+17]. The correctness of the shuffled multiplication can be seen by looking at the complete multiplication, which results in $(a(x) * x^i) * (b(x) * x^{N-i}) = a(x) * b(x) * x^N$. As x^N equals one in the ring $\mathcal{R}_q$, the multiplication result is equal to the unshuffled variant. The overhead of this countermeasure is low since multiplication by x^i and x^{N-i} corresponds to a clockwise (x^i) or counterclockwise (x^{N-i}) rotational shift of the coefficients of a polynomial. In addition, the implementation of the multiplication does not have to be changed.

Algorithm 6 Shuffled multiplication [WWZ+17, Algorithm 2]

```
Input: $a(x) \in \mathcal{T}(d+1, d)$ or $\mathcal{B}(d)$; $b(x) \in R_q$
 1: $i \xleftarrow{\$} \{0, \dots, N-1\}$
 2: $a(x) = a(x) * x^i$
 3: $b(x) = b(x) * x^{N-i}$
 4: $c(x) = a(x) * b(x)$                    ▷ Algorithm 4 or Algorithm 5
Output: $c(x) \in R_q$
```

3.4.4 Performance Evaluation

In this section, the proposed implementations are evaluated regarding their performance, i.e., execution time overhead. The cycle counts for both NTRUEncrypt parameter sets (c.f. Table 3.1) on an ARM Cortex-M4 are shown in Table 3.2. The cycle measurements are done on an STMicroelectronics `STM32F303DISCOVERY` board with an `STM32F303VC` microcontroller running at 48 MHz. The code is compiled with version 13.2.0 of `arm-none-eabi-gcc` and an `-O3` optimization. The presented cycle counts are averaged over 10 000 measurements.

	Unmasked	Sequential	Sequential (shuffled)	Parallel	Parallel (shuffled)
NTRU-443	1 867 266[a]	3 863 936	3 902 881	1 871 197	1 906 567
NTRU-743	5 372 737[a]	11 128 983	11 182 445	5 373 387	5 438 931

[a]) Note that it is possible to also speed up the unmasked implementation with SIMD instructions increasing the performance by a factor of 2.

Table 3.2 Performance evaluation of the implementations presented in this section.

We use the ternary index-based multiplication (Algorithm 5) as a baseline for comparison, which is denoted as Unmasked in Table 3.2. The sequential masked implementation (Section 3.4.1) has an overhead of 107 % (NTRU-443) and 107.14 % (NTRU-743), which mainly comes from executing the multiplication twice for both shares. In contrast, the parallel masked implementation (Section 3.4.2) only has an overhead of 0.21 % (NTRU-443) and 0.012 % (NTRU-743). All assembly operations between the parallel implementation and the baseline require the same amount of clock cycles. This overhead is due to the additional required unmasking of the result during the modulo reduction in lines 10 and 11 of Algorithm 5. Adding the additional shuffling countermeasure increases the required clock cycles by 1.01 % (NTRU-443) and 0.48 % (NTRU-743) for the sequential implementation, while the parallel implementation shows an increase of 1.89 % (NTRU-443) and 1.22 % (NTRU-743).

3.5 Side-Channel Evaluation Results

In this section, we show the results of a CPA attack on a non-masked implementation using ternary polynomials as well as first- and second-order attacks against the two masked implementations. In addition, we show attack results for the masked implementations in combination with the shuffling countermeasure.

All attacks are performed with power measurements of a STM32F303RCT7 ARM Cortex-M4 microcontroller mounted on the NewAE CW308 UFO board. The target device is built with a 12 Ω shunt resistor placed in the VDD line, and the corresponding power consumption can be measured through an SMA connector on the CW308 board. The measurements are performed with a Picoscope 6402D and a sampling frequency of 156.25 MHz. As the power is measured between VDD and GND, a Minicircuits BLK-89+ DC Block is used in order to utilize the whole input range of the oscilloscope. The clock for our Device Under Test (DUT) is fixed to 10 MHz and is provided by a Keysight 33500B waveform generator. In order to provide aligned traces, the device clock and the sampling clock of the oscilloscope are synchronized through the waveform generator.

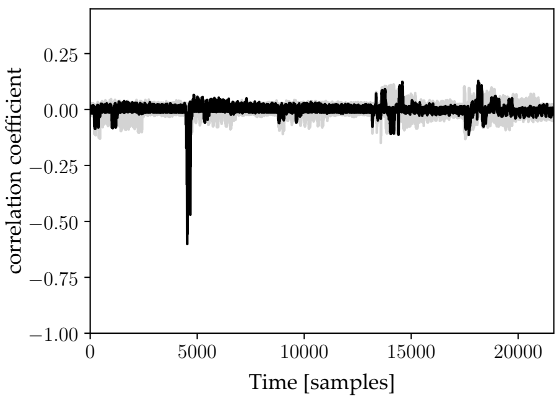
3.5.1 CPA on Ternary Polynomials

First, the results of the CPA against the unmasked ternary multiplication according to Algorithm 5 from [Sch17] are presented. As described in Section 3.3, the attack retrieves the differences between the indices of ones and minus ones in $\mathbf{f}$. This results in $2d + 1$ CPAs to retrieve all w^1 and w^{-1} and an additional attack on w^0, which translates to a sum of 288 different attacks for, e.g., NTRU-443. Although the attack has been successfully performed for both NTRU parameter sets, it is not possible to show this amount of attack results (i.e., 288 for NTRU-443 and 495 for NTRU-743) in a reasonable manner. Therefore, the results for a smaller instance for a multiplication given the private key $f \in \mathcal{T} = [3, 7, 10], [1, 4]$ and a maximum degree of $N = 20$ is shown in Fig. 3.7.

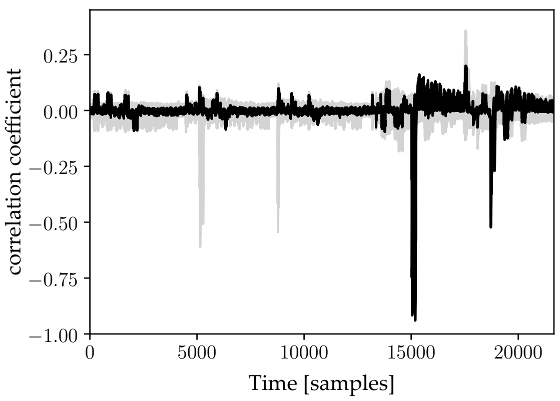
Exemplarily the correlation graphs that reveal w_0^1, w^0, and w_0^{-1} are provided. However, all different CPA attacks on the corresponding key indexes are successful. The correlation over time is shown for the whole execution of lines 4 to 9 in Algorithm 5. It can be seen that the attack is successful even without the restriction of the measurements to the corresponding attacked operations. Nevertheless, a restriction would remove additional correlation peaks. In contrast to the attack in [LSCH10], the attack uses the Hamming weight power model and targets the result of the addition of t_i with e_0 as $\mathrm{HW}(t_i + e_0)$ for retrieving all w^1. The Hamming weight is used, as it usually corresponds to the bus transfer leakage on microcontrollers, which shows a high amount of exploitable leakage. This is due to the comparable large length of the bus wires, which causes a high power consumption for transfers on the bus. In [Sch17] this is experimentally verified and the storage of the final result $t_i + e_0$ is found to be a suitable target for the attack with a higher exploitable leakage than the Hamming distance of the computation.

3.5.2 Second-Order Attacks on the Masking Countermeasure

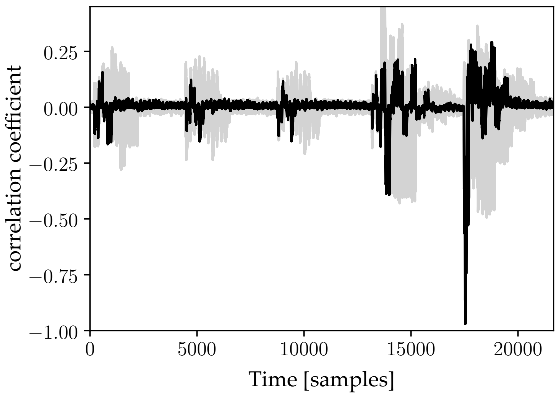
After the attack has been shown to be working on an unprotected implementation, this section discusses the results of second-order attacks against the masked implementations. We again limit the amount of shown attack results and the NTRU instance, which does not influence the transferability of results. For the attack, we take measurements for the respective implementation with the parameters $N = 20$ and private key $f \in \mathcal{T}(2, 1) = [3, 7], [5]$. Attack results are only given for the second round ($j = 1$) in Algorithm 5 corresponding to the addition of ones in the ciphertext. In other words, results are shown for the difference between $f_{ones}[0]$ and $f_{ones}[1]$, which is the first attackable key index difference. The presented attacks are also applicable to the remaining key index differences. We first show attack results on the sequential implementation, followed by results for the parallel variant. In a second step, we provide a comparison of both implementations regarding the required amount of traces for a successful attack.



(a) Result for w_0^1



(b) Result for w^0

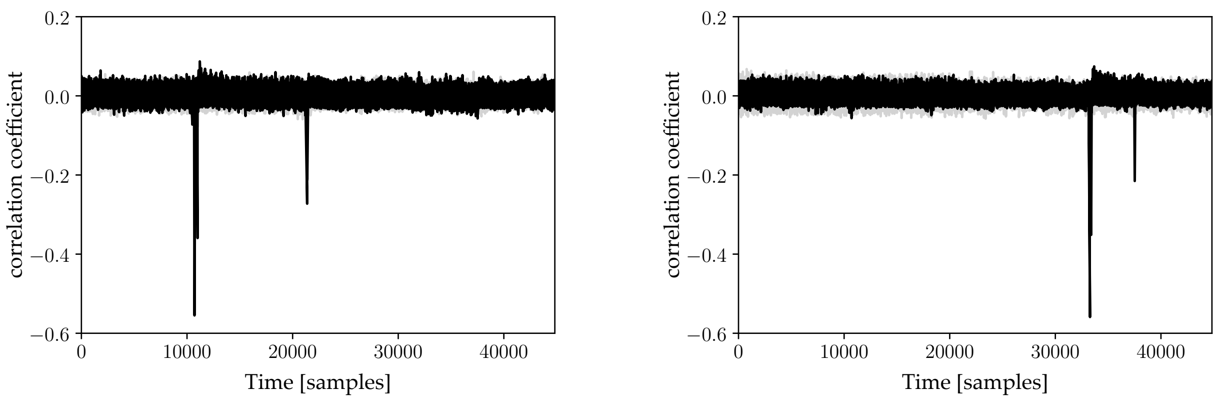


(c) Result for w_0^{-1}

Figure 3.7 CPA results for $N = 20$ and $f \in \mathcal{T} = [3, 7, 10], [1, 4]$ with a total amount of 10 000 traces. Using the initially developed implementation from [Sch17], the shown attack results are regenerated for [SMS19] using traces captured with the described measurement setup.

Sequential Implementation

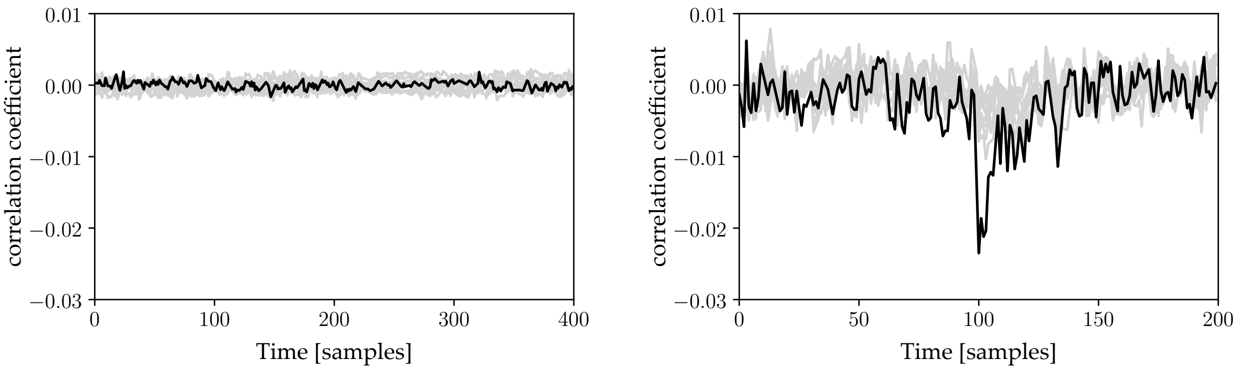
In this implementation, the masked value and the mask itself are processed during different clock cycles. Therefore, a multivariate second-order attack is able to defeat the masking countermeasure through the combination of the corresponding leakages. In [PRB09, SVCO$^+$10], the normalized product preprocessing function, which was initially proposed in [CJRR99], is stated to be the optimal way of combination for a Hamming weight leakage model. As we are targeting Hamming weight leakage, we make use of this combination function by a multiplication of the corresponding mean-free sample points. The location of the leakage points is found by a separate CPA on the masked intermediate value and the mask itself. Figure 3.8 shows the correlation of both shares, indicating the points in time where the individual values are processed. This is possible since we want to evaluate the attack under the best possible conditions, and therefore, we store the masks during the trace measurement. If an attacker does not know the corresponding mask, she has to perform an educated guess on the possible leakage areas and try all possible combinations of samples, usually as a multiple of clock cycles. This is an advantage of the sequential implementation since the computation of the correlation for all these sample combinations becomes computationally expensive. Results for the first- and second-order attacks are shown in Fig. 3.9. No significant correlation is visible for the first-order attack using up to two million trace measurements. In contrast, the second-order attack is successful with two hundred thousand traces.



(a) Masked value

(b) Mask

Figure 3.8 Leakage points of the masked value and the corresponding mask in time for the sequential implementation. The points with the highest correlation are used for the multivariate second-order attack.



(a) First-order attack (2M traces)

(b) Second-order attack (200k traces)

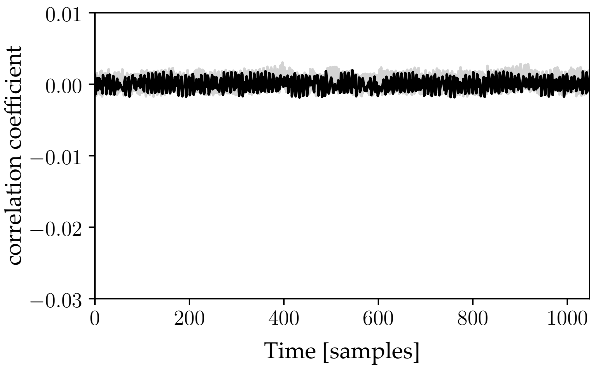
Figure 3.9 Attack results for the sequential masked implementation. Here, a subsection of 200 samples centered around the leakage of the mask is shown. For the preprocessing step, the traces are shifted 1445 clock cycles.

Parallel Implementation

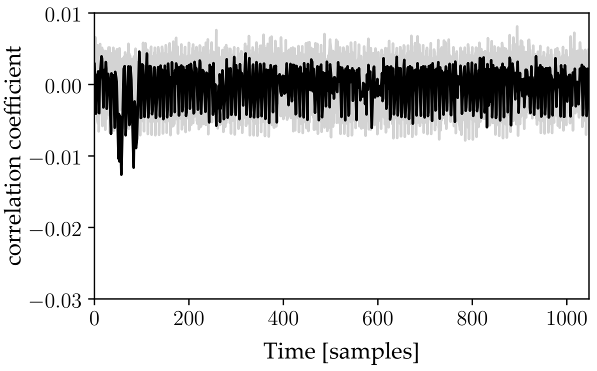
As the parallel implementation processes the mask and the masked value at the same time, a zero-offset second-order attack can be used to attack this implementation. In order to perform this attack, the individual samples of the traces are mean-free squared. In Fig. 3.10, attack results for the masked implementation with the parallel construction are shown. No first-order leakage could be found using an amount of two million trace measurements. On the other hand, it can be seen that the proposed second-order attack is successful in retrieving the correct key index difference w_0^1 for an amount of two hundred thousand traces.

Comparison

In order to compare both implementations, we provide correlation plots of the corresponding main leakage points, taken from the attack results in Figs. 3.9 and 3.10, for an increasing number of trace measurements in Fig. 3.11. The correlation is shown for up to two hundred thousand measurements. It can be seen that a second-order attack is less effective on the parallel implementation. In contrast, the sequential implementation shows significant leakage with less than 50 000 traces. In practice, a designer has to make a trade-off between both implementations. The parallel implementation requires a significantly increased amount of traces for a successful attack and comes with a reduced execution time in comparison to the sequential one. The sequential implementation has the benefit that in a



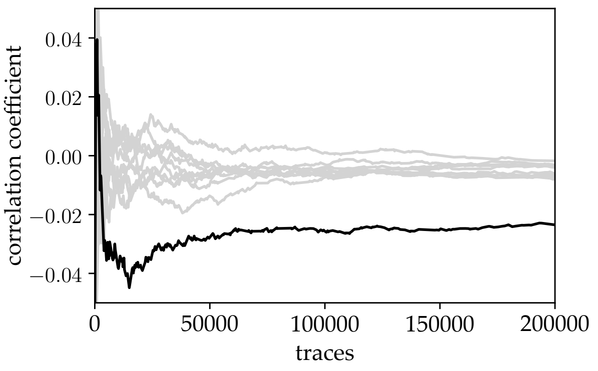
(a) First-order attack (2M traces)



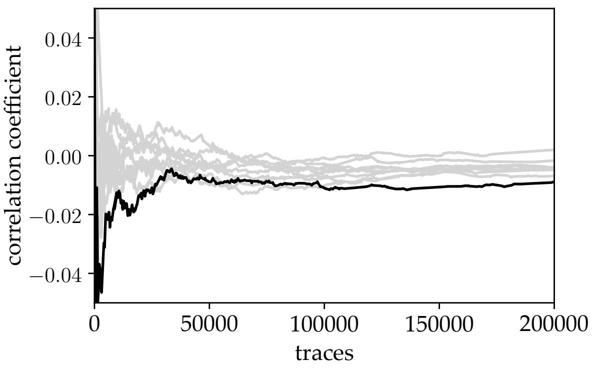
(b) Second-order attack (200k traces)

Figure 3.10 Attack results for the parallel masked implementation. The shown samples belong to the execution of the first two coefficient additions corresponding to $f_{ones}[1]$.

black box scenario, an attacker has to find the correct sample combinations that enable a second-order attack, which can become computationally expensive.



(a) Sequential implementation



(b) Parallel implementation

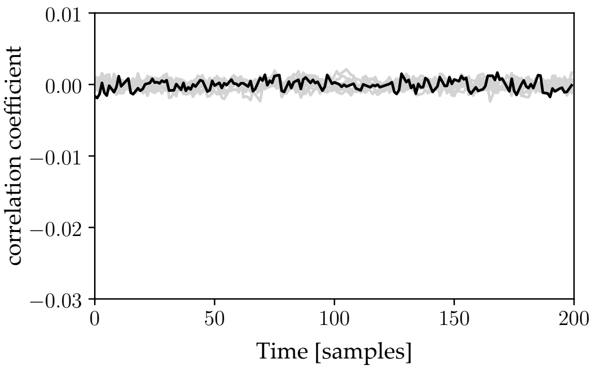
Figure 3.11 Evolution of the correlation for both implementations with an increasing number of traces. The correlation of all hypotheses is shown for up to two hundred thousand measurements.

3.5.3 Second-Order Attack on the Combination of Masking and Shuffling

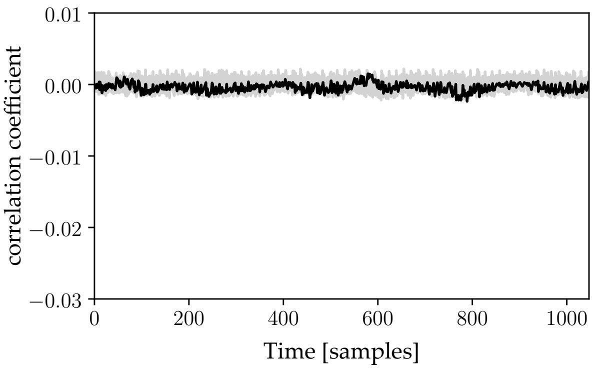
We have shown that both implementations can be attacked with a second-order attack. As the shuffling countermeasure described in Section 3.4.3 can be employed without changes to the underlying multiplication, we propose its use in combination with our masked implementations. For the used parameter set, the degree of polynomials is $N = 20$, which implies a shuffling with twenty possible ways of multiplication. The attack results for both shuffled and masked implementations are shown in Fig. 3.12. It can be seen that the resulting implementation shows no significant second-order leakage for up to two million trace measurements.

3.6 Discussion

The attack on NTRUEncrypt described in this chapter has been developed very early in the NIST competition. It is, therefore, based on a version that was available at that time, namely the standardized version [IEE09] with modern parameter sets [HPS$^+$17] and the first round NIST submission [CHWZ17]. This section gives an overview of algorithm changes for NTRUEncrypt during the ad-



(a) Sequential implementation



(b) Parallel implementation

Figure 3.12 Second-order attack results for both masked implementations in combination with the random key rotation countermeasure using two million trace measurements.

vancing NIST competition and discusses its influence on the presented attack and countermeasures. Additionally, the usage of index-based multiplication in NTRUEncrypt is discussed.

Usage of Index-based Multiplication As already briefly discussed in Section 3.2.3, the index-based multiplication has the downside of not being executed in constant time on systems with caches since its usage of secret-dependent array accesses. Nevertheless, the algorithm is developed for resource-constrained devices like microcontrollers that usually do not contain caches, making it suitable for such devices. During the time of performing the research, index-based multiplication was the standard method for polynomial multiplication by the NTRUEncrypt authors, as it, e.g., can be seen in their open source library named `libntruencrypt`[2]. An attack on index-based multiplication had practical relevance as it was an attack on the reference implementation of the algorithm. Another argument for the relevance of the presented research is its applicability to performant hardware multipliers for ternary polynomials as used by Fritzman et al. [FSF+19] and Farahmand et al. [FDNG19]. These accelerators are built in a Linear Feedback Shift Register (LFSR) structure that performs a clockwise addition or substruction of coefficients dependent on the ternary polynomial with an implicit reduction in the ring. An attack can be performed by building a hypothesis for a specific ternary coefficient using the Hamming distance for the update of the registers holding the polynomial coefficients of the result. This again allows to perform multiple CPAs to successively retrieve all private key coefficients. Ternary multiplication can be performed in constant time using these hardware accelerators.

Changes to NTRUEncrypt With the merge of NTRUEncrypt and NTRU-HRSS-KEM in the second round of the NIST competition to the NTRU submission [CDH+19], several changes to the algorithm have to be considered regarding the presented side-channel attack. Although the reference implementations of all these submissions decided not to use index-based multiplication anymore, we first discuss if its usage is possible in theory.

For the NTRU submission, the authors merged their submissions with the purpose of unifying them. This results in four different parameter sets, for which three parameter sets keep the property of fixed weight polynomials, as in NTRUEncrypt. These parameter sets define an instance that is called NTRU-HPS, while the additional one follows the design principles of NTRU-HRSS-KEM and is therefore called NTRU-HRSS. The resulting parameter sets are shown in Table 3.3.

The parameter sets for NTRU-HPS allow the usage of index-based multiplication, although it has to be noted that $\mathbf{f}$ is now uniformly sampled from $\mathcal{T}$ and therefore does not have a fixed amount of ones and minus ones anymore. The NTRU-HRSS parameter set no longer uses polynomials with a

[2]`https://github.com/jschanck-si/NTRUEncrypt`, last accessed 3rd November 2023

Parameter set	N	q	p	$\mathbf{f} \in$	$\mathbf{g} \in$	$\mathbf{r} \in$
ntruhps2048509	509	2048	3	$\mathcal{T}$	$\mathcal{T}(q/8-2, q/8-2)$	$\mathcal{T}$
ntruhps2048677	667	2048	3	$\mathcal{T}$	$\mathcal{T}(q/8-2, q/8-2)$	$\mathcal{T}$
ntruhps4096821	821	4096	3	$\mathcal{T}$	$\mathcal{T}(q/8-2, q/8-2)$	$\mathcal{T}$
ntruhrss701	701	8192	3	$\mathcal{T}_+$	$(x-1) \cdot \mathbf{v} : \mathbf{v} \in \mathcal{T}_+$	$\mathcal{T}$

Table 3.3 Parameter sets of the NTRU submission [CDH+19].

fixed weight but uniformly samples all polynomials from $\mathcal{T}$, introducing the additional notation of ternary polynomials in $\mathcal{T}_+$, which includes only polynomials with a higher or equal amount of ones than minus ones. This parameter set also allows the usage of index-based multiplication. However, the additional multiplication of $(x-1)$ during the sampling of $\mathbf{g}$ requires additional attention, as after the multiplication, $\mathbf{g}$ is not a ternary polynomial anymore. This problem can be solved by first multiplying the ternary part of $\mathbf{g}$ with the second operand and, in a second step, multiplying it with $(x-1)$. This is possible due to the commutative properties of the used polynomial ring.

As index-based multiplication can be used for both types of parameter sets in the NTRU submission, we have to discuss the influence of these changes on the presented CPA and countermeasures. The CPA is still working as described, but it has to be adapted to the fact that $\mathbf{f}$ is now sampled uniformly from $\mathcal{T}$. In this case, an attacker does not know the amount of non-zero elements in $\mathbf{f}$, and therefore, she does not know the amount of different CPAs needed to find all key index differences. A strategy to cope with this limitation is to perform the attack under the assumption of $d = N$ ones followed by $d = N$ minus ones in $\mathbf{f}$. Then, if an attack on a certain w_i^1 or w_i^{-1} does not result in a significant correlation, she can reason that all respective ones or minus ones have been found. The changes to the algorithm and parameters do not influence the proposed masked implementations, as operations are still linear, and changes to the mask can be computed accordingly. The adaption of the modulus q still allows the use of the parallel masked implementation, as q is still a power of two for all parameter sets, and since $q \leq 2^{16}$, the implicit reduction by the SIMD instructions has no influence on the results. As the ring modulus is still $(x^N - 1)$, the shuffling countermeasure can be used unaltered.

4 Chosen-Chiphertext Attacks on Hamming Quasi-Cyclic (HQC)

This chapter is based on the following publications:

- *Schamberger/Renner/Sigl/Wachter-Zeh: A Power Side-Channel Attack on the CCA2-Secure HQC KEM* published in *Smart Card Research and Advanced Applications (CARDIS)*, 2021 [SRSWZ21]
- *Schamberger/Holzbaur/Renner/Wachter-Zeh/Sigl: A Power Side-Channel Attack on the Reed-Muller Reed-Solomon Version of the HQC Cryptosystem* published in *Post-Quantum Cryptography (PQCrypto)*, 2022 [SHR+22]

The attack strategy on HQC-BCH shown in Section 4.5 and published in [SRSWZ21] was jointly developed with Julian Renner. Additionally, the method to retrieve the final bits of the secret key in Section 4.4.3, as well as the utilization of partial attack results through information set decoding in Section 4.4.4, was also developed by Julian Renner. The attack strategy on HQC-RMRS in Section 4.6 was jointly developed and published in [SHR+22] with Lukas Holzbaur. The formal proof of the attack strategy on HQC-RMRS in Appendix A was developed by Lukas Holzbaur. The discussion on possible countermeasures is partly based on the master thesis of Tim Kaiser [Kai22]. The ideas for the countermeasures in Sections 4.7.2 and 4.7.3 were initially proposed by me and have been jointly developed further during this thesis.

4.1 Introduction

The code-based cryptosystem HQC is a fourth-round candidate in the NIST post-quantum standardization competition. It is a promising candidate in the standardization of an alternative KEM based on a different mathematical problem than the already selected lattice-based CRYSTALS-Kyber [ABD+20]. In [Nat22], NIST summarizes HQC as offering strong security assurances in combination with a mature decryption failure rate analysis while showing reasonable public key and ciphertext sizes. While the first two rounds of the NIST competition were dominated by research on possible performance improvements and optimized implementations, from the beginning of the third round, the focus shifted to the side-channel security of systems, which was encouraged by NIST [Nat20, Nat22]. This section presents contributions in this direction with two chosen-ciphertext attacks through the use of a power side-channel on first the original version (HQC-BCH) and the third round version (HQC-RMRS) with updated error-correcting codes. A first evaluation of countermeasures suggests that a complete masking scheme for the HQC algorithm has to be developed in order to provide a secure implementation against the presented attacks.

Related Work In 2019, Wafo-Tapa et al. presented an ePrint version of a timing side-channel attack on the non-constant-time implementation of HQC-BCH, which was later officially published in [WBB+22]. Their attack consists of a chosen-ciphertext attack that utilizes the variable execution time of the implemented Bose-Chaudhuri-Hocquenghem (BCH) decoder that depends on the number of errors that have to be corrected. They propose a constant-time implementation of the decoder that has been integrated into the reference implementation of HQC-BCH. In [PT20], the authors show an additional timing side-channel on HQC-BCH, where they also utilize the timing variance of the BCH decoder. They use the timing information to reconstruct the spectrum, i.e., the difference in the indices of entries, of the secret key. With the knowledge of the spectrum, they utilize a known search algorithm to determine the actual support of the key. Their attack requires a minimum of 400 million queries to the algorithm. Guo and Johansson [GJ20] show a decryption failure attack on HQC-BCH that uses a pre-computation step to compute inputs that show a significantly higher failure rate. Their overall attack complexity is estimated as 2^{246}, which is slightly below the security level of HQC-BCH-256. As the precomputation has to be performed only once and allows an attack on multiple keys, the authors propose to additionally include the public key in the hash used to derive the randomness for the KEM version of HQC.

In [GHJ$^+$22], the authors present a timing attack on the fixed-weight polynomial sampler of HQC-RMRS. They identify that the rejection sampling approach requires a variable amount of randomness that has to be generated with additional calls to the Pseudo Random Number Generator (PRNG), which influences the execution time. The authors construct an input to the algorithm that requires a large amount of additional randomness and then additively query the algorithm with modifications of this input, where a faster execution time (less randomness) is likely caused by a part of the secret key. A countermeasure to this attack is presented in [Sen21], which is included in the reference implementation of the fourth round HQC submission. In [UXT$^+$21] and [XIU$^+$21a], the authors identify that a misuse attack on the Public Key Encryption Scheme (PKE) version of HQC-BCH published in [BDHD$^+$19, HDV20] can be used to construct a chosen-ciphertext attack on HQC-RMRS. They construct the required oracle for the attack from a power side-channel on the used SHAKE256 extendable-output function [UXT$^+$21] or from directly observing incorrect KEM outputs through a fault attack on the message check of the used variant of the Fujisaki-Okamoto (FO) transformation [XIU$^+$21a]. Nevertheless, these two attacks are shown to be non-functional in this thesis.

Shortly after the publication of the attack on HQC-RMRS presented in this thesis, Goy et al. [GLG22] also published a chosen-ciphertext attack on HQC-RMRS. This attack directly utilizes the Reed-Muller (RM) decoder result by querying every bit of a RM block individually and observing through a power side-channel if the decoder has to correct an additional error or if the error is reduced by one. Although their attack claims to be "less complex" it has several disadvantages compared to the attack presented in this thesis. First, in order to construct the side-channel oracle, Goy et al. have to distinguish several distinct classes corresponding to a fixed amount of errors corrected by the RM code. This is a significantly stronger assumption for a side-channel attack. Additionally, this requires the attacker to correctly identify the execution of the RM decoder in the measurement traces. In contrast, the oracle used in this thesis only has to distinguish two classes. Therefore, the complete side-channel information of the HQC decapsulation, starting from the decoding step, can theoretically be used, as discussed in this thesis. Additionally, their attack requires 240 000 profiling traces in contrast to the 1000 traces used in this thesis, which outweighs the additional traces required during the attack phase of the presented attack.

Contributions This chapter proposes two chosen-ciphertext attacks on HQC as well as its updated third-round version and shows the applicability of the attacks through practical side-channel evaluations. The contributions can be summarized as follows:

- We present the first power side-channel based chosen-ciphertext attack on HQC-BCH that utilizes leakage of the BCH decoder and show practical attack results on an STM32F415 ARM Cortex-M4 microcontroller using 20 000 traces. The attack strategy is able to retrieve a large portion of possible keys, e.g., 93.2 % of HQC-128.

- We provide a complete discussion of the attack, including a method based on linear algebra that is able to retrieve a small part of the secret key that is not included in the input of the decoder and, therefore, not attackable through the side-channel.
- The state of the art for attacks on HQC-RMRS is revisited, and it is shown that the change of the used codes in HQC-RMRS to a Reed-Muller (RM) and Reed-Solomon (RS) code induces a new form of decoder that breaks attack attempts in [UXT+21, XIU+21a].
- As the attack of [UXT+21] is shown to be inapplicable, we again present the first power side-channel attack on HQC-RMRS, including a proof of the attack based on a condition that is fulfilled with very high probability for all parameter sets. Additionally, a discussion of published side-channel oracles for HQC-RMRS is given, and it is shown how and if those oracles can be instantiated for the use with the proposed attack strategy. This includes a practical side-channel evaluation of the RS decoder.
- We discuss multiple countermeasures against the presented attacks and propose a novel countermeasure through the insertion of additional errors during decoding, for which we provide an evaluation of its effectiveness.

Outline In Section 4.2, an introduction to error-correcting codes is given with a focus on concepts and codes used in the different versions of the HQC cryptosystem. The HQC cryptosystem is presented in Section 4.3, including a discussion of the different changes to the system as well as details on the used error-correcting codes. In Section 4.4, the general concept of the attack strategy for both versions of the cryptosystem is presented. The attack on HQC-BCH is described in Section 4.5 and the attack on HQC-RMRS in Section 4.6. Finally, countermeasures are discussed in Section 4.7.

4.2 Error-Correcting Codes

In this section, an overview of the field of error-correcting codes is given. As this field of research is very extensive, the focus is set on an explanation of the concepts and error-correcting codes used in the HQC cryptosystem. For further information, the reader is referred to [LC04] and [Rot06]. This section is structured in accordance with [WZ21].

First, a general introduction to the topic is given in Section 4.2.1. This is followed by an overview of linear block codes and their general encoding and decoding in the Hamming metric in Section 4.2.2. An introduction to RS codes is provided in Section 4.2.3 followed by an introduction to BCH codes in Section 4.2.4, which are subcodes of RS codes. In Section 4.2.5 RM codes are introduced. Finally, Section 4.2.6 describes the concept of concatenated codes, which are used in HQC to achieve the required error correction capability.

4.2.1 Introduction to Error-Correcting Codes

The purpose of error-correcting codes is to store or transmit data in a systematic or structured way given a noisy environment (usually modeled as a communication channel), such that a receiver is able to reconstruct the transmitted message even in the presence of errors. In order to allow this reconstruction, a certain amount of redundancy has to be added to a transmitted message. The challenge in the design of error-correcting codes is to find a tradeoff between a low amount of redundancy and the resulting error-correcting capability (how many errors can be corrected by the code).

In order to encode a message given its information vector $\boldsymbol{u} = (u_0, u_1, \dots, u_{k-1})$ with symbols usually chosen from a finite field $\mathbb{F}_q$, it is mapped using an *encoder* to a codeword vector $\boldsymbol{c}$ of larger length n. More formally, the encoding is defined as the mapping:

$$\boldsymbol{u} \in \mathbb{F}_q^k \mapsto \boldsymbol{c} \in \mathbb{F}_q^n, \text{with } \boldsymbol{c} \in \mathcal{C},$$

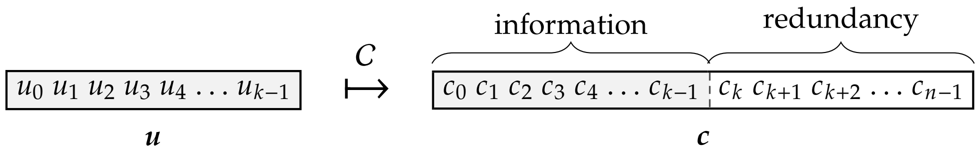

information
redundancy
$u_0\ u_1\ u_2\ u_3\ u_4 \dots u_{k-1}$
$\overset{C}{\longmapsto}$
$c_0\ c_1\ c_2\ c_3\ c_4 \dots c_{k-1}$
$c_k\ c_{k+1}\ c_{k+2} \dots c_{n-1}$
$\boldsymbol{u}$
$\boldsymbol{c}$

Figure 4.1 Example of a systematic encoding of the vector $\boldsymbol{u}$.

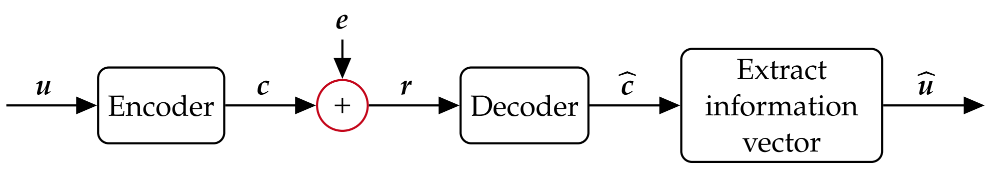


Figure 4.2 Encoding and decoding of a message vector $\boldsymbol{u}$.

where C is a subset of $\mathbb{F}_q^n$ and denotes the used error-correcting code. An example of a codeword in systematic encoding, meaning the first k symbols of the codeword contain the information symbols, is shown in Fig. 4.1. The resulting codeword $\boldsymbol{c}$ can then be transmitted over a channel[1] that can add errors, modeled as the error vector $\boldsymbol{e}$, resulting in the received codeword $\boldsymbol{r} = \boldsymbol{c}+\boldsymbol{e} = (r_0, r_1, r_2, \dots, r_{n-1})$ at the receiver. Due to the used error-correcting code, it is possible to reconstruct (decode) the codeword $\boldsymbol{c}$ from $\boldsymbol{r}$. As the decoding is only possible up to a limited amount of errors, the result after decoding is defined as $\widehat{\boldsymbol{c}}$, where $\widehat{\boldsymbol{c}} = \boldsymbol{c}$ holds for a successful decoding. Finally, from $\widehat{\boldsymbol{c}}$ the information vector $\widehat{\boldsymbol{u}}$ can be reconstructed, while for a systematic encoding the location of the corresponding symbols is known. Frequently, this reconstruction is considered to be part of the decoder and therefore it is assumed that the decoder directly returns the information vector $\widehat{\boldsymbol{u}}$. The process of encoding and decoding is visualized in Fig. 4.2.

4.2.2 Linear Block Codes

Linear block codes are the most prominent subclass of error-correcting codes, as they fit the purpose of data transmission with arbitrary-length messages that can be decomposed into individual message blocks. These blocks are then encoded to their respective codewords independently of the adjacent codewords. A *linear* block code implies that the linear combination of two codewords from the same code C is again a valid codeword. It follows from this property that the all-zero codeword $\mathbf{0} = (0, \dots, 0)$ is a valid codeword for any linear code.

Formally, a linear block code is a k-dimensional linear subspace of the vector space $\mathbb{F}_q^n$ with the minimum Hamming distance d, where q defines the underlying alphabet size. For $q = 2$ the resulting codes are called *binary*. The characteristics of a code can be described using the notation $[n, k, d]_q$ where:

- n is the *length* of the code (i.e., the number of symbols in a codeword),
- k the *dimension* (i.e., the number of symbols to be encoded),
- $n - k$ the number of *redundancy* symbols,
- q^k the *cardinality* (i.e., the number of different codewords),
- d the *minimum distance* of the code.

[1]A channel model is used to model the probability of an error for a certain symbol of the transmitted codeword. In our case we only consider a binary symmetric channel where each bit of a codeword is flipped with probability p independently.

The codes that we consider are defined in the Hamming metric, where the minimum distance d of the code C is the minimum Hamming distance (HD) between two codewords, which can be simplified for linear codes [LC04, Section 3.3]:

$$d = \min_{\substack{x,y \in C \\ x \neq y}} \mathrm{HD}(\boldsymbol{x}, \boldsymbol{y}) = \min_{\substack{x \in C \\ x \neq \boldsymbol{0}}} \mathrm{HW}(\boldsymbol{x}).$$

It has to be noted that the Hamming weight (HW) and the Hamming distance (HD) between two codewords in C is defined as the number of non-zero symbols and the number of coordinates with different symbols, respectively. Only for binary codes ($q = 2$) this corresponds to the definition for calculating the Hamming weight/Hamming distance of the binary representation of a value.

The minimum distance of a code can be used to describe its error-correcting capability. For a code C with minimum distance d and an error vector with $\mathrm{HW}(\boldsymbol{e}) \leq \left\lfloor \frac{d-1}{2} \right\rfloor$, a decoder for the given code can always correct the induced errors. The process of decoding an erroneous codeword that resides in the decoding spheres of the decoder is called *unique* error correction and a decoder using this principle is called *bounded distance* decoder. Such a decoder with its unique decoding radius is visualized in Fig. 4.3a. There is also a second type of decoder called *maximum likelihood*[2] (ML) or *nearest codeword* decoder. In the Hamming metric, such a decoder returns the codeword that has the smallest Hamming distance to the received vector $\boldsymbol{r}$. Formally, an ML decoder returns the codeword

$$\widehat{\boldsymbol{c}} = \arg\max_{c \in C} P(\boldsymbol{r}|\boldsymbol{c}) \,,$$

which is equivalent in the HQC case, i.e., without considering soft information, to minimizing the Hamming distance between $\boldsymbol{r}$ and $\boldsymbol{c}$, resulting in

$$\widehat{\boldsymbol{c}} = \arg\min_{c \in C} \mathrm{HD}(\boldsymbol{r}, \boldsymbol{c}) \,. \tag{4.1}$$

The decoder is visualized in Fig. 4.3b with the borders for the decoding decision marked with gray lines. It can be seen that an ML decoder is able to correct additional errors that exceed the unique decoding radius. Additionally, the decision boundaries show a distinctive margin depending on the "location" of the received word $\boldsymbol{r}$, which reflects the fact that the amount of correctible errors is also influenced by the support of the error, i.e., the positions of the errors in the received word. It is to be noted that for a non-distinct solution, i.e., the received word lies on the decision boundary between codewords, the decision is not specified and is fully dependent on the implementation of the decoder.

Encoding The encoding of an information vector $\boldsymbol{u}$ to a codeword $\boldsymbol{c}$ is done by multiplying $\boldsymbol{u}$ with a $\mathbb{F}_q^{k \times n}$ generator matrix $\boldsymbol{G}$ as

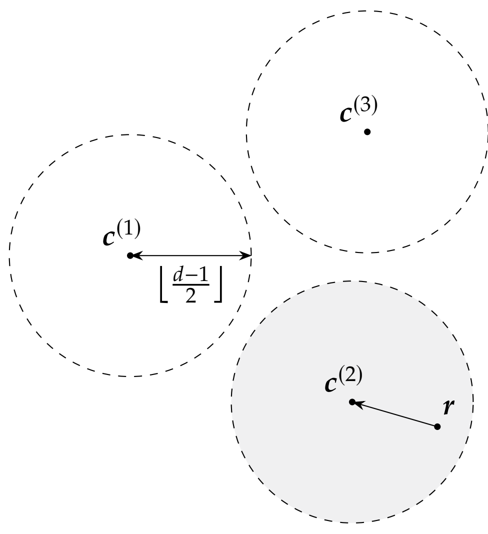
$$\boldsymbol{c} = \boldsymbol{u} \cdot \boldsymbol{G} \,.$$

In the case of a desired systematic encoding, $\boldsymbol{G}$ can be transferred into its systematic form $\boldsymbol{G}_{sys} = \begin{pmatrix} \boldsymbol{I}_k & \boldsymbol{A} \end{pmatrix}$, where $\boldsymbol{A}$ is calculated through Gaussian elimination and $\boldsymbol{I}_k$ denotes the $k \times k$ identity matrix. A systematic encoding of $\boldsymbol{u}$ is therefore defined as

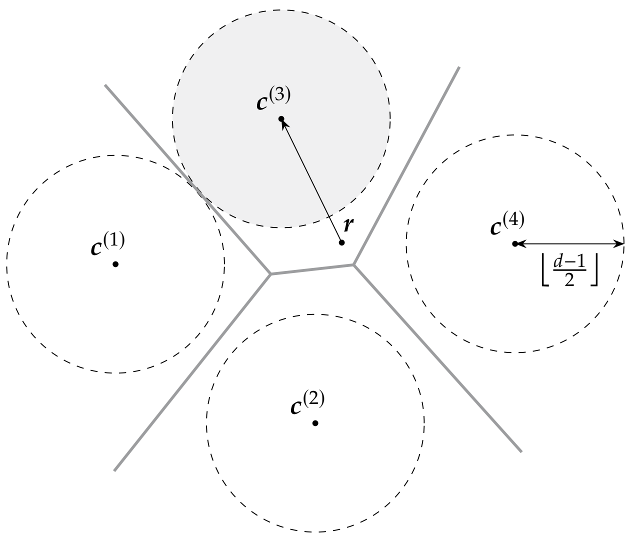
$$\boldsymbol{c} = \boldsymbol{u} \cdot \boldsymbol{G}_{sys} = \boldsymbol{u} \cdot \begin{pmatrix} \boldsymbol{I}_k & \boldsymbol{A} \end{pmatrix} = \begin{pmatrix} \boldsymbol{u} & \boldsymbol{u} \cdot \boldsymbol{A} \end{pmatrix} \,. \tag{4.2}$$

Please note that $\boldsymbol{G}_{sys}$ is still a valid generator matrix of C, but only the mapping to codewords is changed. An example of an encoding using a generator matrix in systematic form is given in Example 4.2.1.

[2]In the case of HQC, where a channel is assumed that has a probability for an error smaller than 0.5, i.e., fewer errors are more likely than more errors, a nearest codeword decoder is also a maximum likelihood decoder.



(a) Bounded distance decoding



(b) Maximum likelihood (ML) decoding

Figure 4.3 Visualization of the different decoding methods. Adapted from [WZ21].

Example 4.2.1: Systematic Encoding of a Single-Parity Check Code. Adapted from [WZ21].

A single-parity check code is a $[n, n-1, 2]_2$ binary code that computes the parity of the input vector, in this case, a binary value of size $k = n-1$. As an example, a $[5, 4, 2]_2$ code is considered, which is used to encode the information vector $\boldsymbol{u} = (0110)$. Using the generator matrix of this code in systematic form, the resulting codeword is

$$\boldsymbol{c} = (0110) \cdot \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} = (01100)\,.$$

It can be seen that the first parts of $\boldsymbol{G}_{sys}$ consist of $\boldsymbol{I}_{n-1}$ and therefore the first k entries of the codeword correspond to $\boldsymbol{u}$.

Parity-Check Matrix The decoding process of an error-correcting code is highly dependent on the used code, but there is a particular class of decoder that is used by most of the codes (BCH and RS) in HQC. This class of decoders is based on computing the *syndrome* of a vector that is to be decoded using a so-called *parity-check matrix* of the code C. A parity-check matrix $\boldsymbol{H}$ is an $(n-k) \times n$ matrix in $\mathbb{F}_q$ that fulfills the property that for any codeword $\boldsymbol{c} \in C$ it holds:

$$\boldsymbol{c} \cdot \boldsymbol{H}^T = \boldsymbol{0}\,. \tag{4.3}$$

Using this definition, the syndrome $\boldsymbol{s} \in \mathbb{F}_q^{n-k}$ of a vector $\boldsymbol{r} \in \mathbb{F}_q^n$ is defined as the result of the multiplication of this vector and the parity-check matrix as

$$\boldsymbol{s} = \boldsymbol{r} \cdot \boldsymbol{H}^T\,. \tag{4.4}$$

If $\boldsymbol{s} = \boldsymbol{0}$, it follows that $\boldsymbol{r}$ is a valid codeword, and a syndrome that is not equal to zero indicates that an error has occurred. This distinction makes it possible to detect errors and to construct a decoder that uses the information contained in the syndrome.

Example 4.2.2: Parity-Check Matrix and Syndrome of a Single-Parity Check Code

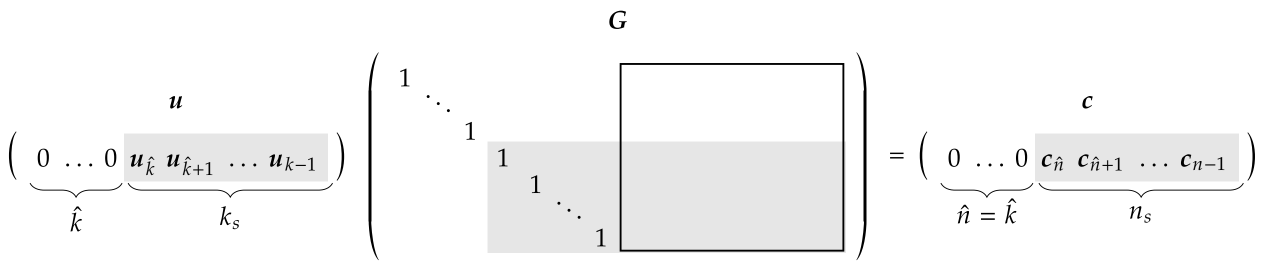
Using the $[5,4,2]_2$ binary single-parity check code from Example 4.2.1, we can calculate the syndrome for a correct codeword and an erroneous codeword. The generator and parity-check matrix of this code is given by

$$\boldsymbol{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}, \quad \boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

For a correct codeword $\boldsymbol{c} = (01100)$, the computation of the syndrome shows that it is a codeword of C, while for an erroneous vector $\boldsymbol{c}_{err} = (11100)$ the syndrome indicates an error:

$$\boldsymbol{s} = (01100) \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = 0, \quad \boldsymbol{s} = (11100) \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = 1\,.$$

Shortened Codes In some use cases, it can be desired to encode a word that is smaller than the dimension k of the code. An example of this is the fixed size of a shared secret during a key exchange using a KEM that normally corresponds to the security level of the algorithm, e.g., 128 bit. This can be achieved through *shortening* of the used code, such that the new dimension k_s is the desired length of the word to be encoded. The process works with a systematic encoding by first identifying all codewords with a "0" at the first positions and removing these zeros from the codewords[3]. This also implies to reduce the length of the codewords to reflect this change. Formally, the shortening of a $[n,k,d]_q$ code results in a $[n_s,k_s,d_s]_q$ code with $k_s = k - \hat{k}$, $n_s = n - \hat{n}$, and $d_s \geq d$ where $\hat{k}, \hat{n}$ correspond to the size of the removed symbols and $\hat{k} = \hat{n}$. The shortening process is visualized in Fig. 4.4.



G
u
0 ... 0 $u_{\hat{k}}$ $u_{\hat{k}+1}$ ... u_{k-1}
$\hat{k}$
k_s
1
1
1
1
1
=
c
0 ... 0 $c_{\hat{n}}$ $c_{\hat{n}+1}$ ... c_{n-1}
$\hat{n} = \hat{k}$
n_s

Figure 4.4 Visualization of the shortening of a given code adapted from [WZ21]. The resulting shortened generator matrix is indicated in gray together with the corresponding input and resulting codeword.

Cyclic Codes Some codes fulfill the property of cyclicity, which states that every cyclic shift of a codeword is again a valid codeword. For cylic codes, a codeword vector $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ can be represented with the corresponding polynomial $c(x) = c_0 + c_1x + c_2x^2 + \ldots + c_{n-1}x^{n-1}$ which is an element of the ring $\frac{\mathbb{F}_q[x]}{x^n-1}$. This implies that a multiplication with x corresponds to a cyclic shift of one position as

$$x \cdot c(x) = x \cdot (c_0 + c_1x + \ldots + c_{n-1}x^{n-1}) = c_{n-1} + c_0x + c_1x^2 + \ldots + c_{n-2}x^{n-1} \mod (x^n - 1)\,.$$

[3]This does not imply that the first position $\boldsymbol{u}_{\hat{k}}$ of the shortened $\boldsymbol{u}$ has to be zero.

For cyclic codes there exists a *generator polynomial* $g(x)$ that divides all codewords $c(x) \in \mathbb{F}_q[x]$ of degree $n-1$ without a remainder polynomial [Rot06, Section 8.2]:

$$c(x) \in C \iff g(x) \mid c(x) \, .$$

This implies that for cyclic codes the encoding of an information vector $\boldsymbol{u}$ in its polynomial representation $u(x)$, with $\deg(u(x)) < k$, can be done through multiplication with the generator polynomial $g(x)$ of degree $n-k$:

$$c(x) = u(x) \cdot g(x) \mod (x^n - 1) \, . \tag{4.5}$$

This is an advantage for practical implementations of cyclic codes, as there exist optimized algorithms for polynomial multiplication or even dedicated hardware constructions like a LFSR [LC04]. Analogously, the parity-check polynomial $h(x)$ corresponding to $g(x)$ can be computed as

$$h(x) = \frac{x^n - 1}{g(x)} \mod (x^n - 1) \, . \tag{4.6}$$

The corresponding generator and parity check matrix to a given generator and parity-check polynomial can be constructed by cyclic shifts of the respective polynomial:

$$\boldsymbol{G} = \begin{pmatrix} g_0 & g_1 & \cdots & g_{n-k} & & & & \\ & g_0 & g_1 & \cdots & g_{n-k} & & & \\ & & \ddots & \ddots & & \ddots & \ddots & \\ & & & g_0 & & g_1 & \cdots & g_{n-k} \end{pmatrix}, \quad \boldsymbol{H} = \begin{pmatrix} h_k & h_{k-1} & \cdots & h_0 & & & & \\ & h_k & h_{k-1} & \cdots & h_0 & & & \\ & & \ddots & \ddots & \ddots & \ddots & & \\ & & & & h_k & h_{k-1} & \cdots & h_0 \end{pmatrix} .$$

4.2.3 Reed-Solomon (RS) Codes

Reed-Solomon (RS) codes are a prominent class of block codes as they achieve optimal error correction capabilities[4] while being efficient to encode and decode. Additionally, since they are defined for symbols over $\mathbb{F}_q$, a particular advantage is the possibility of correcting consecutive bit errors, so-called burst errors. An $[n, k, d]_q$ RS code has the following parameters:

- Block length: $n = q - 1$
- Minimum distance: $d = n - k + 1$
- Number of parity check symbols: $n - k = 2\delta$, where δ is the error correcting capability defined as $\delta = \left\lfloor \frac{d-1}{2} \right\rfloor$
- Dimension: $k = q - 1 - 2\delta$

In the following, only *primitive* RS codes are described as only this subclass of RS codes is used in HQC. A primitive RS code is usually defined over an extension field of $\mathbb{F}_2$ with extension degree m using a primitive element $\alpha \in \mathbb{F}_{2^m}$ that is used to generate the field. Additionally, the code length n must be dividable by $2^m - 1$ and α has to be of order n, which means that n is the smallest integer such that $\alpha^n = 1$. This results in the following generator and parity check matrices of a primitive RS code based on the primitive element α:

$$\boldsymbol{G}_{\text{RS}} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{(k-1)} & \alpha^{2(k-1)} & \cdots & \alpha^{(n-1)(k-1)} \end{pmatrix}, \quad \boldsymbol{H}_{\text{RS}} = \begin{pmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{(n-1)} \\ 1 & \alpha^2 & (\alpha^2)^2 & \cdots & (\alpha^2)^{(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{(n-k)} & (\alpha^{(n-k)})^2 & \cdots & (\alpha^{(n-k)})^{(n-1)} \end{pmatrix} .$$

[4]More formally, they achieve the Singleton bound and are, therefore, maximum distance separable codes.

As a primitive RS code is cyclic, a generator polynomial can also be used to describe its encoding. The construction of the parity check matrix $\boldsymbol{H}_{\mathrm{RS}}$ and the fact that Eq. (4.3) holds implies that every codeword has the sequential powers of α as its roots, i.e., $c(\alpha) = c(\alpha^2) = \ldots = c(\alpha^{(n-k)}) = 0$. Thus, the generator polynomial $g(x)$ of a primitive RS code is given by

$$g(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^3) \ldots (x - \alpha^{(n-k)}) \,. \tag{4.7}$$

Using Eq. (4.6), the corresponding parity check polynomial is defined as

$$h(x) = \frac{x^n - 1}{g(x)} = (x - \alpha^{(n-k+1)})(x - \alpha^{(n-k+2)})(x - \alpha^{(n-k+3)}) \ldots (x - \alpha^{(n)}) \,. \tag{4.8}$$

Encoding The encoding of a message using RS codes can be done through the multiplication of the message polynomial with the generator polynomial (c.f. Eq. (4.5)). Nevertheless, this produces an encoding in non-systematic form, which requires an additional division of the decoding result by $g(x)$ to retrieve the message from $\boldsymbol{u}$. This downside is removed by a systematic encoding, which works for RS codes as follows:

- First, the message polynomial $u(x)$ is multiplied by $x^{(n-k)}$.
- Then the remainder $b(x)$ for the division of $x^{(n-k)} \cdot u(x)$ with the generator polynomial is computed.
- In a final step the codeword is obtained as $c(x) = b(x) + x^{(n-k)} \cdot u(x)$.

Please note that this systematic encoding defines the rightmost symbols or the coefficients that are equal or higher than $x^{(n-1)}$ as information symbols and the leftmost symbols or coefficients smaller than $x^{(n-1)}$ as the redundancy. This stands in contrast to the definition in Eq. (4.2) and has to be considered during the retrieval of the message after decoding.

Unique Syndrome-Based Decoding The most prominent method for decoding RS codes is based on computing the syndromes of the received word $\boldsymbol{r}$ that should be decoded. In order to describe the decoding process the error polynomial $e(x) = e_0 + e_1 x + \ldots + e_{n-1} x^{(n-1)}$ has to be introduced, which is defined as

$$e(x) = r(x) - c(x) \,. \tag{4.9}$$

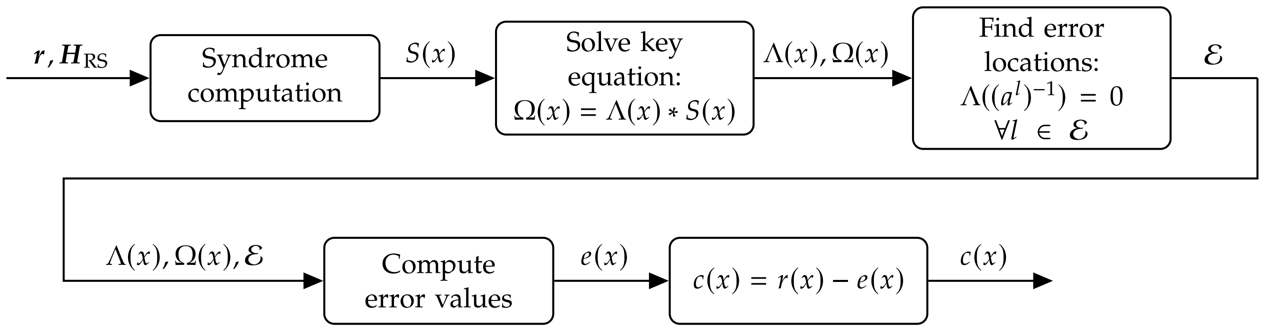
According to $e(x)$ the set of error locations $\mathcal{E} = \operatorname{supp}(\boldsymbol{e})$ is defined, which contains all the nonzero coefficients of $e(x)$, i.e., $\mathcal{E} = \{i : e_i \neq 0, i = 0, \ldots, n-1\}$. The described decoder results in a unique decoding result, which implies $|\mathcal{E}| = \mathrm{HW}(\boldsymbol{e}) \leq \left\lfloor \frac{d-1}{2} \right\rfloor$ (c.f. Section 4.2.2). The goal of the decoding process is to determine the error polynomial $e(x)$ in order to compute the codeword $c(x)$ using Eq. (4.9). The decoding consists of the following steps[5], which are additionally visualized in Fig. 4.5:

1) **Syndrome Computation**

   The syndrome with its coefficients $\boldsymbol{s} = (s_0, s_1, \ldots, s_{d-2}) \in \mathbb{F}_q^{d-1}$ is calculated by multiplication of the received word with the parity check matrix as $\boldsymbol{s} = \boldsymbol{r} \cdot \boldsymbol{H}_{\mathrm{RS}}^T$. Please note that the syndrome is only dependent on the error $\boldsymbol{e}$ of the received word, since $\boldsymbol{c} \cdot \boldsymbol{H}_{\mathrm{RS}}^T = \boldsymbol{0}$ and therefore:

$$\boldsymbol{s} = \boldsymbol{r} \cdot \boldsymbol{H}_{\mathrm{RS}}^T = (\boldsymbol{c} + \boldsymbol{e}) \cdot \boldsymbol{H}_{\mathrm{RS}}^T = \underbrace{\boldsymbol{c} \cdot \boldsymbol{H}_{\mathrm{RS}}^T}_{\boldsymbol{0}} + \boldsymbol{e} \cdot \boldsymbol{H}_{\mathrm{RS}}^T \,. \tag{4.10}$$

[5]Please note we describe the decoder for primitive RS codes, which implies that the code locators are powers of a primitive element $\alpha \in \mathbb{F}_q$ and therefore $\alpha_i = \alpha^i$. In literature [LC04, Rot06], the decoder is usually described for generalized RS codes using the general α_i. These generalized RS codes additionally introduce non-zero column multipliers $v_i = v_0, v_1, \ldots, v_{n-1} \in \mathbb{F}_q$ that are used to scale the columns of the parity check matrix. For primitive RS codes $v_i = \alpha_i = \alpha^i$. For consistency, all α_i and v_i are substituted in the presented formulas of the decoder.



r, H_{RS}
Syndrome computation
$S(x)$
Solve key equation: $\Omega(x) = \Lambda(x) * S(x)$
$\Lambda(x), \Omega(x)$
Find error locations: $\Lambda((a^l)^{-1}) = 0$ $\forall l \in \mathcal{E}$
$\mathcal{E}$
$\Lambda(x), \Omega(x), \mathcal{E}$
Compute error values
$e(x)$
$c(x) = r(x) - e(x)$
$c(x)$

Figure 4.5 Syndrome-based unique decoding of primitive RS codes.

The individual syndrome coefficients for $i = 0, \dots, d-2$ are

$$s_i = \sum_{j=0}^{n-1} r_j \alpha^j \alpha^{j \cdot i} = \sum_{j=0}^{n-1} e_j \alpha^j \alpha^{j \cdot i} = \sum_{j \in \mathcal{E}} e_j \alpha^j \alpha^{j \cdot i} .$$

2) **Solving the Key Equation**

The key equation states the relation between the error locator polynomial $\Lambda(x)$, which defines the locations of errors, and the error evaluator polynomial $\Omega(x)$, which defines the error values, as

$$\Omega(x) = \Lambda(x) \cdot S(x) \mod (x^{d-1}), \text{with } \deg(\Omega(x)) < \deg(\Lambda(x)) \leq \left\lfloor \frac{d-1}{2} \right\rfloor . \tag{4.11}$$

The error error locator polynomial $\Lambda(x)$ is defined by

$$\Lambda(x) = \prod_{i \in \mathcal{E}} (1 - \alpha^i x) , \tag{4.12}$$

with roots $(\alpha^l)^{-1}$ for $l \in \mathcal{E}$, which means that only if l is an erroneous position it holds that $\Lambda((\alpha^l)^{-1}) = 0$. The error evaluator polynomial $\Omega(x)$ is defined by

$$\Omega(x) = \sum_{i \in \mathcal{E}} e_i \alpha^i \prod_{j \in \mathcal{E} \setminus \{i\}} (1 - \alpha^j x) . \tag{4.13}$$

For a known amount of errors t in $\boldsymbol{e}$, i.e., $t = \mathrm{HW}(\boldsymbol{e}) = |\mathcal{E}|$, the key equation (Eq. (4.11)) is equal to the following system of equations:

$$\begin{pmatrix} s_0 & 0 & 0 & \dots & 0 \\ s_1 & s_0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ s_{t-1} & s_{t-2} & \dots & s_0 & 0 \\ \hline s_t & s_{t-1} & \dots & s_1 & s_0 \\ s_{t+1} & s_t & \dots & s_2 & s_1 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ s_{d-2} & s_{d-3} & \dots & s_{d-t-1} & s_{d-t-2} \end{pmatrix} \cdot \begin{pmatrix} \Lambda_0 \\ \Lambda_1 \\ \vdots \\ \Lambda_t \end{pmatrix} = \begin{pmatrix} \Omega_0 \\ \Omega_1 \\ \vdots \\ \Omega_{t-1} \\ \hline 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} . \tag{4.14}$$

The last equations in Eq. (4.14) for $d - 1 - t \geq t$ do not depend on $\Omega(x)$ and therefore, this linear system of equations can be solved in order to determine $\Lambda(x)$. In practice, the number of errors in $\boldsymbol{e}$ is unknown during decoding, and therefore t must be determined. This can be done by subsequently checking the rank of the following matrix with syndrome coefficients

$$\boldsymbol{S}_v = \begin{pmatrix} s_{v-1} & \dots & s_1 & s_0 \\ s_v & \dots & s_2 & s_1 \\ \vdots & \ddots & \vdots & \vdots \\ s_{2v-2} & \dots & s_v & s_{v-1} \end{pmatrix},$$

for $v = \lfloor \frac{d-1}{2} \rfloor, \dots, 1$ that only has full rank, i.e., is invertible, if $v = t$ [Pet60]. Please note that there are more efficient algorithms to compute $\Lambda(x)$, like the Berlekamp-Massey algorithm, that does not rely on the knowledge of t. For a detailed discussion on this algorithm and other ways to solve the key equation, the reader is referred to [Rot06, Chapter 6]. Finally, $\Omega(x)$ can be computed using Eq. (4.11) or inserting the now known $\Lambda(x)$ in Eq. (4.14).

3) **Determining $\mathcal{E}$ from $\Lambda(x)$**

   The definition of $\Lambda(x)$ in Eq. (4.12) states that $\Lambda((\alpha^l)^{-1}) = 0$ for all $l \in \mathcal{E}$. Therefore, the error locations $\mathcal{E}$ can be found through finding the roots of $\Lambda(x)$. As the number of possible roots (the inverses of all α^l) is limited by n, it is possible to perform a brute-force search to identify them. An efficient way to perform this brute-force search in practice is the so-called *chien search.* For a detailed discussion, the reader is again referred to the literature [LC04, Chapter 7].

4) **Finding the Error Values**

   The error values, i.e., the individual coefficients e_i of $e(x)$, can be computed using *Froney's formula* that states:

$$e_i = \frac{1}{\alpha^i} \cdot \frac{\Omega((\alpha^i)^{-1})}{\prod_{j \in \mathcal{E} \setminus \{i\}} (1 - \alpha^j (\alpha^i)^{-1})} \tag{4.15}$$

5) **Retrieve the Codeword $c(x)$**

   The codeword is computed as

$$c(x) = r(x) - e(x) \; .$$

4.2.4 Bose-Chaudhuri-Hocquenghem (BCH) Codes

The following section describes *primitive* Bose-Chaudhuri-Hocquenghem (BCH) codes that are also an important class of cyclic error-correcting codes. Especially, we focus on binary BCH codes used in the HQC cryptosystem. For primitive BCH codes, the block length is defined as $n = q^m - 1 = 2^m - 1$, as for binary BCH codes $q = 2$. Given a primitive element α of $\mathbb{F}_{2^m}$ and $\mathcal{M} = C_{i_1} \cup C_{i_2} \cup \dots \cup C_{i_l}$ for $l \geq 1$ with the different C_i being the cyclotomic cosets with respect to n, the generator polynomial of the corresponding primitive BCH code is defined as

$$g(x) = \prod_{i \in \mathcal{M}} (x - \alpha^i) = \sum_{i=0}^{|\mathcal{M}|} g_i x^i, \text{with } g_i \in \mathbb{F}_2 \; . \tag{4.16}$$

The desired minimum distance d of the code is defined as the smallest integer of all cyclotomic cosets that is not in $\mathcal{M}$[6]. The dimension of the code is defined as $k = n - |\mathcal{M}|$. A cyclotomic coset with respect to an integer $n = q^m - 1$ is defined as

$$C_i = \{i \cdot q^j \bmod n, \; j = 0, 1, \dots, n_i - 1\}, \tag{4.17}$$

where n_i is the smallest positive integer such that $i \cdot q^{n_i} = i$. For a detailed explanation, the reader is referred to [Rot06, Chapter 7.5].

[6]Another definition is that d is the highest number of consecutive numbers (starting from 1) in $\mathcal{M}$ added by one.

It is important to note that although the primitive element α is defined in $\mathbb{F}_{2^m}$, the described BCH codes are binary, i.e., the symbols of resulting codewords are in $\mathbb{F}_2$, and therefore the codewords are elements of $\mathbb{F}_2^n$. This is due to the construction of the generator polynomial from the cyclotomic cosets and their corresponding minimal polynomials such that $g(x)$ has coefficients in $\mathbb{F}_2$. This implies that a binary primitive BCH code is a subfield-subcode of the respective primitive RS code of the same block length.

Encoding Since BCH codes are cyclic codes, the encoding of a message can be done by representing it as a message polynomial and multiplying it with the generator polynomial as defined in Eq. (4.5). The systematic encoding of BCH codes works analog to primitive RS codes by first multiplying the message polynomial $u(x)$ by $x^{(n-k)}$. In a second step, the remainder $b(x)$ of the division $(x^{(n-k)} \cdot u(x))/g(x)$ is computed. Finally, the codeword is obtained as $c(x) = b(x) + x^{(n-k)} \cdot u(x)$.

Decoding As BCH codes can be seen as RS codes with binary coefficients, they can be decoded in the same way as a RS code, which is described in Section 4.2.3. For binary BCH codes, the decoding is even simplified since for binary coefficients, it is sufficient to determine the locations of errors $\mathcal{E}$ from $\Lambda(x)$ and flipping the respective bits in the received word. This allows skipping the computation of the error values during decoding (Step 4 of the RS decoding). The decoding of BCH codes is visualized in Fig. 4.6.

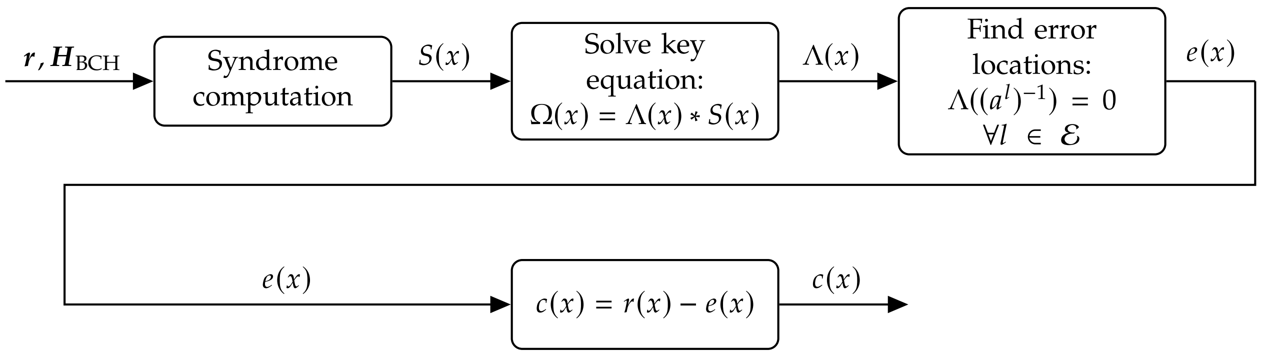

r, H_{BCH}
Syndrome computation
$S(x)$
Solve key equation: $\Omega(x) = \Lambda(x) * S(x)$
$\Lambda(x)$
Find error locations: $\Lambda((a^l)^{-1}) = 0$ $\forall l \in \mathcal{E}$
$e(x)$
$e(x)$
$c(x) = r(x) - e(x)$
$c(x)$

Figure 4.6 Syndrome-based unique decoding of binary primitive BCH codes analogous to the decoding of primitive RS codes.

4.2.5 Reed-Muller (RM) Codes

In this section, Reed-Muller (RM) codes are described, which are among the oldest error-correcting codes. Their main advantage is their simple construction, and their decoding can be performed relatively easily in contrast to other code classes through the use of majority logic. The section is loosely based on [MS77, Chapter 13,14] and [LC04], with a focus on binary first-order RM codes, and the reader is referred to these references for a more in-depth discussion. A binary RM code of order r is denoted as $\mathcal{RM}(r, m)$ and has the following properties:

- Block length: $n = 2^m$
- Dimension: $k = \sum_{i=0}^{r} = \binom{m}{i}$
- Minimum distance: $d = 2^{m-r}$

The generator matrix of an $\mathcal{RM}(r, m)$ code is given by the binary vectors $\boldsymbol{p}_i \in \mathbb{F}_2^m$ for $1 \leq i \leq m$ of the form

$$\boldsymbol{p}_i = (\underbrace{0\ldots0}_{2^{(i-1)}}, \underbrace{1\ldots1}_{2^{(i-1)}}, \underbrace{0\ldots0}_{2^{(i-1)}}, \ldots, \underbrace{1\ldots1}_{2^{(i-1)}}), \tag{4.18}$$

where the amount of alternating all-zero and all-one sequences of size $2^{(i-1)}$ is given by $2^{(m-i+1)}$. Additionaly, $\boldsymbol{p}_0$ is defined as the vector in $\mathbb{F}_2^m$ with all ones, i.e., $\boldsymbol{p}_0 = (1, 1, \ldots, 1)$. These vectors can also be interpreted as polynomials for which operations, like addition or multiplication, are obtained from a boolean function $f(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_m)$, which implies that, e.g., $\boldsymbol{p}_1 + \boldsymbol{p}_2$, is defined as the *exclusive or* and $\boldsymbol{p}_1 \cdot \boldsymbol{p}_2$ as the binary *and* of the individual coefficients. The generator matrix $\boldsymbol{G}_{\text{RM}}$ of an $\mathcal{RM}(r, m)$ code is given by the set of vectors

$$\boldsymbol{G}_{\text{RM}}(r, m) = \{\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots \boldsymbol{p}_m, \boldsymbol{p}_1\boldsymbol{p}_2, \boldsymbol{p}_1\boldsymbol{p}_3, \ldots, \boldsymbol{p}_{m-1}\boldsymbol{p}_m, \ldots, \text{up to all products of degree r}\} \quad (4.19)$$

arranged as the rows of the generator matrix. An example for the generator matrix of the first- and second-order RM codes for $m = 4$ is shown in Example 4.2.3.

Example 4.2.3: Generator Matrix $\boldsymbol{G}_{\text{RM}}$ of $\mathcal{RM}(1, 4)$ and $\mathcal{RM}(2, 4)$

The $\mathcal{RM}(1, 4)$ code corresponds to the $[16, 5, 8]_2$ code. Therefore, the corresponding generator matrix is of size $F_2^{5\times16}$ (c.f. Section 4.2.2). A first-order RM code ($r = 1$) implies that only the elements with degree ≤ 1 of the set defined in Eq. (4.19) are considered for the generator matrix. This results in the set $\{\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4\}$ and the usage of Eq. (4.18) defines the generator matrix as

$$\boldsymbol{G}_{\mathcal{RM}(1,4)} = \left(\begin{array}{c} \boldsymbol{p}_0 \\ \hline \boldsymbol{p}_4 \\ \boldsymbol{p}_3 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_1 \end{array}\right) = \left(\begin{array}{cccccccccccccccc} 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\ \hline 0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1 \\ 0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1 \\ 0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1 \\ 0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \end{array}\right).$$

In contrast, the $\mathcal{RM}(2, 4)$ code corresponds to the $[16, 11, 4]_2$ code and therefore the generator matrix is of size $F_2^{11\times16}$. It is a second-order RM code, and therefore all elements of degree ≤ 2 defined in Eq. (4.19) specify the generator matrix. This results in the set $\{\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4, \boldsymbol{p}_1\boldsymbol{p}_2, \boldsymbol{p}_1\boldsymbol{p}_3, \boldsymbol{p}_1\boldsymbol{p}_4, \boldsymbol{p}_2\boldsymbol{p}_3, \boldsymbol{p}_2\boldsymbol{p}_4, \boldsymbol{p}_3\boldsymbol{p}_4\}$ that define the generator matrix as

$$\boldsymbol{G}_{\mathcal{RM}(2,4)} = \left(\begin{array}{c} \boldsymbol{p}_0 \\ \hline \boldsymbol{p}_4 \\ \boldsymbol{p}_3 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_1 \\ \hline \boldsymbol{p}_3\boldsymbol{p}_4 \\ \boldsymbol{p}_2\boldsymbol{p}_4 \\ \boldsymbol{p}_1\boldsymbol{p}_4 \\ \boldsymbol{p}_2\boldsymbol{p}_3 \\ \boldsymbol{p}_1\boldsymbol{p}_3 \\ \boldsymbol{p}_1\boldsymbol{p}_2 \end{array}\right) = \left(\begin{array}{cccccccccccccccc} 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\ \hline 0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1 \\ 0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1 \\ 0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1 \\ 0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \\ \hline 0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1 \\ 0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1 \\ 0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1 \\ 0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1 \\ 0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1 \end{array}\right).$$

Note that the generator matrices show a lot of structure and that the elements of degree two (e.g., $\boldsymbol{p}_3\boldsymbol{p}_4$) indeed are the result of a binary *and* of their individual vector components. Additionally, the weight of the individual $\boldsymbol{p}_i$ except $\boldsymbol{p}_0$ is given by $\text{HW}(\boldsymbol{p}_i) = 2^{m-1}$ and in contrast the elements of degree two, e.g., $\boldsymbol{p}_3\boldsymbol{p}_4$, are of weight $\text{HW}(\boldsymbol{p}_i) = 2^{m-2}$.

In literature there is also another definition of RM codes [MS77], that defines the code as all vectors defined by an evaluation of the boolean function $f(\boldsymbol{x})$ with $\boldsymbol{x} = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_m)$, where $f(\boldsymbol{x})$ is a polynomial of degree at most r and the coefficients of f are in $\mathbb{F}_2$. More formally the binary $\mathcal{RM}(r, m)$ code is defined as

$$\mathcal{RM}(r, m) = \left\{\text{ev}(f(\boldsymbol{x})) \,\middle|\, f(\boldsymbol{x}) \in \mathbb{F}_2[\boldsymbol{x}], \deg(f(\boldsymbol{x})) \leq r\right\}, \quad (4.20)$$

where ev($f(\boldsymbol{x})$) denotes the evaluation of the polynomial $f(\boldsymbol{x})$ for all possible coefficients. An example using this definition is given in Example 4.2.4. In this example, it can be seen that the polynomial that defines $f(\boldsymbol{x})$ can be a single element of $\boldsymbol{x}$ and therefore, the different elements of $\boldsymbol{x}$ are analogously called polynomial in the later described attack on the HQC system (Section 4.6.2).

Example 4.2.4: Definition of $\mathcal{RM}(1,3)$ as ev($f(\boldsymbol{x})$)

For the first-order $\mathcal{RM}(1,3)$ code the degree of $f(\boldsymbol{x})$ is given as $\deg(f(\boldsymbol{x})) \leq 1$. The elements of $\boldsymbol{x} = (\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3)$ are generated according to Eq. (4.18) which results in

$$\begin{aligned}\boldsymbol{p}_3 &= (0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 1)\\ \boldsymbol{p}_2 &= (0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1)\ .\\ \boldsymbol{p}_1 &= (0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1)\end{aligned}$$

The resulting polynomial of degree one is

$$f(\boldsymbol{x}) = a_0\mathbf{1} + a_1\boldsymbol{p}_1 + a_2\boldsymbol{p}_2 + a_3\boldsymbol{p}_3, \text{with } a_i \in \mathbb{F}_2\ .$$

Now the codewords of $\mathcal{RM}(1,3)$ are constructed as ev($f(\boldsymbol{x})$) which results in the $2^k = 16$ codewords

$$\begin{aligned}
\mathbf{0}\ (a_0=0,a_1=0,a_2=0,a_3=0) &= (0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0)\\
\boldsymbol{p}_3\ (a_0=0,a_1=0,a_2=0,a_3=1) &= (0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 1)\\
\boldsymbol{p}_2\ (a_0=0,a_1=0,a_2=1,a_3=0) &= (0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1)\\
\boldsymbol{p}_2+\boldsymbol{p}_3\ (a_0=0,a_1=0,a_2=1,a_3=1) &= (0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0)\\
\boldsymbol{p}_1\ (a_0=0,a_1=1,a_2=0,a_3=0) &= (0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1)\\
\boldsymbol{p}_1+\boldsymbol{p}_3\ (a_0=0,a_1=1,a_2=0,a_3=1) &= (0\ \ 1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 1\ \ 0)\\
\boldsymbol{p}_1+\boldsymbol{p}_2\ (a_0=0,a_1=1,a_2=1,a_3=0) &= (0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0)\\
\boldsymbol{p}_1+\boldsymbol{p}_2+\boldsymbol{p}_3\ (a_0=0,a_1=1,a_2=1,a_3=1) &= (0\ \ 1\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 1)\\
\mathbf{1}\ (a_0=1,a_1=0,a_2=0,a_3=0) &= (1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1)\ .\\
\mathbf{1}+\boldsymbol{p}_3\ (a_0=1,a_1=0,a_2=0,a_3=1) &= (1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0)\\
\mathbf{1}+\boldsymbol{p}_2\ (a_0=1,a_1=0,a_2=1,a_3=0) &= (1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0)\\
\mathbf{1}+\boldsymbol{p}_2+\boldsymbol{p}_3\ (a_0=1,a_1=0,a_2=1,a_3=1) &= (1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1)\\
\mathbf{1}+\boldsymbol{p}_1\ (a_0=1,a_1=1,a_2=0,a_3=0) &= (1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0)\\
\mathbf{1}+\boldsymbol{p}_1+\boldsymbol{p}_3\ (a_0=1,a_1=1,a_2=0,a_3=1) &= (1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1)\\
\mathbf{1}+\boldsymbol{p}_1+\boldsymbol{p}_2\ (a_0=1,a_1=1,a_2=1,a_3=0) &= (1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1)\\
\mathbf{1}+\boldsymbol{p}_1+\boldsymbol{p}_2+\boldsymbol{p}_3\ (a_0=1,a_1=1,a_2=1,a_3=1) &= (1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\ \ 0)
\end{aligned}$$

Note that the addition of two elements of $\boldsymbol{x}$ is defined by the boolean function as an *exclusive or* of the coefficients.

Encoding The encoding of binary RM codes can be done in the classical way by a multiplication of the information vector $\boldsymbol{u}$ with the generator matrix $\boldsymbol{G}_{\text{RM}}$. Alternatively, the evaluation of the boolean function $f(\boldsymbol{x})$, as defined in Eq. (4.20), can be used for the encoding. This is done by using the coefficients of $\boldsymbol{u}$ as the coefficients a_i of this polynomial $f(\boldsymbol{x})$. An example is given in Example 4.2.5.

Example 4.2.5: Encoding of $\boldsymbol{u}$ in $\mathcal{RM}(1,3)$

This example discusses the encoding of the message $\boldsymbol{u} = (0101)$ with the code $\mathcal{RM}(1,3)$. First, the resulting codeword is computed using the generator matrix as

$$\boldsymbol{c} = \boldsymbol{u} \cdot \boldsymbol{G}_{\mathcal{RM}(1,3)} = \begin{pmatrix} 0 & 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix} .$$

The same result can be computed by evaluating $f(0101)$ which corresponds to $a_1 = 1$ and $a_3 = 1$ that is the polynomial $f(0101) = \boldsymbol{p}_1 + \boldsymbol{p}_3$, which evaluates to

$$\begin{array}{r} \boldsymbol{p}_1 = (0 \;\; 0 \;\; 0 \;\; 0 \;\; 1 \;\; 1 \;\; 1 \;\; 1) \\ \boldsymbol{p}_3 = (0 \;\; 1 \;\; 0 \;\; 1 \;\; 0 \;\; 1 \;\; 0 \;\; 1) \\ \hline \boldsymbol{c} = \boldsymbol{p}_1 + \boldsymbol{p}_3 = (0 \;\; 1 \;\; 0 \;\; 1 \;\; 1 \;\; 0 \;\; 1 \;\; 0) \end{array} .$$

Decoding A general method for decoding binary RM codes is the *Reed decoding algorithm* that works by majority logic decoding. In the following, the concept is explained for the $\mathcal{RM}(1,3)$ code of Example 4.2.4. The polynomial of this code is given by $f(x) = a_0\boldsymbol{p}_0 + a_1\boldsymbol{p}_1 + a_2\boldsymbol{p}_2 + a_3\boldsymbol{p}_3$ for which a decoder has to distinguish the different a_i from a received word. When the vectors $\boldsymbol{p}_i$ are displayed with their individual coefficients as $(b_0, b_1, \ldots, b_{2^m-1})$, multiple equations can be constructed for the value of an a_i. If we use the values for, e.g., $\boldsymbol{p}_1$ of $\mathcal{RM}(1,3)$ (c.f. Example 4.2.4), there are the following equations to determine a_1:

$$a_1 = b_0 + b_1 = b_2 + b_3 = b_4 + b_5 = b_6 + b_7 .$$

The evaluation of these four equations gives four votes on the value of a_1, and the majority vote is still correct even in the presence of one error, which is the error correction capability of the $\mathcal{RM}(1,3)$ code. The process to determine a_2 and a_3 works analogously. After all a_i for $i \geq 1$ have been determined, the final equation for a_0 can be constructed, which is $\boldsymbol{x}' = \boldsymbol{r} - a_1\boldsymbol{p}_1 - a_2\boldsymbol{p}_2 - a_3\boldsymbol{p}_3$. The majority of zeros or ones in $\boldsymbol{x}'$ then define the value of a_0. For further explanation, the reader is referred to [MS77].

For binary first-order RM codes, there exists an even more practical form of decoder as these codes can be efficiently decoded using maximum likelihood (ML) decoding as defined in Eq. (4.1). In essence, a ML decoding works by comparing a received vector $\boldsymbol{r}$ with all codewords of $\mathcal{RM}(1,r)$ and then decoding to the codeword that has the smallest Hamming distance to $\boldsymbol{r}$. If there is a tie between codewords, the decision for a codeword is only dependent on the actual implementation of the decoder. In practice, there is a faster method to perform this comparison for all codewords through the Hadamard transformation. The Hadamard transformation is defined through the Hadamard transformation matrix H_i of size $2^i \times 2^i$, which can be recursively defined starting with $H_0 = 1$ as:

$$H_i = \begin{pmatrix} H_{i-1} & H_{i-1} \\ H_{i-1} & -H_{i-1} \end{pmatrix} . \tag{4.21}$$

For a decoding using the Hadamard transformation the received word $\boldsymbol{r} = (r_1, r_2, \ldots, r_{2^m})$ first has to be transferred to $\hat{\boldsymbol{r}} = ((-1)^{r_1}, (-1)^{r_2}, \ldots, (-1)^{r_{2^m}})$. Then, the largest value of $|\hat{\boldsymbol{r}} \cdot H_{2^m}|$ has to be determined. The coordinate (the index in the resulting vector) corresponding to the largest value is defined as x, where the binary representation of x defines the coefficients a_i of $f(\boldsymbol{x})$. The sign of the largest value determines the final step, which adds $\mathbf{1}$ to the resulting codeword if the value is negative. An example of ML decoding for RM codes is shown in Example 4.2.6.

Example 4.2.6: ML Decoding of $\mathcal{RM}(1,3)$

The codeword $\boldsymbol{c} = (00000000)$ with a single error $\boldsymbol{e} = (00100000)$ results in the word $\boldsymbol{r} = (00100000)$ that has to be decoded by the ML decoder. The decoding works by a comparison of the Hamming distance of $\boldsymbol{r}$ to all codewords, which is given by:

$\boldsymbol{r}$	(0 0 1 0 0 0 0 0)		$\boldsymbol{r}$	(0 0 1 0 0 0 0 0)	
$\boldsymbol{c}_0$	(0 0 0 0 0 0 0 0)	$\rightarrow \mathrm{HD}(\boldsymbol{r}, \boldsymbol{c}_0) = 1$	$\boldsymbol{c}_8$	(1 1 1 1 1 1 1 1)	$\rightarrow \mathrm{HD}(\boldsymbol{r}, \boldsymbol{c}_8) = 7$
$\boldsymbol{c}_1$	(0 0 0 0 1 1 1 1)	$\rightarrow \mathrm{HD}(\boldsymbol{r}, \boldsymbol{c}_1) = 5$	$\boldsymbol{c}_9$	(1 1 1 1 0 0 0 0)	$\rightarrow \mathrm{HD}(\boldsymbol{r}, \boldsymbol{c}_9) = 3$
$\boldsymbol{c}_2$	(0 0 1 1 0 0 1 1)	$\rightarrow \mathrm{HD}(\boldsymbol{r}, \boldsymbol{c}_2) = 3$	$\boldsymbol{c}_{10}$	(1 1 0 0 1 1 0 0)	$\rightarrow \mathrm{HD}(\boldsymbol{r}, \boldsymbol{c}_{10}) = 5$
$\boldsymbol{c}_3$	(0 0 1 1 1 1 0 0)	$\rightarrow \mathrm{HD}(\boldsymbol{r}, \boldsymbol{c}_3) = 3$	$\boldsymbol{c}_{11}$	(1 1 0 0 0 0 1 1)	$\rightarrow \mathrm{HD}(\boldsymbol{r}, \boldsymbol{c}_{11}) = 5$
$\boldsymbol{c}_4$	(0 1 0 1 0 1 0 1)	$\rightarrow \mathrm{HD}(\boldsymbol{r}, \boldsymbol{c}_4) = 5$	$\boldsymbol{c}_{12}$	(1 0 1 0 1 0 1 0)	$\rightarrow \mathrm{HD}(\boldsymbol{r}, \boldsymbol{c}_{12}) = 3$
$\boldsymbol{c}_5$	(0 1 0 1 1 0 1 0)	$\rightarrow \mathrm{HD}(\boldsymbol{r}, \boldsymbol{c}_5) = 5$	$\boldsymbol{c}_{13}$	(1 0 1 0 0 1 0 1)	$\rightarrow \mathrm{HD}(\boldsymbol{r}, \boldsymbol{c}_{13}) = 3$
$\boldsymbol{c}_6$	(0 1 1 0 0 1 1 0)	$\rightarrow \mathrm{HD}(\boldsymbol{r}, \boldsymbol{c}_6) = 3$	$\boldsymbol{c}_{14}$	(1 0 0 1 1 0 0 1)	$\rightarrow \mathrm{HD}(\boldsymbol{r}, \boldsymbol{c}_{14}) = 5$
$\boldsymbol{c}_7$	(0 1 1 0 1 0 0 1)	$\rightarrow \mathrm{HD}(\boldsymbol{r}, \boldsymbol{c}_7) = 3$	$\boldsymbol{c}_{15}$	(1 0 0 1 0 1 1 0)	$\rightarrow \mathrm{HD}(\boldsymbol{r}, \boldsymbol{c}_{15}) = 5$

It can be seen that the codeword with the smallest Hamming distance to the received word is $\boldsymbol{c}_0$, and therefore the decoding is successful. Now, the decoding using the Hardamard transformation is shown. First, $\hat{\boldsymbol{r}}$ is computed as

$$\hat{\boldsymbol{r}} = ((-1)^0 + (-1)^0 + (-1)^1 + (-1)^0 + (-1)^0 + (-1)^0 + (-1)^0 + (-1)^0) = \begin{pmatrix} 1 & 1 & -1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} .$$

Then the multiplication of $\hat{\boldsymbol{r}}$ with the Hadamard matrix H_8 is computed, which results in

$$\begin{aligned} \hat{\boldsymbol{r}} \cdot H_8 &= \begin{pmatrix} 1 & 1 & -1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix} \\ &= \begin{pmatrix} 6 & -2 & 2 & 2 & -2 & -2 & 2 & 2 \end{pmatrix} . \end{aligned}$$

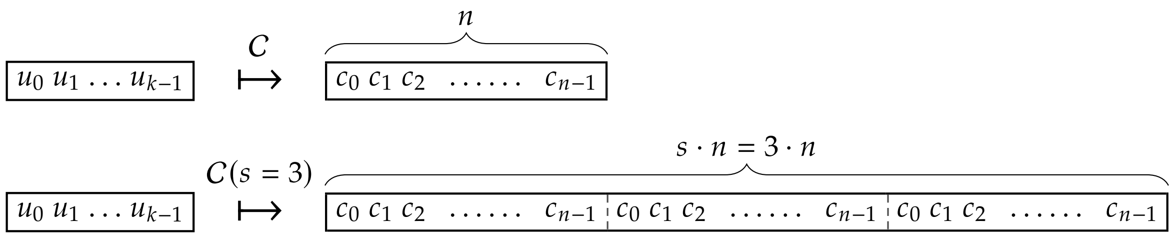
The hightest value of $|\hat{\boldsymbol{r}} \cdot H_{2^m}|$ is the first index with value 6, which gives $x = 0$. The binary representation of x (0000) defines the different a_i as $(a_3 = 0, a_2 = 0, a_1 = 0, a_0 = 0)$. This results in the codeword $\boldsymbol{c}$ as

$$\boldsymbol{c} = 0 * p_0 + 0 * p_1 + 0 * p_2 + 0 * p_3 = (00000000) .$$

The highest value in $|\hat{\boldsymbol{r}} \cdot H_{2^m}|$ is 6, which is a positive value and therefore the resulting codeword is $\boldsymbol{c}$, which indicates a successful decoding.

Duplicated RM codes For RM codes, there is the possibility of duplicating the code multiple times, where the total amount of duplicated codewords is given by s, which is called the multiplicity of the code. After the duplication, a $RM(r, m)$ code that corresponds to the $[2^m, \sum_{i=0}^{r} \binom{m}{i}, 2^{m-r}]$ code results in an $[s \cdot 2^m, \sum_{i=0}^{r} \binom{m}{i}, s \cdot 2^{m-r}]$ code. As an example the $RM(1, 3)$ code with $[8, 4, 4]$ and multiplicity $s = 3$ results in the $[24, 4, 12]$ code. It can be seen that the dimension of the code is unchanged, but the length and minimum distance increase. Therefore, the error-correcting capability of the duplicated code increases accordingly. For the given example, the single error-correcting code ($[8, 4, 4]$) is transferred to a code, which is able to correct up to 5 errors.

The encoding with a duplicated RM code of a vector $\boldsymbol{u}$ is done by first encoding it with the underlying $\mathcal{RM}(r, m)$ code and duplicating the resulting codeword $s - 1$ times through binary concatenation. The process is visualized in Fig. 4.7.



n
C
$u_0\ u_1 \dots u_{k-1}$
$\mapsto$
$c_0\ c_1\ c_2\ \dots\dots\ c_{n-1}$
$s \cdot n = 3 \cdot n$
$C(s=3)$
$u_0\ u_1 \dots u_{k-1}$
$\mapsto$
$c_0\ c_1\ c_2\ \dots\dots\ c_{n-1}$ $c_0\ c_1\ c_2\ \dots\dots\ c_{n-1}$ $c_0\ c_1\ c_2\ \dots\dots\ c_{n-1}$

Figure 4.7 Visualization of the duplicated encoding of a RM code. The top part shows an encoding with the normal code, while the bottom part shows the encoding for the duplicated code with multiplicity $s = 3$.

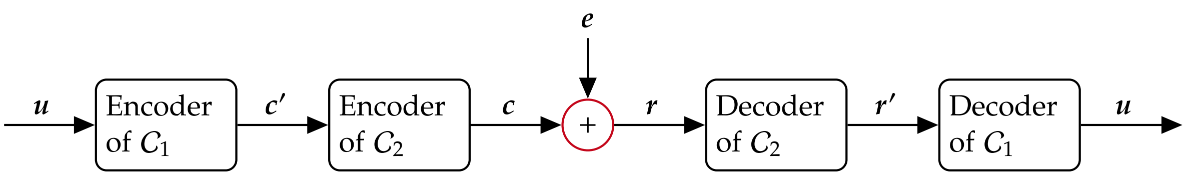


Figure 4.8 Encoding and decoding of concatenated codes. Adapted from [WZ21].

For the decoding, only the mapping of the received word to $\hat{\boldsymbol{r}}$ must be adapted to reflect the now duplicated code. Given the received word of the duplicated code as $\boldsymbol{r}_{\text{dubl}} = \begin{pmatrix} \boldsymbol{r}_0 & \boldsymbol{r}_1 & \dots & \boldsymbol{r}_{s-1} \end{pmatrix}$ and the individual as $\boldsymbol{r}_i = \begin{pmatrix} r_{i_0} & r_{i_1} & \dots & r_{i_{n-1}} \end{pmatrix}$ then $\hat{\boldsymbol{r}}_{\text{dubl}}$ is defined as

$$\hat{\boldsymbol{r}}_{\text{dubl}} = \begin{pmatrix} (-1)^{r_{0_0}} + (-1)^{r_{1_0}} + \ldots + (-1)^{r_{s-1_0}} & \dots & (-1)^{r_{0_{n-1}}} + (-1)^{r_{1_{n-1}}} + \ldots + (-1)^{r_{s-1_{n-1}}} \end{pmatrix} .$$

After this mapping, the length of $\hat{\boldsymbol{r}}_{\text{dubl}}$ is equal to the one of the non-duplicated code, and therefore the decoding follows the same steps as described for the non-duplicated RM code.

4.2.6 Concatenated Codes

In practice, the problem exists that the execution time and resource requirements of a decoder as well as the overall complexity of the decoding increase with the length of the code. A prominent example are code-based encryption schemes where the requirements for the decryption failure rates lead to code lengths of $n > 20\,000$. A solution for this problem is the concept of *code concatenation*, where the desired length of the code is achieved by combining multiple shorter codes, for which the individual decoding is efficient. In the following, we describe the code concatenation of two error-correcting codes C_1 and C_2 to the concatenated code C. In this scenario the *outer* code C_1 encodes an input $\boldsymbol{u}$ to the codeword $\boldsymbol{c}'$ and the *inner* code C_2 divides $\boldsymbol{c}'$ into blocks of its dimension k_{C_2} that are individually encoded and their concatenation is the codeword $\boldsymbol{c}$ of C. For practical reasons, the dimension of C_2 is often chosen as the length of a symbol of C_1, as then n_{C_1} is dividable by k_{C_2}. The parameters of the concatenated code C built from C_1 with $[n_{C_1}, k_{C_1}, d_{C_1}]_{2^m}$ and C_2 with $[n_{C_2}, k_{C_2} = m, d_{C_2}]_2$ are $n = n_{C_1} n_{C_2}$, $k = k_{C_1} k_{C_2}$, and $d \geq d_{C_1} d_{C_2}$. A visualization of this code concatenation is shown in Fig. 4.8.

4.3 Hamming Quasi-Cyclic (HQC)

This section gives an introduction to HQC. First, the notation is introduced in Section 4.3.1. In Section 4.3.2, the HQC cryptosystem is described with its changes during the NIST competition. Finally, details on the used error-correcting codes for both versions of HQC are provided in Section 4.3.3.

4.3.1 Notation

Let $\mathbb{F}_2$ be the finite field of size 2. $\mathbb{F}_2^{m\times n}$ is used to denote the set of all $m\times n$ matrices over $\mathbb{F}_2$, $\mathbb{F}_2^n = \mathbb{F}_2^{1\times n}$ for the set of all row vectors of length n over $\mathbb{F}_2$, and define the set of integers $[a,b] := \{i : a \leq i \leq b\}$. We index rows and columns of $m\times n$ matrices by $0,\dots,m-1$ and $0,\dots,n-1$, where the entry in the i-th row and j-th column of the matrix $\boldsymbol{A}$ is denoted by $A_{i,j}$.

The Hamming weight of a vector $\boldsymbol{a}$ is indicated by $\mathrm{HW}(\boldsymbol{a})$ and the support of $\boldsymbol{a}$ is denoted by $\mathrm{supp}(\boldsymbol{a}) := \{i \in \mathbb{Z} : a_i \neq 0\}$. A set $\mathcal{A}$ is called super support (ssupp) of $\boldsymbol{a}$ if $\mathcal{A} \supset \mathrm{supp}(\boldsymbol{a})$.

Let $\mathcal{V}$ be a vector space of dimension n over $\mathbb{F}_2$. We define the product of $\boldsymbol{u},\boldsymbol{v} \in \mathcal{V}$ as

$$\boldsymbol{u}\boldsymbol{v} = \boldsymbol{u}\,\mathrm{rot}(\boldsymbol{v})^\top = \boldsymbol{v}\,\mathrm{rot}(\boldsymbol{u})^\top = \boldsymbol{v}\boldsymbol{u}, \tag{4.22}$$

where

$$\mathrm{rot}(\boldsymbol{v}) := \begin{bmatrix} v_0 & v_{n-1} & \dots & v_1 \\ v_1 & v_0 & \dots & v_2 \\ \vdots & \vdots & \ddots & \vdots \\ v_{n-1} & v_{n-2} & \dots & v_0 \end{bmatrix} \in \mathbb{F}_2^{n\times n}. \tag{4.23}$$

As a consequence of this definition, elements of $\mathcal{V}$ can be interpreted as polynomials in the ring $\mathcal{R}_2 := \mathbb{F}_2[X]/(X^n - 1)$.

4.3.2 Cryptosystem

The code-based cryptosystem Hamming Quasi-Cyclic (HQC) was first published in 2018 [AMBD+18] and is a promising alternative candidate in the NIST post-quantum cryptography competition. It is still present in the ongoing fourth round of the competition as it offers the following advantages. In contrast to established code-based cryptosystems like McEliece [ABC+20] and its derivatives, its security does not rely on hiding the structure of the used error-correcting code. Instead, the structure of the used code and the efficient decoding algorithm are publicly known, reducing its security to instances of the Quasi-Cyclic Syndrome Decoding (QCSD) problem, which is a well-understood problem in coding theory. Furthermore, HQC features attractive key sizes and allows precise estimations of its decryption failure rate.

The HQC scheme is based on two different codes. It consists of a public code $\mathcal{C} \subseteq \mathbb{F}_2^{n_1 n_2}$ of length $n_1 n_2$ and dimension k, where it is assumed that both an efficient encoding algorithm $\mathcal{C}.\textsc{Enc}$ and an efficient decoding algorithm $\mathcal{C}.\textsc{Dec}$ are known publicly. Further, the decoding algorithm can correct δ errors with high probability but fails for errors of large weight. HQC is also based on a second code of length $2n$ and dimension n which has a parity-check matrix $(\boldsymbol{I}, \mathrm{rot}(\boldsymbol{h})) \in \mathbb{F}_2^{n\times 2n}$, where $\boldsymbol{I}$ denotes the $n\times n$ identity matrix. Contrary to $\mathcal{C}$, it is assumed that no party possesses an efficient decoding algorithm for the second code. Please note that knowledge of this second code is not needed for either encryption or decryption, but it is instead used to model the security of the system.

We start by introducing the PKE version of the algorithm as shown in Algorithms 7 to 9. First, the secret and public key is generated using HQC-KeyGen (Algorithm 7) such that the secret key consists of two polynomials $\boldsymbol{x},\boldsymbol{y}$ with fixed Hamming weight w and the public key of the polynomials $\boldsymbol{h},\boldsymbol{s}$. With HQC-Encrypt (Algorithm 8), a message of size k can be encrypted to the ciphertext c consisting of the two polynomials $\boldsymbol{u},\boldsymbol{v}$. This ciphertext is decrypted with HQC-Decrypt using the secret key $\boldsymbol{y}$ to the message $\boldsymbol{m}$. Within these algorithms, several polynomials are uniformly sampled from $\mathcal{R}_2$, denoted as $\xleftarrow{\$}$, with the optional argument of specifying the Hamming weight w of the polynomial. The required randomness for the sampler is generated using a PRNG (SHAKE256 is used in the presented version of HQC) that is initialized with a random seed. For HQC-Encrypt this seed is defined by the input θ such that the encryption is deterministic, which is a requirement of the KEM version of HQC.

Algorithm 7 HQC-KeyGen

```
Input: param
  1: $h \xleftarrow{\$} \mathcal{R}_2$
  2: $x \xleftarrow{\$(w)} \mathcal{R}_2$        ▷ $\mathrm{HW}(x) = w$
  3: $y \xleftarrow{\$(w)} \mathcal{R}_2$        ▷ $\mathrm{HW}(y) = w$
  4: $s \leftarrow x + hy$
Output: pk = $(h, s)$, sk = $(x, y)$
```

Algorithm 8 HQC-Encrypt

```
Input: pk = $(h, s)$, $m$ and seed $\theta$
  1: Initialize the PRNG used by the sampler with $\theta$
  2: $e' \xleftarrow{\$(w_e)} \mathcal{R}_2$        ▷ $\mathrm{HW}(e') = w_e$
  3: $r_1 \xleftarrow{\$(w_r)} \mathcal{R}_2$        ▷ $\mathrm{HW}(r_1) = w_r$
  4: $r_2 \xleftarrow{\$(w_r)} \mathcal{R}_2$        ▷ $\mathrm{HW}(r_2) = w_r$
  5: $u \leftarrow r_1 + hr_2$
  6: $v \leftarrow \mathcal{C}.\mathrm{Enc}(m) + sr_2 + e'$
Output: $c = (u, v)$
```

Algorithm 9 HQC-Decrypt

```
Input: sk = $y$, ct = $(u, v)$
  1: $v' \leftarrow v - uy$
  2: $m \leftarrow \mathcal{C}.\mathrm{Dec}(v')$
Output: $m$
```

	shortened BCH code $[n_1, k, d_{BCH}]$	Repetition code $[n_2, k_{Rep}, d_{Rep}]$	n	k	w	$w_r = w_e$
HQC-BCH-128	[766, 256, 115]	[31, 1, 31]	23 869	256	67	77
HQC-BCH-192	[766, 256, 115]	[59, 1, 59]	45 197	256	101	117
HQC-BCH-256	[796, 256, 121]	[87, 1, 87]	69 259	256	133	153

Table 4.1 Parameter sets of HQC-BCH according to the updated second-round NIST submission from 21.04.2020 [MAB+20].

	shortened RS code $[n_1, k^{a)}, d_{RS}]$	duplicated RM code $[n_2, k_{RM}, d_{RM}, s]$	n	k	w	$w_r = w_e$
HQC-RMRS-128	[46, 16, 31]	[384, 8, 192, 3]	17 669	128	66	75
HQC-RMRS-192	[56, 24, 33]	[640, 8, 320, 5]	35 851	192	100	114
HQC-RMRS-256	[90, 32, 59]	[640, 8, 320, 5]	57 637	256	131	149

a) Note that the RS code uses $\mathbb{F}_{2^8}$.

Table 4.2 Parameter sets of HQC-RMRS according to the third round NIST submission [MAB+20].

The correctness of the encryption scheme lies in the decoding capability of the decoder during Line 2 of Algorithm 9. C.Dec correctly decodes $\boldsymbol{v} - \boldsymbol{u}\boldsymbol{y}$ in C if

$$\begin{aligned}
&\mathrm{HW}(\boldsymbol{s} \cdot \boldsymbol{r}_2 - \boldsymbol{u} \cdot \boldsymbol{y} + \boldsymbol{e}') \leq \delta \\
&\mathrm{HW}((\boldsymbol{x} + \boldsymbol{h} \cdot \boldsymbol{y}) \cdot \boldsymbol{r}_2 - (\boldsymbol{r}_1 + \boldsymbol{h} \cdot \boldsymbol{r}_2) \cdot \boldsymbol{y} + \boldsymbol{e}') \leq \delta \\
&\mathrm{HW}(\boldsymbol{x} \cdot \boldsymbol{r}_2 - \boldsymbol{r}_1 \cdot \boldsymbol{y} + \boldsymbol{e}') \leq \delta\,. \qquad (4.24)
\end{aligned}$$

The parameters for the algorithm have been chosen in a way that the resulting decryption failure probability is lower than the specified security level of the parameter set. Besides the parameters, an instance of HQC is defined by the used code C. The HQC authors specify two instances with a different concatenation of two error-correcting codes. The first one, which we call HQC-BCH, is based on a concatenation of a BCH code in combination with a repetition code. The corresponding parameter sets are shown in Table 4.1. The most recent instance, called HQC-RMRS, is based on a concatenation of an RM code in combination with an RS code. The corresponding parameter sets are shown in Table 4.2. A detailed discussion about the used error-correcting codes is given in Section 4.3.3.

The HQC authors use a variant of the FO transformation from [HHK17] to construct an IND-CCA2 secure KEM from the PKE version of the system. Using this KEM, a random shared secret K can be exchanged where the sender applies encapsulation (Algorithm 10) and the receiver decapsulation (Algorithm 11). These algorithms use three different hash functions: $\mathcal{G}$,$\mathcal{H}$, and $\mathcal{K}$, which are based on SHAKE256 with 512 bits of output. In order to counteract chosen-chipertext attacks, the decrypted message is re-encrypted and compared to the original ciphertext input. Only if both ciphertexts are equal, K is released, otherwise decapsulation is aborted. The reference version of the third round version of HQC returns an all-zero value in the case of a failed decapsulation. In order for the re-encryption to be possible, the sampling of the random elements has to be deterministic, which is ensured by deriving a seed from the message, which is then utilized to initialize the PRNG used to derive the required randomness of the sampler.

Changes of the HQC Cryptosystem During the duration of the NIST post-quantum cryptography competition, there have been several updates to the HQC cryptosystem. An overview of the different updates with their respective timeline can be seen in Fig. 4.9.

Algorithm 10 HQC-Encapsulate

```
Input: pk = $(\boldsymbol{s}, \boldsymbol{h})$
1: $\boldsymbol{m} \xleftarrow{\$} \mathbb{F}_2^k$
2: $\theta \leftarrow \mathcal{G}(\boldsymbol{m})$
3: $c \leftarrow$ HQC-Encrypt(pk, $\boldsymbol{m}, \theta$)
4: $K \leftarrow \mathcal{K}(\boldsymbol{m}, c)$
5: $\boldsymbol{d} \leftarrow \mathcal{H}(\boldsymbol{m})$
Output: $K, c = (\boldsymbol{u}, \boldsymbol{v}), \boldsymbol{d}$
```

Algorithm 11 HQC-Decapsulate

```
Input: sk = $(\boldsymbol{y}), c, \boldsymbol{d}$
1: $\boldsymbol{m}' \leftarrow$ HQC-Decrypt(sk, $c$)
2: $\theta' \leftarrow \mathcal{G}(\boldsymbol{m}')$
3: $c' \leftarrow$ Encrypt(pk, $\boldsymbol{m}', \theta'$)
4: if $c \neq c'$ and $\boldsymbol{d} \neq \mathcal{H}(\boldsymbol{m}')$ then
5:     abort                    ▷ The HQC reference implementation sets $K$ to all zeros.
6: else
7:     $K \leftarrow \mathcal{K}(\boldsymbol{m}, c)$
Output: $K$
```

The original HQC submission for the first round of the NIST competition defined a parameter set with the use of a concatenated BCH in combination with a repetition code (HQC-BCH). For the second round version, the authors introduced some changes to the security proof of the system and changed the parameters such that, e.g., parameter sets that showed a decryption failure rate above the desired security level were discarded. In April 2020, the authors updated the system outside the NIST standardization interval. They introduced a new variant in addition to HQC-BCH that is based on a combination of a RM and a RS code (HQC-RMRS). The improved error correction capability of HQC-RMRS results in a significant decrease in the parameter sizes of the algorithm, e.g., the size of the public key is reduced by 17 % for the smallest parameter set. For the third round, the HQC-BCH version was discarded because the HQC-RMRS version achieved a strictly better performance. Two additional changes allowed a further reduction of parameter sizes. First, the authors provide an improved theoretical bound for the decryption failure rate of the RM code, allowing a precise estimation and, therefore, reduced parameter sizes. Second, the authors adapt the transferred message size k to the security level of the algorithm instead of fixing it to 256 for all parameter sets. This improves the decoding capacity of the used codes, allowing further lowering of parameter sizes. In June 2021, the authors proposed only using SHAKE256 for the required hash function of the KEM, constructing the different functions $\mathcal{G}$,$\mathcal{H}$, and $\mathcal{K}$ through domain separation. For the fourth round submission, the authors mainly mitigate published attacks on the cryptosystem. They propose to compute the randomness θ by concatenating the message $\boldsymbol{m}$ additionally with the public key and a salt to mitigate a multi-ciphertext attack. In addition, they include a countermeasure [Sen21] against the timing attack on the fixed Hamming weight sampler published in [GHJ+22].

For the description of the HQC system in this section, the specification of the third-round version in the NIST competition is used [MAB+20]. The parameter sets for HQC-BCH as well as the reference implementation for the actual side-channel evaluation is taken from the updated second-round submission package from the 21.04.2020 [MAB+20]. The parameter sets for HQC-RMRS and the corresponding reference implementation correspond to the official third-round NIST submission package [MAB+20].

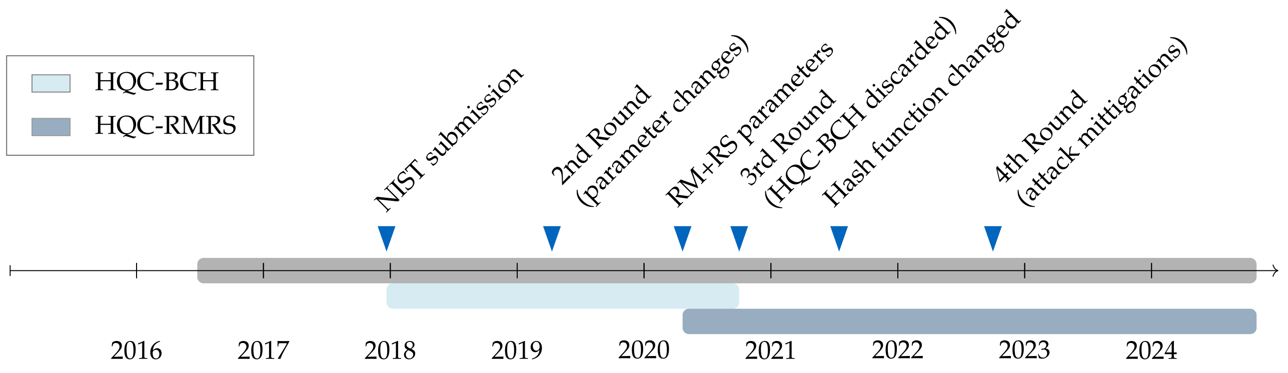


Figure 4.9 Updates to the HQC cryptosystem during the NIST competition.

4.3.3 Choice of the Error-Correcting Code

The HQC cryptosystem as defined in Section 4.3.2 can be described without specifying the used error-correcting code C. In theory, any error-correcting code could be used to instantiate the scheme if the parameters are chosen to satisfy Eq. (4.24) with a low enough decryption failure rate[7]. In HQC, the authors instantiate their scheme as HQC-BCH, with a combination of an BCH with a repetition code, which was later superseded by HQC-RMRS using a combination of a RM and RS code that shows an increased error correction capability.

Code Concatenation in HQC Both versions of the HQC algorithm make use of a code concatenation of two error-correcting codes (c.f. Section 4.2.6). This means that there is an *outer* code C_1 and an *inner* code C_2[8] that are used in combination in order to encode a message $\boldsymbol{m}$ in C such that $\mathbb{F}_2^k \longrightarrow \mathbb{F}_2^{n_1 n_2}$. The two-stage process is started by encoding the message $\boldsymbol{m} \in \mathbb{F}_2^k$ in C_1 resulting in $\boldsymbol{m}_1 \in \mathbb{F}_2^{n_1}$. Subsequently, each coordinate $\boldsymbol{m}_{1,i}$ of $\boldsymbol{m}_1$ for $i = 0, \dots, n_1 - 1$ is encoded in C_2. The final codeword $\tilde{\boldsymbol{m}} \in \mathbb{F}_2^{n_1 n_2}$ is constructed by concatenating the result of the individual encodings as $\tilde{\boldsymbol{m}} = (\tilde{\boldsymbol{m}}_{1,0}, \dots, \tilde{\boldsymbol{m}}_{1,n_1-1})$. The decoding in C works analogously by first decoding an input to the decoder $\boldsymbol{v}'$ in C_2 to get $\tilde{\boldsymbol{v}}$. This means the individual elements $\boldsymbol{v}'_i$ of $\boldsymbol{v}'$ for $i = 0, \dots, n_1 - 1$ of size n_2 are decoded individually. The results are concatenated and form the n_1 coordinates $\tilde{\boldsymbol{v}}_i$ of $\tilde{\boldsymbol{v}}$ for $i = 0, \dots, n_1 - 1$. This $\tilde{\boldsymbol{v}}$ is then decoded in C_1 to retrieve the message $\boldsymbol{m}$. The concatenation for both the encoder and decoder of HQC is visualized in Fig. 4.10.

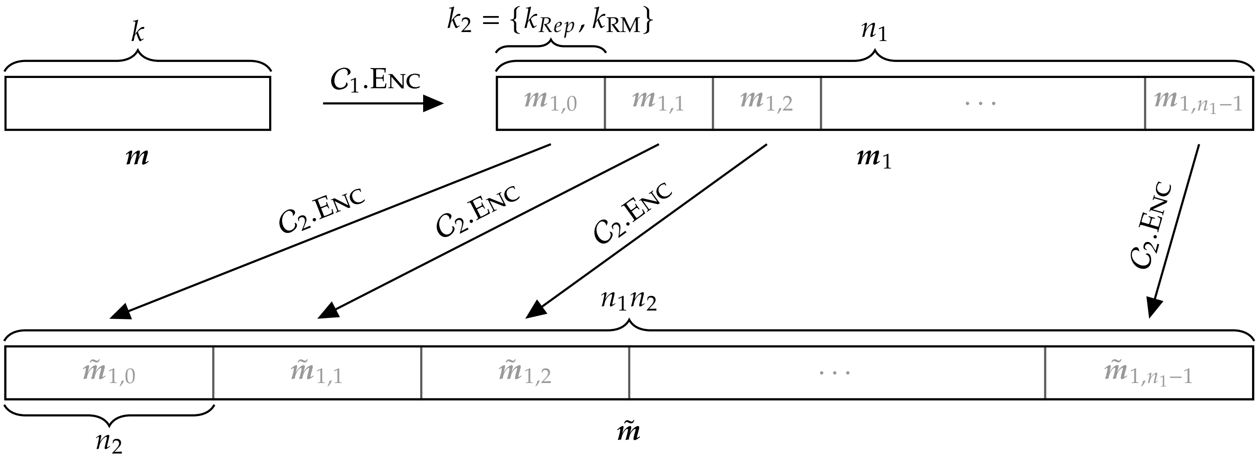
It is important to note that the elements in HQC are polynomials in $\mathcal{R}_2$ and can therefore be represented as elements in $\mathbb{F}_2^n$. As it has been discussed in this paragraph, the result after $C.\textsc{Enc}(\boldsymbol{m})$ is an element in $\mathbb{F}_2^{n_1 n_2}$. This causes a difference of $n - n_1 n_2$ in the length of the codeword and the desired length in $\mathcal{R}_2$. This difference comes from the fact that n is chosen to be the smallest primitive prime greater than $n_1 n_2$, which avoids algebraic attacks using polynomial factorization. In order to again achieve the desired length in $\mathbb{F}_2^n$, the resulting codeword $\tilde{\boldsymbol{m}}$ is padded with zeros. Analogously during HQC-Decrypt (Algorithm 9) only the first $n_1 n_2$ coefficients of $\boldsymbol{v}'$ are used as input to $C.\textsc{Dec}$.

Repetition and BCH Code For HQC-BCH, the code C is constructed using a $[n_1, k, d_{BCH}]$ shortened BCH code as C_1 and a $[n_2, 1, d_{Rep}]$ repetition code as C_2. Formally, the encoding is defined as

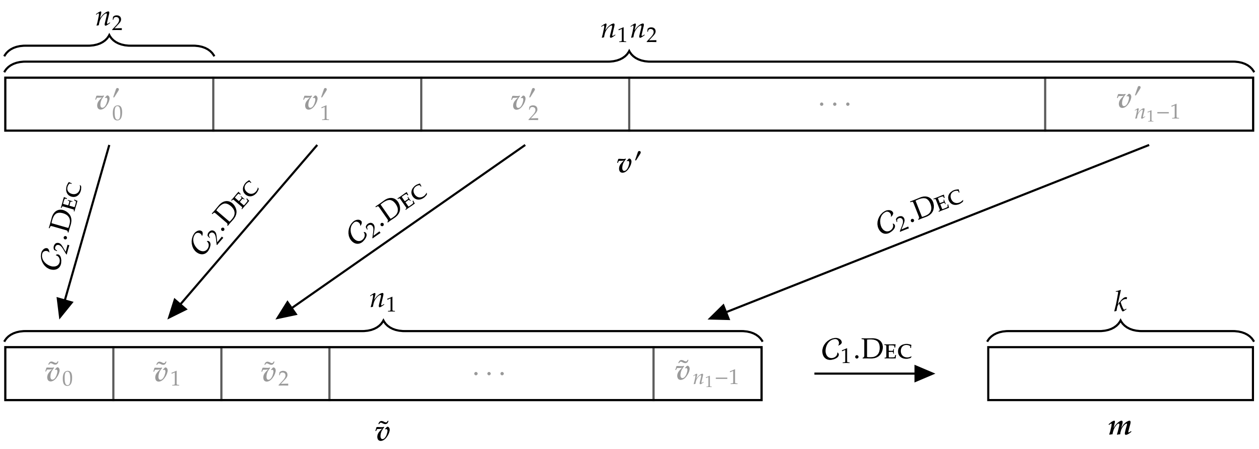
$$
\begin{aligned}
C.\textsc{Enc} : \mathbb{F}_2^k &\longrightarrow \mathbb{F}_2^{n_1 n_2}, \\
\boldsymbol{m} &\longmapsto (\underbrace{\boldsymbol{m}_{1,0}, \dots, \boldsymbol{m}_{1,0}}_{n_2 \text{ times}}, \underbrace{\boldsymbol{m}_{1,1}, \dots, \boldsymbol{m}_{1,1}}_{n_2 \text{ times}}, \dots, \underbrace{\boldsymbol{m}_{1,n_1-1}, \dots, \boldsymbol{m}_{1,n_1-1}}_{n_2 \text{ times}}),
\end{aligned}
$$

[7]For completeness, it has to be noted that the resulting parameters also have to show a sufficient security margin with respect to the best-known attacks on the underlying 2-QCSD and 3-QCSD problem.

[8]In the HQC submission this is named analogously as external and internal code.



(a) HQC encoder



(b) HQC decoder

Figure 4.10 Visualization of the code concatenation used in HQC.

where $\boldsymbol{m}_1 = (\boldsymbol{m}_{1,0}, \dots, \boldsymbol{m}_{1,n_1-1}) = \boldsymbol{m}\boldsymbol{G}_1$ and $\boldsymbol{G}_1 \in \mathbb{F}_2^{k\times n_1}$ is a generator matrix of the $[n_1, k, d_{BCH}]$ shortened BCH code $\mathcal{C}_1$. Given an input vector $\boldsymbol{v}' = (\boldsymbol{v}'_0, \dots, \boldsymbol{v}'_{n_1-1}) \in \mathbb{F}_2^{n_1 n_2}$ the decoding algorithm $\mathcal{C}$.Dec consists of two steps. First the algorithm decodes the vectors $\boldsymbol{v}'_0, \dots, \boldsymbol{v}'_{n_1-1}$ separately in the repetition code $\mathcal{C}_2$ using majority decoding to a vector $\tilde{\boldsymbol{v}} = (\tilde{v}_0, \dots, \tilde{v}_{n_1-1}) \in \mathbb{F}_2^{n_1}$, where

$$\tilde{v}_i = \begin{cases} 1 & \text{if } \sum_{j=0}^{n_2-1} v'_{ij} \geq \lceil \frac{n_2+1}{2} \rceil \\ 0 & \text{otherwise.} \end{cases}$$

In the second step, the algorithm decodes $\tilde{\boldsymbol{v}}$ in the BCH code $\mathcal{C}_1$ to the vector $\boldsymbol{m} \in \mathbb{F}_2^k$.

The actual implemented BCH codes with parameters given in Table 4.1 are shortened versions of the $[1023, 513, 115]_2$ (HQC-BCH-128, HQC-BCH-192), and the $[1023, 483, 121]_2$ (HQC-BCH-256) BCH code. The encoding is implemented in systematic form as described in Section 4.2.4 using a LFSR division described in [LC04, Chapter 5.3]. A syndrome-based decoder, as shown in Fig. 4.6, is implemented to decode the shortened BCH codes. The syndromes are computed using the transpose of the additive Fast Fourier Transformation as proposed in [BCS13]. Then, the error locator polynomial is determined using a modification of Berlekamp's simplified algorithm for binary BCH codes [JK95]. Finally, the error locations are computed with an additive Fast Fourier Transform [GM10].

Reed-Muller and Reed-Solomon Code For HQC-RMRS the code $\mathcal{C}$ is instantiated with a $[n_1, k, d_{RS}]$ shortened RS code as $\mathcal{C}_1$ and a duplicated $[n_2, 8, d_{RM}]$ RM code as $\mathcal{C}_2$. First the outer RS code is used to encode $\boldsymbol{m}$ into $\boldsymbol{m}_1 \in \mathbb{F}_{2^8}^{n_1}$, followed by encoding each coordinate $\boldsymbol{m}_{1,i}$ of $\boldsymbol{m}_1$ into $\tilde{\boldsymbol{m}}_{1,i} \in \mathbb{F}_2^{n_2}$ using the inner duplicated RM code. The duplicated encoding works in two phases. First, each $\boldsymbol{m}_{1,i}$ is encoded with the underlying [128,8,64] RM code to obtain $\bar{\boldsymbol{m}}_{1,i}$, which is then duplicated based on the multiplicity s (see Table 4.2) resulting in $\tilde{\boldsymbol{m}}_{1,i}$. In other words the final encoding result is constructed as $\tilde{\boldsymbol{m}} = (\tilde{\boldsymbol{m}}_{1,0}, \dots, \tilde{\boldsymbol{m}}_{1,n_1-1}) \in \mathbb{F}_2^{n_1 n_2}$. For the decoding $\mathcal{C}$.Dec an input $\boldsymbol{v}' \in \mathbb{F}_2^{n_1 n_2}$ is decoded to the message $\boldsymbol{m} \in \mathbb{F}_2^k$. First the individual $\boldsymbol{v}'_i$ are decoded with the duplicated RM decoder ($\mathfrak{D}_{\mathcal{RM}}$), which results in the input to the RS decoder ($\mathfrak{D}_{\mathcal{RS}}$) as an element in $\mathbb{F}_{2^8}^{n_1}$. Finally, the RS decoding results in the message $\boldsymbol{m} \in \mathbb{F}_2^k$. The implemented RS codes are shortened versions of the $[255, 225, 31]_{2^8}$ (HQC-RMRS-128), $[255, 223, 33]_{2^8}$ (HQC-RMRS-192), and $[255, 197, 59]_{2^8}$ (HQC-RMRS-256) codes. The encoding is performed in systematic form as described in Section 4.2.3. A syndrome-based decoder, as visualized in Fig. 4.5, is used for the decoding. The syndrome computation is performed through multiplication with the parity check matrix. Subsequently, the key equation is solved using the Berlekamp-Massey algorithm [LC04]. Further, the error locations are found as the roots of the error-locator polynomial $\Lambda(x)$ through the use of an additive Fast Fourier Transform [GM10]. Finally, the error values are derived by a small modification to Forney's formula as detailed in [LC04, Chapter 7.3]. The duplicated RM codes are encoded through multiplication with the respective generator matrix, while the decoder is implemented using the Hadamard transformation as detailed in Section 4.2.5. If a tie is detected, the decoder decides on the codeword of a smaller lexicographical order, i.e., the codeword with the smaller index.

4.4 General Attack Strategy using SCA Oracles

In this section, we propose the general concept of a chosen ciphertext attack on HQC to retrieve the secret key $\boldsymbol{y} = (\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}) \in \mathbb{F}_2^n$, where $\boldsymbol{y}^{(0)} \in \mathbb{F}_2^{n_1 n_2}$ and $\boldsymbol{y}^{(1)} \in \mathbb{F}_2^{n - n_1 n_2}$. Please note that although the secret key additionally consists of $\boldsymbol{x}$, it is sufficient to only retrieve $\boldsymbol{y}$, as it is the only secret needed for a successful decryption (c.f. Section 4.3.2). First, the support distribution of $\boldsymbol{y}$ is analyzed in Section 4.4.1, as $\boldsymbol{y}$ is a sparse polynomial, and therefore it is sufficient to retrieve its support. In Section 4.4.2, the general concept of the attack on $\boldsymbol{y}^{(0)}$ is discussed, which is based on a decoding oracle of the used error-correcting code. Additionally, the used oracles for the attacks on HQC-BCH and HQC-RMRS are defined. This is followed by a discussion on the retrieval of all parts of the secret

w_{max}	HQC-BCH-128	HQC-BCH-192	HQC-BCH-256
1	5.59 %	0.11 %	≈ 0 %
2	93.20 %	77.98 %	58.99 %
3	99.86 %	99.25 %	97.99 %

Table 4.3 Probabilities that $\boldsymbol{y}$ is generated such that the weight of all $\boldsymbol{y}_i^{(0)}$ for $i = 0, \dots, n_1 - 1$ is at most w_{max} for the different parameter sets of HQC-BCH.

key in Section 4.4.3, which consists of additionally finding $\boldsymbol{y}^{(1)}$. Finally, in Section 4.4.4, it is shown how to utilize partial attack results through information set decoding.

4.4.1 Support Distribution of the Secret Key $\boldsymbol{y}$

The distribution of the non-zero positions in the secret $\boldsymbol{y}$, i.e., the support of $\boldsymbol{y}$, is essential for the proposed attack. To simplify the notation, we decompose the vector $\boldsymbol{y}$ as follows

$$\boldsymbol{y} = (\boldsymbol{y}_0^{(0)}, \dots, \boldsymbol{y}_{n_1-1}^{(0)}, \boldsymbol{y}^{(1)}) \in \mathbb{F}_2^n ,$$

where $\boldsymbol{y}_0^{(0)}, \dots, \boldsymbol{y}_{n_1-1}^{(0)} \in \mathbb{F}_2^{n_2}$ and $\boldsymbol{y}^{(1)} \in \mathbb{F}_2^{n-n_1 n_2}$. This decomposition is visualized in Fig. 4.11.

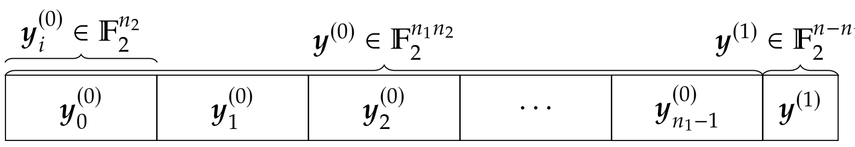

$\boldsymbol{y}_i^{(0)} \in \mathbb{F}_2^{n_2}$
$\boldsymbol{y}^{(0)} \in \mathbb{F}_2^{n_1 n_2}$
$\boldsymbol{y}^{(1)} \in \mathbb{F}_2^{n-n_1 n_2}$
$\boldsymbol{y}_0^{(0)}$
$\boldsymbol{y}_1^{(0)}$
$\boldsymbol{y}_2^{(0)}$
$\dots$
$\boldsymbol{y}_{n_1-1}^{(0)}$
$\boldsymbol{y}^{(1)}$

Figure 4.11 Different parts of the secret key $\boldsymbol{y} \in \mathbb{F}_2^n$.

Note that as described in Section 4.3.3, $\boldsymbol{y}$ has to be larger than $n_1 n_2$ to withstand algebraic attacks. Therefore, a part of $\boldsymbol{y}$, which we defined as $\boldsymbol{y}^{(1)}$, is not part of the codeword and is therefore not processed by the decoder. As the proposed attack utilizes side-channel information of the decoder, $\boldsymbol{y}^{(1)}$ can not be directly attacked. Methods to retrieve it from a known $\boldsymbol{y}^{(0)}$ are discussed in Section 4.4.3.

The proposed method of attacking $\boldsymbol{y}^{(0)}$ targets all codewords of C_2 individually, which defines the split of $\boldsymbol{y}^{(0)}$ into its corresponding chunks $\boldsymbol{y}_i^{(0)}$ with $0 \le i \le n_1 - 1$. Additionally, for all HQC parameter sets, $\boldsymbol{y}$ is sampled as a sparse polynomial with $\mathrm{HW}(\boldsymbol{y}) = w$, which implies that $\mathrm{HW}(\boldsymbol{y}_i^{(0)})$ is close to zero with high probability. It is, therefore, sufficient for a successful attack to only retrieve the support of $\boldsymbol{y}^{(0)}$. In the following, the support distribution of $\boldsymbol{y}^{(0)}$ is analyzed as the attacks for the different HQC versions have restrictions on the HW of the individual blocks $\boldsymbol{y}_i^{(0)}$.

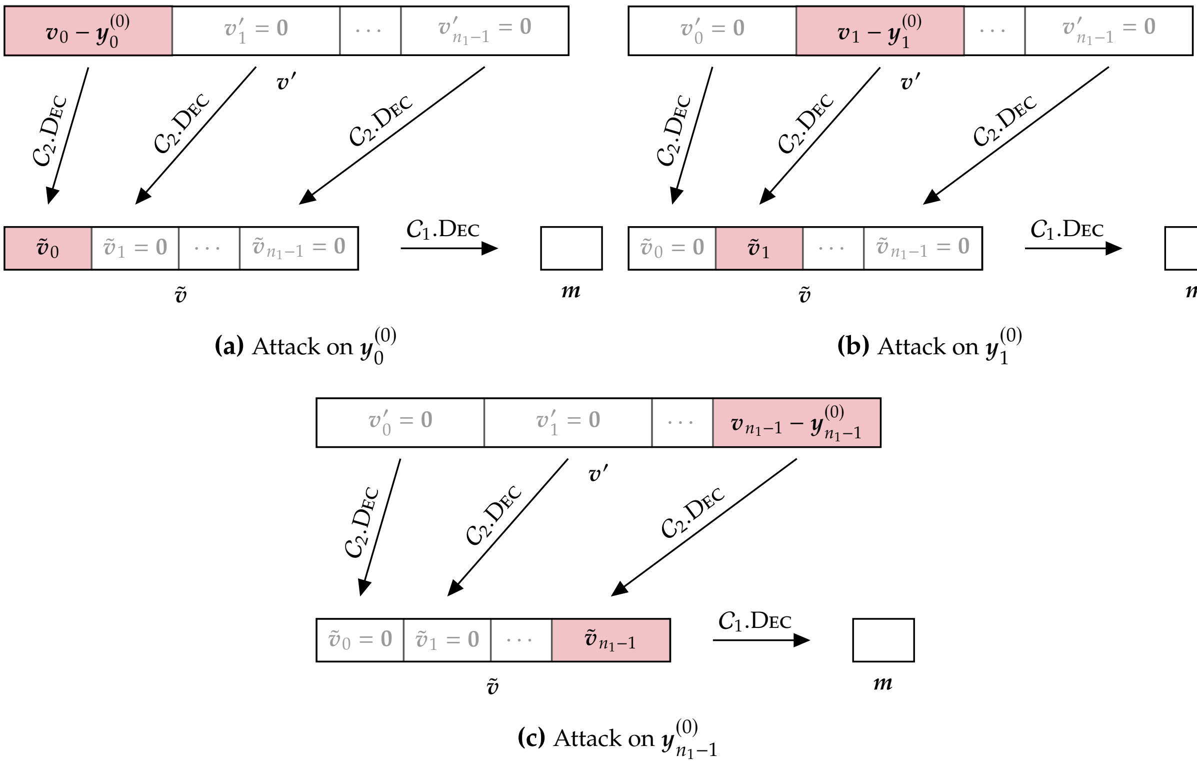
For the attack on HQC-BCH, there is a weight restriction on the maximum HW of all $\boldsymbol{y}_i^{(0)}$ such that $\mathrm{HW}(\boldsymbol{y}_i^{(0)}) \le w_{max}$ for $i = 0, \dots, n_1 - 1$. The probabilities that $\boldsymbol{y}$ is generated such that this restriction is fulfilled for all blocks of $\boldsymbol{y}_i^{(0)}$ are shown in Table 4.3. The results are obtained through a simulation that generates one million secret keys of the different parameters of HQC-BCH. It can be seen that most of the secret keys show a w_{max} of at most 3.

We show a proof of the attack strategy for HQC-RMRS with the restriction that the maximum Hamming weight of all $\boldsymbol{y}_i^{(0)}$ is smaller than $y_{w,max}$, where $y_{w,max} = \max\{\mathrm{HW}(\boldsymbol{y}_0^{(0)}), \dots, \mathrm{HW}(\boldsymbol{y}_{n_1-1}^{(0)})\}$. We determine the probabilities of the different $y_{w,max}$ by simulating the weight distribution of ten million samples of $\boldsymbol{y}$ with the results shown in Table 4.4.

The probability that $\mathrm{HW}(\boldsymbol{y}^{(1)}) > 0$ can be computed by $1 - \binom{n_1 \cdot n_2}{w} / \binom{n}{w}$. Considering the parameters of HQC-RMRS-128, HQC-RMRS-192, and HQC-RMRS-256 we determine the respective probabilities to be 1.85 %, 3.02 %, and 8.07 %. For HQC-BCH the resulting probabilities are 29.29 %, 0.66 %, and 1.34 % for HQC-BCH-128, HQC-BCH-192 and HQC-BCH-256, respectively. Therefore, in most cases it suffices to determine $\boldsymbol{y}^{(0)}$ because there are no ones in $\boldsymbol{y}^{(1)}$.

$y_{w,max}$	3	4	5	6	7	8	9
HQC-RMRS-128	3.75%	48.77%	86.49%	97.44%	99.59%	99.94%	99.99%
HQC-RMRS-192	0.01%	10.74%	57.96%	88.50%	97.57%	99.56%	99.93%
HQC-RMRS-256	0.09%	20.83%	71.87%	93.94%	98.96%	99.84%	99.97%

Table 4.4 Probabilities that $\boldsymbol{y}^{(0)}$ is generated such that $\max\{\mathrm{HW}(\boldsymbol{y}_0^{(0)}), \ldots, \mathrm{HW}(\boldsymbol{y}_{n_1-1}^{(0)})\}$ is at most $y_{w,max}$ for all parameter sets of HQC-RMRS.



$\boldsymbol{v}_0 - \boldsymbol{y}_0^{(0)}$
$v'_1 = 0$
$\cdots$
$v'_{n_1-1} = 0$
$\boldsymbol{v}'$
C_2.Dec
$\tilde{\boldsymbol{v}}_0$
$\tilde{v}_1 = 0$
$\tilde{v}_{n_1-1} = 0$
C_1.Dec
$\tilde{\boldsymbol{v}}$
$\boldsymbol{m}$
(a) Attack on $\boldsymbol{y}_0^{(0)}$
$v'_0 = 0$
$\boldsymbol{v}_1 - \boldsymbol{y}_1^{(0)}$
$\tilde{v}_0 = 0$
$\tilde{\boldsymbol{v}}_1$
(b) Attack on $\boldsymbol{y}_1^{(0)}$
$\boldsymbol{v}_{n_1-1} - \boldsymbol{y}_{n_1-1}^{(0)}$
$\tilde{\boldsymbol{v}}_{n_1-1}$
(c) Attack on $\boldsymbol{y}_{n_1-1}^{(0)}$

Figure 4.12 Visualization of the attack strategy to reveal the individual $\boldsymbol{y}_i^{(0)}$ for $0 \leq i \leq n-1$.

4.4.2 Retrieving $y^{(0)}$ Using a Side-Channel Oracle

With our chosen-ciphertext attack, we are able to determine all parts of the secret key $\boldsymbol{y}_i^{(0)}$ individually and in a sequential manner. The following describes the general attack strategy using an oracle obtained through a side-channel of the implementation. Note that the individual ways to obtain such an oracle are detailed in the respective attack sections (Section 4.5, Section 4.6) and that for a description of the attack strategy, it is sufficient to assume access to the oracle.

Within the HQC algorithm, the only operation utilizing the secret key is the decoding of the vector $\boldsymbol{v}' = \boldsymbol{v} - \boldsymbol{u}\boldsymbol{y}$ during decryption (c.f. Algorithm 9). By setting $\boldsymbol{u} = (1, 0 \ldots, 0) \in \mathbb{F}_2^n$ the decoder input results in $\boldsymbol{v}' = \boldsymbol{v} - \boldsymbol{y}$. As $\boldsymbol{v}$ is part of the ciphertext $c = (\boldsymbol{u}, \boldsymbol{v})$, it is controllable by the attacker. By setting it to a valid codeword, $\boldsymbol{y}$ can be seen as the error $\boldsymbol{e}$ that has to be corrected by the decoder. Note that due to the sparseness of $\boldsymbol{y}$, it is sufficient to only retrieve its support as discussed in Section 4.4.1. Now, the general idea of the attack is to choose $\boldsymbol{v}$ such that the decoding result only depends on a single $\boldsymbol{y}_i^{(0)}$, revealing its support. This is done by setting all $\boldsymbol{v}_i$ to the all-zero codeword except for the one corresponding to the targeted part of the secret key. The iterative attack process is visualized in Fig. 4.12.

In the case of HQC, an attacker requires access to the individual decoder results of C_2, as the respective input consists of $\boldsymbol{y}_i^{(0)}$ subtracted from its corresponding $\boldsymbol{v}_i$. It is not possible to directly attack the decapsulation of HQC (c.f. Algorithm 11), as it includes a check for the validity of the ciphertext that does not reveal information about $\boldsymbol{m}$ in case of failure. Nevertheless, a side-channel can be used in order to construct an oracle that again reveals information about the decoding result. For the presented attacks in this thesis, a separate oracle is defined that is tailored to the respective HQC version[9]. For HQC-BCH, we define a decoding oracle $O_{01}^{Dec}(\boldsymbol{v})$ that is able to determine whether an error is corrected by the BCH code (C_1). After the oracle has been initialized, it can be queried for different inputs $\boldsymbol{v}$ and returns 1 if an error had to be corrected and 0 otherwise. For HQC-RMRS, we define an oracle that is able to determine whether the RM decoder (C_2) decoded to the all-zero or a non-zero codeword in a given position. The oracle is formally defined in Definition 1. Note in Fig. 4.12 that through setting all $\boldsymbol{v}_i$ except the targeted one to zero, the decoding result of C_1.DEC can also be used to identify the decoding result of the inner code C_2.

Definition 1 (Close-to-zero Oracle)**.** *Let C be an $\mathcal{RM}(1, m)$ code. Define $\mathfrak{D}_0^{\boldsymbol{e}} : \mathbb{F}_q^n \mapsto \{\mathsf{True}, \mathsf{False}\}$ with $\boldsymbol{e} \in \mathbb{F}_q^n$ to be the function given by*

$$\mathfrak{D}_0^{\boldsymbol{e}}(\boldsymbol{r}) = \begin{cases} \mathsf{True}, & \text{if } \mathfrak{D}_{\mathcal{RM}}(\boldsymbol{r} + \boldsymbol{e}) = \boldsymbol{0}, \\ \mathsf{False}, & \text{else}, \end{cases}$$

where $\mathfrak{D}_{\mathcal{RM}}$ denotes a decoder for the RM code.

By querying the oracle and, therefore, gaining access to the decoding result, the attack strategy is based on two steps. First, an input $\boldsymbol{v}$ has to be found such that the resulting codeword after the subtraction of the corresponding $\boldsymbol{y}_i^{(0)}$ lies exactly at the decoding boundary of the C_2 decoder[10]. An example is to find an input that lies exactly one error above the boundary, meaning it results in a decoding error, i.e., in the decoder not returning the all-zero codeword $\boldsymbol{0}$, and therefore, e.g., $\mathfrak{D}_0^{\boldsymbol{e}}$ returns False. This implies that if we set an additional bit in the input, which is in the support of $\boldsymbol{y}_i^{(0)}$, we reduce the error resulting in a successful decoding indicated by, e.g., $\mathfrak{D}_0^{\boldsymbol{e}}$ as True. In the second attack step, an attacker can query the oracle by successively inverting each bit of the input found in the first step. In this process, an oracle result of True, i.e., a successful decoding to the all-zero codeword, indicates that this position is in the support of $\boldsymbol{y}_i^{(0)}$. This allows to retrieve the whole support of the attacked secret key block. By repeating this approach for all n_1 blocks of C_2 codewords, the complete $\boldsymbol{y}^{(0)}$ can be retrieved.

4.4.3 Retrieval of $\boldsymbol{y}^{(1)}$

With our attack strategy, we are able to attack $\boldsymbol{y}^{(0)}$, as only this part acts as an input to the decoder. Nevertheless, we discuss methods to retrieve the whole $\boldsymbol{y}$ in this section if $\boldsymbol{y}^{(0)}$ has been retrieved completely and error-free. In the case that $\boldsymbol{y}^{(0)}$ is only partially retrieved, information set decoding can be used, as described in Section 4.4.4, which directly reveals the complete secret key $\boldsymbol{y}$ from this partial information.

As discussed in Section 4.4.1, the size of $\boldsymbol{y}^{(1)}$ is given by $n - n_1 n_2$, which results in the sizes shown in Table 4.5. With the resulting sizes being reasonably small and $\mathrm{HW}(\boldsymbol{y}^{(1)})$ close to zero with high probability, the remaining entries in $\boldsymbol{y}^{(1)}$ can be found through an exhaustive search. For the HQC-BCH-128 parameter set, a reasonable restriction for $\mathrm{HW}(\boldsymbol{y}^{(1)})$ has to be set in order to achieve a feasible brute-force complexity. In the following, a brute-force method based on linear algebra is presented.

[9]In order to be consistent with the published version of the attacks in [SRSWZ21] and [SHR+22], the oracles are presented as defined in the papers.

[10]Note that the attack strategy on HQC-BCH presented in Section 4.5 only finds codewords that are at the decoding boundary given a maximum hamming weight of the attacked $\boldsymbol{y}_i^{(0)}$. Therefore, not all keys of HQC-BCH can be attacked as discussed in Section 4.4.1. In contrast, the attack strategy on HQC-RMRS reveals codewords mostly independent of the hamming weight of $\boldsymbol{y}_i^{(0)}$. In other words, there is a relaxed restriction on the maximum hamming weight with a sufficient margin that allows to attack honestly generated keys.

	$n - n_1 n_2$		$\mathcal{W}_{\mathrm{L}}(\mathrm{HW}(\boldsymbol{y}^{(1)}) \leq 2)$	
	BCH	RMRS	BCH	RMRS
HQC-*-128	123	5	$2^{28.42}$	$2^{18.43}$
HQC-*-192	3	11	$2^{18.05}$	$2^{21.91}$
HQC-*-256	7	37	$2^{21.47}$	$2^{26.19}$

Table 4.5 Length of $\boldsymbol{y}^{(1)}$ that has to be found through an exhaustive search and the corresponding work factor of the presented approach based on linear algebra.

Let $\mathcal{J} = \{j_0, \dots, j_{t-1}\}$ denote the known support of $\boldsymbol{y}^{(0)}$ and let $\mathcal{L} = \{l_0, \dots, l_{w-t-1}\}$ be the support of $\boldsymbol{y}^{(1)}$ that we want to determine. First, observe that

$$\boldsymbol{s} = \boldsymbol{x} + \boldsymbol{h}\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{H}_{n+j_0}^{\top} + \dots + \boldsymbol{H}_{n+j_{t-1}}^{\top} + \boldsymbol{H}_{n+l_0}^{\top} + \dots + \boldsymbol{H}_{n+l_{w-t-1}}^{\top},$$

where $\boldsymbol{H}_i$ denotes the i-th column of $\boldsymbol{H} = (\mathbf{1}, \mathrm{rot}(\boldsymbol{h}))$. Since $\boldsymbol{s}$, $\boldsymbol{h}$ and $\mathcal{J}$ are known, we can compute

$$\tilde{\boldsymbol{s}} = \boldsymbol{s} + \boldsymbol{H}_{n+j_0}^{\top} + \dots + \boldsymbol{H}_{n+j_{t-1}}^{\top} = \boldsymbol{x} + \boldsymbol{H}_{n+l_0}^{\top} + \dots + \boldsymbol{H}_{n+l_{w-t-1}}^{\top}.$$

Then, we repeatedly sample $w - t$ indices $\hat{l}_0, \dots, \hat{l}_{w-t-1}$ from $\{0, \dots, n - n_1 n_2 - 1\}$ and compute

$$\hat{\boldsymbol{x}} := \tilde{\boldsymbol{s}} + \boldsymbol{H}_{n+\hat{l}_0}^{\top} + \dots + \boldsymbol{H}_{n+\hat{l}_{w-t-1}}^{\top}$$

until $\mathrm{HW}(\hat{\boldsymbol{x}}) = w$. In this case, $\hat{\boldsymbol{x}} = \boldsymbol{x}$ which means that $\{\hat{l}_0, \dots, \hat{l}_{w-t-1}\} = \mathcal{L}$. We finally output $\mathcal{J} \cup \{\hat{l}_0, \dots, \hat{l}_{w-t-1}\}$ as estimation of $\mathrm{supp}(\boldsymbol{y})$. The probability that $\{\hat{l}_0, \dots, \hat{l}_{w-t-1}\} = \mathcal{L}$ is $\binom{n-n_1n_2}{w-t}^{-1}$ and checking whether $\{\hat{l}_0, \dots, \hat{l}_{w-t-1}\}$ is equal to $\mathcal{L}$ requires $w - t$ column additions which is in $O\big(n(w-t)\big)$. This results in a work factor of

$$\mathcal{W}_{\mathrm{L}} = n(w-t)\binom{n - n_1 n_2}{w - t}.$$

Assuming $w - t \leq 2$, the resulting work factor is shown in the right column of Table 4.5.

4.4.4 Utilizing Partial Attack Results through Information Set Decoding

For the proposed attacks, it is important to recognize that retrieving the secret key $\mathsf{sk} = (\boldsymbol{x}, \boldsymbol{y})$ from the public key $\mathsf{pk} = (\boldsymbol{h}, \boldsymbol{s})$ is equal to solving an instance of the 2-QCSD problem. This can be seen by

$$\boldsymbol{s} = \boldsymbol{x} + \boldsymbol{h}\boldsymbol{y} = (\boldsymbol{x}, \boldsymbol{y})\begin{pmatrix} \mathbf{1}^{\top} \\ \mathrm{rot}(\boldsymbol{h})^{\top} \end{pmatrix} = \boldsymbol{e}\boldsymbol{H}^{\top},$$

where $\boldsymbol{e} := (\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{F}_2^{2n}$ with $\mathrm{HW}(\boldsymbol{x}) = \mathrm{HW}(\boldsymbol{y}) = w$ and $\boldsymbol{H} := (\mathbf{1}, \mathrm{rot}(\boldsymbol{h})) \in \mathbb{F}_2^{n \times 2n}$. The vector $\boldsymbol{s}$ can be interpreted as the syndrome of the error $\boldsymbol{e}$ and the parity-check matrix $\boldsymbol{H}$.

Therefore, this problem can be solved by information set decoding (ISD) algorithms like Prange's algorithm [Pra62] or one of the improvements, e.g., Stern's algorithm [Ste89] revealing the complete secret key $\boldsymbol{y}$. The idea of Prange's ISD algorithm is to guess an error-free information set, i.e., a set of indices $\{I_0, \dots, I_{n-1}\} \subset \{0, \dots, 2n-1\}$ such that $e_{I_0} = \dots = e_{I_{n-1}} = 0$ and such that the columns $\bar{I}_0, \dots, \bar{I}_{n-1}$ of $\boldsymbol{H}$ are linearly independent, where $\{\bar{I}_0, \dots, \bar{I}_{n-1}\} = \{0, \dots, 2n-1\} \setminus \{I_0, \dots, I_{n-1}\}$. We can use the partial attack results $\mathcal{P} = \{p_0, \dots, p_{t-1}\} \subset \mathrm{supp}(\boldsymbol{y})$ to reduce the complexity of a used information set decoding algorithm. This is done by modifying the syndrome vector to include partial information as

$$\boldsymbol{s}' = \boldsymbol{s} + \boldsymbol{H}_{n+p_0}^{\top} + \dots + \boldsymbol{H}_{n+p_{t-1}}^{\top},$$

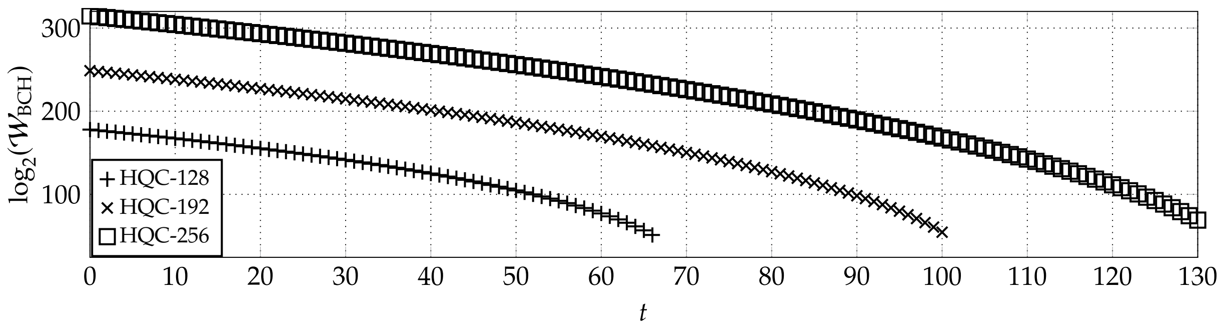


Figure 4.13 Complexity $\mathcal{W}_{\mathrm{BCH}}$ of Prange's algorithm for all parameter sets of HQC-BCH as a function of t, where t is the number of non-zero positions in $\boldsymbol{y}^{(0)}$ that are correctly obtained by the proposed side-channel attack [SRSWZ21, Figure 4].

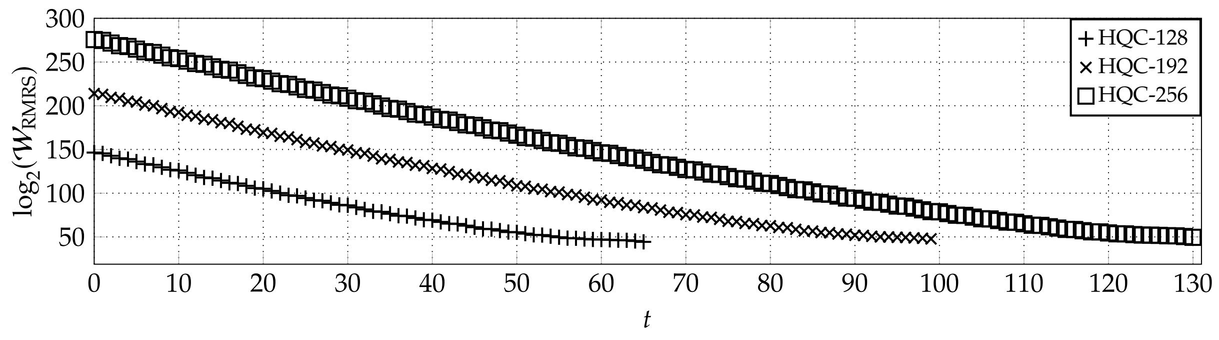


Figure 4.14 Complexity $\mathcal{W}_{\mathrm{RMRS}}$ of Stern's algorithm for all parameter sets of HQC-RMRS as a function of t, where t is the number of non-zero positions in $\boldsymbol{y}^{(0)}$ that are correctly obtained by the proposed side-channel attack [SHR+22, Figure 2].

where $\boldsymbol{H}_i$ denotes the i-th column of $\boldsymbol{H} = (\boldsymbol{1}, \mathrm{rot}(\boldsymbol{h}))$. There are two main reasons for an attacker to use partial attack results. Either there is a limit to the number of possible oracle calls due to the number of decryptions the attacker can observe, or the side-channel used to create the oracle does not result in perfect oracle answers. A limit on the oracle calls allows to only attack a subset of all $\boldsymbol{y}_i^{(0)}$ and the partial attack results consist of the successfully attacked blocks. In case of incorrect oracle results, the partial attack results can be built by only using retrieved $\boldsymbol{y}_i^{(0)}$ that show the highest classification results as established for attacks on other public-key cryptosystems [HIM+14, KSTS21].

In Fig. 4.13 and Fig. 4.14, the resulting work factor given the knowledge of t elements of the support of $\boldsymbol{y}^{(0)}$ are shown for HQC-BCH, and HQC-RMRS, respectively. As the concrete modifications of the respective ISD algorithms are developed by other authors, this section only presents the resulting work factors, and the reader is referred to the published works, which is [SRSWZ21] for HQC-BCH and [SHR+22] for HQC-RMRS. For a general approach on how to incorporate partial side-channel attack results with information set decoding, the reader is referred to [HPR+22].

4.5 Attack Strategy on HQC-BCH

This section presents the attack strategy on the second round version of the HQC algorithm (HQC-BCH) that combines a repetition code with a BCH code. First, the attack strategy is detailed in Section 4.5.1. In Section 4.5.2, the method to construct the required oracle O_{01}^{Dec} through a power side-channel is described. Section 4.5.3 shows attack results of this oracle for a software implementation of HQC-BCH-128 running on an ARM-Cortex M4 microcontroller. Finally, the attack strategy is compared with related work in Section 4.5.4.

4.5.1 Attack Strategy

In this section, we propose a chosen ciphertext attack that retrieves $\boldsymbol{y}_i^{(0)}$ for $i = 0, \dots, n_1 - 1$. First, we fix $\boldsymbol{u}$ to $(1, 0, \dots, 0) \in \mathbb{F}_2^n$ such that the vector that is fed into the decoder of $\mathcal{C}$ is given by

$$\boldsymbol{v}' = \boldsymbol{v} - \boldsymbol{y},$$

as it can be seen in Algorithm 9. Now, the oracle $O_{01}^{Dec}(\boldsymbol{v})$, as defined in Section 4.4.2, can be queried for different inputs $\boldsymbol{v}$, and using the results allows to obtain the individual $\boldsymbol{y}_i^{(0)}$.

To derive the required vectors $\boldsymbol{v}$, recall that the code $\mathcal{C}$ is a code concatenation consisting of a BCH code $\mathcal{C}_1$ of length n_1 and a repetition code $\mathcal{C}_2$ of length n_2, and only the first $n_1 n_2$ positions of $\boldsymbol{v}'$ are decoded in $\mathcal{C}$. The decoder for $\mathcal{C}$ divides the first $n_1 n_2$ positions of $\boldsymbol{v}'$ into chunks $\boldsymbol{v}'_0, \dots, \boldsymbol{v}'_{n_1-1}$ of size n_2 that are separately decoded in the repetition code. Decoding in the repetition code $\mathcal{C}_2$ is performed by a majority voting, meaning vectors of Hamming weight at least $\lceil \frac{n_2}{2} \rceil$ are mapped to 1 and the remaining vectors are mapped to 0. The outputs of the repetition decoder are then decoded in the BCH code. Since the all-zero vector is in $\mathcal{C}_1$ and vectors of Hamming weight one[11] are not in $\mathcal{C}_1$, we observe the following. Setting $\lceil \frac{n_2}{2} \rceil$ entries of $\boldsymbol{v}_i$ to 1 and $\boldsymbol{v}_j$ to the all-zero vector, where $j \in [0, n_1 - 1] \setminus \{i\}$, results in two cases that we can distinguish using O_{01}^{Dec}:

1. $|\operatorname{supp}(\boldsymbol{y}_i^{(0)}) \cap \operatorname{supp}(\boldsymbol{v}_i)| > \frac{\mathrm{HW}(\boldsymbol{y}_i^{(0)})}{2}$: This leads to $\mathrm{HW}(\boldsymbol{v}'_i) < \lceil \frac{n_2}{2} \rceil$ and the repetition decoder outputs 0. Then, no error is corrected in the BCH code.

2. $|\operatorname{supp}(\boldsymbol{y}_i^{(0)}) \cap \operatorname{supp}(\boldsymbol{v}_i)| \leq \frac{\mathrm{HW}(\boldsymbol{y}_i^{(0)})}{2}$: This leads to $\mathrm{HW}(\boldsymbol{v}'_i) \geq \lceil \frac{n_2}{2} \rceil$ and the repetition decoder outputs 1, which is corrected in the BCH code.

This observation allows to determine the support of $\boldsymbol{y}_i^{(0)}$ in a two-step approach, where we first determine a super support of $\boldsymbol{y}_i^{(0)}$, which is an approximation of the support (c.f. Section 4.3.1), and then refine these approximate locations to obtain the exact non-zero positions of $\boldsymbol{y}_i^{(0)}$. Note that all $\boldsymbol{y}_0^{(0)}, \dots, \boldsymbol{y}_{n_1-1}^{(0)}$ are examined separately in sequential manner.

Finding a Super Support of $\boldsymbol{y}_i^{(0)}$

In the following, we derive how to choose $\boldsymbol{v}_i$ to determine a super support of $\boldsymbol{y}_i^{(0)}$ under the assumption that $\mathrm{HW}(\boldsymbol{y}_i^{(0)}) \leq 2$. As shown in Table 4.3, this already covers a large part of the possible keys. Nevertheless, a generalization of the proposed method to cases where $\mathrm{HW}(\boldsymbol{y}_i^{(0)}) > 2$ works accordingly. Assuming $\mathrm{HW}(\boldsymbol{y}_i^{(0)}) \leq 1$, a super support of $\boldsymbol{y}_i^{(0)}$ can be found by using only two patterns of $\boldsymbol{v}_i$. For pattern 0, we choose $\operatorname{supp}(\boldsymbol{v}_i) = [0, \lceil \frac{n_2}{2} \rceil - 1]$ and for pattern 1, we choose $\operatorname{supp}(\boldsymbol{v}_i) = [\lceil \frac{n_2}{2} \rceil - 1, n_2 - 1]$. The patterns for $n_2 = 31$ (HQC-BCH-128) are illustrated in Fig. 4.15.

[11]This follows from the fact that the BCH code has a minimum distance larger than 1.

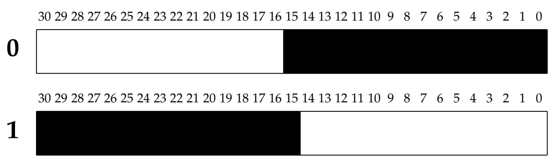


Figure 4.15 Patterns of v_i to find a super support of $y_i^{(0)}$ for $n_2 = 31$ and $\mathrm{HW}(y_i^{(0)}) \leq 1$. The black part indicates positions with value 1, and the white part positions with value 0.

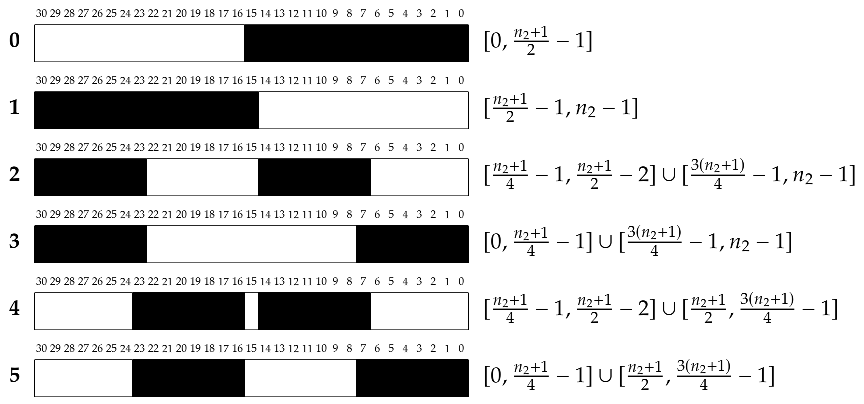

30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 $[0, \frac{n_2+1}{2} - 1]$
1 $[\frac{n_2+1}{2} - 1, n_2 - 1]$
2 $[\frac{n_2+1}{4} - 1, \frac{n_2+1}{2} - 2] \cup [\frac{3(n_2+1)}{4} - 1, n_2 - 1]$
3 $[0, \frac{n_2+1}{4} - 1] \cup [\frac{3(n_2+1)}{4} - 1, n_2 - 1]$
4 $[\frac{n_2+1}{4} - 1, \frac{n_2+1}{2} - 2] \cup [\frac{n_2+1}{2}, \frac{3(n_2+1)}{4} - 1]$
5 $[0, \frac{n_2+1}{4} - 1] \cup [\frac{n_2+1}{2}, \frac{3(n_2+1)}{4} - 1]$

Figure 4.16 Patterns of v_i to find a super support of $y_i^{(0)}$ for n_2=31 and $\mathrm{HW}(y_i^{(0)}) \leq 2$. The black part indicates positions with value 1 and the white part entries with value 0. In addition, the support of v_i dependent on n_2 is given.

If the BCH decoder has to correct an error for both patterns, it follows that $\mathrm{HW}(y_i^{(0)}) = 0$ and in case no error was corrected by the BCH code in both cases, we conclude that $\mathrm{supp}(y_i^{(0)}) = \mathrm{ssupp}(y_i^{(0)}) = \{\lceil \frac{n_2}{2} \rceil - 1\}$. Furthermore, if the BCH decoder has to correct an error for the first pattern but not for the second pattern, we know that $\mathrm{supp}(y_i^{(0)}) \cap \mathrm{ssupp}(y_i^{(0)}) = [\lceil \frac{n_2}{2} \rceil, n_2 - 1]$. Given the BCH decoder does not correct an error in the first case but in the second we know that $\mathrm{supp}(y_i^{(0)}) \cap \mathrm{ssupp}(y_i^{(0)}) = [0, \lceil \frac{n_2}{2} \rceil - 2]$.

Given the HQC-BCH parameter sets and the case that $\mathrm{HW}(y_i^{(0)}) \leq 2$, the attack can be generalized as follows. Instead of only two patterns, we construct six different patterns of v_i. An illustration of the six patterns for $n_2 = 31$ together with the general formulas for the sets dependent on n_2 is given in Fig. 4.16. Similar to before, we can determine a super support of $y_i^{(0)}$ based on the output of the oracle for the different patterns of v_i, where the logic is given in Table 4.6. In the table, either one or two sets per row are shown. The union of these sets gives a super support of $y_i^{(0)}$, and each set has a non-empty intersection with the support of $y_i^{(0)}$. The latter property is important since it reduces the complexity of the next step.

Finding the support of $y_i^{(0)}$

From the super support of $y_i^{(0)}$, we can determine $\mathrm{supp}(y_i^{(0)})$ using the fact that all indices of $y_i^{(0)}$ that are not in $\mathrm{ssupp}(y_i^{(0)})$ correspond to entries with value zero. As already discussed, we describe the proposed method for $\mathrm{HW}(y_i^{(0)}) \leq 2$.

Assume that $\mathrm{HW}(y_i^{(0)}) = 1$. We can find the support of $y_i^{(0)}$ by setting $\lceil \frac{n_2}{2} \rceil - 1$ entries in v_i to 1 for indices which are not in $\mathrm{ssupp}(y_i^{(0)})$. Keeping these entries fixed, we iterate through all vectors v_i with $|\,\mathrm{supp}(v_i) \cap \mathrm{ssupp}(y_i^{(0)})| = 1$. This procedure is depicted for $n_2 = 31$ and $\mathrm{ssupp}(y_i^{(0)}) = \{0, \ldots, 14\}$

O_{01}^{Dec}(Pattern $\star$)						$\mathrm{ssupp}(\boldsymbol{y}_i^{(0)})$
0	1	2	3	4	5	
1	1	1	1	1	1	$\{\,\}$
0	0	-	-	-	-	$\{\frac{n_2+1}{2}-1\}$
0	1	-	-	-	-	$[0:\frac{n_2+1}{2}-1]$
1	0	-	-	-	-	$[\frac{n_2+1}{2}-1:n_2-1]$
1	1	0	1	1	1	$[\frac{n_2+1}{4}:\frac{n_2+1}{2}-2]$, $[\frac{3(n_2+1)}{4}:n_2-1]$
1	1	1	0	1	1	$[0:\frac{n_2+1}{4}-2]$, $[\frac{3(n_2+1)}{4}:n_2-1]$
1	1	1	1	0	1	$[\frac{n_2+1}{4}:\frac{n_2+1}{2}-2]$, $[\frac{n_2+1}{2}:\frac{3(n_2+1)}{4}-2]$
1	1	1	1	1	0	$[0:\frac{n_2+1}{4}-2]$, $[\frac{n_2+1}{2}:\frac{3(n_2+1)}{4}-2]$
1	1	0	0	1	1	$\{\frac{n_2+1}{4}-1\}$, $[\frac{3(n_2+1)}{4}:n_2-1]$
1	1	0	1	0	1	$[\frac{n_2+1}{4}:\frac{n_2+1}{2}-2]$, $\{\frac{3(n_2+1)}{4}-1\}$
1	1	1	0	1	0	$[0:\frac{n_2+1}{4}-2]$, $\{\frac{3(n_2+1)}{4}-1\}$
1	1	1	1	0	0	$\{\frac{n_2+1}{4}-1\}$, $[\frac{n_2+1}{2}:\frac{3(n_2+1)}{4}-2]$
1	1	0	0	0	0	$\{\frac{n_2+1}{4}-1\}$, $\{\frac{3(n_2+1)}{4}-1\}$

Table 4.6 Super support of $\boldsymbol{y}_i^{(0)}$ depending on the oracle output for different patterns of $\boldsymbol{v}_i$ (see Fig. 4.16) and $\mathrm{HW}(\boldsymbol{y}_i^{(0)}) \leq 2$.

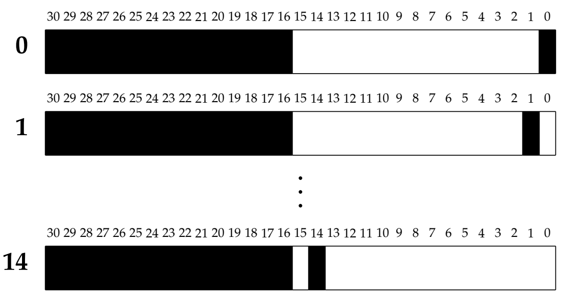


Figure 4.17 Patterns to determine $\mathrm{supp}(\boldsymbol{y}_i^{(0)})$ from $\mathrm{ssupp}(\boldsymbol{y}_i^{(0)})$ for $n_2 = 31$ and $\mathrm{ssupp}(\boldsymbol{y}_i^{(0)}) = \{0, \ldots, 14\}$.

in Fig. 4.17. Every time the BCH decoder corrects an error, we know that $\mathrm{supp}(\boldsymbol{v}_i) \cap \mathrm{ssupp}(\boldsymbol{y}_i^{(0)}) \neq \mathrm{supp}(\boldsymbol{y}_i^{(0)})$ and when the BCH decoder does not correct an error, we can conclude that $\mathrm{supp}(\boldsymbol{v}_i) \cap \mathrm{ssupp}(\boldsymbol{y}_i^{(0)}) = \mathrm{supp}(\boldsymbol{y}_i^{(0)})$.

For $\mathrm{HW}(\boldsymbol{y}_i^{(0)}) = 2$, we fix $\lceil \frac{n_2}{2} \rceil - 2$ entries in $\boldsymbol{v}_i$ to 1 for indices which are not in $\mathrm{ssupp}(\boldsymbol{y}_i^{(0)})$. In case Table 4.6 refers to one set as the super support of $\boldsymbol{y}_i^{(0)}$, we brute-force all vectors $\boldsymbol{v}_i$, where $|\,\mathrm{supp}(\boldsymbol{v}_i) \cap \mathrm{ssupp}(\boldsymbol{y}_i^{(0)})| = 2$. In case Table 4.6 refers to two sets, we iterate through all vectors $\boldsymbol{v}_i$ that have a non-empty intersection with both sets. As before, every time the BCH decoder corrects an error, we know that $\mathrm{supp}(\boldsymbol{v}_i) \cap \mathrm{ssupp}(\boldsymbol{y}_i^{(0)}) \neq \mathrm{supp}(\boldsymbol{y}_i^{(0)})$ and when the BCH decoder does not correct an error, we state $\mathrm{supp}(\boldsymbol{v}_i) \cap \mathrm{ssupp}(\boldsymbol{y}_i^{(0)}) = \mathrm{supp}(\boldsymbol{y}_i^{(0)})$.

4.5.2 Instantiation of the Decoding Oracle through a Power Side-Channel

This section introduces a method to construct a decoding oracle O_{01}^{Dec} through the power side-channel, which allows an attacker to identify whether the BCH decoder has to correct an error during the decoding step of the decryption of HQC-BCH. As explained in Section 4.5.1, this allows the attacker

O_{01}^{Dec}	$\boldsymbol{u} \in \mathbb{F}_2^n$	$\boldsymbol{v} = (v_0, \ldots, v_{n_1-1}) \in \mathbb{F}_2^{n_1 n_2}$ with $v_i \in \mathbb{F}_2^{n_2}$
0 (no error)	0	$(0, \ldots, 0)$
1 (error)	0	$(\mathrm{HW}(v_0) = \lceil \frac{n_2}{2} \rceil, 0, \ldots, 0)$

Table 4.7 Ciphertext input used for the initialization of the oracle.

to retrieve the used secret key $\boldsymbol{y}^{(0)}$. The oracle can be constructed using a t-test (c.f. Section 2.1) for POI detection in combination with a template matching approach through a sum of squared differences metric.

In [RSRCB20] Ravi et al. mounted a successful attack against the NIST candidates LAC [LLJ+19] and Round5 [BBF+19] by utilizing a power side-channel that allows to distinguish whether the used error correction had to correct an error. This section introduces their attack methodology based on a POI-reduced template matching approach with respect to its application on HQC. The oracle O_{01}^{Dec} that returns 0 if no error had to be corrected by the BCH decoder and outputs 1 otherwise can be constructed in a two-stage approach. In the first step, templates for the different classes are built based on POIs identified by the t-test. In the second step, an attacker can query the oracle in the sense that she performs template matching for the different attack traces.

In order to initialize the oracle, templates for the two different classes are built using the ciphertext inputs shown in Table 4.7. To understand these patterns, we have to recall that the used error-correcting code $\mathcal{C}$ is a code combination consisting of a BCH code $\mathcal{C}_1$ and a repetition code $\mathcal{C}_2$ of length n_2. During $\mathcal{C}$.DEC, the repetition code uses majority decoding, which results in a codeword in $\mathcal{C}_1$ that is then fed to the BCH decoder. If a single $\mathrm{HW}(v_i)$ exceeds the threshold $\lceil \frac{n_2}{2} \rceil$, the BCH decoder has to correct an error, while otherwise, the result is the all-zero codeword, for which no correction is required. We refer to Section 4.5.1 for a detailed explanation of how these values are derived. Please note that an attacker does not need to know the used secret key $\boldsymbol{y}$ in order to construct the templates. This allows to directly build the templates on the device under attack, which significantly increases the strength of the attack.

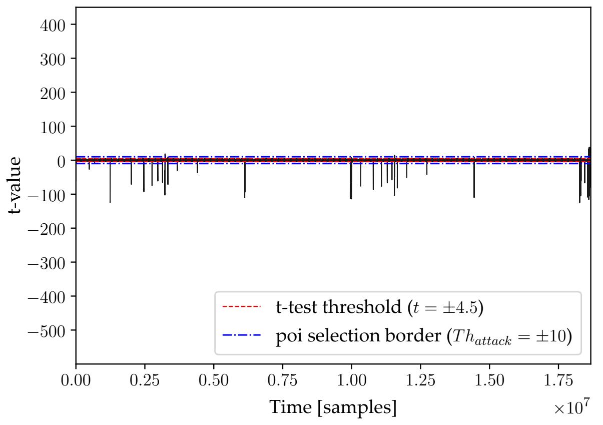
To start building the templates, a limited amount of N_t power traces for both classes, which will be denoted as $\mathcal{T}_0$ and $\mathcal{T}_1$, is recorded during the BCH decoding step of the function $\mathcal{C}$.DEC in the decryption algorithm of HQC (c.f. Algorithm 9). In order to cope with environment changes during the measurement phase, e.g., DC offsets, the individual power traces $\boldsymbol{t_i}$ are normalized for both classes. This is done by subtracting the respective mean $\bar{t}_i$, such that $\boldsymbol{t_i'} = \boldsymbol{t_i} - \bar{t}_i\mathbf{1}$. Now, a t-test is used to identify measurement samples that can be used to distinguish between the two classes. Based on these t-test results and a chosen threshold value Th_{attack} both trace sets can be reduced to their respective POIs resulting in $\mathcal{T}_0'$ and $\mathcal{T}_1'$. Finally, the templates for both classes can be calculated as the mean over all traces in their respective set, resulting in $\boldsymbol{t_m^0}$ and $\boldsymbol{t_m^1}$, respectively.

In order to evaluate the oracle for a given ciphertext input $(\boldsymbol{u}, \boldsymbol{v})$, the corresponding power trace $\boldsymbol{t_c}$ has to be captured by the attacker. The classification process is performed by an evaluation of the sum of squared differences SSD_* to both templates. The trace $\boldsymbol{t_c}$ is classified as the class with the lowest SSD value. Note that $\boldsymbol{t_c}$ also has to be reduced to the previously found POI. If both the templates $\boldsymbol{t_m^1}, \boldsymbol{t_m^0}$ and attack trace $\boldsymbol{t_c}$ are seen as a vector of their respective sample values, the evaluation can be written as

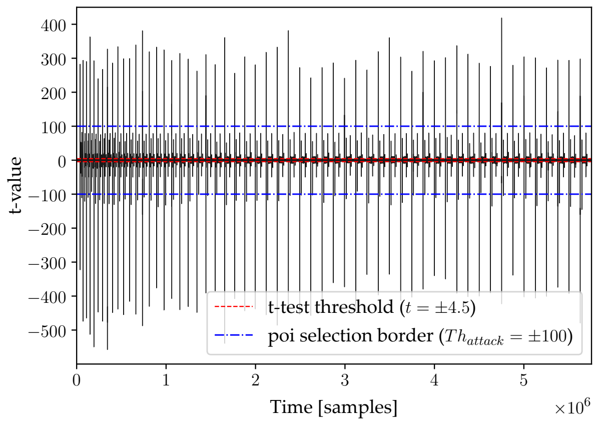
$$SSD_0 = ((\boldsymbol{t_c} - \bar{t}_c\mathbf{1}) - \boldsymbol{t_m^0})^T \cdot ((\boldsymbol{t_c} - \bar{t}_c\mathbf{1}) - \boldsymbol{t_m^0})$$
$$SSD_1 = ((\boldsymbol{t_c} - \bar{t}_c\mathbf{1}) - \boldsymbol{t_m^1})^T \cdot ((\boldsymbol{t_c} - \bar{t}_c\mathbf{1}) - \boldsymbol{t_m^1}).$$

4.5.3 Experimental Results

In order to evaluate the oracle O_{01}^{Dec}, we implemented the reference implementation of HQC-BCH-128 (c.f. Section 4.3.2 for a description of the used submission package) on our test platform consisting



(a) Compute Syndromes



(b) Error locator polynomial

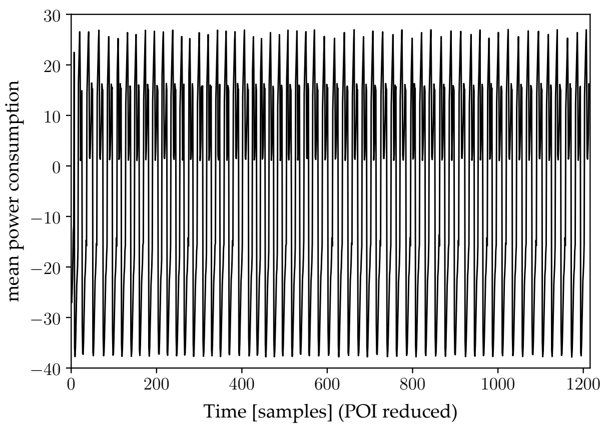
Figure 4.18 T-test results using 1000 measurements for different functions of the BCH decoding during the HQC decryption. The computation of syndromes corresponds to the function `compute_syndromes()` and the error locator polynomial computation to the function `compute_elp()` of the reference implementation.

of an STM32F415RGT6 ARM Cortex-M4 microcontroller. The microcontroller is part of a CW308T-STM32F target board, which is mounted on a CW308 UFO board running at a clock frequency of 10 MHz. The clock is provided by an external clock generator. We measured the power consumption through an integrated shunt resistor with a PicoScope 6402D USB oscilloscope at a sampling rate of 156.25 MHz. A dedicated GPIO pin is used to provide a trigger signal to the oscilloscope, indicating the duration of the function that is evaluated.

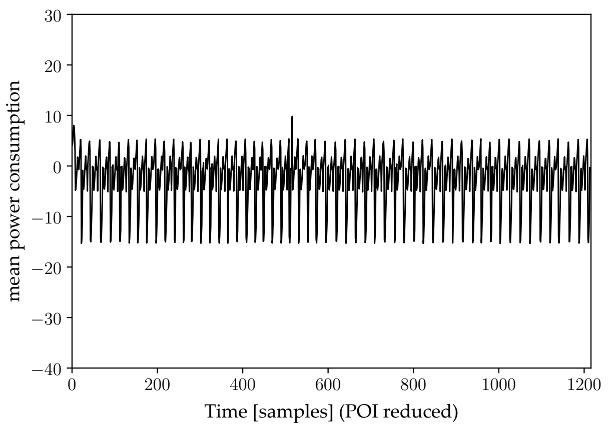
First, we evaluated if both classes can be distinguished through the power side-channel using our setup. Therefore, we performed a t-test on 1000 measurements of the BCH decoder with randomly chosen classes. As described in Section 4.3.3 and visualized in Fig. 4.6, there are three main steps during the BCH decoding, where each step could potentially be used for the distinction. In the original proposal of the attack methodology by Ravi et al. [RSRCB20], the authors find the computation of syndromes a suitable operation during the decoding. The t-test results for this attack vector are shown in Fig. 4.18a. It can be seen that there are some measurement samples with a t-value that indicates a sufficient amount of side-channel leakage that allows to distinguish both classes. Nevertheless, we opted to additionally examine the computation of the error locator polynomial, as seen in Fig. 4.18b. The result shows significantly higher leakage in addition to a reduced execution time compared to the syndrome computation. Therefore, we used the computation of the error locator polynomial as our attack target.

In a second step, we prove the efficiency of the oracle by building the templates $\boldsymbol{t}_m^0$ and $\boldsymbol{t}_m^1$ using a total of 1000 template traces (500 for each class) using only the POI given by the attack threshold Th_{attack} depicted in Fig. 4.18b. The resulting templates are shown in Fig. 4.19. It can be clearly seen that there is a significant difference between both templates. After the initialization, we evaluated 20 000 queries to the oracle, given traces with a randomly chosen class. The oracle was able to classify all measurements correctly. In an effort to lower the required amount of traces for the initialization, we iteratively evaluated the classification results with a decreasing number of template traces. As a result, we were able to successfully classify all 20 000 traces with exactly two template traces for each class.

Finally, we were able to successfully retrieve the complete secret key $\boldsymbol{y}$ of the reference implementation of HQC-BCH-128 using our measurement setup. In addition to the traces needed to initialize the oracle, the complete attack, given a maximum $\mathrm{HW}(\boldsymbol{y}_i^{(0)}) = 1$, requires 1532 traces to find $\mathrm{ssupp}(\boldsymbol{y}^{(0)})$ and 1005 traces for the final $\mathrm{supp}(\boldsymbol{y}^{(0)})$. In case of a maximum $\mathrm{HW}(\boldsymbol{y}_i^{(0)}) = 2$, the amount of pattern increases to six and therefore 4596 traces are needed to find $\mathrm{ssupp}(\boldsymbol{y}^{(0)})$. The amount of traces to



(a) t_m^0



(b) t_m^1

Figure 4.19 Computed templates after the initialization step of the oracle using 500 traces for each class.

retrieve the final $\text{supp}(\boldsymbol{y}^{(0)})$ is highly dependent on the result of $\text{ssupp}(\boldsymbol{y}^{(0)})$ and therefore, we only provide a worst-case estimation, which is a total of 3976 traces.

4.5.4 Discussion

The main related work to the described attack in this chapter is the attack by Wafo-Tapa et al. [WBB+22], which exploits a timing side-channel in the BCH decoder. Although their attack targets a timing side-channel, the general attack strategy could be used with the decoding oracle shown in Section 4.5.2. Nevertheless, there are two main differences to the presented attack approach.

First, the method to find the $\text{supp}(\boldsymbol{y}_i^{(0)})$ from knowing a super support of $\boldsymbol{y}_i^{(0)}$ is different. They identify that $\text{ssupp}(\boldsymbol{y}_i^{(0)})$ can be iteratively cut in half while proceeding only with the promising half. In case $\text{HW}(\boldsymbol{y}_i^{(0)}) = 1$, the attack of Wafo et al. determines the support of $\boldsymbol{y}_i^{(0)}$ in $\lfloor \log_2 n_2 \rfloor + 1$ requests, while our attacks requires $\left\lfloor \frac{n_2}{2} \right\rfloor$ requests to the oracle. As these requests have to be performed for each block w times, our attack requires more requests to retrieve the support of $\boldsymbol{y}_i^{(0)}$. However, the approach by Wafo et al. requires the output of the oracle before constructing the ciphertext $\boldsymbol{v}_i$ for the next query of the oracle. In the case of our power side-channel attack, this would require us to perform multiple measurements additively based on the previous attack result. In contrast, with our attack methodology, we can obtain attack traces for finding $\text{supp}(\boldsymbol{y}_i^{(0)})$ with a single measurement campaign. From a side-channel perspective, this advantage outweighs the additional queries that are needed. The difference in the required amount of traces can be found in Table 4.8. Additionally, a second difference is that Wafo et al. does not provide a method to retrieve $\boldsymbol{y}^{(1)}$.

	HQC-BCH-128	HQC-BCH-192	HQC-BCH-256	Adaptive Measurements
This work	5601	7525	10 495	no
[WBB+22]	5411	5852	6631	yes

Table 4.8 Comparision of the required oracle queries for an attack with $\text{HW}(\boldsymbol{y}_i^{(0)} \leq 2)$.

4.6 Attack Strategy for HQC-RMRS

In this section, the developed attack against the third-round version of HQC (HQC-RMRS) is presented. First, a discussion of related work is given in Section 4.6.1, where it is shown that published attacks on HQC-RMRS through the power side-channel are not applicable since the used RM decoder is implemented as a maximum likelihood decoder. This section additionally presents a counterexample for the proposed attack strategy of related work. In Section 4.6.2, an attack strategy on HQC-RMRS is presented that considers the maximum likelihood decoder of the RM code. This strategy is proven to be effective given a nonrestrictive assumption on the maximum Hamming weight of a single block of the secret key. Finally, in Section 4.6.3, we discuss different side-channel targets of the algorithm that can be used to build the required oracle for the attack strategy.

4.6.1 Unapplicability of Related Work

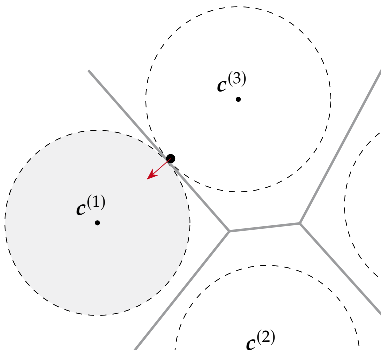
Attacks against the current version of HQC, namely HQC-RMRS, with a combination of a Reed-Muller and Reed-Solomon code, use parts of the implemented variant of the Fujisaki-Okamoto transformation to build a plaintext-checking oracle. This allows distinguishing if crafted ciphertexts decrypt to the same plaintext dependent on the secret key, again resulting in a possible attack on the whole key using multiple queries to the oracle. Xagawa et al. [XIU$^+$21b] use a fault injection to skip the ciphertext comparison setup of the transformation, resulting in direct access to the plaintext, while Ueno et al. [UXT$^+$21] attack the used PRNG required in the transformation through a power side-channel. Both use an adaption of an attack described in [BDHD$^+$19, HDV20] to HQC-RMRS.

With the change of the used codes in HQC-RMRS, the first attack step, as discussed in Section 4.4.2, namely finding an input that lies at the decoding border of the inner code, has to be changed due to the use of a different decoder type. This is the case as the decoder of the now used RM code is implemented as a maximum likelihood (ML) decoder, where ties are resolved in favor of the word of smaller lexicographical order (c.f. Eq. (4.1)). Note that ML decoding is known to be very complex and, therefore, rarely used in practice. However, for a few code classes, such as first-order RM codes, efficient decoders are known, a fact that is exploited in the HQC-RMRS system. Most other systems based on algebraic codes, such as Classic McEliece [ABC$^+$20] or HQC-BCH, instead employ bounded-distance decoders, which decode *any* error up to a given weight and fail if no codeword is within this specified radius[12]. On the other hand, for a symmetric memoryless channel, an ML decoder *always* returns (one of) the codeword(s) closest to the received word, regardless of its distance to the received word. Importantly, this implies that the behavior of this decoder does not only depend on the *number of errors* but also on the *positions of these errors*. An example of this behavior is visualized in Fig. 4.20.

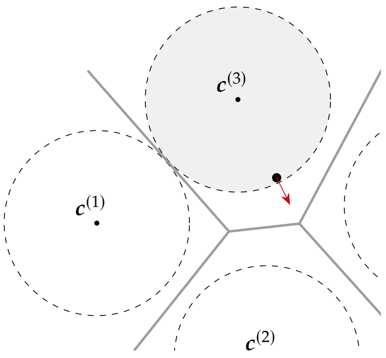
However, this independence of the error positions in a bounded-distance decoder is essential to the attack strategies in [BDHD$^+$19, HDV20]. Hence, while the setup might look similar, these methods cannot be directly applied to a system employing an ML decoder. For instance, the side-channel attack in [XIU$^+$21b, Section C.7][13] claims that the method for determining an additive error vector from oracle outputs, given in [BDHD$^+$19, Figure 7], also applies to HQC-RMRS. In the following, we show that this leads to incorrect outputs of the algorithm, which are caused by exactly this difference in behavior between an ML and a bounded-distance decoder, rendering their described attack unsuccessful.

[12]The HQC-BCH cryptosystem employs repetition codes of odd length instead of RM codes. It is well-known that this class of codes is *perfect*, i.e., the unique decoding error balls centered on the codewords fill the entire space. In this specific case, a bounded-distance decoder with radius $(d-1)/2$ is equivalent to an ML decoder. Note that first-order RM codes are *not perfect*, so this special case does not apply here.

[13]Note that the description of the attack in [UXT$^+$21] is based on the same assumptions, as it directly refers to [XIU$^+$21b].



(a) Different codeword



(b) Same codeword

Figure 4.20 Decoding results of an ML decoder for two different inputs at the decoding boundary after an additional error has been added to the input. For a), the decoding results in a different codeword $\boldsymbol{c}^{(1)}$ while for b), the same codeword $\boldsymbol{c}^{(3)}$ is returned even though the input exceeds the decoding boundary by one.

Algorithm 12 Learning a String of Small Hamming Weight [BDHD+19, Figure 7]

```
Input: $x = (00\dots0) \in \mathbb{F}_2^{n_2}, y = (11\dots1) \in \mathbb{F}_2^{n_2}$
 1: while HW$(y) > 1$ do
 2:     split at random $y = u + v$ with $u \wedge v = 0$, HW$(u) = \left\lfloor \frac{\text{HW}(y)}{2} \right\rfloor$, and HW$(v) = \left\lceil \frac{\text{HW}(y)}{2} \right\rceil$
 3:     if BOO$(x + u) = 1$ then
 4:         $x \leftarrow x + u$
 5:         $y \leftarrow v$
 6:     else
 7:         $y \leftarrow u$
 8: $z \leftarrow x$
 9: for $i = 0$ to $n_2 - 1$ do                ▷ [BDHD+19, Figure 7] skips all $i \in \text{supp}(y)$
10:     $x_{\text{test}} \leftarrow x$
11:     $x_{\text{test}}[i] \leftarrow x_{\text{test}}[i] + 1$
12:     if BOO$(x_{\text{test}}) = 1$ then
13:         $z[i] \leftarrow z[i] + 1$
Output: $z$
```

Counterexample to the Attack Strategy in [UXT+21, XIU+21b] The attacks [UXT+21, XIU+21b] claim that Algorithm 12 from [BDHD+19, Figure 7] allows for a successful attack on the individual $\boldsymbol{y}_i^{(0)}$ of HQC-RMRS given an oracle BOO$(\boldsymbol{r})$ [BDHD+19] that is defined as

$$\text{BOO}(\boldsymbol{r}) = \begin{cases} \text{True}, & \text{if HW}(\boldsymbol{r} + \boldsymbol{e}) \le \tau, \\ \text{False}, & \text{else.} \end{cases}$$

In other words, the oracle provides the information whether the sum of the input $\boldsymbol{r}$ and the error $\boldsymbol{e}$ is corrected to zero by a bounded-distance decoder of radius τ. Note that BOO$(\boldsymbol{r})$ is similar to the oracle $\mathfrak{D}_0^{\boldsymbol{e}}(\boldsymbol{r})$ (Definition 1) in this thesis, except that it assumes a bounded-distance decoder, i.e., a fixed decoding threshold, instead of an ML decoder. However, the fixed decoding threshold of the bounded-distance decoder is essential to Algorithm 12, and in the following we show that replacing the BOO$(\boldsymbol{r})$ oracle with the $\mathfrak{D}_0^{\boldsymbol{e}}(\boldsymbol{r})$ oracle, i.e., considering an ML decoder instead of a bounded-distance decoder, causes this algorithm to return incorrect results. Note that this choice of decoder is inherent to the system and cannot be influenced by the attacker. The while loop in lines 1-7 of Algorithm 12 corresponds to the first step of the attack strategy (c.f. Section 4.4.2), which is to find an input that lies exactly at the decoding boundary. Lines 9-13 of Algorithm 12 individually query the BOO oracle with a single bit of this input flipped, which then determines if the corresponding position is set in $\boldsymbol{y}_i^{(0)}$.

However, this only applies if the resulting input to the decoder is at the decoding threshold *for every position*. In Example 4.6.1, a vector in $\mathcal{RM}(1,4)$ is given, for which this is only the case for a subset of positions, which leads to an incorrect output even in the case of a single error. In addition to this counterexample, we implemented the attack strategy and performed a simulated attack by directly accessing the RM decoder results of the HQC-RMRS reference implementation. We were not able to correctly retrieve the support $\boldsymbol{y}_i^{(0)}$ with these simulations.

Example 4.6.1: Counterexample for Algorithm 12 using $\mathcal{RM}(1,4)$

The used $\mathcal{RM}(1,4)$ code is a $[16,5,8]_2$ code and therefore codewords are of size $n = 16$. The error $\boldsymbol{e}$ is fixed to $\boldsymbol{e} = (1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$. When evaluating $\mathfrak{D}_0^{e}(\boldsymbol{r})$, recall that an ML decoder returns the codeword with the smallest HD to all other codewords according to Eq. (4.1). The algorithm executes in the following way:

First while iteration		
v	$(0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1)$	
u	$(1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0)$	$\longrightarrow \mathfrak{D}_0^{e}(x+u) =$ False
$y \leftarrow u$	$(1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0)$	
Second while iteration		
v	$(1,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0)$	
u	$(0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0)$	$\longrightarrow \mathfrak{D}_0^{e}(x+u) =$ False
$y \leftarrow u$	$(0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0)$	
Third while iteration		
v	$(0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0)$	
u	$(0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0)$	$\longrightarrow \mathfrak{D}_0^{e}(x+u) =$ True
$x \leftarrow x+u$	$(0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0)$	
$y \leftarrow v$	$(0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0)$	
Fourth while iteration		
v	$(0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0)$	
u	$(0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0)$	$\longrightarrow \mathfrak{D}_0^{e}(x+u) =$ True
$x \leftarrow x+u$	$(0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0)$	
$y \leftarrow v$	$(0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0)$	
$\mathrm{HW}(y) = 1$	Terminate while loop.	

As claimed by the algorithm, x is now an input that is at the decoding boundary, i.e., if a position inside its support is flipped, we decode to the all-zero codeword ($\mathfrak{D}_0^{e}(x) =$ True). Now lines 9-13 of Algorithm 12 are executed:

Initialize z		
$z \leftarrow x$	$(0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0)$	
Test first position		
$x + e^{(1)}$	$(1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0)$	$\longrightarrow \mathfrak{D}_0^{e}(x+e^{(1)}) =$ True
$z \leftarrow z + e^{(1)}$	$(1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0)$	
Test second position		
$x + e^{(2)}$	$(0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0)$	$\longrightarrow \mathfrak{D}_0^{e}(x+e^{(2)}) =$ True
$z \leftarrow z + e^{(2)}$	$(1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0)$	

Test third position

$$x + e^{(3)} \quad (0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0) \quad \rightarrow \mathfrak{D}_0^e(x + e^{(3)}) = \text{True}$$

$$z \leftarrow z + e^{(3)} \quad (1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0)$$

Test fourth position

$$x + e^{(4)} \quad (0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0) \quad \rightarrow \mathfrak{D}_0^e(x + e^{(4)}) = \text{False}$$

The oracle indicates no update to z.

After the first eight positions, the algorithm has computed $z = (1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$. For $e^{(9)}$, the difference between a bounded-distance decoder and an ML decoder begins to affect the decoding decision:

$$x + e^{(9)} \quad (0,1,1,0,1,0,0,0,1,0,0,0,0,0,0,0) \quad \rightarrow \mathfrak{D}_0^e(x + e^{(9)}) = \text{True}$$

$$z \leftarrow z + e^{(9)} \quad (1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0)$$

While $x + e^{(9)}$ has a HW of more than $d/2$, the decoder still decides for the all-zero word. This holds for all remaining positions in x, so at the end of the algorithm, z is given by

$$z = \quad (1,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1)\,.$$

Hence, the strategy does not return the correct error vector in this example.

4.6.2 Attack Strategy

Algorithm 13 FindWordAtBorder

```
Input: Oracle function $\mathfrak{D}_0^e$, Sets $\hat{\mathcal{I}}, \check{\mathcal{I}}$
Output: Vector $\boldsymbol{r} \in \mathbb{F}_2^{n_2}$
1: $\boldsymbol{r} \leftarrow \boldsymbol{0}$
2: for $\xi \in \hat{\mathcal{I}} \cap \check{\mathcal{I}}$ do
3:     if $\mathfrak{D}_0^e(\boldsymbol{r})$ = True then
4:         $r_\xi \leftarrow 1$
5:     else
6:         return Vector $\boldsymbol{r} \in \mathbb{F}_2^{n_2}$
7: for $\xi \in \hat{\mathcal{I}} \setminus \check{\mathcal{I}}$ do
8:     if $\mathfrak{D}_0^e(\boldsymbol{r})$ = True then
9:         $r_\xi \leftarrow 1$
10:    else
11:        return Vector $\boldsymbol{r} \in \mathbb{F}_2^{n_2}$
```

Algorithm 14 FindError

```
Input: Oracle function $\mathfrak{D}_0^e$, Sets $\mathcal{I}_1, \mathcal{I}_2$
Output: Vector $\tilde{\boldsymbol{e}} \in \mathbb{F}_2^{n_2}$
1: $\tilde{\boldsymbol{e}} \leftarrow \boldsymbol{0}$
2: for $\hat{\mathcal{I}} \in \{\mathcal{I}_1, [0, n_2 - 1] \setminus \mathcal{I}_1\}$ do
3:     for $\check{\mathcal{I}} \in \{\mathcal{I}_2, [0, n_2 - 1] \setminus \mathcal{I}_2\}$ do
4:         $\boldsymbol{r} \leftarrow$ FindWordAtBorder($\mathfrak{D}_0^e, \hat{\mathcal{I}}, \check{\mathcal{I}}$)
5:         $\hat{\boldsymbol{e}} \leftarrow \boldsymbol{r}$
6:         for $\xi \in \hat{\mathcal{I}} \cap \check{\mathcal{I}}$ do
7:             $r_\xi \leftarrow r_\xi + 1$
8:             if $\mathfrak{D}_0^e(\boldsymbol{r})$ = True then
9:                 $\hat{e}_\xi \leftarrow \hat{e}_\xi + 1$
10:            $r_\xi \leftarrow r_\xi + 1$
11:        $\tilde{\boldsymbol{e}}_{\hat{\mathcal{I}} \cap \check{\mathcal{I}}} \leftarrow \hat{\boldsymbol{e}}_{\hat{\mathcal{I}} \cap \check{\mathcal{I}}}$
12: return Vector $\tilde{\boldsymbol{e}} \in \mathbb{F}_2^{n_2}$
```

In this section, an attack strategy is introduced that considers the characteristics of the ML decoder of the RM code and, therefore, again allows for the correct retrieval of $\boldsymbol{y}^{(0)}$. The strategy is based on two algorithms (Algorithms 13 and 14) that exploit the well-understood structure of RM codewords as well as the support of the intersection of two codewords. The strategy is formally proven to successfully retrieve $\boldsymbol{y}^{(0)}$ if the Hamming weight of the respective RM block is smaller than $\frac{d_{\mathcal{RM}}}{4}$. The simulation of $y_{w,max}$ (see Table 4.4) indicates that this condition holds for nearly all possible keys of HQC, as 99.9 % of simulated keys show a $y_{w,max}$ of 9 with $d_{\mathcal{RM}}$ being 192 for HQC-RMRS-128 and 320 for HQC-RMRS-192/HQC-RMRS-256 (c.f. Table 4.2), respectively. First, the algorithms are introduced with the respective lemmas leading to the final theorem of Algorithm 14. In order to increase the readability, the respective proofs are given in Appendix A. Finally, this section concludes with a discussion of the required oracle calls of the presented strategy compared to related work.

To begin, Lemma 1 shows some general results on the intersection of the supports of $\mathcal{RM}(1,m)$ codewords. Note that there exists extensive literature on RM codes and their supports are well understood. The statements in the following heavily rely on the properties of the multivariate polynomials associated with each RM codeword as defined in Eq. (4.20). The construction of the different polynomials of a RM code is detailed in Section 4.2.5. Note that $p + \mathbf{1}$ is equal to the binary inverse of the polynomial p, as an addition by $\mathbf{1}$ consists of an xor with the all-one vector.

Lemma 1. *Consider two polynomials $\hat{p}, \check{p} \in \mathbb{F}_2[x]$ with $\deg(\hat{p}) = \deg(\check{p}) = 1$ and $\check{p} \notin \{\hat{p}, \hat{p}+1\}$. Denote $d = 2^{m-1}$. Then, for any $f \in \mathcal{RM}(1,m)$ we have*

$$|\operatorname{supp}(f) \cap \operatorname{supp}(\hat{p}\check{p})| = \mathrm{HW}(f\hat{p}\check{p}) = \begin{cases} 0, & \text{if } f \in \{0, \hat{p}+1, \check{p}+1, \hat{p}+\check{p}\} \\ \frac{d}{2}, & \text{if } f \in \{1, \hat{p}, \check{p}, \hat{p}+\check{p}+1\} \\ \frac{d}{4}, & \text{else.} \end{cases}$$

Using Lemma 1 allows to construct Algorithm 13 for which two polynomials $\hat{p}$ and $\check{p}$ with $\check{p} \notin \{\hat{p}, \hat{p}+1\}$ are chosen, that define $\hat{\mathcal{I}} = \operatorname{supp}(\hat{p})$ and $\check{\mathcal{I}} = \operatorname{supp}(\check{p})$ as the input to the algorithm. The resulting output $\boldsymbol{r}$ is an input to the decoder at the decoding boundary, which concludes the first part of the general attack strategy (c.f. Section 4.4.2). With this choice of polynomials, the algorithm always results in an input vector that causes a specific and controllable ML decoding result as stated in Lemma 2.

Lemma 2. *Denote by $\hat{p}, \check{p} \in \mathbb{F}_2[x]$ two polynomials with $\deg(\hat{p}) = \deg(\check{p}) = 1$ and $\check{p} \notin \{\hat{p}, \hat{p}+1\}$. Then, for $r = \mathsf{FindWordAtBorder}(\mathfrak{D}_0^e, \operatorname{supp}(\hat{p}), \operatorname{supp}(\check{p}))$ as in Algorithm 13 and $\mathcal{F} = \{\hat{p}, \check{p}, \hat{p}+\check{p}+1\}$ it holds that $\mathfrak{D}_{\mathcal{RM}}(r+e) \in \mathcal{F}$ and the decision is not the result of a tie in the distance with some other codeword of $\mathcal{RM}(1,m) \setminus \mathcal{F}$.*

Due to the specific structure of the words in the set $\mathcal{F}$, i.e., the possible outputs of the ML decoder for the considered input, we are now able to make a statement on the behavior of the oracle when a single bit of this input is flipped (Lemma 3).

Lemma 3. *Denote by $\hat{p}, \check{p} \in \mathbb{F}_2[x]$ two polynomials with $\deg(\hat{p}) = \deg(\check{p}) = 1$ and $\check{p} \notin \{\hat{p}, \hat{p}+1\}$. Then, for $r = \mathsf{FindWordAtBorder}(\mathfrak{D}_0^e, \operatorname{supp}(\hat{p}), \operatorname{supp}(\check{p}))$ as in Algorithm 13 and any $\xi \in \operatorname{supp}(\hat{p}\check{p})$ it holds that*

$$\mathfrak{D}_0^e(r + u^{(\xi)}) = \begin{cases} \mathsf{True}, & \text{if } r_\xi + e_\xi = 1 \\ \mathsf{False}, & \text{else,} \end{cases}$$

where $u^{(\xi)} \in \mathbb{F}_2^{2^m}$ denotes the (polynomial corresponding to the) ξ-th unit vector.

This is exactly the behavior required for a successful attack strategy as the decoder now is at the decoding boundary for all positions of $\operatorname{supp}(\hat{p}\check{p})$. Now, if $r_\xi + e_\xi = 1$, the error is reduced by one, and the decoder returns the all-zero codeword, which is indicated by the oracle $\mathfrak{D}_0^e$. This again allows to successively test all positions in $\boldsymbol{r}$ restricted by $\operatorname{supp}(\hat{p}\check{p})$. This restriction to the support of the intersection between $\hat{p}$ and $\check{p}$ is the reason for Algorithm 14. With this algorithm, multiple inputs $\boldsymbol{r}$ are found through the use of Algorithm 13 initialized with different polynomials such that the resulting supports span the whole length of a RM codeword and thus allow for a test of all positions. Using Lemma 3 it can be derived that $\mathrm{HW}(\hat{p}\check{p}) = \frac{d}{2}$ and for a first-order RM code it holds that $n = 2d$ (c.f. Section 4.2.5). Therefore, four inputs at the decoding boundary have to be found by Algorithm 13 in order to be able to test the complete length of a codeword. Due to the structure of first-order RM codewords it is sufficient to use all combinations of $\{\hat{p}, \hat{p}+1\}$ and $\{\check{p}, \check{p}+1\}$ as shown in Example 4.6.2.

Example 4.6.2: Intersection of RM Polynomials used in Algorithm 14

In this example, it is shown why it is sufficient to use all combinations of $\{\hat{p}, \hat{p}+1\}$ and $\{\check{p}, \check{p}+1\}$ in lines 2-3 of Algorithm 14. Using a $\mathcal{RM}(1,4)$ code, we choose the polynomials $\hat{p} = \boldsymbol{p}_4$ and $\check{p} = \boldsymbol{p}_3$ as given by Example 4.2.3, which results in

$$\begin{array}{cc} \hat{p} & (0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1) \\ \check{p} & (0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1) \end{array}.$$

The binary inversion of these polynomials is given by an addition with the all-ones vector

$$\begin{array}{cc} \hat{p}+1 & (1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0) \\ \check{p}+1 & (1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0) \end{array}.$$

Finally, through multiplication, the different combinations are given by

$$\begin{array}{ll} \hat{p}\check{p} & (0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1) \\ \hat{p}(\check{p}+1) & (0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0) \\ (\hat{p}+1)\check{p} & (0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0) \\ (\hat{p}+1)(\check{p}+1) & (1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0) \end{array}.$$

Note that the union of all supports $\mathrm{supp}(\hat{p}\check{p}) \cup \mathrm{supp}(\hat{p}(\check{p}+1)) \cup \mathrm{supp}((\hat{p}+1)\check{p}) \cup \mathrm{supp}((\hat{p}+1)(\check{p}+1))$ includes all positions in a codeword.

Finally, in Theorem 1 it is shown that Algorithm 14 is always successful in recovering the correct vector $\boldsymbol{e}$, given $\mathrm{HW}(\boldsymbol{e}) < \frac{d}{4}$. Note that the theorem includes the multiplicity s of the used RM code (c.f. Section 4.2.5) and therefore $d = s2^{n_2-1}$. Accordingly, the input of the algorithm consists of the duplicated versions of the polynomials p_1 and p_2, which results in $\mathcal{I}_1 = \mathrm{supp}^{\times s}(p_1)$ and $\mathcal{I}_2 = \mathrm{supp}^{\times s}(p_2)$.

Theorem 1. *Let $\mathfrak{O}_0^e$ be an oracle for the code $\mathcal{RM}^{\times s}(1,m) \subset \mathbb{F}_2^{s2^m}$ of minimum distance $d = s2^{m-1}$, where $\boldsymbol{e} \in \mathbb{F}_2^{s2^m}$ with $\mathrm{HW}(\boldsymbol{e}) < \frac{d}{4}$. Consider two polynomials $p_1, p_2 \in \mathbb{F}_2[x]$ with $\deg(p_1) = \deg(p_2) = 1$ and $p_2 \notin \{p_1, p_1+1\}$. Then, the output of Algorithm 14 is* $\mathsf{FindError}(\mathfrak{O}_0^e, \mathrm{supp}^{\times s}(p_1), \mathrm{supp}^{\times s}(p_2)) = \boldsymbol{e}$.

Required Oracle Calls The strategy, as described in this section, requires at most $4 \cdot (\frac{2 \cdot n_2}{4} + \frac{n_2}{4})$ oracle calls dependent on the length of the RM code. Note that the algorithm has to be repeated for all n_1 blocks of $\boldsymbol{y}^{(0)}$ introducing an additional factor of n_1. We compare the required oracle calls with the timing attack by Guo et al. [GHJ+22] in Table 4.9. In addition to some disadvantages of the exploited timing side-channel (see Section 4.6.3 for a detailed discussion), this approach shows a largely increased number of required oracle calls. In essence, their attack works by randomly increasing the Hamming weight of an input to the RM decoder until they reach the decoding boundary. Then, the oracle can be queried with the individual bits of the input flipped. From the now found error positions, only those that are not self-introduced in the first step are counted as a valid part of the support of $\boldsymbol{y}_i^{(0)}$. Therefore, the attack steps have to be repeated until each position is evaluated and optionally a certain threshold for each position is reached. This makes the attack non-deterministic, and therefore, the authors report the amount of required ideal timing oracle calls as the median of 6096 attacks. For HQC-RMRS-128 our attack strategy reduces the required oracle queries by a factor of 16.34. We additionally observed that the attack strategy shown by Guo et al. [GHJ+22] is also useable with our close-to-zero oracle. We implemented their strategy targeting a single $\boldsymbol{y}_i^{(0)}$ block and simulated the required oracle calls[14] for 400 000 attacks with $y_{w,max}$ in the range $1 \leq y_{w,max} \leq 10$ given

[14]The simulation integrates the C reference implementation of the RM decoder and directly uses the decoding result to build the required oracle.

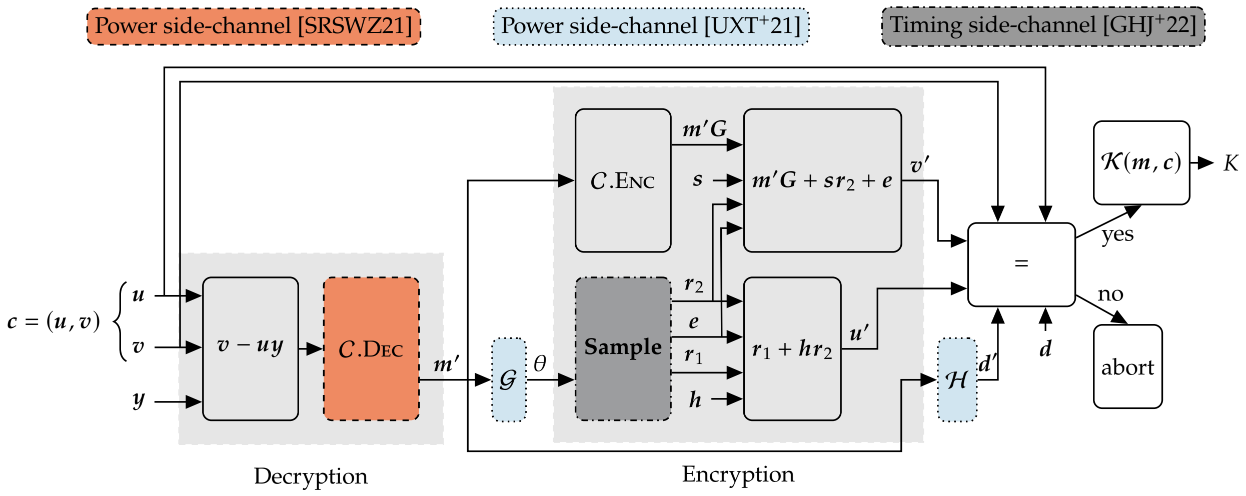


Figure 4.21 Building blocks of the HQC-RMRS decapsulation (c.f. Algorithm 11) with the side-channel attack targets used in related work.

a threshold of one (each position has to be evaluated once). The resulting median of the required oracle calls is shown in the third column of Table 4.9. These numbers are reported to ensure a fair comparison that is not influenced by the different types of oracles. To summarize, the presented attack strategy in comparison to [GHJ+22] requires by a factor of 11.44 (HQC-128) and 12.07 (HQC-192/256) less oracle queries when using the proposed close-to-zero oracle. In addition, it is proven to be successful for $\mathrm{HW}(y_i^{(0)}) < \frac{d_{\mathcal{RM}}}{4}$, where $d_{\mathcal{RM}} = s \cdot 2^{m-1}$.

	This work	Timing Attack [GHJ+22]	Strategy of [GHJ+22] using $\mathfrak{O}_0^e$
HQC-128	1152*46	18 829*46[a]	13 174*46
HQC-192	1920*56	-[b]	23 170*56
HQC-256[c]	1920*90	-[b]	23 170*90

[a] The authors report a median of 866 143 oracle calls to retrieve the whole $y^{(0)}$. To provide comparable numbers, we report the required calls per block.
[b] Numbers not shown in [GHJ+22].
[c] The same simulation results as for HQC-RMRS-192 are shown since both use the same RM code.

Table 4.9 Comparison of required oracle queries for the different attack strategies.

4.6.3 Side-Channel Targets to Build the Required Oracle

There are several possible side-channels that can be used to construct the close-to-zero oracle $\mathfrak{O}_0^e(\boldsymbol{r})$ as given in Definition 1. In this section, we describe the results of directly attacking the implemented RS decoder of the HQC-RMRS reference implementation using the power side-channel described in Section 4.5.2. In addition, the approaches in related work are discussed, and it is shown how or whether these side-channels can be adapted to build the required oracle for the presented attack strategy. An overview of the different side-channel targets of the HQC decapsulation is shown in Fig. 4.21.

Power Side-Channel of the RS Decoder It is possible to construct our required oracle from the decoding result of the RS decoder. First, we have to recall that our oracle indicates whether the RM decoder is able to correctly decode to the all-zero codeword or the decoding fails, and any other codeword is returned. Transferring this behavior to be observable through the RS decoder requires setting all the remaining $y_i^{(0)}$ that are not attacked to zero. Then, if the RS decoder has to correct an error, we know that the RM decoder was not able to return the correct all-zero codeword, and

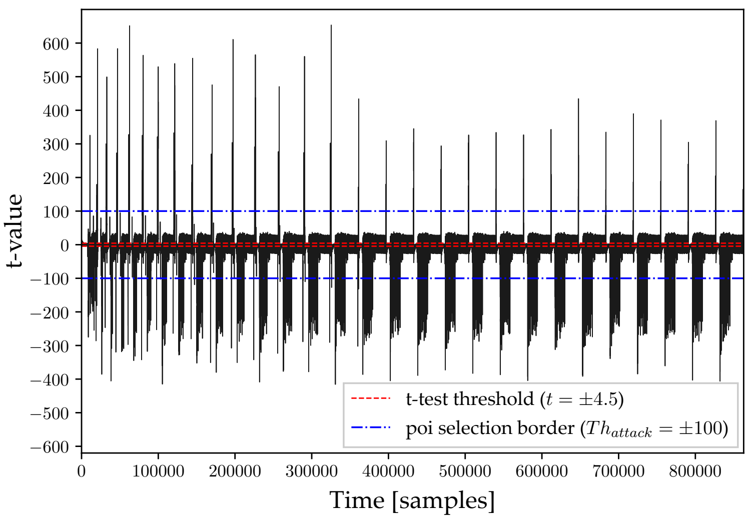


Figure 4.22 Resulting t-values for the error-locator polynomial computation (`compute_elp()`) of the HQC-RMRS-128 reference implementation. The marked Th_{attack} is used as a threshold for the actual attack.

the oracle result is False. In order to observe that the RS decoder has to correct an error, we use the template matching approach shown in Section 4.5.2. Note that although this method targets a BCH decoder, we can still use the approach in our setting. This is due to the fact that BCH codes are subcodes of RS codes, and their decoder is similar as discussed in Section 4.2.4. The attack target for building the templates is the computation of the error-locator polynomial, as it showed the highest amount of exploitable leakage. As the input can be directly controlled by an attacker, templates for both classes can be constructed through the power side-channel. Then, for each oracle call, the constructed input is fed into the decapsulation and the respective power trace is compared to the templates through the use of a sum-of-squared-difference metric. As a result, the class with the smallest metric is chosen as the oracle result.

We evaluated the oracle with a power side-channel setup consisting of a STM32F415RGT6 ARM Cortex-M4 microcontroller mounted on a Chipwhisperer CW308 UFO board running at 10 MHz. The power consumption is measured through the integrated shunt resistor with a PicoScope 6402D USB-oscilloscope running at a sampling rate of 156.25 MHz. As an attack target, we use the HQC-RMRS-128 reference implementation (c.f. Section 4.3.2 for a description of the used submission package). With this setup, a total amount of 1000 template traces is used for the initialization of the oracle using a t-test threshold of 100 for POI detection. The t-test results are shown in Fig. 4.22, where the shown samples correspond to the complete execution time of the error-locator polynomial computation. Using the initialized oracle, we are able to correctly classify 100 000 attack traces. As this number is above the required number of oracle calls for a complete key recovery, the attack on the complete secret key is considered successful.

Power Side-Channel of the used Hash Functions In [UXT+21], the authors show how to create a plaintext-checking oracle for HQC-RMRS by observing a power side-channel of the used extendable-output function SHAKE256. With their oracle, they are able to distinguish if a certain input results in a fixed message m' or if the result is different from this fixed message. As m' is directly used as an input to $\mathcal{G}$ and $\mathcal{H}$, the authors identify these hash functions as an attack target. In order to instantiate their oracle, they use a machine learning classifier based on a CNN. They evaluate their CNN on the SHAKE software implementation of `pqm4` [KRSS] with the same side-channel platform and microcontroller as used in the evaluation of the RS decoder in this thesis. Using 30 000 training traces, they achieve an accuracy of 0.998 when classifying 10 000 test traces, which can be further increased through the combination of multiple classifications.

This oracle can not be directly used with the proposed attack strategy, as the resulting message after decryption is always zero. It can nevertheless be adapted to work as our close-to-zero oracle (Definition 1) by setting the resulting input to $\mathcal{C}.\textsc{Dec}$ such that the input to the RS decoder is set to its decoding boundary. This can be done by setting $(d_{\mathcal{RS}} - 1)/2$ blocks of $\boldsymbol{y}^{(0)}$ that are not currently attacked to be decoded as a non-zero value, thus acting as an error for the all-zero RS codeword. Then the RM decoding result of the attacked $\boldsymbol{y}_i^{(0)}$ determines if the resulting message is zero (True) or unequal to zero (False), which is observable through their oracle.

Timing Side-Channel of the used Sampler Guo et al. [GHJ+22] showed a timing side-channel in the implementation of the sampler of the HQC-RMRS reference implementation that is used to generate the random fixed-weight vectors $\boldsymbol{e}$, $\boldsymbol{r}_1$, and $\boldsymbol{r}_2$. This is the case as the sampler implements rejection sampling, which requires varying calls to the PRNG to generate potentially required additional randomness. As the seed θ for the PRNG is derived from the message $\boldsymbol{m}$, the amount of additional required PRNG calls is dependent on $\boldsymbol{m}$ and therefore also the execution time of the decapsulation. This timing side-channel allows to build a plaintext-checking oracle for which the authors show an attack strategy. In order for two different messages to be distinguishable through their oracle, the initial message is chosen such that it requires at least three additional calls to the PRNG, which has a low probability of 0.58 % for all possible messages. Due to the inherent uncertainty of timing side-channels and this probability still leaving room for ambiguity, the authors introduce a majority threshold for the classification of each bit. Their empirical results show a classification success rate of 87 % with a majority threshold of five.

This oracle can not be used with the presented attack strategy as its resulting message $\boldsymbol{m}$ is always zero. Unfortunately, this message does not require multiple calls to the PRNG, and therefore, it is not easily distinguishable through this timing side-channel. In contrast, the developed attack strategy allows the usage of both described power side-channels, which show a better classification result and eliminate the inherent uncertainty of a timing side-channel, removing the need for a majority decision.

4.7 Countermeasures

In this section, countermeasures against the developed attacks in this thesis are discussed. First, a general countermeasure against chosen-ciphertext attacks through the detection of attack-specific input patterns is discussed in Section 4.7.1. This is followed by an evaluation of an established countermeasure by Merli et al. [MSS13] in Section 4.7.2 that is used to secure a decoder of an error-correcting code in the context of Physical Unclonable Functions (PUFs). In Section 4.7.3, a countermeasure, which is based on the insertion of additional errors before the decoding of the outer code, is presented. This countermeasure is able to significantly reduce the side-channel leakage of the decoder but comes with the downside of required modifications of the used code. Finally, Section 4.7.4 concludes the section with a discussion and an outlook on future work required to develop a secure implementation of HQC.

4.7.1 Detection of Attack Patterns

The attacks described in Sections 4.5 and 4.6 set the ciphertext for the decapsulation of HQC, such that the secret key acts as the error added to a valid codeword. This is achieved by a ciphertext input of $\boldsymbol{u} = (1, 0 \ldots, 0) \in \mathbb{F}_2^n$ and successively querying the algorithm with varying $\boldsymbol{c}$, where only the attacked codeword of the inner code is set to a non-zero value (c.f. Section 4.4.2). Therefore, in order to counteract these attacks, it is possible to detect the specific pattern at the beginning of the decapsulation (Algorithm 11). If an active attack is detected, the decapsulation can be aborted before any HQC operations are performed, resulting in no exploitable side-channel leakage.

The input $\boldsymbol{u} = (1, 0 \ldots, 0)$ can be detected through its characteristic HW of one. Additionally, this also allows to detect all cyclic shifts of $\boldsymbol{u}$, which would allow an attack on the shifted version of the secret key. For the presented attack, all patterns for $\boldsymbol{v}$ can be detected, as only one codeword of the inner code is set to a non-zero value. Nevertheless, in practice, this detection is aggravated as the attacker only has to make sure that the decoding result of all inner codewords, except the currently targeted one, is the all-zero codeword. This allows an attacker to randomly choose the support of non-targeted codewords of the inner code such that their decoding still results in the all-zero codeword, i.e., there are many inputs to the decoder that show the desired decoding results. Therefore, the detection of invalid inputs has to be carefully crafted in order to not falsely discard valid ciphertexts. This shows a strong downside of such a detection approach, as there is the possibility that an attack is adapted to queries that cannot be detected from valid ciphertext without performing the actual decryption. An example of such an attack has been published by Hermelink et al. [HPP21] for the lattice-based KEM Kyber. With this attack, the authors are able to defeat their statistical countermeasure [PPHH] that uses a test if the ciphertext follows a binomial distribution. Only ciphertexts that show a positive result for this test get decrypted by the algorithm, thwarting published chosen-chipertext attacks. The attack in [HPP21] defeats such a statistical test as it only has to manipulate a single bit of a valid ciphertext, which can not be identified anymore.

Due to the mentioned downside, dedicated side-channel countermeasures for the outer code of HQC are explored in the following sections.

4.7.2 Codeword Masking of the Outer Decoder

Merli et al. [MSS13] proposed a masking scheme in order to secure an error-correcting code against side-channel attacks, which they call *codeword masking*. With their scheme, they exploit a property of linear codes, which states that the sum of two codewords is again a codeword of the same code C. This allows to randomly sample a second information vector $\boldsymbol{u}_m$ that is encoded to the codeword $\boldsymbol{c}_m = C.\text{Enc}(\boldsymbol{u}_m)$. An input $\boldsymbol{r}$ to a decoder can then be masked through an addition with $\boldsymbol{c}_m$ before the decoding as

$$\boldsymbol{u}' = C.\text{Dec}(\boldsymbol{r} + \boldsymbol{c}_m) .$$

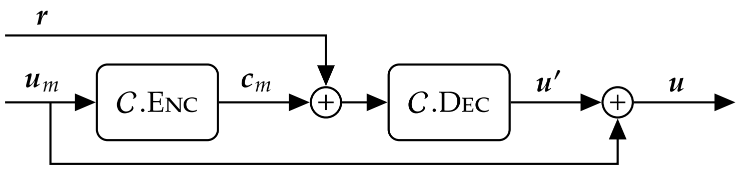


Figure 4.23 Visualization of codeword masking for error-correcting codes as proposed by Merli et al. [MSS13].

The resulting information vector after decoding $\boldsymbol{u}'$ then consists of the decoding result of the combined codewords as $\boldsymbol{u}' = \boldsymbol{u} + \boldsymbol{u}_m$. As the mask $\boldsymbol{u}_m$ is known, the final unmasked decoding result $\boldsymbol{u}$ can be retrieved through subtraction of the mask according to

$$\boldsymbol{u} = \boldsymbol{u}' - \boldsymbol{u}_m \; .$$

The concept of codeword masking is visualized in Fig. 4.23. With the choice of the mask as a valid codeword, the error correction capabilities of the code are unaltered, making it, in principle, suitable for the use in HQC as the amount of correctable errors has to be unaltered in order to achieve the desired security level and decryption failure rate. A masked variant of the decoder for the outer code in HQC-Decrypt (Algorithm 9) is shown in Algorithm 15.

Algorithm 15 Codeword masking of the outer code during HQC-Decrypt (Algorithm 9)

```
Input: $\tilde{v} = C_2.\mathrm{Dec}(v')$                ▷ Decoding result of the inner code $C_2$
1: $\boldsymbol{m}_m \xleftarrow{\$} \mathbb{F}_2^k$
2: $\boldsymbol{c}_m = C_1.\mathrm{Enc}(\boldsymbol{m}_m)$
3: $\boldsymbol{m}' = C_1.\mathrm{Dec}(\tilde{v} + \boldsymbol{c}_m)$        ▷ This results in $\boldsymbol{m}' = \boldsymbol{m} + \boldsymbol{m}_m$
4: $\boldsymbol{m} = \boldsymbol{m}' - \boldsymbol{m}_m$
Output: $\boldsymbol{m}$
```

Codeword masking has been successfully utilized by Merli et al. [MSS13] in order to counteract a CPA on the used error correction of a PUF. Their attack targets the input register of a hardware implementation of a BCH code by using the HD of the updated register content as its power model. With a successful attack on this input register, an attacker can infer the PUF response, which breaks the security of the system as a used key is derived from this response. It has to be noted that the attack targets a single point in time, i.e., the register update, and since the attack is differential, it requires a model of the power consumption. With codeword masking, the content of the input register is masked such that a first-order CPA is not successful anymore.

For the chosen-ciphertext attacks shown in this thesis, the attacker builds a side-channel oracle that only needs to distinguish between two classes as detailed in the respective section of both attacks on the different HQC versions (Sections 4.5.2 and 4.6.3). Therefore, the complete side-channel information of the execution of the attacked decoder can be used to build the required oracle. This leads to codeword masking not being effective for a syndrome-based decoder, as only the syndrome calculation is masked through the additional codeword $\boldsymbol{c}_m$, while the resulting syndromes only depend on the error as shown in Eq. (4.10), which is not affected by the masking scheme. An attacker can build the oracle from all parts of the decoder after the syndrome computation, even if codeword masking is used. In practice, since the highest amount of exploitable side-channel information is obtained from the computation of the error locator polynomial (c.f. Fig. 4.18), this does not make an attack more difficult.

4.7.3 Inserting Additional Errors

As established masking methods for error-correcting codes are not applicable, as discussed in Section 4.7.2, a different countermeasure approach is discussed in the following. The idea of this countermeasure is to insert an additional error vector $\boldsymbol{e}_m \in \mathbb{F}_q^{n_1}$ before the decoding of the outer code C_1 during HQC-Decrypt, forcing the decoder to correct an error for all inputs. This breaks the assumption required by the side-channel oracles discussed in Sections 4.5.2 and 4.6.3. Two different methods for insertion of additional errors are presented in this section. First, a fixed amount of errors is added to the input, while in a second attempt the amount of errors is chosen randomly from an interval. Unfortunately, a downside of this approach is that it changes the desired error-correcting capabilities of the code. This has to be considered as the parameters, and thus, the used codes are chosen by the HQC authors such that a desired security level with its required decryption failure rate is achieved. Therefore, this section concludes with a discussion on how to change the used code in order to account for the additional errors.

Fixed Number of Errors In this case, the error vector $\boldsymbol{e}_m \in \mathbb{F}_q^{n_1}$ is chosen with a fixed weight $\mathrm{HW}(\boldsymbol{e}_m) = \phi$ but the position of the errors is sampled randomly. Note that for $q \neq 2$, which is the case for the RS code of HQC-RMRS, the hamming weight is defined as the number of nonzero symbols. The resulting algorithm for the decoding of the outer code C_1 is shown in Algorithm 16.

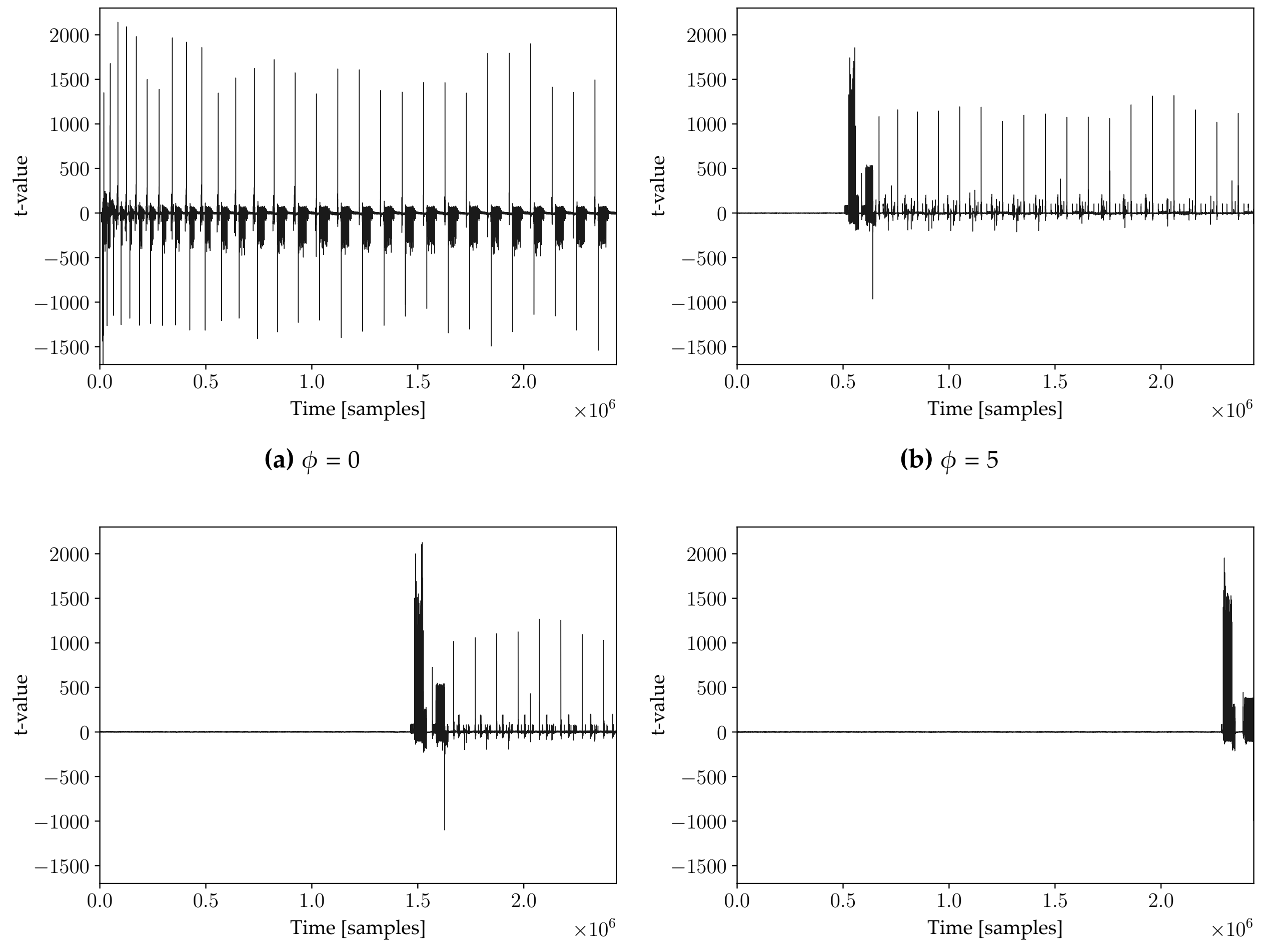
Algorithm 16 Adding a fixed Number of Errors to the outer code of HQC-Decrypt (Algorithm 9)

```
Input: Number of errors $\phi$, $\tilde{v} = C_2.\textsc{Dec}(v')$
1: $\boldsymbol{e}_m \xleftarrow{\$(w=\phi)} \mathbb{F}_q^{n_1}$        ▷ For $q \neq 2$ erroneous symbols are randomly chosen
2: $\boldsymbol{m} = C_1.\textsc{Dec}(\tilde{v} + \boldsymbol{e}_m)$
Output: $\boldsymbol{m}$
```

In order to discuss the effectiveness of this countermeasure, we have to recall the required oracle results for the presented attacks. The attacks require access to an oracle that distinguishes exactly two classes, namely $C_1.\textsc{Dec}$ has to correct an error or the input to the decoder is the all-zero codeword and the decoder does not have to correct an error[15]. With the additional error vector $\boldsymbol{e}_m$, the two classes are transformed to a correction of $\phi + 1$ errors and the correction of ϕ errors, respectively. The results of a t-test using 2000 traces of the error locator polynomial computation of the RS decoder of HQC-RMRS-128 for different values of ϕ are shown in Fig. 4.24. If the countermeasure is deactivated ($\phi = 0$), the whole computation shows exploitable leakage. In contrast, for an activated countermeasure, the part of the computation that shows exploitable leakage is narrowed down with an increasing amount of inserted errors. In this case, the first peak that occurs in the t-test corresponds to the point of the computation where both classes differ in the amount of errors that have to be corrected. Although this decreases the amount of exploitable POIs, the resulting t-values and, therefore, the leakage is not reduced significantly. The reason for this characteristic lies in the used constant time version of Berlekamp-Massey algorithm [WBB⁺22, Appendix A] that is used in the reference implementation of HQC-RMRS. This algorithm iteratively computes the error locator polynomial for which the implemented constant time version requires $2\delta - 1$ iterations. Nevertheless, the algorithm already successfully retrieves the error locator polynomial at an earlier iteration corresponding to two times the amount of errors in $\tilde{v} + \boldsymbol{e}_m$ independently of the position of the errors. To achieve the required constant time execution, dummy operations are performed for the remaining iterations. Therefore, until the error locator polynomial is retrieved, actual updates are computed and both classes ($\phi + 1$ or ϕ errors) are hard to distinguish, which is also shown in Fig. 4.24. For the next iteration, an update has to be computed in the case of $\phi + 1$ errors, while for ϕ errors dummy operations are performed. This leads to distinguishable leakage, which is shown by the t-test as the first characteristic peak. For the remainder of the computation only dummy operations are performed for both classes. As these depend on the number of errors, they lead to different leakage for both classes shown in Fig. 4.24 with the additional peaks for the remaining iterations of the algorithm. It

[15]The oracle for the attack on HQC-RMRS as defined in Definition 1 requires the oracle output of the RM code (C_2). As discussed in Section 4.6.3, for the actual instantiation of this oracle, the decoding result of the RS code (C_1) can be used.

can therefore be concluded that the additional insertion of a fixed number of errors only shifts the *distinction boundary*, i.e., the distinction between zero or one error is shifted to ϕ or $\phi + 1$ errors, but it is not suitable as a countermeasure for the presented attacks.



(a) $\phi = 0$ **(b)** $\phi = 5$

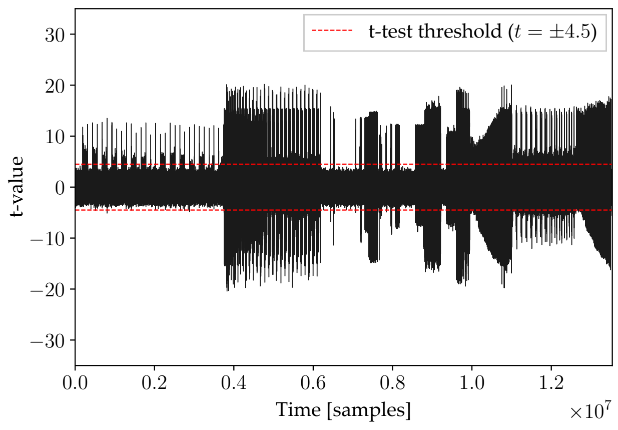
(c) $\phi = 10$ **(d)** $\phi = 14$

Figure 4.24 T-test results of the error locator polynomial computation of the RS decoder of HQC-RMRS-128 for different values of ϕ with a total of 2000 traces.

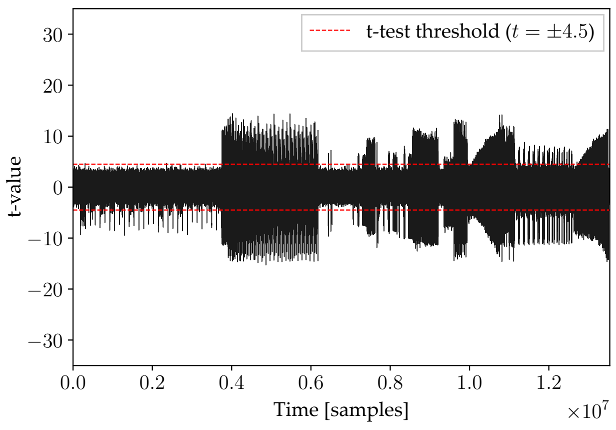
Random Error Insertion With this countermeasure the amount of inserted errors ϕ is chosen randomly from the set $\{0, \ldots, \phi_{\max}\}$, resulting in Algorithm 17. Therefore, the outer decoder has to correct a varying amount of errors for each execution of HQC-Decrypt. The t-test results for the RS decoder of HQC-RMRS-128 for multiple $\phi_{\max}$ are shown in Fig. 4.25. The results show that even for $\phi_{\max} = 1$ (c.f. Fig. 4.25a), the leakage is reduced significantly in contrast to an unsecured decoder. Through an increase of $\phi_{\max}$, this leakage can be reduced further, with $\phi_{\max} = 10$ exceeding the threshold of 4.5 only marginally.

Algorithm 17 Random Error Insertion for the outer code of HQC-Decrypt (Algorithm 9)

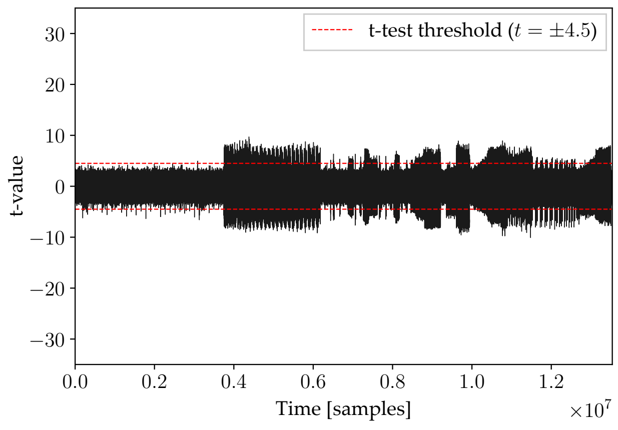
```
Input: Maximum amount of errors $\phi_{\max}$, $\tilde{v} = C_2.\textsc{Dec}(v')$
1: $\phi \xleftarrow{\$} \{0, \ldots, \phi_{\max}\}$
2: $e_m \xleftarrow{\$(w=\phi)} \mathbb{F}_q^{n_1}$        ▷ Erroneous symbols are randomly chosen
3: $m = C_1.\textsc{Dec}(\tilde{v} + e_m)$
Output: $m$
```



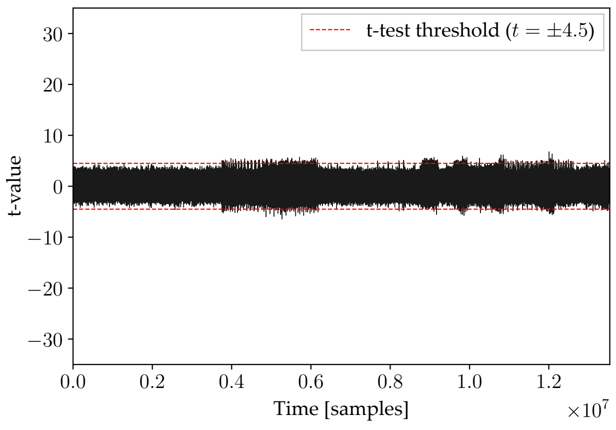
(a) $\phi_{\max} = 1$



(b) $\phi_{\max} = 2$



(c) $\phi_{\max} = 5$



(d) $\phi_{\max} = 10$

Figure 4.25 T-test results for the RS decoder of HQC-RMRS-128 secured with the random error insertion countermeasure. All results are shown for a total amount of 500 traces.

Required Changes to C The additional inserted errors have to be compensated by an increased error correction capability of the used code C in order to achieve the desired decryption failure rate. As C is built through a code concatenation, the length of either C_1 or C_2 can be increased. If C_2 is enlarged, the outer code has the correct fewer errors, and therefore, this margin can be used for the countermeasure. In the case of HQC-RMRS, the ML decoding of the used RM code implies that its decoding probability can not be stated exactly, which makes it hard to determine the exact margin for which additional errors can be inserted. Therefore, we only discuss an enlargement of the outer code C_1 in this thesis. This implies increasing the error correction capability such that the additional inserted errors can be corrected, which results in

$$\delta' = \delta + \phi_{\max} \, ,$$

where δ is the error correction capability of the unsecured code.

In the following, an adaption of the RS code is discussed, as HQC-RMRS is the current version of the HQC cryptosystem. The length of the unshortened RS code is fixed to $n_1 = 2^8 - 1$ since the code concatenation is designed for symbols of C_1 with $q = 2^8$. Therefore, in order to achieve δ', the number of symbols k has to decrease as given by

$$k = n_1 - 2\delta' = n_1 - 2(\delta + \phi_{\max}) \, .$$

In consequence, the length of the shortened code has to increase, as the corresponding k of the shortened code is fixed and has to reflect the length of the shared secret of the KEM, i.e., $k = 16$

	RS code	shortened RS code				Overhead	
	$[n, k, \delta']$	$[n_1, k, \delta']$	n_2	$n_1 n_2$	n	$\vert\mathsf{pk}\vert$	$\vert c\vert$
HQC-RMRS-128	[255, 205, 15+10=25]	[66, 16, 25]	384	25 344	25 349	42.8 %	42.7 %
HQC-RMRS-192	[255, 203, 16+10=26]	[76, 24, 26]	640	48 640	48 661	35.4 %	35.5 %
HQC-RMRS-256	[255, 177, 29+10=39]	[110, 32, 39]	640	70 400	70 507	22.2 %	22.2 %

Table 4.10 Resulting parameter sets of HQC-RMRS for $\phi_{\max} = 10$. For completeness, it has to be noted that the correct parameter for $\phi_{\max}$ should be evaluated for each parameter set individually and the shown parameter sizes are provided as an estimate of the resulting overhead of the countermeasure.

for HQC-RMRS-128. The resulting parameters of HQC-RMRS for $\phi_{\max} = 10$ are given in Table 4.10, where n is chosen to be the next primitive prime larger than $n_1 n_2$. With the countermeasures, the increase in the length n results in an overhead of the public key size[16] of 42.8 %, 35.4 %, and 22.2 %, while the ciphertext size shows an increase of 42.7 %, 35.5 %, and 22.2 %.

Although the increase in the code length is able to compensate for the additional errors, the security of the system has to be reevaluated for the now larger instances of the underlying QCSD problems. This includes the choice of the parameters w, w_r, and w_e such that Eq. (4.24) is satistied. In order to construct a secure cryptosystem, these evaluations have to be carefully performed by experts in the field of coding theory and are considered out of scope for this thesis.

4.7.4 Conclusion

In this section, several countermeasures against the chosen-ciphertext attacks developed in this thesis were discussed. With the detection of characteristic attack patterns, the proposed attacks can be prevented with low additional resources. Nevertheless, an adaption of the attacks to patterns that can not be distinguished from valid ciphertexts without decryption again defeats this type of countermeasure. Such an adaption has already been shown for an attack on a lattice-based system.

The established codeword masking countermeasure is not applicable due to the syndrome-based decoder of the HQC reference implementation and the fact that side-channel information corresponding to the complete decapsulation, starting from the execution of the decoder, can be used to build the required oracle. The option to secure other decoder types of an RS code with codeword masking was additionally evaluated in [Kai22] under the supervision of the author of this thesis. Here, the assumption was that for a decoder that is not syndrome-based, the countermeasure is able to mask the complete execution of the decoder. First, an implementation of a Welch-Berlekamp decoder was developed and analyzed with the result that even though codeword masking is employed, this decoder still shows exploitable leakage. Further experiments show that the primary source of leakage is the constant time pivoting strategy during the required Gaussian elimination step of the decoder. In a second step, a list decoder for the RS code was developed. The results show that codeword masking is also ineffective for this decoder type, as the required degree function depends on the number of erroneous symbols, which exhibits characteristic leakage. Additionally, in order to achieve a constant-time implementation, the decoder has to perform dummy operations after all errors have been corrected, which is identified as another source of exploitable leakage. It can, therefore, be concluded that codeword masking is not applicable as a countermeasure for the proposed attacks.

The random insertion of additional errors before the decoding of the outer code C_1 during HQC-Decrypt has shown to be effective as it forces the decoder to correct an error for all inputs to the algorithm. Nevertheless, the effectiveness of this countermeasure depends on the maximum number of additional inserted errors $\phi_{\max}$, which is limited for a particular adaptation of the used code. With this adaptation, the used code has to be enlarged in order to be able to correct the additional inserted

[16]Note that the secret key of HQC is not increased in size by this parameter change as it is only stored as the seed used for its generation.

errors. This ultimately increases the parameter sizes and additionally implies that the security of the system has to be reevaluated as the instances of the underlying QCSD problems change. As changes to the system with their required reevaluation of the security have to be performed carefully by experts in the field, a countermeasure with such a requirement is usually not desired. Nevertheless, the current status of the NIST competition would allow the authors of HQC to design a version of the system with a specific error margin for this countermeasure.

For future work, a complete masking scheme of the whole HQC algorithm has to be developed in order to provide a secure implementation against the attacks presented. Such a scheme is usually designed in a certain model, e.g., the d-probing model [ISW03], which splits the sensitive variable into $d + 1$ shares providing security against a d-th order attack. In order to prove the security of the masking scheme, the algorithm is split into small parts, called gadgets, for which the security is proven individually. Multiple gadgets can then be securely composed in order to construct the masked implementation of the whole algorithm. Work in the direction of provable masked implementations usually targets block ciphers like AES [DR20] or sponge constructions like Keccak [Dwo15], which consists of boolean operations and multiplications for which efficient and secure gadgets are known. Nevertheless, the used codes of HQC utilize operations that require the development of new kinds of gadgets for which, to the best of this author's knowledge, no prior work exists. A first work in this direction has been shown by Demange and Rossi [DR24], who have developed multiple novel gadgets in order to provide a masked implementation of the Quasi-Cyclic Moderate Density Parity-Check (QC-MDPC) code used in the code-based cryptosystem BIKE [ABB+20a].

Part II

Explainability for Deep Learning-based Side-Channel Analysis (DL-SCA)

5 Preliminaries

This chapter provides the preliminaries for the work on DL-based SCA presented in this thesis. It is based on parts of the following publications:

- *Egger/Schamberger/Tebelmann/Lippert/Sigl: A Second Look at the ASCAD Databases* published in *Constructive Side-Channel Analysis and Secure Design (COSADE)*, 2022 [EST+22]

- *Schamberger/Egger/Tebelmann: Hide and Seek: Using Occlusion Techniques for Side-Channel Leakage Attribution in CNNs* published in *Applied Cryptography and Network Security Workshop (ACNS)*, 2023 [SET23].

In particular, the presentation of the evaluation metrics in Section 5.1.4 is based on [SET23] and the introduction to the ASCAD databases in Section 5.2 is based on [EST+22].

5.1 Deep Learning-based Side-Channel Analysis (DL-SCA)

Machine learning techniques can be used to perform profiled side-channel attacks (c.f. Section 2.3). In particular, this thesis discusses deep learning-based methods for side-channel analysis, where the leakage of the device is modeled with a Deep Neural Network (DNN). A simplified distinction between deep learning methods and other machine learning techniques lies in the required feature engineering effort: deep learning methods can be directly applied to raw data with a light preprocessing like, e.g., normalization, while other machine learning methods require human-engineered features in order to produce satisfying results. The profiling stage is given by training the DNN, whereas the trained network is used to classify the unknown attack traces in the attack phase. This section gives an introduction to DL-based SCA.

In Section 5.1.1, the notation is introduced and the problem of performing a profiled side-channel attack is reformulated using this notation. The elemental layers of deep neural networks and the used types of network architectures are presented in Section 5.1.2. The training process of a neural network is detailed in Section 5.1.3 and Section 5.1.4 introduces the used evaluation metrics of the attack results. Finally, in Section 5.1.5, possible advantages and disadvantages of DL-based SCA are discussed.

5.1.1 Introduction and Notation

We define the set of side-channel traces as $\mathcal{T}$ and the number of available traces as $N_t = |\mathcal{T}|$, with each trace consisting of N_s samples. The i-th trace of a set of traces is defined as $\boldsymbol{t}_i$ and individual samples of this trace are defined as $\boldsymbol{t}_i[j]$ with j being the sample index. When referring to a range of samples we use $\boldsymbol{t}_i[j_{start}, j_{end}]$ with $\boldsymbol{t}_i[0, 99]$ indicating samples 0 to 99 of the i-th trace.

In the setting of deep learning-based side-channel attacks, the available traces in $\mathcal{T}$ are split into the sets $\mathcal{X}_{train}$, $\mathcal{X}_{val}$, and $\mathcal{X}_{attack}$, where each set is defined as $\mathcal{X} = \{\boldsymbol{X}, \boldsymbol{y}\}$. The feature matrix $\boldsymbol{X} \in \mathbb{R}^{N \times N_s}$ is a matrix of available side-channel traces in the set, and $\boldsymbol{y}$ is a vector of labels associated with the individual traces. A single feature vector $\boldsymbol{x}_i \in \boldsymbol{X}$ contains one side-channel trace $\boldsymbol{t}_i$. As DL-based side-channel attacks are profiled side-channel attacks, the goal is to use all pairs of $(\boldsymbol{x}_i, y_i) \in \mathcal{X}_{train}$ to learn the function $y = f(\boldsymbol{x})$ that provides a mapping of an input vector $\boldsymbol{x}$ to the correct label y. In the

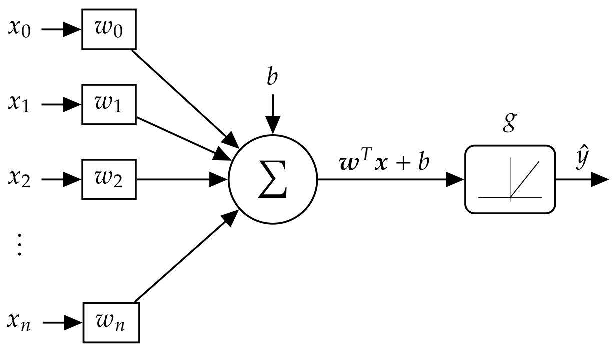


Figure 5.1 Visualization of a perceptron. Adapted from [Mit97, Figure 4.6].

attack phase of the profiled side-channel attack, the learned function can then be used to classify the traces in the attack set $\mathcal{X}_{attack}$ to the correct label.

5.1.2 Elemental Network Layers and Used Architectures

The function f is modeled as a DNN that consists of the composition of multiple smaller functions called layers into a directed acyclic graph[1] [GBC16]. The information on the composition of layers is also called the architecture of a DNN. During the training of a DNN, the parameters of the individual layers are determined such that the whole network achieves a mapping according to f. The details of the training of a DNN are described in Section 5.1.3. In the following, the different layers of the architectures used in this thesis are described, starting with the description of a single perceptron, the connection of multiple layers of parallel perceptrons as an MLP, and finally, the CNN that introduces convolution layers before fully connected layers.

Perceptron The term neural network is inspired by neuroscience, which models the function of a brain as multiple interconnected neurons that perform a computation on multiple inputs and forward the result to its connected neurons. The simplest form of neural network contains a single layer with one neuron and is called *perceptron*, which is visualized in Fig. 5.1. A perceptron takes a feature vector $\boldsymbol{x} = (x_0, \dots, x_n)$ as input and outputs a weighted sum of all features and an additional bias b, where the individual contribution of each feature x_i is defined by its corresponding weight w_i given by a weight vector $\boldsymbol{w} = (w_0, \dots, w_n)$. The weight vector and bias are parameters of the perceptron that are optimized during the training process. As the weighted sum can also be seen as a linear combination of features, the perceptron includes an additional non-linear *activation function* g that provides non-linearity of the output[2]. For modern neural networks the Rectified Linear Unit (ReLU) defined as $g(z) = \max\{0, z\}$ (also shown in Fig. 5.1) is used as the default [GBC16].

MLPs A Multi Layer Perceptron (MLP) is based on multiple fully connected layers of parallel neurons (also referred to as units or nodes), where the output of each neuron in a layer is connected with all neurons in the subsequent layer. As the input features are therefore fed from the *input layer* over multiple *hidden layers* to the final *output layer*, a MLP is a type of a so-called feedforward neural network. The network architecture is called multilayer perceptron since the individual neurons of each layer are implemented as single perceptrons. The individual layers are also called *dense layers*, which are defined by the number of neurons per layer and the used activation function. For the remainder of this thesis, the notation DENSE(n_d) is used to describe a dense layer with n_d neurons. The input layer usually contains the feature vector $\boldsymbol{x}$ and the output layer is based on a dense layer

[1]There are also recurrent neural networks that include feedback connections in the graph, which are not considered in this thesis.

[2]The original perceptron fixes the activation function to a Heaviside step function but the modern usage of a perceptron allows to freely choose a suitable activation function.

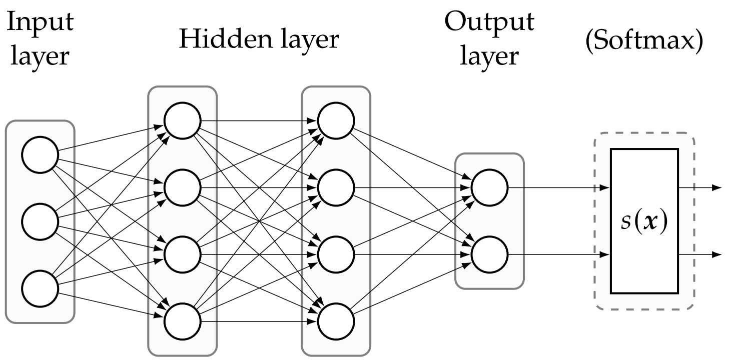


Figure 5.2 Visualization of an MLP with two hidden layers. The architecture of the network can be described as [INPUT(3), DENSE(4), DENSE(4), SM(2)]. Note that the activation function is implicitly included in each perceptron of the hidden layers, while the softmax activation function is shown to emphasize its usage in the output layer.

with N_c neurons, where N_c is the number of possible classes for the label y. The output of each neuron in the output layer is a score corresponding to the network's prediction that the input belongs to the corresponding class (or label), where a higher score indicates a higher certainty. In order to convert the resulting scores to probabilities, the softmax activation function can be used. In the classification setting, it is therefore usually used as the activation function of the dense output layer, which is denoted as SM(N_c) in this thesis. An MLP with two hidden layers is visualized in Fig. 5.2.

CNNs A Convolutional Neural Network (CNN) introduces additional *convolutional layers*, which are defined by a specified amount of filters with their respective kernels that each perform a convolution operation on their input. An activation function is applied to the result of the convolution for each filter in order to produce the output of the layer. The trainable parameters of a convolutional layer are the individual kernel weights of each filter. We denote the convolutional layer as CONV($n_{filter}, n_{kernel}, n_{stride}$), where n_{filter} defines the number of filters, n_{kernel} the size of the kernel, and n_{stride} the stride, i.e., the step size of the kernel during the convolution. The idea behind this layer is that each filter is able to individually learn a more complex representation of its input features (e.g., edges in images), while the complexity of features increases with each subsequent layer. A convolutional layer can be followed by a *pooling layer* that combines multiple features of the individual filters of a convolutional layer into a single one, i.e., it achieves a downsampling of the input to a lower dimension. The pooling layer is characterized by two parameters: the number of features n_{pool}, which defines the width of the pooling window and thus the number of features to be combined, as well as the stride n_{stride} that specifies the step size of the pooling window. There are two types of pooling layers which differ in the combination function, where the *average pooling* layer POOL_avg(n_{pool},n_{stride}) uses the average and the *max pooling* layer POOL_max(n_{pool},n_{stride}) the maximum value of the pooling window. The classification step of a CNN is performed by a set of fully connected dense layers at the output of the network. This requires reshaping the output of the final convolutional layer (or its corresponding pooling layer) so that the output of the individual filters of the layer can be fed into the first dense layer. As this process can be seen as a flattening of the filter dimension, the corresponding layer for this task is called *flatten* layer, which is denoted as FLATTEN in this thesis. A CNN architecture for a one-dimensional input data (such as a time series signal like a side-channel trace) is shown in Fig. 5.3.

The convolutional layers of a CNN are linear, and thus they can be represented as a dense layer where the weight vectors of the individual neurons are constraint, e.g., they are sparse (most of the weights are zero) or several weights are shared between neurons. Therefore, compared to an MLP that only has fully connected neurons in each layer, the memory requirements and statistical efficiency are improved [GBC16]. Additionally, as complex features are inherently learned by the architecture, the final fully connected layer can be significantly reduced in size in comparison with an MLP that has to directly learn these features. An additional advantage of CNNs is that, due to the pooling layers

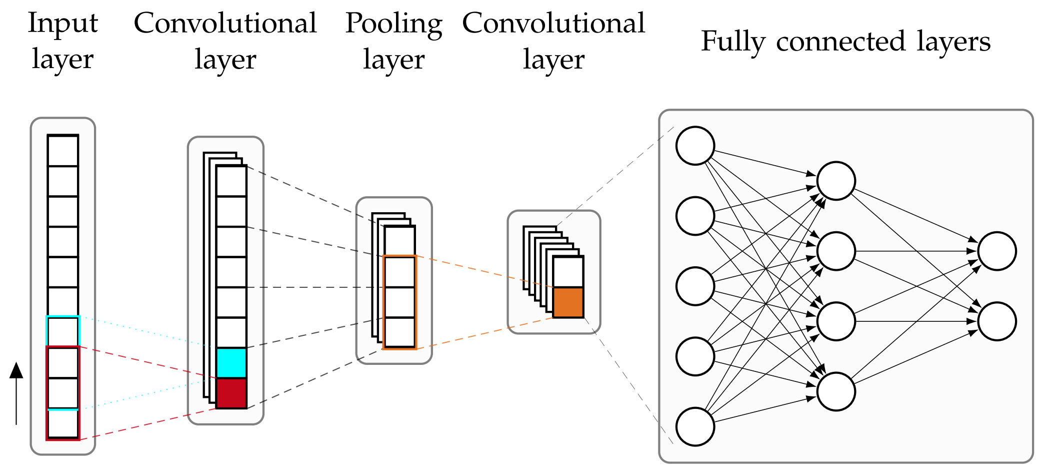


Figure 5.3 Visualization of a one dimensional CNN. The architecture is described by [`INPUT`(10), `CONV`(3, 3, 1), `POOL_avg`(2, 2), `CONV`(6, 3, 1), `FLATTEN`, `DENSE`(5), `DENSE`(4), `DENSE`(2)].

and the stride of the convolution, the networks become robust to variances (shifts in the location of features) in the input data. This is especially of interest in the side-channel domain as it helps to defeat hiding countermeasures that aim at reducing the SNR due to a misalignment of traces.

5.1.3 Training and Architecture Optimization

The training process of a neural network, i.e., the optimization of its parameters, is performed for a fixed architecture, which requires that the architecture is defined before the training process. The process of evaluating multiple architectures and adapting them based on the achieved results is called hyperparameter optimization, where the hyperparameters include the details of the architecture as well as parameters that define the training process itself. This optimization can be performed manually through expert knowledge or automatically using, e.g., random grid search, Bayesian optimization, or advanced methods like reinforcement learning. As a detailed discussion of these methods is out of scope for this thesis, the interested reader is referred to [GBC16, Chapter 11.4].

The optimization of the network parameters, i.e., the weights of the individual neurons in each layer, is done by minimizing the empirical risk defined by a loss function L that describes the error of the prediction $f(\boldsymbol{x}) = \hat{y}$ given the correct label y as $L(\hat{y}, y)$. For a classification task, cross-entropy is usually used as a loss function. Using this loss function, the training of the network can be performed by different variants of the *stochastic gradient descent* algorithm, which is based on the so-called *backpropagation*. Backpropagation works by evaluating the loss function for a given input feature vector $\boldsymbol{x}_i \in \mathcal{X}_{train}$ and computing the gradient of the loss function regarding the parameters of the network $\boldsymbol{\theta}$ as

$$\nabla_{\boldsymbol{\theta}} L = \frac{\partial L(f(\boldsymbol{x}), y)}{\partial \boldsymbol{\theta}} , \tag{5.1}$$

where $\boldsymbol{\theta}$ includes all individual weights of the neurons in each layer of the network. With the knowledge of the gradient, the contribution of the individual parameters to the loss can be determined with the goal of updating the parameters such that a minimum of the loss function is reached. The update of the parameters $\Delta\boldsymbol{\theta}$ is performed in the direction of the gradient scaled with a positive value η called the *learning rate* as

$$\Delta\boldsymbol{\theta} = -\eta \cdot \nabla_{\boldsymbol{\theta}} L . \tag{5.2}$$

In order to compute the exact gradient of the loss function, all training examples[3] of the training set $\mathcal{X}_{train}$ have to be used. Nevertheless, the *stochastic* gradient descent algorithm only uses a statistical

[3]These examples are often called samples of the training set, which is not used in this thesis as the term sample is used to refer to a single point in time of a side-channel measurement.

estimation of the gradient by dividing the training set into multiple smaller sets, called minibatches. The estimation of the gradient for each minibatch (often also called batch) is then used to separately update the parameters according to the learning rate. This tradeoff between the accuracy of the gradient and required computation time is motivated by the fact that there is redundancy in the training data, which implies that not all examples of the training set are required to compute a reasonable approximation of the gradient, and a faster convergence due to the multiple parameter updates [GBC16]. The training is performed for multiple passes through the whole training set, where a single pass is called a training *epoch* until a specified amount of epochs or a stopping criterion is reached. The goal of training is to achieve generalization, which means that the trained network also correctly classifies data that is not used during the training process. Therefore, the evaluation set $\mathcal{X}_{val}$ is used to monitor the training result, e.g., after each epoch, such that a possible underfitting or overfitting to the training data can be identified. This allows to use *early stopping* as a stopping criterion, which states that the training should be stopped if the classification error of the evaluation set does not decrease for a given number of iterations.

In practice, modern variants of the stochastic gradient descent algorithm are used. These algorithms optimize the training process through the use of an individual adaptive learning rate for each parameter [GBC16, Chapter 8.5].

5.1.4 Evaluation Metrics

In the attack phase of a DL-based side-channel attack, the trained network is used to classify each trace $\boldsymbol{t}$ of $\mathcal{X}_{attack}$, which results in the probabilities $p(l|\boldsymbol{t})$ denoted as $p_{l,\boldsymbol{t}}$ for each class l at the output of the softmax layer[4]. For an attack on AES, the classes are usually chosen as the S-box output of byte i after the first round $y = \mathrm{S}(ptxt_i \oplus k_i)$ (ID model) or its Hamming weight $y = \mathrm{HW}\left(\mathrm{S}(ptxt_i \oplus k_i)\right)$ (HW model), according to the label from training. Thus, to get the probabilities $p_{c,\boldsymbol{t}}$ for the $N_c = 256$ different key byte candidates $\hat{k}_c$ with $c \in \{0, \ldots, 255\}$, a transformation is required. In case of the identity (ID) model, where $N_l = N_c$ holds, an inversion the S-box maps to the respective key byte, while for HW labels the $N_l = 9$ softmax output probabilities $p_{l,\boldsymbol{t}}$ are assigned to $N_c > N_l$ different $\hat{k}_c$, i.e., several $\hat{k}_c$ share the same $p_{l,\boldsymbol{t}}$. In the following, the $N_c \times N_t$ matrix $\mathbf{P}$ contains the $p_{c,\boldsymbol{t}}$ as its entries, i.e., the probabilities for the candidate $\hat{k}_c$ and trace $\boldsymbol{t}$, for all N_c key byte candidates and $N_t = |\mathcal{X}_{attack}|$ traces available. Usually, a single prediction does not suffice to reliably predict the correct secret and therefore several predictions have to be combined. This combination is computed as the product of the probabilities $p_{c,\boldsymbol{t}}$ for a given subset of traces with cardinality $N \leq N_t$, which is usually determined by the sum of logarithmic probabilities

$$p_c(N) = \prod_{t=0}^{N-1} p_{c,\boldsymbol{t}} = \sum_{t=0}^{N-1} \log(p_{c,\boldsymbol{t}}) \tag{5.3}$$

to avoid numerical underflows or overflows.

Ultimately, an attacker is interested in the capability of the network to retrieve the correct value of the attacked secret key byte denoted as $k^\star$. The *key rank* [PGA+23] is used as a measure for the remaining uncertainty about the correct key value given the predictions of the network. Taking the probabilities p_c for all N_c key candidates, a rank vector $rank = [r_0, r_1, \cdots, r_{N_c-1}]$ can be computed by sorting all p_c in decreasing order with r_0 presenting the most likely candidate. Now, the key rank, which we define as $\mathrm{KR}(N)$, represents the position of $k^\star$ in $rank$, where a key rank of zero indicates that the correct key is found. Note that other works frequently use guessing entropy as an attack metric, which is defined as the mean key rank of several attacks (not to be confused with several traces used for a single attack). We additionally express the key rank in terms of entropy in bits as

$$H_{\mathrm{KR}}(N) = \log_2\left(\mathrm{KR}(N) + 1\right), \tag{5.4}$$

[4]The class index l is used in order to emphasize that the number of classes for which the networks provides predictions can be different from the amount possible values N_c of the targeted intermediate.

ranging from 0 to $\log_2(N_c)$. An entropy of 0 corresponds to a successful attack.

We additionally introduce the metric N' that defines the number of traces for which the final key rank after N_t traces occurs for the first time, i.e.,

$$N' := \min_N (N \mid \mathrm{KR}(N) \leq \mathrm{KR}(N_t)) \,. \tag{5.5}$$

The implicit assumption of the metric is that the key rank decreases with an increasing number of traces and reaches an optimum after a certain amount. The number of traces for which the key rank reaches zero is denoted N_0. The metric provides an insight into whether additional traces would potentially further decrease the key rank. If $N' = N_t$, the network requires all available traces for its best performance and might still be improving with more traces. Similarly, if $N' << N_t$ most likely additional traces do not add information and the network reaches its optimal performance.

5.1.5 Advantages and Disadvantages of DL-SCA

After Maghrebi et al. [MPP16] showed that DL-based SCA is possible, there have been numerous publications about advantages in comparison with classical profiled side-channel attacks like template attacks. This section gives an overview of these advantages in combination with possible disadvantages leading to a motivation for the presented work in Chapter 6 and Chapter 7.

A first advantage is that no assumption on the leakage model is needed for a DL-based SCA as due to the *universal approximation theorem* any function (of the leakage) can be represented by an MLP given it is large enough [GBC16, Chapter 6.4.1]. Therefore, the fixed assumption of a template attack in modeling the leakage as a multivariate Gaussian distribution is not needed anymore, and the attack can possibly benefit from a more accurate model of the leakage. Second, DL-based SCA significantly reduces or even eliminates the required preprocessing of side-channel traces. As discussed in Section 2.3, template attacks usually require a preprocessing of traces to identify POIs or perform dimensionality reduction techniques. This preprocessing is needed to only build the templates from samples with a high SNR and therefore reduce noise and limit the number of used samples in order to achieve a feasible computational complexity. With the training of a neural network, the samples that positively influence classification are identified through backpropagation and therefore a POI detection is performed automatically. Nevertheless, some preprocessing steps are still required, such as normalizing the traces and selecting a suitable sample interval that includes the computation of the targeted intermediate. Another advantage that goes in the same direction, i.e., reduces the preprocessing effort, is the ability of deep learning approaches to cope with a misalignment of traces. For classical side-channel attacks, the available traces must be correctly aligned to achieve a high SNR. A misalignment of traces is usually induced by the measurement setup through, e.g., an unstable clock of the target device or dedicated hiding countermeasures (c.f. Section 2.4). Therefore, different alignment methods are frequently used in SCA to preprocess traces, but a successful realignment is not always possible. With DL-based SCA methods, the misalignment of traces can be inherently learned during the training process and especially CNNs have been shown to perform well on misaligned traces due to their spatial invariance of learned features [CDP17, BPS+20]. Finally, with DL-based SCA, it is possible to attack masked implementations without a manual combination of the respective samples that correspond to each share of the targeted intermediate [BPS+20]. This combination is needed to estimate the higher-order statistical moments (c.f. Section 2.4) and usually requires narrowing down possible sample ranges of shares to achieve a feasible computational complexity. With the use of a neural network, the correct combination can be learned during the training and successful attacks have been shown for large sample ranges ($\geq$ 10 000 samples) without preprocessing [LZC+21, PWP22a, HAS+24].

A first disadvantage of DL-based SCA is the large number of possible hyperparameter configurations that have to be evaluated in order to find the optimal model for a given implementation or dataset, as the optimal combination is highly dependent on the attacked implementation (types of countermeasures) or the dataset (number of available traces, samples per trace or the SNR of the measurements) [KLP+22]. Usually, a full exploration of the hyperparameter space is not feasible due to computation time and resource constraints. Therefore, the reason why an attack is not able to retrieve the correct secret can be an ineffective or wrong network architecture, which can lead to wrong conclusions about the security of the implementation by an evaluator. Further, as POIs are learned during training and the network is able to approximate arbitrary functions of the leakage, it is important to correctly understand and evaluate the datasets on which research on DL-based SCA is conducted. This is especially important for higher-order attacks on masked implementations, where the network is claimed to combine the leakage of multiple shares. In order to claim a successful multivariate higher-order attack by the network, it has to be made sure that the evaluated dataset does not contain first-order leakage or univariate leakage, which can be exploited by the network instead of combining the leakage of multiple shares. Finally, a major downside of DL-based SCA is the lack of explainability of attack results. As the networks perform POI detection internally, an evaluator does not know which samples and their corresponding leakage are used in order to perform a successful attack. Nevertheless, in order to remove these leakages and therefore again harden an implementation, the exact leaking samples and the corresponding operations have to be identified.

5.2 ANSSI SCA Databases (ASCAD)

The ANSSI SCA Database (ASCAD) was published by researchers of the French cyber security agency (ANSSI) in combination with their paper [BPS+20][5] that provides an in-depth study of DL-based SCA approaches using the database. Since its publication, ASCAD is typically used as the standard dataset for research in the field due to its public availability [BPS+18] and detailed documentation, as well as the availability of example code for the used network architectures, which can be utilized to replicate results and serves as a starting point for further research. Over time, the authors published multiple versions of the database in order to facilitate research on additional attack scenarios and provide an attack target with stronger countermeasures. This section gives an overview of the different ASCAD databases that are used in this thesis, starting with the presentation of the underlying first-order masked AES implementation of the databases in Section 5.2.1. This is followed by an introduction to the different versions of the databases with their respective characteristics in Section 5.2.2. Finally, Section 5.2.3 presents the neural network architectures that are used to attack ASCAD in this thesis.

5.2.1 AES Implementation

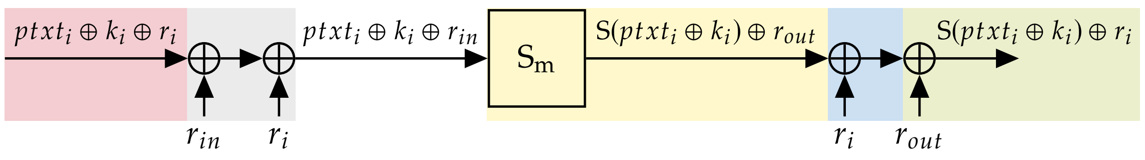
The used ASCAD databases in this thesis are based on a first-order boolean masked implementation of AES with a key size of 128 bit that is written in assembly [BLPR18]. As the AES block cipher is the primary target of research on SCA, this thesis assumes knowledge about the algorithm, and for a detailed introduction, the reader is referred to [DR20]. The implementation masks each key byte k_i using a mask byte r_i[6]. In order to protect the S-box look-up, a table recomputation method [PR07, RPD09] is used that computes a masked S-box ($\mathrm{S_m}$) by determining for all possible entries $x \in \{0, \ldots, 255\}$ the masked S-box output values according to

$$\mathrm{S_m}(x) = \mathrm{S}(x \oplus r_{in}) \oplus r_{out} \ , \tag{5.6}$$

where r_{in} and r_{out} refer to the S-box input and output masks that are equal for all bytes. A block diagram of the whole masked SubBytes operation is shown in Fig. 5.4. In total, 18 masks are required to protect the implementation.

[5] Note that [BPS+20] is the peer-reviewed version of the original publication from 2018 [PSB+18], which was published on the Cryptology ePrint Archive. For the remainder of this thesis, the peer-reviewed version [BPS+20] of the paper is used as reference.

[6] Note that we use byte indices according to the AES standard in contrast to the indexing introduced with the ASCAD databases [BPS+20] starting with index 1.



$ptxt_i \oplus k_i \oplus r_i$
$ptxt_i \oplus k_i \oplus r_{in}$
S_m
$S(ptxt_i \oplus k_i) \oplus r_{out}$
$S(ptxt_i \oplus k_i) \oplus r_i$
r_{in}
r_i
r_i
r_{out}

Figure 5.4 Masked SubBytes operation of the ASCAD implementation.

Note that for the measurements contained in the ASCAD databases, the S-box look-ups corresponding to key bytes k_0 and k_1 are not masked. This is due to the fact that the implementation supports a shuffling countermeasure that determines the order of S-box accesses from the values of the first two masks r_0 and r_1, which are set to `0x00` to disable the shuffling and facilitate the analysis. The sequence of the S-box accesses is therefore fixed to the following processing order of key bytes:

$$15 \rightarrow 12 \rightarrow 13 \rightarrow \mathbf{1} \rightarrow 8 \rightarrow 10 \rightarrow \mathbf{0} \rightarrow 3 \rightarrow 7 \rightarrow 6 \rightarrow 9 \rightarrow 5 \rightarrow 11 \rightarrow 2 \rightarrow 4 \rightarrow 14, \tag{5.7}$$

where the unmasked bytes are highlighted in bold. Another interesting property of the implementation is that it uses additional operations that aim at clearing the value at a destination address/register or on the read/write bus to prevent HD leakage. An example of such an operation, which will be referred to as *security load*, is done as follows

$$\text{security load} = \mathrm{S_m}(\underbrace{\mathrm{S}(k_{\underline{i}-1} \oplus ptxt_{\underline{i}-1}) \oplus r_{\underline{i}-1}}_{:=\mathrm{S_{prev}}}) = \mathrm{S}(\mathrm{S_{prev}} \oplus r_{in}) \oplus r_{out}, \tag{5.8}$$

where $\underline{i}$ denotes the byte index of the shuffled order and $\underline{i}-1$ indicates the previous byte according to Eq. (5.7). Results in Chapter 6 show that leakage of this particular operation is present in the trace segment used for ASCAD.

5.2.2 Versions of the Database

The first ASCAD database, called ASCAD fixed key (*ASCAD FIX*) in this thesis, was published as supplementary material of [PSB+18] and contains measurements of the first-order boolean masked AES implementation detailed in Section 5.2.1. The measurements are captured on an 8-bit ATmega8515 AVR microcontroller running at a clock frequency of $f_{clk} = 4\,\mathrm{MHz}$[7]. The database is structured into a training and attack set and contains the required metadata for evaluation, such as the used key, plaintext, ciphertext, and the values of random masks. *ASCAD FIX* has the property that the same key is used for all measurements in the database. This does not correspond to a realistic attack scenario, as the attacker should not know the targeted key, and has some undesired implications. In Chapter 6, this thesis shows that training and attacking on the same fixed key is an easier task for the evaluated networks compared to training on variable keys. Additionally, it has been shown by Hoang et al. [HHO20] that by addition of the plaintext to the used training label, the network is able to provide correct classifications even if the traces are replaced by random values, as the network learns the bijection of the correct label and the plaintext. In order to provide a different attack scenario and overcome the possible downsides of an equal key, the authors published an updated version of ASCAD with a variable key for the training set. This updated version is referred to as ASCAD variable key (*ASCAD VARIABLE*) for the remainder of this thesis. For completeness, it has to be noted that there exists a second version of ASCAD that includes stronger countermeasures (shuffling in combination with affine masking) measured on a microcontroller with a modern ARM Cortex-M4 architecture [RBT18]. This database is nevertheless not considered in this thesis.

An overview of the characteristics of both ASCAD databases is given in Table 5.1. *ASCAD FIX* consists of 60 000 traces and *ASCAD VARIABLE* provides of a total amount of 300 000 traces. Both

[7] `https://github.com/ANSSI-FR/ASCAD/issues/2`, last accessed 22nd April 2024

	N_{train}	N_{attack}	N_s	N_s/clock cycle	Key of training set
ASCAD fix	50 000	10 000	700	50	fixed
ASCAD variable	200 000	100 000	1400	125	random

Table 5.1 Overview of the ASCAD databases.

databases show a different split of the traces into the training set $\mathcal{X}_{train}$ with $|\mathcal{X}_{train}| = N_{train}$[8] and attack set $\mathcal{X}_{attack}$ with $|\mathcal{X}_{attack}| = N_{attack}$ according to Table 5.1. The databases contain raw traces with 100 000 (*ASCAD fix*) and 250 000 (*ASCAD variable*) samples that correspond to almost the first two AES rounds. In order to facilitate analysis, the authors propose the usage of an excerpt of N_s samples corresponding to the SubBytes operation processing the third key byte k_2, which is commonly referred to as the ASCAD database in related work. This sample range is chosen to include the leakage of $\mathrm{S}(ptxt_i \oplus k_i) \oplus r_i$ and $\mathrm{S}(ptxt_i \oplus k_i) \oplus r_{out}$ with the corresponding masks such that second-order attacks are feasible. It also implies the training label, which is chosen as $\mathrm{S}(ptxt_i \oplus k_i)$ (ID model) and $\mathrm{HW}(\mathrm{S}(ptxt_i \oplus k_i))$ (HW model). Finally, it has to be emphasized that both ASCAD databases differ in their measurement characteristics. *ASCAD variable* is measured with a sampling frequency of $f_s = 500\,\mathrm{MS/s}$ and *ASCAD fix* with $f_s = 200\,\mathrm{MS/s}$[9], which leads to 125 and 50 samples per clock cycle.

5.2.3 Used Network Architectures

Multiple CNN-based neural network architectures are used to provide the experimental attack results in Chapters 6 and 7. This section gives an overview of the used models in combination with their respective hyperparameters of the training process.

As a baseline, the reference architecture of the ASCAD authors is used, which is called $\mathrm{CNN}_{\mathrm{best}}$ for the remainder of this thesis. It is based on the popular VGG-16 [SZ15] architecture used for image recognition or object detection tasks. In [BPS+20], the ASCAD authors perform a manual hyperparameter search using the VGG-16 architecture as a baseline and derive $\mathrm{CNN}_{\mathrm{best}}$ as the hyperparameter combination that shows the best attack results on *ASCAD fix*. With the publication of *ASCAD variable*, the authors still use $\mathrm{CNN}_{\mathrm{best}}$ as the reference architecture, but the input layer has to be increased due to the increased N_s of the chosen trace excerpt[10]. For the training, a batch size of 200, the Root Mean Square Propagation (RMSProp) optimizer, a learning rate of $\eta = 10^{-5}$, and a typical amount of 100 training epochs is used. The used architectures for $\mathrm{CNN}_{\mathrm{best}}$ are detailed in Table 5.2.

In contrast, we use four additional architectures from [RWPP21] for which the hyperparameter search and optimization were performed automatically through the use of reinforcement learning. The resulting architectures are tailored to a specific dataset and the used training label (ID or HW model). All found architectures through the reinforcement learning process are small in terms of trainable parameters and show a good attack performance. Due to the compact size of the architectures, they are called $\mathrm{CNN}_{\mathrm{small}}$ for the remainder of this thesis. For the training process of the $\mathrm{CNN}_{\mathrm{small}}$ architectures, we follow the authors of [RWPP21] and use a batch size of 400, the Adam optimizer [KB15], and a custom cyclic learning rate as proposed by [Smi17]. The number of training epochs is set to 50 for the results presented in Chapter 6 and has been increased to 100 for Chapter 7. The resulting architectures are shown in Table 5.3. It can be seen that the amount of trainable parameters is reduced significantly in comparison to $\mathrm{CNN}_{\mathrm{best}}$.

[8]The traces for the evaluation set $\mathcal{X}_{val}$ are subtracted from the available training traces.

[9]There has been uncertainty regarding the used sampling frequency of *ASCAD fix*. The sampling frequency is not $f_s = 2\,\mathrm{GS/s}$ as provided by [BPS+20] and generally adopted by related work, as in this case, the whole trace segment would only show 1.4 clock cycles, which is not consistent with the leakage evaluation of intermediates (c.f. Fig. 6.1). During the work on this thesis, a sampling frequency of 200 MS/s was considered the most convincing variant. See `https://github.com/ANSSI-FR/ASCAD/issues/13`, last accessed 22nd April 2024, for a discussion with the ASCAD authors.

[10]`https://github.com/ANSSI-FR/ASCAD/blob/master/ASCAD_train_models.py`, last accessed 25th April 2024

	network architecture	trainable parameters
ASCAD fix		
CNN_{best} (ID)	INPUT(700), CONV(64,11,2), POOL_avg(2,2), CONV(128,11,1), POOL_avg(2,2), CONV(256,11,1), POOL_avg(2,2),CONV(512,11,1), POOL_avg(2,2), CONV(512,11,1), POOL_avg(2,2), FLATTEN, DENSE(4096), DENSE(4096), SM(256)	43 583 872
CNN_{best} (HW)	INPUT(700), CONV(64,11,2), POOL_avg(2,2), CONV(128,11,1), POOL_avg(2,2), CONV(256,11,1), POOL_avg(2,2), CONV(512,11,1), POOL_avg(2,2), CONV(512,11,1), POOL_avg(2,2), FLATTEN, DENSE(4096), DENSE(4096), SM(9)	42 571 913
ASCAD variable		
CNN_{best} (ID)	INPUT(1400), CONV(64,11,2), POOL_avg(2,2), CONV(128,11,1), POOL_avg(2,2), CONV(256,11,1), POOL_avg(2,2),CONV(512,11,1), POOL_avg(2,2), CONV(512,11,1), POOL_avg(2,2), FLATTEN, DENSE(4096), DENSE(4096), SM(256)	66 652 544
CNN_{best} (HW)	INPUT(1400), CONV(64,11,2), POOL_avg(2,2), CONV(128,11,1), POOL_avg(2,2), CONV(256,11,1), POOL_avg(2,2), CONV(512,11,1), POOL_avg(2,2), CONV(512,11,1), POOL_avg(2,2), FLATTEN, DENSE(4096), DENSE(4096), SM(9)	65 640 585

Table 5.2 Overview of the used CNN_{best} architectures. The CONV and DENSE layers use the ReLU activation function. Note that the original CNN_{best} architecture for *ASCAD fix* uses a stride of one in the first convolutional layer, while the architecture for *ASCAD variable* uses a stride of two. For this thesis, we use a stride of two for both databases since even with this change, similar attack results for *ASCAD fix* are achieved (c.f. [BPS+20, Figure 14] and Fig. 6.5a).

	network architecture	trainable parameters
ASCAD fix		
$\text{CNN}_{\text{small}}$ (ID)	INPUT(700), CONV(128,25,1), POOL_avg(25,25), FLATTEN, DENSE(20), DENSE(15), SM(256)	79 439
$\text{CNN}_{\text{small}}$ (HW)	INPUT(700), CONV(16,100,1), POOL_avg(25,25), FLATTEN, DENSE(15), DENSE(4), DENSE(4), SM(9)	8480
ASCAD variable		
$\text{CNN}_{\text{small}}$ (ID)	INPUT(1400), CONV(128,3,1), POOL_avg(75,75), FLATTEN, DENSE(30), DENSE(2), SM(256)	70 492
$\text{CNN}_{\text{small}}$ (HW)	INPUT(1400), CONV(8,3,1), POOL_avg(25,25), FLATTEN, DENSE(30), DENSE(30), DENSE(20), SM(9)	15 241

Table 5.3 Overview of the used $\text{CNN}_{\text{small}}$ architectures. The CONV and DENSE layers use the Scaled Exponential Linear Unit (SeLU) [KUMH17] activation function.

6 A Second-Look at the ASCAD Databases

This chapter is based on the publication *Egger/Schamberger/Tebelmann/Lippert/Sigl: A Second Look at the ASCAD Databases* published in *Constructive Side-Channel Analysis and Secure Design (COSADE)*, 2022 [EST+22].
Parts of the presented work were developed during the research internship of Maximilian Egger, including most of the framework to generate the presented DL-based SCA results. The visualization of the second-order CPA results in Fig. 6.4 was developed during the master thesis of Florian Lippert [Lip20].

6.1 Introduction

Most research on the ASCAD databases has tried to propose, improve, and compare deep learning approaches in the SCA realm and consequently, little attention has been paid to the details of the underlying datasets. The analysis of the leakage behavior has been limited to a few intermediate values and a single key byte of the *ASCAD fix* dataset [BPS+20], i.e., the greater part has not yet been analyzed in detail. Nevertheless, in order to interpret attack results and reason about the attack capabilities of evaluated networks, a thorough understanding of the used datasets is required. This chapter closes this gap with a detailed evaluation of both ASCAD datasets with classical SCA and a comparison of both datasets. Additionally, in the context of this evaluation, the locations of trace segments for all key bytes of the implementation are identified from the provided raw traces, allowing the use of multiple attack locations for the evaluation of DL-based SCA methods. In particular, this chapter provides results from leakage evaluation, first-order, and uni- and multivariate second-order SCA results for *all key bytes* of the ASCAD AES implementation. The findings show that research can benefit from using the different key bytes of the ASCAD databases to evaluate robustness under varying leakage conditions. Furthermore, it is shown that for the ASCAD databases, training and attacking on the same key is a substantially easier task for CNNs than training on randomized keys and attacking on a fixed key. This highlights that the *ASCAD variable* dataset is to be preferred for DL-based SCA research, and raises the question if results based on the *ASCAD fix* dataset are similar if re-evaluated on *ASCAD variable*.

Related Work The main body of research is devoted to the proposal and improvement of network architectures, and little attention has been paid to analyzing the datasets by classical SCA techniques. Benadjila et al. [BPS+20] provide some analysis in terms of the SNRs of intermediate values related to mask values of the round mask r_2 and the output mask r_{out}, but limit their analysis to the processing of key byte k_2 only. They show further that there is no SNR related to the unmasked value of the S-box output of the first round $\mathrm{S}(ptxt_2 \oplus k_2)$ for the correct key byte k_2 and the respective plaintext byte $ptxt_2$. Furthermore, they note that key bytes k_0 and k_1 are unmasked. Other works that compute leakage behavior also use the *ASCAD fix* dataset and use the results for comparison with attribution techniques that are supposed to highlight important features identified by the networks: Masure et al. [MDP19] show the SNR of r_{out} and $\mathrm{S}(k_2 \oplus ptxt_2) \oplus r_{out}$ to compare it to their results on gradient visualization. Timon [Tim19] computes CPA results with known values for the round mask r_2 and the S-box value $\mathrm{S}(k_2 \oplus ptxt_2) \oplus r_2$ to evaluate his sensitivity analysis. Kuroda et al. [KFYF21] show first-order CPA results for all key bytes of *ASCAD fix* using only 30 % of available traces. As proposed by Benadjila et al. [BPS+20], in most research, the third key byte k_2 is selected as a target, and the proposed POIs (in the form of the chosen trace segment) are used for training and attack. However, there are some exceptions: Wu and Picek [WP20] simulate shuffling of the masked bytes for *ASCAD FIX* but do not compare attacks for different bytes. Zhou and Standaert [ZS20] compare rank results of all 16 S-boxes of the *ASCAD FIX* dataset and the CNN_{best} network from [BPS+20]. They show that unmasked bytes can be attacked using a single trace and observe that significantly more traces are needed for k_{10} compared to other bytes. However, there is no analysis of the similarities or differences in the leakage behavior between bytes to explain the performance differences. Kuroda et al. [KFYF21] evaluate results of all key bytes with non-profiled DL-based SCA using 34 % of the entire sample range of *ASCAD FIX*. Lu et al. [LZC+21] introduce an architecture that takes the entire trace of *ASCAD FIX* and *ASCAD VARIABLE* instead of the preselected POIs proposed by Benadjila et al. [BPS+20]. The analysis is limited to k_2, for which fewer traces are required for a successful attack when using the entire trace. Bronchain et al. [BCS21] use the entire traces of *ASCAD VARIABLE* and derive an attack based on template matching with LDA preprocessing. They pass the results to a belief propagation algorithm to reduce the number of traces for an attack. The results from [BCS21, LZC+21] suggest that training benefits from leakage points outside the proposed POI range. Rijsdijk et al. [RWPP21] optimize network hyperparameter search to reduce the number of traces for a successful attack as well as the number of trainable parameters and compare to similar approaches. They provide the number of traces to reach an average key rank of zero for *ASCAD FIX* and *ASCAD VARIABLE* datasets simultaneously. Results suggest that *ASCAD FIX* requires fewer traces for ID labels, while for models using the HW label *ASCAD VARIABLE* can be at least as easily attacked. Regarding differences in the difficulty of training on fixed or variable key datasets, Hoang et al. [HHO20] show that if *ASCAD FIX* is used and the plaintext is added as an input label to the S-box output, the model can learn the bijection of the S-box from the labels.

Since the publication of the paper on which this chapter is based, several related works have been published. First, Perin et al. [PWP22a] show attack results for both ASCAD databases using a downsampled variant of the entire trace, optimized POIs with high SNR using the entire trace as well as the sample range proposed by the ASCAD authors. Their presented attack results also show performance differences between the individual key bytes, which are aligned with the results shown in this chapter. Additionally, Hajra et al. [HAS+24] used the complete sample ranges of both ASCAD databases. The results of [SET23], which are presented in Chapter 7, use the leakage evaluation shown in this chapter in order to interpret the results of the developed attribution methods and show that used architectures actually exploit the additional leakage of intermediates identified in this thesis. This is also confirmed in [YBP23] with their developed attribution method using *ASCAD VARIABLE*.

Contributions This chapter provides a detailed leakage analysis and classical SCA of the ASCAD datasets that allow for new insights beneficial for deep-learning attribution techniques. A comparison of the impact of fixed-key vs. variable-key training suggests that the *ASCAD VARIABLE* dataset should be used instead of the *ASCAD FIX* dataset. Finally, attack results for different bytes highlight that depending on the hyperparameter architecture, differing leakage from operations is challenging for CNNs. In particular, the contributions are the following:

- We provide a detailed leakage analysis for all bytes of the ASCAD implementation that highlights leakage from additional intermediate values not considered by related work. In addition, this leakage also largely differs between individual bytes. The findings are important to better understand how networks learn from side-channel traces, as they can be used to interpret the results of attribution techniques.

- We show first-order and second-order univariate vulnerabilities that are unexpected for a masked implementation, as the key byte can be recovered from a single sample. The first-order vulnerabilities occur only for up to two bytes when using the proposed trace segments of the ASCAD authors, while most (*ASCAD FIX*) and all (*ASCAD VARIABLE*) bytes can be attacked using the entire traces. This highlights a direction for future work on understanding if DL-based SCA approaches implement multivariate higher-order attacks or rather directly exploit these identified vulnerabilities when using the whole sample ranges contained in the ASCAD databases.

- We show that training on the same fixed key as during the attack yields significantly better attack results than training on variable keys for both ASCAD datasets. This suggests that results on the *ASCAD FIX* dataset overestimate the performance of networks and that the *ASCAD VARIABLE* dataset should be used in future work.

- Finally, we provide DL-based SCA results for all key bytes of the *ASCAD VARIABLE* dataset that show that traces of identical operations on the same dataset pose challenges to CNNs. The substantial variations for different bytes indicate that, in addition to evaluating different datasets, research benefits from using the entire ASCAD datasets to improve the robustness of results.

Outline First, Section 6.2 presents the results of the classical side-channel analysis of both ASCAD databases. This includes the leakage evaluation for all key bytes as well as the first- and second-order attack results. This is followed by a presentation of the DL-based SCA results in Section 6.3. First, the influence of the fixed key for the training set of *ASCAD FIX* is shown, followed by the evaluation of the attack results for all key bytes of the databases. Finally, the chapter is concluded in Section 6.4.

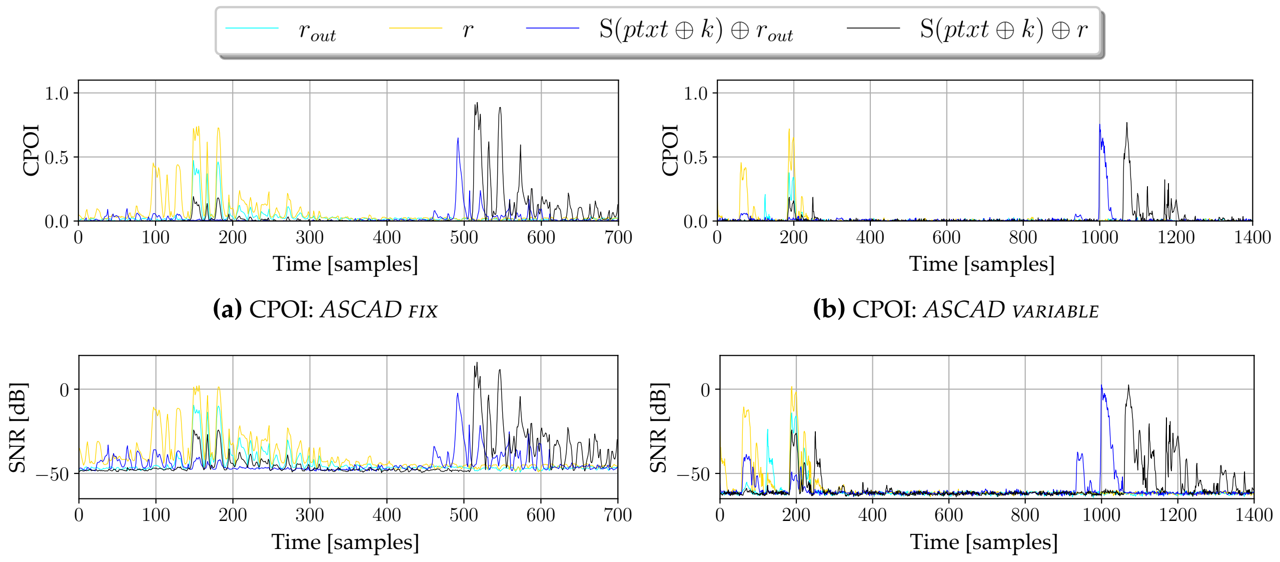
6.2 Classical Side-Channel Analysis of ASCAD

This section discusses the results of classical side-channel analysis for both ASCAD databases (see Section 5.2 for an introduction to the databases). First, Section 6.2.1 presents the results of a leakage evaluation for all key bytes of the chosen trace segment by the ASCAD authors. Additional results are shown for the execution of an entire SubBytes operation in order to match leakage to executed assembly instructions of the implementation. In Section 6.2.2, results of a first-order as well as a univariate and multivariate second-order CPA are presented for all key bytes of both the trace segment and the entire sample range of the databases.

6.2.1 Leakage Evaluation

We use the results of a leakage analysis of different intermediate values in the processing of the protected SubBytes operation detailed in Section 5.2.1 to establish a link between the implementation [BLPR18] and points in time of the measurements. We leverage the CPOI method (c.f. Section 2.1), which is a correlation-based leakage detection based on profiling instead of an abstract leakage model. The CPOI method uses k-fold cross-validation, i.e., the correlation is repeated with k distinct test sets, and results are averaged. In this thesis, all CPOI results are calculated using a total of 20 000 traces and a two-fold cross-validation ($k = 2$).

We start the CPOI evaluation with the trace segment proposed by the ASCAD authors, which contains the processing of key byte k_2. This analysis is subsequently extended to all key bytes, where we identify the respective trace segments corresponding to the same operations as for k_2. In a second step, we provide results for the whole masked SubBytes operation of key byte k_2. This is used to fit the different assembly instructions (with their corresponding line number in the source) to the best of our knowledge onto the CPOI plot, allowing further research about the origin of exploitable leakage used by a model. Our results show that the proposed segments contain additional leakage that has not been considered in related work.



(a) CPOI: *ASCAD FIX*

(b) CPOI: *ASCAD VARIABLE*

(c) SNR: *ASCAD FIX*

(d) SNR: *ASCAD VARIABLE*

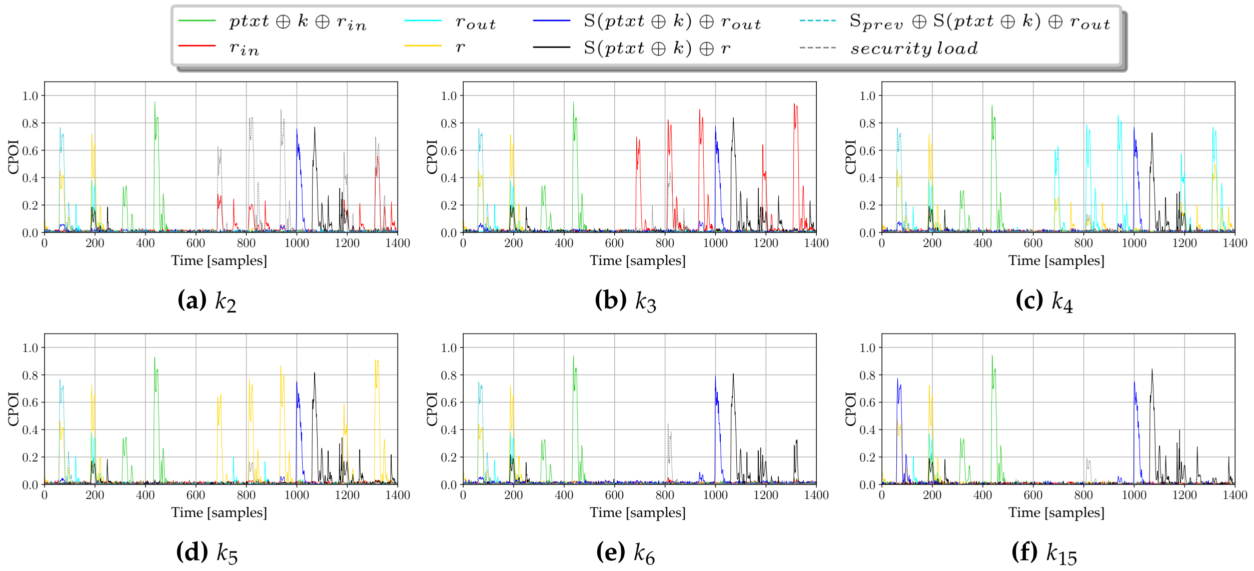
Figure 6.1 Leakage analysis of k_2 for the sample ranges and state-of-the-art intermediates from [BPS+20] proposed for both ASCAD datasets. The results of the SNR evaluation in Figs. 6.1c and 6.1d were generated using [CB23].

Analysis of Proposed Trace Segments In order to facilitate DL-based SCA, the ASCAD authors propose to use a subsegment of the traces that only contains some operations of the whole masked SubBytes operation as detailed in Section 5.2.2: For key byte k_2, a range of 700 (*ASCAD FIX*) and 1400 (*ASCAD VARIABLE*) samples is used. The rationale for this selection is to include intermediate values $S(k_2 \oplus ptxt_2) \oplus r_{out}$ and $S(k_2 \oplus ptxt_2) \oplus r_2$ as well as the respective mask values r_{out} and r_2, allowing for a second-order SCA attack. The leakage of these intermediates has been evaluated in [BPS+20] by calculating SNR values for the *ASCAD FIX* dataset.

In Figs. 6.1a and 6.1b, the difference between both datasets is shown in terms of CPOI results for key byte k_2 for the limited ranges and the state-of-the-art intermediates from [BPS+20]. For comparison, we additionally provide an SNR evaluation in Figs. 6.1c and 6.1d, where in contrast to [BPS+20] the SNR is visualized in logarithmic scale. As both methods perform equally well, CPOI is used for the remainder of this chapter due to the more intuitive nature of correlation results, which are confined to the interval between 0 and 1. By direct comparison of Fig. 6.1a and Fig. 6.1b, the leakage from *ASCAD FIX* measurements is present during the whole clock cycle and in addition spread among multiple clock cycles. In contrast, *ASCAD VARIABLE* shows a different, temporal confined leakage characteristic. We attribute these differences to changes in the measurement setup; see [MM13] for a detailed discussion of setup characteristics that can induce this behavior. A possible explanation is that the measurements in *ASCAD FIX* are subject to low-pass filtering[1]. It can also be noted that both ranges do not correspond to the same time interval. Due to the factor of 2.5 between sampling rates (200 MS/s vs. 500 MS/s), the trace segments of 1400 samples for *ASCAD VARIABLE* correspond to a shorter time interval than the trace segment of 700 samples for *ASCAD FIX*, i.e., the first 92 samples and the last 48 samples in Fig. 6.1a do not have a correspondence in Fig. 6.1b. Nevertheless, the important leakages are contained in the proposed segments of both datasets.

We conclude that the leakages do not differ significantly between both datasets (except that leakage is spread over multiple clock cycles in the *ASCAD FIX* case) as expected from the identical underlying implementation. In the following, we only show results for the *ASCAD VARIABLE* measurements because the temporal confinement eases the match of the intermediate values from the masked SubBytes operation (c.f. Fig. 5.4) and POIs.

[1]The authors revised their claim in [BPS+20] that both datasets consist of electromagnetic (EM) measurements to them being power measurements in `https://github.com/ANSSI-FR/ASCAD/issues/13`, last accessed 24th April 2024.



(a) k_2 **(b)** k_3 **(c)** k_4

(d) k_5 **(e)** k_6 **(f)** k_{15}

Figure 6.2 CPOI analysis of the *ASCAD* VARIABLE dataset for key bytes k_2, k_3, k_4, k_5, k_6 and k_{15}. The bytes k_7 to k_{14} show a qualitatively similar leakage behavior as k_6 and are therefore shown in Appendix B.

Comparision of Key Bytes To obtain comparable DL-based SCA results for all key bytes, it is crucial to provide the network with input data that contains the same operations among the different key bytes. Therefore, corresponding sample ranges for all key bytes have to be found from the entire sample range of the databases. For this purpose, we identify the leakage of the intermediate value $\mathrm{S}(ptxt_i \oplus k_i) \oplus r_{out}$ as a suitable target for alignment, as its maximum value is easily identified[2] for all key bytes. We align the sample index of this characteristic leakage in accordance with its location in the proposed range of key byte k_2. Consequently, in the case of the *ASCAD* FIX dataset, the maximum CPOI is fixed to sample index 492. In the *ASCAD* VARIABLE case, the leakage is fixed to sample index 1000. The exact sample ranges for the different bytes are given in Table 6.1.

	ASCAD FIX		*ASCAD* VARIABLE	
Byte	min	max	min	max
0	30 824	31 524	45 943	47 343
1	24 577	25 277	30 942	32 342
2	45 400	46 100	80 945	82 345
3	32 906	33 606	50 943	52 343
4	47 482	48 182	85 946	87 346
5	41 235	41 935	70 945	72 345
6	37 071	37 771	60 944	62 344
7	34 989	35 689	55 943	57 343
8	26 660	27 360	35 942	37 342
9	39 154	39 854	65 944	67 344
10	28 742	29 442	40 942	42 342
11	43 318	44 018	75 945	77 345
12	20 413	21 113	20 941	22 341
13	22 495	23 195	25 941	27 341
14	49 565	50 265	90 946	92 346
15	18 330	19 030	15 940	17 340

Table 6.1 Sample indices for the alignment of all key bytes on CPOI values.

The CPOI results for the remaining key bytes of *ASCAD* VARIABLE after alignment are shown in Fig. 6.2[3]. To increase readability, we omit the byte index i in the legends of the plots, and the respective byte is given in the caption. In contrast to the leakage evaluated in the state of the art and shown in Fig. 6.1, we identify additional leakage contained in the segments. A similarity between all

[2]Note that for key byte k_{15} of the *ASCAD* VARIABLE dataset there is one additional peak compared to the other bytes. We manually corrected the range such that the same operations are contained.

[3]For completeness, the CPOI evaluations for all key bytes of *ASCAD* FIX are additionally shown in Appendix B.

bytes is the additional leakage of $ptxt \oplus k \oplus r_{in}$ from sample 300 to 500 (green color). It can also be seen that the CPOI plots of the individual bytes largely differ in two regions, namely from samples 600 to 1000 and 1100 to 1400. In these regions, an additional leakage of either the security load (k_2; dashed gray), r_{in} (k_2, k_3; red), r_{out} (k_4; cyan), or r_i (k_5; yellow) is present. The remaining bytes do not show this leakage characteristic. For k_{15} an additional leakage of $\mathrm{S}(k \oplus ptxt) \oplus r_{out}$ (blue) is visible at the beginning of the trace segment (around sample 70). A reason for this could be that k_{15} is the byte that is processed first (c.f. Eq. (5.7)) and therefore registers are still empty, which leads to additional leakage.

Two conclusions can be drawn from the CPOI evaluation in Fig. 6.2. First, there is a significant difference between the leakage observed for the individual key bytes. As most of the related work on the ASCAD databases has been conducted on k_2, the performance of DL-based SCA for other key bytes (with different leakage characteristics) might differ. We evaluate this assumption in Section 6.3.3. Second, in addition to the intermediate values r_i and r_{out} provided by related work, the results show that trace segments include additional leakage like the input mask r_{in} and $ptxt_i \oplus k_i \oplus r_{in}$ that might be utilized by the networks for the attack. In particular, for key byte k_2, there is a significant leakage of these values that has not been considered by related work to the best of our knowledge. The presence of leakage resulting from additional intermediate values is of interest for attribution techniques to determine important features for DL-based SCA. The additional leakages we outline can lead to a better understanding of attribution difficulties for the ASCAD database of existing methods [HGG20, MDP19], which consider only the leakage evaluation by Benadjila et al. [BPS+20]. This thesis shows that the identified leakages are indeed helpful, as they allow an interpretation of attribution results for the developed method in Chapter 7.

Analysis of the Entire Masked SubBytes Operation This section analyzes the leakage behavior of the entire masked SubBytes operation for key byte k_2 of *ASCAD VARIABLE*. The resulting CPOI analysis is shown in Fig. 6.3. By analyzing the assembly code, the correlation peaks can be related to the respective parts of the masked SubBytes operation, which are denoted by the color bars on top of the figure in the same colors as for the parts of Fig. 5.4. The trace segment chosen by the ASCAD authors and evaluated in the previous section is marked by a horizontal bar with the label *ASCAD*. The resulting sample range mainly covers the second part of the masked SubBytes operation depicted in Fig. 5.4, namely after the lookup of the masked S-box. We further fitted the individual assembly instructions of the implementation onto the CPOI plot in Fig. 6.3 to ease the analysis of the leakage origins[4]. The different line numbers of the implementation are depicted by vertical lines. Code lines marked with *a* and *b* represent subcycles of assembler instructions that take more than one clock cycle to be executed. Despite a thorough analysis and study of the microcontroller architecture as well as the implementation, some correlation peaks and their corresponding code lines can not be easily explained. We attribute this to leakage behavior observable between all registers of AVR microcontrollers, as analyzed, e.g., in [SR16]. The processing of the next byte according to Eq. (5.7) is marked by another horizontal bar and a corresponding label in Fig. 6.3. Note that since several registers[5] do not get cleared after the calculations of each key byte, the intermediate value $\mathrm{S}(ptxt \oplus k) \oplus r$ is also visible during the processing of the subsequent SubBytes operation, e.g., around samples 2500 to 2800.

As a summary, it can be concluded that the trace segment chosen by the ASCAD authors includes only a part of the whole masked SubBytes operation. This is clearly due to the fact that the authors chose a trade-off between the size of the sample range and therefore the input size of the network, which has an influence on the required training time and the contained information. Nevertheless,

[4]We map the beginning of a correlation peak to the rising edge of a clock cycle, as the highest current change is reflected by a high leakage. We build our mapping on load and store operations that normally exhibit higher leakage for microcontrollers as the data transfer over the bus consumes more power than normal ALU operations. The first operation, namely the load of r_i in line 407, is mapped to the corresponding first correlation peak. The second operation is the store of the final S-box result $\mathrm{S}(k_i \oplus ptxt_i) \oplus r_i$ in line 428.

[5]Namely `r3`, `r24`, `r26` and `r27`, for details, the reader is referred to the assembly implementation.

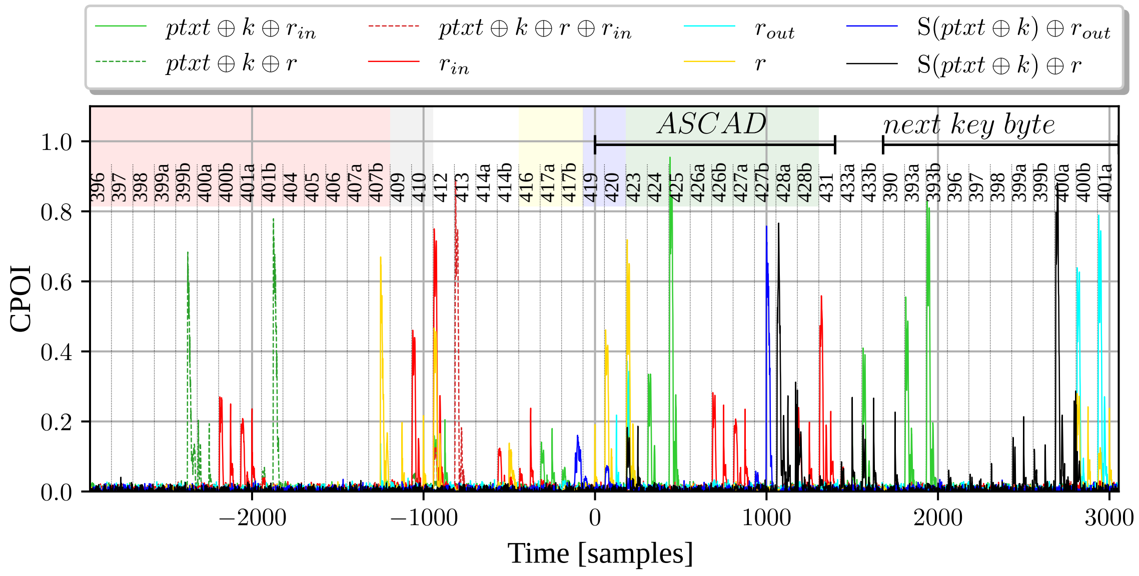


Figure 6.3 CPOI analysis for the entire masked SubBytes operations for key byte k_2 of *ASCAD* VARIABLE (samples 78 000 to 84 000). The indices are aligned such that samples 0-1399 correspond to the trace segment proposed by ASCAD [BPS+20].

the narrow selection of samples omits additional information and leakages that could further be used to improve training and attacks of DL-based SCA.

6.2.2 First/Second-Order CPA Results

One of the main benefits of DL-based SCA is that it reduces the attacker's effort to perform higher-order side-channel attacks. In a classical higher-order CPA, the leakage of the individual shares, e.g., $S(k_i \oplus ptxt_i) \oplus r_i$ and r_i, has to be combined. The individual leakage points are not known in a masked implementation and therefore all combinations of a chosen sample range have to be evaluated. For large numbers of sample points, this requires large computational resources and may even be infeasible. Therefore, a lot of effort and expertise is needed to find a promising small sample range for an attack. With DL-based SCA, the correct combinations are learned during the network training, reducing the attacker's effort. In order to claim this benefit, it has to be verified that there is no first-order or second-order univariate leakage contained in the traces. This could be an easier target for the network since it does not need to learn the correct sample combinations but rather learns directly from this leakage. In addition, the results of classical SCA are also valuable to interpret DL-based attack results for the different key bytes and to perform attribution of learned features by a network.

We, therefore, evaluate both databases regarding first-order and univariate/multivariate second-order CPA. In order to generate the results, the Jlsca toolkit [BK] was used. For all evaluations, we use the HW power model and target the intermediate $S(ptxt \oplus k)$, which corresponds to the intermediate used as training label. For the second-order attacks, we use the normalized (mean-free) product as a preprocessing function to combine the respective samples and perform a CPA on the preprocessed traces. This preprocessing has been shown to be optimal under the assumption of HW leakage [PRB09]. For the first- and second-order univariate attacks, we use all available traces in the datasets, while for the second-order multivariate attack, we limit the amount of attack traces to 10 000. This is due to the vastly increasing computational complexity of multivariate attacks, as all possible sample combinations have to be evaluated. The correlation is evaluated every 20 traces for all bytes except k_0 and k_1, where the correlation is updated every trace. In a first step, we show results for all key bytes using the trace segments described in Section 6.2.1. We conclude with results using the entire sample range of the traces contained in the databases.

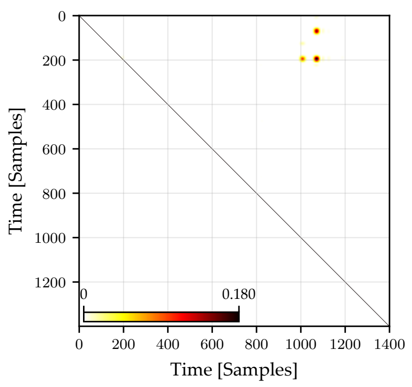
	Order	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}
ASCAD fix	1st	19	12	–	–	–	1960	–	–	–	–	–	–	–	–	–	–
	2nd (uni.)	x	x	5440	2060	4900	3160	4880	9400	5180	2360	2940	5200	8580	7920	1980	2730
	2nd (mult.)	x	x	620	280	540	260	200	480	340	1340	400	460	620	460	240	300
ASCAD variable	1st	10	24	–	–	85 700	1580	–	–	–	–	–	–	–	–	–	–
	2nd (uni.)	x	x	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	2nd (mult.)	x	x	560	640	900	540	880	740	680	960	900	1220	1100	1380	520	660

Table 6.2 First- and second-order CPA results for both ASCAD databases. For each key byte, the number of traces after which a key rank of zero occurs for the first time is given. For entries marked with "–" the correct key could not be found with the provided traces.

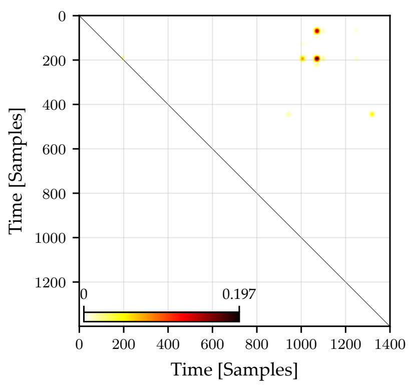
Evaluation of Trace Segments The attack results for the trace segments of the individual key bytes (c.f. Section 6.2.1) are shown in Table 6.2, where we provide the amount of attack traces, after which a key rank of zero first occurs. There are successful first-order attacks for both datasets, which should be unexpected for a first-order secure implementation[6]. Namely, k_5 can be successfully attacked in both datasets and additionally k_4 is retrieved in *ASCAD* VARIABLE. In comparison to the unmasked key bytes k_0 and k_1 where at most 24 traces are sufficient, k_5 requires 1960 (*ASCAD* FIX) and 1580 (*ASCAD* VARIABLE) traces for a successful attack. In contrast, k_4 of *ASCAD* VARIABLE can be attacked with 85 700 traces, which is almost the whole attack set of the database. As *ASCAD* FIX contains only 60 000 measurements, it can be assumed that the CPA is not successful due to the lack of measurements. Note that masked software implementations are known to exhibit security order reductions due to unexpected and therefore unconsidered transitional leakage of the target platform [BGG+15, LCGD18].

For the second-order univariate attack, we show successful attacks on all masked key bytes in *ASCAD* FIX, while no successful attack can be observed for *ASCAD* VARIABLE. A plausible explanation of this difference is that due to differences in the measurement setup or preprocessing of *ASCAD* FIX (see Section 6.2.1), multivariate leakage is transformed to univariate [MM13]. The amount of required attack traces for a successful univariate second-order attack ranges from a minimum of 1980 (k_{14}) up to 9400 (k_7). In the case of a multivariate second-order attack, the results in Table 6.2 show that the chosen amount of 10 000 traces is more than sufficient for a successful attack on all key bytes. As this is expected behavior, we additionally provide an evaluation of the different sample combinations that lead to a successful attack for *ASCAD* VARIABLE. The resulting correlation results for all sample combinations are shown in Fig. 6.4. Note that for visualization, a convolution filter is applied to increase the size of the found sample combination points. Furthermore, correlations of less than $4/\sqrt{10000}$, which is an estimate of a threshold for a significant correlation derived in [MOP07, Section 6.4.2], are depicted in white. We limit the evaluation to key bytes with different leakage characteristics as shown in Fig. 6.2, while the remaining bytes are shown in Appendix C. For all these key bytes, it can be concluded that samples corresponding to r_{out} (around 70) and r_i (200) can be combined with their corresponding masked S-box value (1000 and 1100). For k_{15}, the additional leakage of $\mathrm{S}(k_{15} \oplus ptxt_{15}) \oplus r_{out}$ allows for an additional combination with samples around index 200. Further, key bytes that are vulnerable against first-order CPA (k_4, k_5) show possible additional combinations.

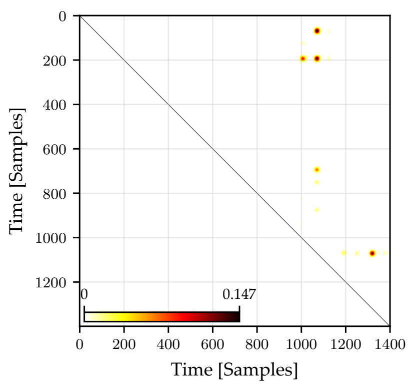
[6]Note that the possibility of a first-order leak has already been discussed in `https://github.com/ANSSI-FR/ASCAD/issues/15`, last accessed 18th April 2024, where a normalization step allows for a sort of second-order univariate attack if observing a particular sample (sample 188) independently. Nevertheless, this does not have an influence in practice, as in a masked setting, an attacker can not perform leakage evaluation to identify this sample directly but rather has to use the maximum correlation among all samples. Additionally, due to the long review cycle of the submission this chapter is based on [EST+22], the first order leak of k_5 has additionally been identified in [KFYF21] for *ASCAD* FIX. Nevertheless, the results are not tailored to the corresponding trace segment identified by the ASCAD authors and only 30 % of the available traces are used for the analysis.



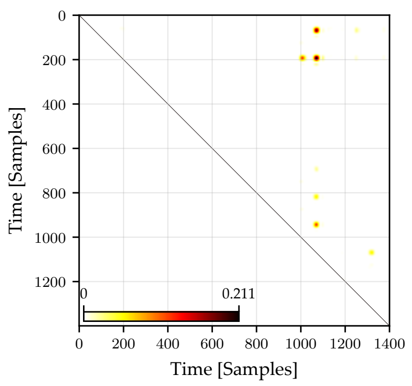
(a) k_2



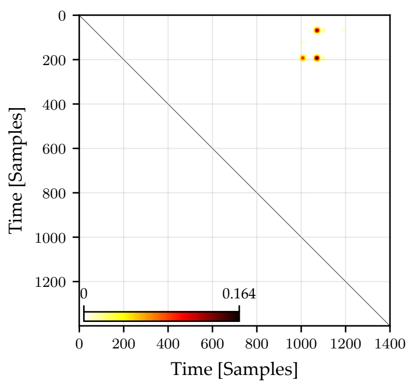
(b) k_3



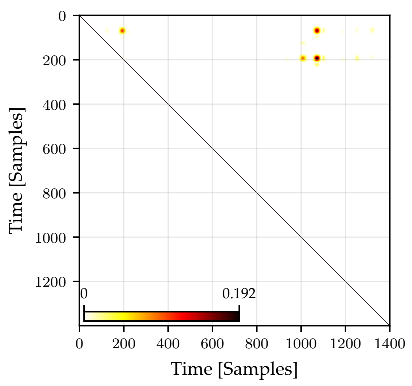
(c) k_4



(d) k_5



(e) k_6



(f) k_{15}

Figure 6.4 Second-order attack results for k_2, k_3, k_4, k_5, k_6 and k_{15} of *ASCAD VARIABLE*. The resulting correlation value for the different sample combinations of the correct key hypothesis is shown.

Evaluation of the Whole Dataset Finally, classical SCA results for the whole sample range of both datasets are shown in Table 6.3. All key bytes of *ASCAD VARIABLE* can successfully be retrieved with a first-order attack, while for *ASCAD FIX* not all key bytes are attackable (k_8, k_{10}, k_{12}, k_{13}, k_{15} are not retrievable). Nevertheless, a univariate second-order attack is still only possible for *ASCAD FIX*.

The possibility of a first-order leak using the whole trace set is worth noticing regarding the interpretation of attack results. Bronchain et al. [BCS21] show a successful attack on all key bytes that is constructed such that first-order leaks cannot be exploited (c.f. [BDMS22] for a detailed discussion). In contrast, the attack methodology of Lu et al. [LZC+21] includes a POI selection step consisting of an encoder before an attention network, making it possible that this method indeed focuses on the first-order leakage. These examples show that a thorough understanding of the underlying dataset is important in order to reason about the capability of a particular architecture or attack approach. In conclusion, the availability of first-order leakage has to be considered in future work in order to understand whether DL-based SCA implements multivariate higher-order attacks using the whole sample range of the ASCAD databases.

6.3 DL-SCA on ASCAD: Impact of Training Scenarios and Varying Key Byte Leakage

In this section, we evaluate the ASCAD datasets regarding DL-based SCA results. First, in Section 6.3.1 we recall the architectures CNN_{best} and $\text{CNN}_{\text{small}}$ that are used in our experiments. Second, in Section 6.3.2 we evaluate the impact of training and attacking on the same fixed key compared to using variable keys during the training process and a fixed key for the attack set. The results show

	Order	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}
ASCAD	1st	14	14	12 960	11 220	11 640	2280	15 240	10 220	–	6980	–	27 580	–	–	34 660	–
fix	2nd (uni.)	x	x	3960	4460	5160	3120	6540	15 560	9820	10 380	6780	6400	12 000	9160	12 840	3100
ASCAD	1st	14	16	17 160	10 900	14 060	2260	7760	22 220	11 720	15 480	13 800	19 360	6120	22 200	12 740	16 520
variable	2nd (uni.)	x	x	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Table 6.3 First- and second-order CPA results using all available samples. For each key byte, the number of traces after which a key rank of zero occurs for the first time is given. For entries marked with "–" the correct key could not be found from the provided traces.

that the former scenario, provided by the *ASCAD* FIX dataset, is an easier task for CNNs. As the latter scenario is more realistic, we provide results for all key bytes of the *ASCAD* VARIABLE dataset in Section 6.3.3. The results show that certain key bytes of the dataset are easier to learn than others. Furthermore, performance differences for different bytes highlight that traces of identical operations on the same dataset pose challenges to CNNs, and therefore research can benefit from using all key bytes of an implementation in order to improve the robustness of results.

6.3.1 Experimental Setup

In the following evaluations, we focus on the difference of the ASCAD datasets regarding the difficulty of training depending on the provided training data and the different bytes of the AES implementation. For a fair comparison of the datasets, we use the ID model variants of the two CNN architectures detailed in Section 5.2.3: The CNN_{best} network by Benadjila et al. [BPS+20] is a relatively large VGG-16-based architecture, whereas hyperparameter optimization based on reinforcement learning by Rijsdijk et al. [RWPP21] provides the smaller $\text{CNN}_{\text{small}}$ architecture. For both models, a five-fold cross-validation is performed for all results, i.e., each model is trained five times with a different split of training and attack set, reducing the effect of initial weights and data selection. The training of CNN_{best} is stopped after 100 epochs while the training for $\text{CNN}_{\text{small}}$ is stopped after 50 epochs.

6.3.2 Fixed Key vs. Variable Key Training

A general question regarding DL-based SCA results is how well the models generalize. On the one hand, hyperparameters such as the number of epochs in the training process can have an effect on the results and the models may overfit, i.e., only learn the specific relation between features and labels in the training dataset. On the other hand, the generalization capabilities are inherently limited by the provided training data. The major drawback of the *ASCAD* FIX dataset is that it uses the exact same key during the training and the attack phase. First, the scenario is unrealistic as an attacker does not know the correct key and can hence not train the model with the same key that is later used for the attack. Second, the question arises whether models benefit from the tailored training set, i.e., whether the results obtained from the *ASCAD* FIX dataset overestimate the abilities of DL-based approaches.

To answer the question, we compare two settings: A) training and attacking are done on traces that use the same key, and B) training is done on traces with variable key values and the attack is carried out on traces with the same key. Setting A corresponds to the use of the *ASCAD* FIX dataset, which only contains traces with the same fixed key. Setting B is targeted by the *ASCAD* VARIABLE dataset, which contains traces with variable keys for training and a fixed key for the attack. In order to have a fair comparison for both settings on the same dataset, we additionally perform training on the fixed key traces from the *ASCAD* VARIABLE dataset.

Fixed Key Training on *ASCAD* FIX To provide a baseline for the fixed key training with CNN_{best} on *ASCAD* VARIABLE, we perform training and attack on the *ASCAD* FIX dataset. With this reference, we can later determine whether the CNN_{best} architecture with the small changes proposed by the ASCAD authors (c.f. Section 5.2.3) performs better or worse on *ASCAD* VARIABLE. For all training processes, we use 48 000 traces for training and validation with a validation split ratio of 1/8, i.e., 42 000 training and 6000 validation traces. For the attack, we use 12 000 traces split into 12 subsets of 1000 traces, for which the attack is carried out separately. Hence, considering the five-fold cross-validation 60 attack sets are available. For CNN$_{\text{small}}$, there exists an optimized version for *ASCAD VARIABLE*, i.e., we do not need the baseline.

Fixed Key Training on *ASCAD VARIABLE* In order to emulate training and attacking on the same key, we use a part of the traces with a fixed key value in the *ASCAD VARIABLE* dataset for the training and validation. As the results are generated on the *ASCAD VARIABLE* dataset, a fair comparison with the setting of a variable key training is possible.

For the CNN$_{\text{best}}$ network, the training and validation uses 48 000 traces and the attack set uses 12 000 traces from the remaining 52 000 traces. We restrict the numbers to be identical with the *ASCAD FIX* setting for a fair comparison. Similarly, we use 1000 traces for the attack, resulting in 60 attack sets. For the CNN$_{\text{small}}$ network, we use 75 000 traces for training and 5000 traces for validation, and 20 000 attack traces resulting in 100 attack sets in total. We use more traces for training, as the model architecture has been evaluated with 100 000 traces during the reinforcement learning and therefore tends to produce worse results with fewer traces. Nevertheless, the fixed key scenario imposes a total limit of 100 000 traces, including training and attack set.

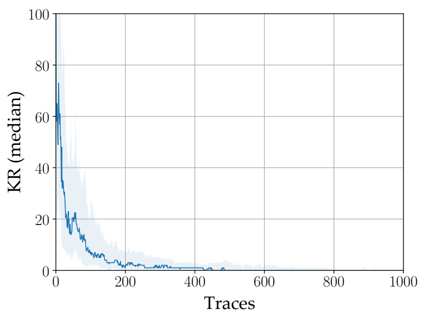
Variable Key Training on *ASCAD VARIABLE* Finally, the training on variable keys for the *ASCAD VARIABLE* dataset is carried out to compare with the fixed key training. We use 48 000 traces for training and validation, but this time with variable keys, and 12 000 traces from the fixed key traces for the attack. The attack traces are split into subsets of 1000 traces, i.e., 60 sets in total can be compared. For the CNN$_{\text{small}}$ network, we use 75 000 traces for training and 5000 traces for validation, and 20 000 attack traces resulting in 100 attack sets in total. Although the training set size could be increased in this scenario, we use the same size as for the fixed key training to keep the results comparable.

Comparison of Results The analysis is carried out on key byte k_2. We calculate the key rank for an increasing number of traces and monitor the evolution after each trace. The performance of the trained models is depicted in Fig. 6.5 where the median key rank from the available attack sets is plotted against the number of evaluated traces.

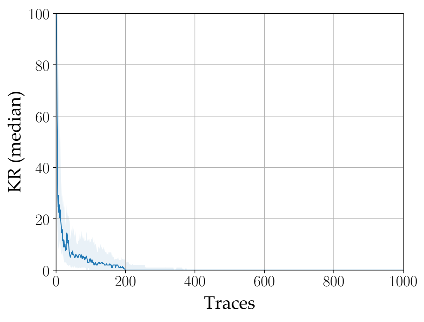
The results for the *ASCAD FIX* dataset and CNN$_{\text{best}}$ in Fig. 6.5a are in line with Benadjila et al. [BPS+20], where approximately 500 traces were needed to reach a key rank of zero using the same model. The median key rank reaches zero at 428 traces, i.e., rank zero is reached for half of the attack sets with less than 428 traces and for the other half with more than 428 traces. The quantiles from 0.25 to 0.75, which are visualized as filled area show that 50% of the attack sets reach a key rank of zero between 147 and around 812 traces. From the 60 attacks sets, 12 do not reach a median key rank of zero with the provided 1000 traces.

The fixed key training with CNN$_{\text{best}}$ on *ASCAD VARIABLE* in Fig. 6.5b reaches a median key rank of zero with 200 traces in half of the cases and the majority of attacks succeed with less than 341 traces. Comparing the results for the fixed key training in Figs. 6.5a and 6.5b, CNN$_{\text{best}}$ requires fewer traces for a successful attack with the *ASCAD VARIABLE* dataset than with the *ASCAD FIX* dataset. Hence, the architecture optimized on *ASCAD FIX* can also be used on *ASCAD VARIABLE* with the proposed small changes like the size of the input layer (c.f. Section 5.2.3) and we use CNN$_{\text{best}}$ to compare the fixed key training against the variable key training on *ASCAD VARIABLE*. A dedicated hyperparameter search for the *ASCAD VARIABLE* dataset could yield even better results. However, as we are mainly concerned with the comparison of the effect of training sets, this is beyond the scope of this thesis. The use of an optimized network is covered by the results generated with CNN$_{\text{small}}$.

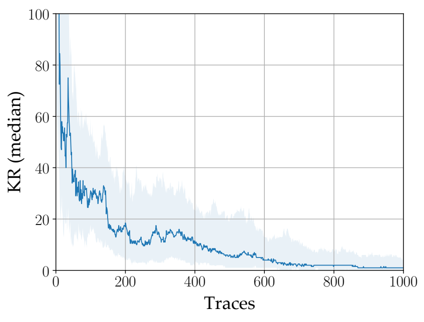
Finally, in Fig. 6.5c, the training of CNN$_{\text{best}}$ on variable keys does not achieve perfect key recovery for more than half of the attacks (37 out of 60) even after 1000 traces. Only a quarter of the attack sets,



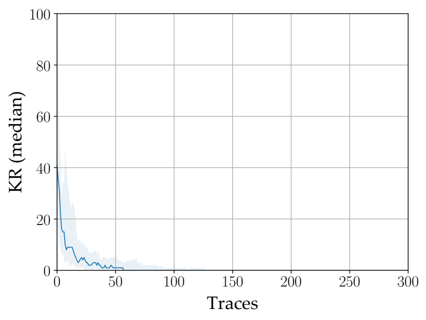
(a) CNN$_{\text{best}}$: *ASCAD FIX*



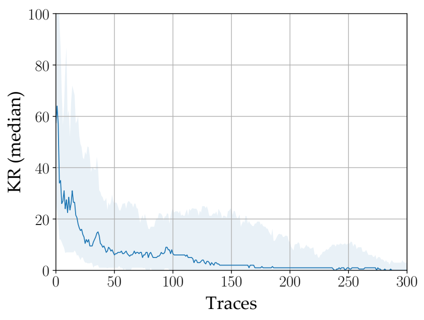
(b) CNN$_{\text{best}}$: Fixed key



(c) CNN$_{\text{best}}$: Variable key



(d) CNN$_{\text{small}}$: Fixed key



(e) CNN$_{\text{small}}$: Variable key

Figure 6.5 Median key rank for different training scenarios for key byte k_2: (a) *ASCAD FIX*, (b)-(e) *ASCAD VARIABLE*. The blue area represents the results between the 0.25 and 0.75 quantiles, i.e., it contains 50% of all results.

corresponding to the 0.25 quantile, can be attacked with less than 638 traces. Compared to the fixed key training in Fig. 6.5b, more than five times as many traces are required for a successful attack.

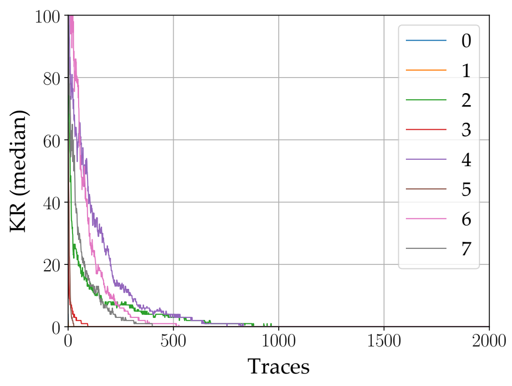
The results for the different training conditions for CNN$_{\text{small}}$ in Figs. 6.5d and 6.5e show similar results. While the optimized hyperparameter architecture achieves key recovery with fewer traces than CNN$_{\text{best}}$, the number of traces to reach a median key rank of zero is about four times higher with variable key training: The training on the same fixed key as during the attack in Fig. 6.5d requires 58 traces for a successful attack. On the other hand, the training on variable keys leads to a median key rank of zero after 239 traces, and for 20 out of 100 attack sets the key byte can not be recovered with 1000 traces.

Summary We compared the results for fixed key and variable key training on the *ASCAD VARIABLE* dataset using two different network architectures. For both networks, successful attacks require at least four times as many traces when training on a variable key compared to training on the same fixed key as used during the attack. The results suggest that the performance of networks based on fixed key training is overestimated. Consequently, as the *ASCAD FIX* dataset only contains measurements from a single key and shows unexpected univariate second-order leakage as detailed in Section 6.2.2, future research should rely on the *ASCAD VARIABLE* dataset with variable training keys. We emphasize that the results are valid for the ASCAD databases, and based on the available data, we made the best effort to support our claim. Nevertheless, training on a random key may not be more difficult in every scenario, especially if the whole AES execution is considered. Measurements with a variable key could contain additional information on, e.g., the key transfer and the key schedule that could be exploited by the models.

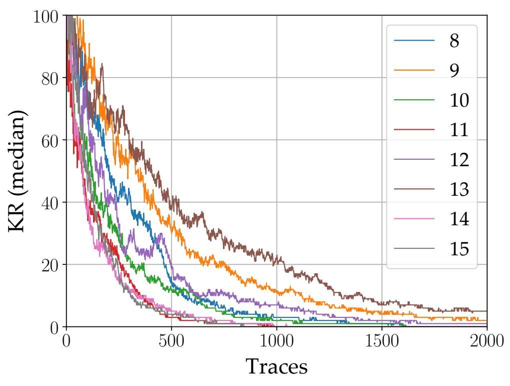
6.3.3 Training on Different Key Bytes

Following the differences in the side-channel leakage for different bytes revealed in Section 6.2, we evaluate how networks perform when training on different key bytes of the *ASCAD* VARIABLE dataset. The expectation that unmasked bytes k_0 and k_1 and bytes k_4 and k_5, which exhibit first-order leaks are easier targets leads to the following question: *1. Are certain key bytes easier to attack independent of the network architecture?* In Section 6.2, we showed that the leakage differs between bytes. Therefore, hyperparameter search may overfit on the leakage behavior of the byte used during the search, and the network may not be able to generalize to other bytes even from the same implementation, which leads to the question: *2. Do networks perform equally well on key bytes other than the one used for the hyperparameter search?* The first question addresses the properties of the dataset, while the second directs towards the generalization capabilities of networks.

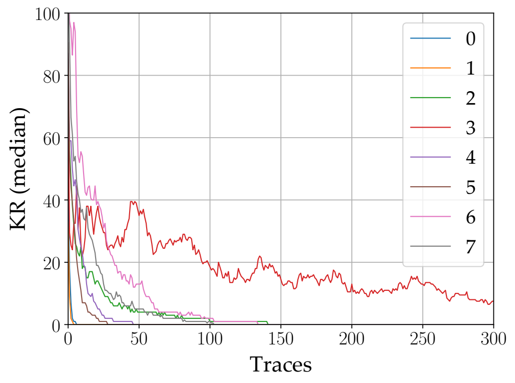
As from Section 6.3.2 a training on variable keys provides the more realistic assessment of the capabilities of the deep-learning models, we use the *ASCAD* VARIABLE dataset for the analysis. For training and validation, we use 110 000 traces with variable keys, with a validation split ratio of 1/11, resulting in 100 000 training and 10 000 validation traces. The two architectures $\mathrm{CNN}_{\mathrm{best}}$ and $\mathrm{CNN}_{\mathrm{small}}$ are trained with traces from a key byte k_i and traces from the same key byte are used to evaluate the attacks. All 100 000 traces with the fixed key are split up into subsets of 2000 traces, i.e., 50 attack sets are tested, yielding a total of 250 attack sets taking into account the five-fold cross-validation.



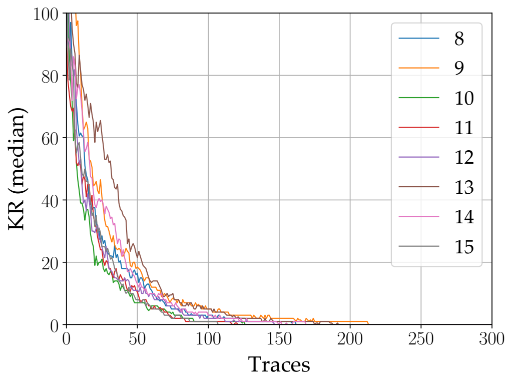
(a) $\mathrm{CNN}_{\mathrm{best}}$: Bytes k_0-k_7



(b) $\mathrm{CNN}_{\mathrm{best}}$: Bytes k_8-k_{15}



(c) $\mathrm{CNN}_{\mathrm{small}}$: Bytes k_0-k_7



(d) $\mathrm{CNN}_{\mathrm{small}}$: Bytes k_8-k_{15}

Figure 6.6 Median key rank of 250 attacks for all key bytes of *ASCAD* VARIABLE.

In Fig. 6.6, the median key rank of all 250 attacks is depicted for the different bytes and networks $\mathrm{CNN}_{\mathrm{best}}$ and $\mathrm{CNN}_{\mathrm{small}}$. Table 6.4 provides further details in terms of the number of traces required to reach a median key rank of zero. For bytes that require more than 2000 traces to converge to zero, we repeated the attacks with 20 attack sets of 5000 traces to provide numbers for the comparison.

	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}
CNN$_{\text{best}}$	2	2	875	95	755	28	517	401	1598	2945[a]	1597	931	2265[a]	2966[a]	966	830
CNN$_{\text{small}}$	7	5	142	1210	47	29	135	104	163	214	127	118	148	176	151	108

a) Results did not converge to zero with 2000 traces. Therefore, given results are generated using 20 attack sets of 5000 traces, corresponding to a median key rank obtained from 100 attacks compared to 250 attacks for other bytes.

Table 6.4 Traces to reach median key rank zero for all bytes of *ASCAD variable*.

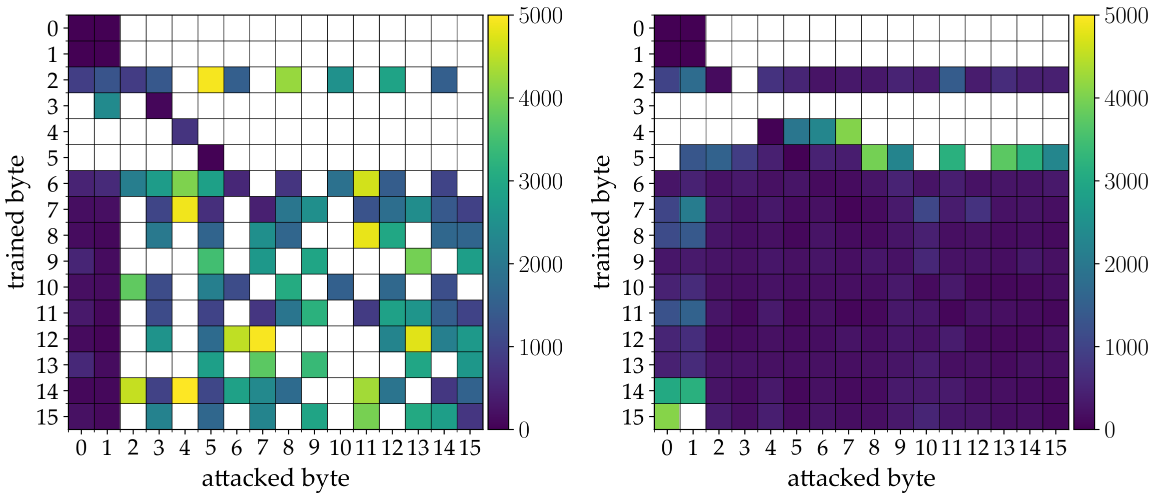
Consequently, the median key ranks are based on a total of 100 attack sets instead of 250 attacks for other bytes.

Regarding the question of whether specific key bytes are easier or harder to attack for both models, a part of the expectation is confirmed by Table 6.4 and Fig. 6.6. The key bytes k_0, k_1, and k_5 reach a median key rank of zero with less than 30 traces compared to more than 100 (CNN$_{\text{small}}$) and more than 400 traces (CNN$_{\text{best}}$) for most other bytes. For the unmasked bytes k_0 and k_1, two traces suffice to recover the correct key with CNN$_{\text{best}}$, which is in line with previous work that reported similar findings on the *ASCAD fix* dataset [ZS20]. The CNN$_{\text{small}}$ network is also able to break k_0 and k_1 but requires 7 and 5 traces, respectively. The fact that k_5 is recovered with fewer traces than other bytes with both networks can be explained by the first-order leak discussed in Section 6.2.2. Either the network directly exploits the first-order leakage, or the leakage at least simplifies the classification task. In either case, as the CPA requires more traces for k_5 compared to the unmasked bytes k_0 and k_1, it is plausible that the network also requires more traces for a successful attack. Key bytes k_9 and k_{13} require more traces compared to the other bytes for both architectures. Interestingly, both bytes are also among the bytes that require most traces for a second-order multivariate CPA in Table 6.2. However, the results for CNN$_{\text{small}}$ vary in a narrow range, i.e., establishing a link between CPA and deep-learning approaches requires further research.

With respect to the question of whether the same network performs equally well among all bytes, it has to be noted that both networks are capable of retrieving all 16 key bytes. Apart from the bytes k_0, k_1, k_3 and k_5 that require less than 100 traces for CNN$_{\text{best}}$, three groups of bytes can be identified in the first row of Table 6.4 and Figs. 6.6a and 6.6b: k_9, k_{12}, k_{13} with more than 2000 traces, k_8 and k_{10} with about 1600 traces, and the remaining seven bytes with 400 to 970 traces. The second row of Table 6.4 and Figs. 6.6c and 6.6d shows leveled results among the different bytes for CNN$_{\text{small}}$: apart from k_0, k_1, k_4 and k_5 with less than 50 traces, most bytes require 100 to 150 traces. The existence of only a few moderate outliers (k_8, k_9, k_{13}) indicates that the network CNN$_{\text{small}}$ performs equally well for most key bytes. The only exception is k_3, which converges to a key rank of zero with more than 1200 traces. A possible explanation is that the hyperparameters of CNN$_{\text{small}}$ are highly optimized for the leakage pattern of k_2, which occurs in most bytes. Therefore, results for CNN$_{\text{small}}$ are constant across different bytes, except that the architecture does not capture the distinct leakage pattern of k_3.

Notably, for key byte k_4, which exhibits a first-order leak with more traces than k_5, CNN$_{\text{small}}$ requires fewer traces than for other bytes, while CNN$_{\text{best}}$ does not show an improved performance on k_4. On the other hand, CNN$_{\text{best}}$ seems to benefit from the different leakage pattern of k_3 shown in Fig. 6.2. A possible explanation is that CNN$_{\text{best}}$ is oversized, i.e., its internal filters do not focus on single events such as the first-order leakage pattern from k_4. Instead, information from multiple points is aggregated, but as the hyperparameters are not highly optimized (as for CNN$_{\text{small}}$), differing leakage as from r_{in} of k_3 can be exploited.

Summing up, k_3 and k_4 are not inherently easier to attack, but the network architecture used for the attacks plays an important role. This shows that the byte used for hyperparameter search exhibits an influence depending on the architecture.



(a) CNN$_{\text{best}}$ **(b)** CNN$_{\text{small}}$

Figure 6.7 Cross-byte analysis on *ASCAD VARIABLE* for CNN$_{\text{best}}$ and CNN$_{\text{small}}$. The color bar represents the traces required to reach a median key rank of zero. Results, where 5000 traces do not suffice to recover the key, are depicted in white.

6.3.4 Cross-Byte Analysis

We conclude the training evaluation by performing attacks on key bytes that are different from the one the network is trained on. The aim of this *cross-byte analysis* is to establish the influence of the different leakage patterns from Fig. 6.2 on the training process. The 100 000 traces with attack traces from *ASCAD VARIABLE* are split into 20 attack sets of 5000 traces resulting in 100 attacks considering the five-fold cross-validation. The results of all possible training and attack combinations for both networks are shown in Fig. 6.7 in terms of the number of traces to reach a median key rank of zero. Combinations for which 5000 traces do not suffice to recover the key are depicted in white. Note that the diagonals consist of the required traces for training and attacking on the same byte from Table 6.4.

As expected, the first two rows in Figs. 6.7a and 6.7b show networks trained on trace segments of the unmasked bytes k_0 and k_1 are not able to successfully attack trace segments from the masked key bytes. Networks trained on bytes that show a different leakage characteristic compared to the majority of other bytes, namely k_3, k_4, k_5 (c.f. Fig. 6.2), tend to only perform well on attacking the byte they are trained for. This highlights that the different leakage characteristics (and possibly intermediate values) are targeted by the CNNs and that trained networks do not easily generalize for attacks on all bytes of the *ASCAD VARIABLE* database. Finally, for CNN$_{\text{small}}$, the key bytes that show a similar leakage behavior in Fig. 6.2 (k_2, k_6-k_{14}) perform reasonably well among each other, while for CNN$_{\text{best}}$ results vary.

To conclude, attack results vary between bytes and trained networks can not easily be used for attacks on all bytes of the *ASCAD VARIABLE* database. This implies that an attacker has to train a separate model for each leakage, i.e., results from a single byte may not be representative. It has to be emphasized that this is not a specific behavior of DL-based SCA but is also observable in classical template attacks. Nevertheless, our results show the extent of this difference and, as most work is done on k_2, results on this byte might be implicitly expected for attacks on other bytes. Alternatively, in order to improve generalization, training on a mixture of leakages from different bytes, which has been shown to be effective for cross-device attacks [BCH^{+}20], could be applied. Other more sophisticated methods used in cross-device learning, like Domain Adaption [CZLG21] or Meta-Transfer Learning [YSPJ21], should also be evaluated in this context. We conclude that cross-byte analysis can serve as a first step to assess the robustness of CNN architectures on the same dataset.

Summary We showed that results of DL-based SCA using CNNs differ for the distinct key bytes of the *ASCAD variable* dataset. For some bytes, the results are consistent for different networks, i.e., they are easier (k_0, k_1, k_5) or more difficult (k_9, k_{13}) to learn. Results coincide with classical SCA in Section 6.2, which highlights that an understanding of the datasets is beneficial to understand and compare DL-based SCA results. The performance differences with the same network architecture for different bytes highlight that traces of identical operations on the same dataset pose challenges to CNNs. The highly optimized architecture $\text{CNN}_{\text{small}}$ shows constant results across different bytes, except for the distinct leakage pattern of k_3. With a large architecture such as CNN_{best}, results vary across different bytes. Hence, we emphasize that in addition to results from different datasets, research can benefit from using the entire ASCAD dataset to test whether models are robust under varying leakage conditions.

6.4 Conclusion

In this chapter, we provided a second look at the ASCAD databases both in terms of classical SCA and regarding the difficulty of CNN-based attacks. In contrast to most related work, we considered all bytes of the implementation. The CPOI analysis demonstrated a contribution of different intermediate values to the leakage, which has not been considered yet. Provided CPA results revealed first-order and univariate second-order leakage for several key bytes that are unexpected for a masked implementation. These results, along with the matching of implementation and observed leakage, can be used to reason about a network's learning behavior.

Subsequently, we showed that training CNNs on the same fixed key as used for the attack yields significantly better results than training on variable keys for the ASCAD databases. Hence, the *ASCAD fix* dataset can be seen as the best-case scenario from an attacker's point of view given the available ASCAD databases. In order to evaluate a more realistic attack, future work should be based on the *ASCAD variable* dataset and a variable key training. Finally, we compared CNN attack results for all bytes of the *ASCAD variable* dataset, which showed that differing leakage poses challenges to CNNs depending on the used architecture. In addition to comparing results on different datasets, the different key bytes of the *ASCAD variable* dataset can be leveraged by future work to improve the robustness of results.

As a final remark, it has to be stressed that the ASCAD databases provide a valuable contribution towards openly available datasets that are crucial for comparable DL-based SCA results. The presented results show that apart from the research on suitable deep-learning architectures, a thorough analysis of the underlying datasets is required to interpret attack results.

7 Occlusion Techniques for DL-SCA Attribution

This chapter is based on the publication *Schamberger/Egger/Tebelmann: Hide and Seek: Using Occlusion Techniques for Side-Channel Leakage Attribution in CNNs* published in *Applied Cryptography and Network Security Workshop (ACNS)*, 2023 [SET23]. The idea of showing occlusion results as waterfall plots was first identified in the master thesis of Florian Lippert [Lip20] that I supervised. The framework for the evaluation of the different occlusion methods was developed by Maximilian Egger and has been revised by me in order to generate the shown results in this chapter. The results in Fig. 7.1 have been generated using the *DeepExplain* framework [ACÖG19a].

7.1 Introduction

In Chapter 6, an evaluation of the ASCAD databases is presented that highlights the downside of the relaxed feature engineering effort of DL-based SCA in contrast to classical profiled side-channel analysis approaches, which lies in the reduced explainability of attack results. The identified additional leakage of intermediates or the first- and second-order leaks can be exploited by the networks and therefore an attacker does not know which leakage is used to defeat the countermeasures of an implementation (see Section 5.1.5 for a discussion). The work that is presented in this chapter tries to solve this problem by developing and evaluating neural network *attribution* methods that can be used to reason about the learned features/samples of a measurement trace during the training process. This knowledge provides valuable insights for designers of masked implementations on how to fix weaknesses in their implementations by comparison with known leakage of intermediates. Additionally, it can also be used to reason about performance differences between classical attacks and DL-based approaches.

In particular, this chapter proposes occlusion techniques, which work by removing (occluding) some parts of the input and comparing the result before and after the occlusion. If the achievable attack result is worse after the occlusion it can be reasoned that the occluded features or sample range is used by the network. In contrast to established gradient-based attribution methods, the proposed occlusion techniques allow a direct interpretation of results. The developed techniques are evaluated for both ASCAD databases, and occlusion parameters that are suitable in the side-channel context are identified. In this context, it is shown that due to side-channel measurement characteristics, multiple adjacent samples have to be occluded simultaneously, which has not been considered in related work. In addition, with the presented higher-order occlusion, we are able to identify leakage combinations that are exploited by a network in order to mount a higher-order attack. Finally, the methods are used to show that networks actually utilize varying leakage characteristics shown in Chapter 6 that are observable for different key bytes of the ASCAD databases. The results show that occlusion is a viable addition to established gradient-based attribution methods.

Related Work When looking at the explainability of neural networks in the side-channel context, there are two different directions. First, the question of *"What is the influence of the different elements, e.g., a single node in a specific layer, on the training outcome?"* can be answered. Wu et al. [WWJ+23] use a method called ablation, where they successively remove parts of the network in order to identify which parts of the network are needed to defeat common hiding countermeasures. A second work in this direction is shown by Perin et al. [PWP22b]. Their approach uses an information-theoretic metric to reason about which layer learns a specific intermediate of a masked implementation.

The second explainability direction is the question of neural network *attribution*, i.e., which input features are learned by the network and therefore are significant for the inference process and hence the classification result. In this category, the majority of techniques are gradient-based visualization approaches that utilize a single forward and backward pass through the network. The basic method is a so-called *saliency map* [SVZ14] that performs attribution by computing the absolute partial derivative of an output class with respect to the input features. An improvement of this method, which takes the signed derivative and multiplies it with the input feature, is called *Gradient*Input* [SGSK16]. A more advanced method called *Layer-wise Relevance Backpropagation (ϵ-LRP)* is proposed by Bach et al. [BBM+15]. It is computed with a backward pass on the network by using a propagation rule that maps the found class relevance of a given layer onto the previous one until the input layer is reached. Ancona et al. [ACÖG19b] provide an in-depth discussion about these gradient-based methods leading to their attribution framework called *DeepExplain* [ACÖG19a], which can be readily integrated into common machine learning frameworks. In the side-channel context, several of the presented methods or their variants have been discussed. Masure et al. [MDP19] propose a method similar to Gradient*Input, while others deviate from the general ML literature like Timon [Tim19] who uses partial derivatives of the first layer network weights and Zaid et al. [ZBHV19] who propose a method based on weight visualization. Perin et al. [PEC19] use a variant of ϵ-LRP. Hettwer et al. [HGG20] extensively use the DeepExplain framework to analyze different publicly available datasets, including ASCAD, and conclude that ϵ-LRP is the most appropriate method. Wouters et al. [WAGP20] use the Gradient*Input method of DeepExplain to reason about the required filter sizes to defeat the misalignment of traces.

A second class of attribution methods can be grouped as *perturbation methods*. These methods work by altering or removing (occluding) some parts of the input with the goal of comparing the results after running a second forward pass on the network. The first occlusion method has been proposed by Zeiler and Fergus [ZF14] in the context of image classification with CNNs where they use a gray patch occluding parts of the image. Prediction Difference Analysis [ZCAW17] improves this method and identifies that a multivariate analysis, e.g., modifying multiple pixels at once, has a significant influence on explainability. Occlusion methods allow for straightforward interpretation and are applicable to any model architecture. However, these methods suffer from large or infeasible computation times for large input feature sizes. Additionally, the number of simultaneously occluded features, as well as the occlusion method, significantly influence the results [ACÖG19b]. Regarding DL-based SCA, occlusion methods have been studied by Hettwer et al. [HGG20], who additionally evaluate the integrated 1-occlusion method of DeepExplain on their datasets. After the publication of the occlusion methods in this chapter, Yap et al. [YBP23] published an occlusion-based method called *Key Guessing Occlusion* that is able to determine one variant of a minimal set of samples for which an attack is still possible. They perform multiple rounds of iteratively occluding single samples with the value of zero and observe the resulting attack result while determining occluded samples for which the occlusion does not result in an unsuccessful attack as irrelevant. Only relevant samples are used for the next iteration until the occlusion results determine all remaining samples as relevant for the attack's success. Their results for k_2 are in accordance with the attribution results shown in this chapter.

Finally, Yap et al. [YBBP23] explore a different direction with the utilization of an architecture called *Truth Table Deep Convolutional Neural Network*, which is built for interpretability. Using this architecture, the authors can identify regions of POIs that are used for an attack. Nevertheless, the attribution results are only valid for this specific architecture and can not be generalized.

Contribution This chapter investigates advanced occlusion techniques for the leakage attribution of CNNs that are used for SCA. It has to be noted that although the developed methods are only evaluated using CNNs, occlusion methods are inherently model-agnostic [ACÖG19b] and therefore the methods should, in theory, be applicable to other network architectures frequently used in SCA, e.g., MLPs. Nevertheless, evaluating other architectures is considered out of scope of this thesis and is left for future work. In particular, the contributions presented in this chapter are as follows:

- We introduce *n-occlusion*, which allows for identifying POIs based on a trained CNN. By adapting the number n of occluded samples, the relevant samples are narrowed down. Compared to 1-occlusion from prior work [HGG20], our approach reliably identifies leakage that is spread across multiple samples, which is usually observed in side-channel measurements.
- We investigate the influence of the occlusion method, i.e., how samples are exchanged, and show that the *gray box method* [ZF14] widely used in image recognition is not the best choice for time series signals like side-channel measurements.
- We propose *higher-order occlusion* that occludes multiple parts of a trace and thus enables the identification of POI combinations for higher-order SCA attacks.
- Finally, we employ our occlusion methods to analyze two different network architectures trained on both ASCAD datasets. A comparison with a leakage analysis allows to foster a better understanding of what CNNs actually deem relevant in these side-channel measurements. Our results show that, especially for other key bytes of the ASCAD databases than the proposed third key byte of the ASCAD authors, the networks infer information from the additional leakage identified in Section 6.2.1.

Outline In Section 7.2, gradient-based attribution methods are revisited, highlighting their attribution difficulties for the ASCAD databases. This is followed by an introduction of the n-occlusion method as well as its higher-order variant in Section 7.3. Within this section, suitable occlusion parameters for both databases are evaluated, leading to a successful evaluation of the chosen trace segment by the ASCAD authors. Using the established occlusion parameters, Section 7.4 provides an evaluation of two additional key bytes for which occlusion shows that the networks exploit the unique leakage of these bytes. Finally, Section 7.5 concludes this chapter.

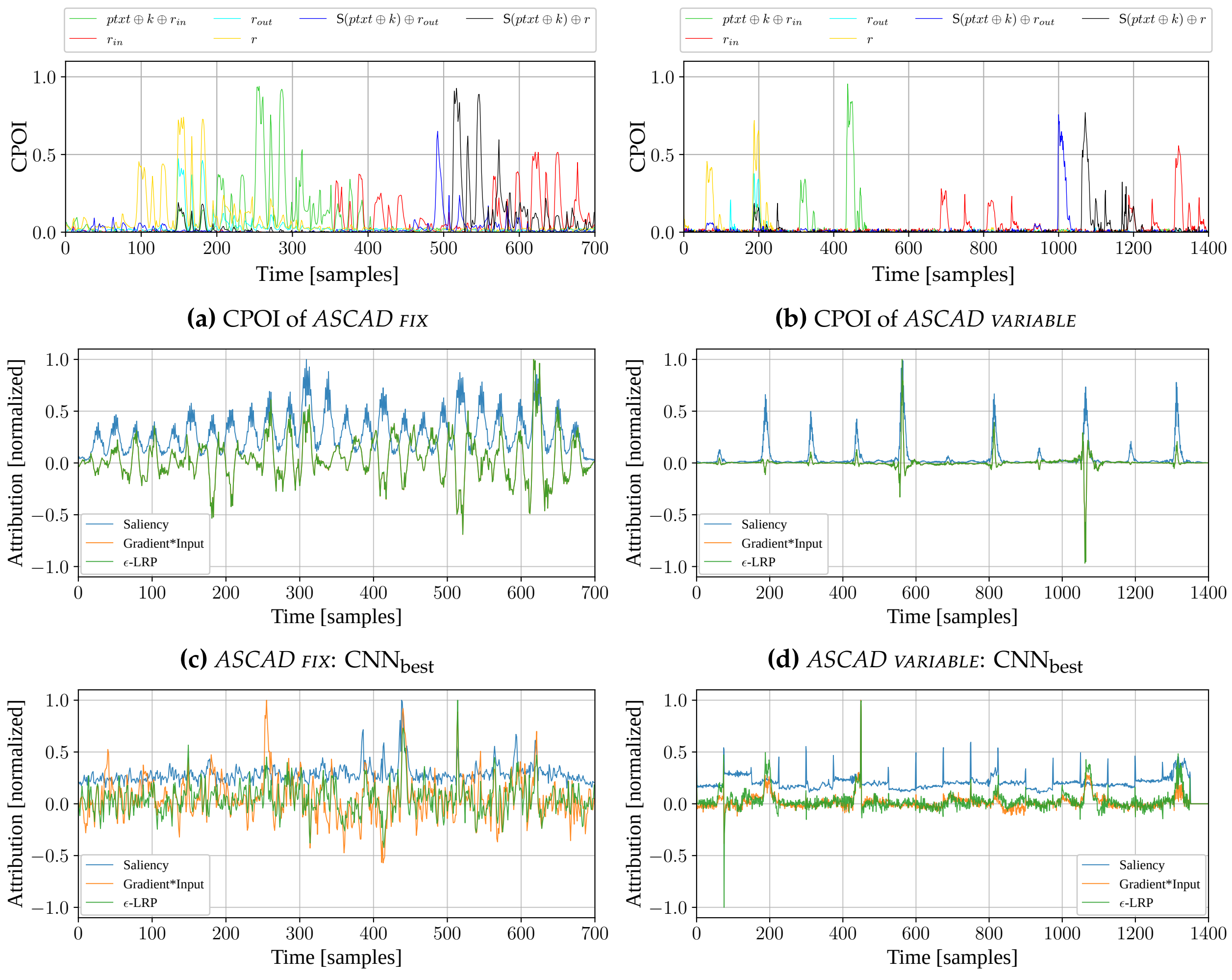
7.2 Revisiting Gradient-based Attribution for ASCAD

In this section, we revisit gradient-based attribution methods for both ASCAD datasets and chosen network architectures (c.f. Section 5.2.3). We use the DeepExplain framework [ACÖG19a] to generate the so-called attribution maps using saliency maps, Gradient*Input, and the ϵ-LRP method after each training epoch. We choose these attribution methods since they correspond to methods used in related work, while saliency maps and the more general Gradient*Input are similar to self-made gradient-based methods that are not established Machine Learning (ML)-based approaches as, e.g., published by Masure et al. [MDP19]. Attribution maps generate an attribution value for each sample point of the used attack traces averaged over the amount of used traces, which we denote as N_{attr}. We additionally normalize the attribution values for each method individually by scaling them relative to their absolute maximum.

The resulting attributions are shown in Fig. 7.1 given a training of 100 epochs on k_2 using the ID model and $N_{attr} = 1000$ traces. For *ASCAD FIX* and *ASCAD VARIABLE*, a total of 48 000 and 100 000 training traces are used, respectively. A direct comparison with a leakage evaluation of intermediates from Section 6.2.1 leads to the following observations:

- Gradient-based attribution does not reliably identify POIs in the setting of related works [MDP19, HGG20], which evaluate their methods on *ASCAD FIX*[1] corresponding to Fig. 7.1c. The resulting

[1]Note that both authors use modified networks for their evaluation. Masure et al. [MDP19] use a network close to CNN_{best} with fewer neurons in the fully connected layers and an additional global pooling layer. Hettwer et al. [HGG20] use their own architecture in order to facilitate an evaluation of multiple datasets.



(a) CPOI of *ASCAD fix* **(b)** CPOI of *ASCAD variable*

(c) *ASCAD fix*: CNN$_{\text{best}}$ **(d)** *ASCAD variable*: CNN$_{\text{best}}$

(e) *ASCAD fix*: CNN$_{\text{small}}$ **(f)** *ASCAD variable*: CNN$_{\text{small}}$

Figure 7.1 Gradient-based attribution results for k_2 in comparison with a leakage evaluation of intermediates from Section 6.2.1. The attribution is performed after 100 training epochs.

attribution indicates several peaks that can hardly be distinguished. In [HGG20, Fig.1], a similar behavior is shown, while in [MDP19], the authors only achieve reliable results for an evaluation at a specific training epoch found with early stopping.

- The methods indicate multiple POIs, which makes it difficult to identify single leakages, in combination with their respective samples, that are essential for the attack success. This ambiguity additionally prevents the identification of sample combinations that are needed to defeat the masking countermeasure of the attacked implementation. This characteristic can be seen for both networks trained on *ASCAD variable* in Figs. 7.1d and 7.1f with the extreme case of using a saliency map for CNN$_{\text{best}}$ shown in Fig. 7.1d.

- The different methods indicate divergent POIs, which is particularly visible in Fig. 7.1e, where each method generates a different result.

Overall, the experiments show that gradient-based attribution methods generate results that are hard to interpret. In addition, there is the problem of choosing the correct training epoch for the attribution due to possible overfitting of the networks, as also discussed in [MDP19]. For our results, we chose to perform attribution after the network training has finished (100 epochs). Additionally, analyzing the results after each epoch did not improve the attribution insights significantly.

7.3 Improvements to the Occlusion Technique

As shown in Section 7.2, gradient-based attribution methods provide results that can be hard to interpret. We, therefore, emphasize the importance of attribution methods that allow for a direct and straightforward interpretation. This property is targeted by the occlusion method, where certain features of the input are removed (occluded) to directly observe their impact on the neural network output. The occlusion method was introduced by Zeiler et al. [ZF14] to determine important pixel ranges for image recognition. However, the amount of occluded features as well as the occlusion method have a significant influence on the attribution results [ACÖG19b]. This limitation has not been investigated in the SCA context. Related work by Hettwer et al. [HGG20] only shows occlusion results by replacing a single measurement sample with the value zero. As side-channel leakage is usually not limited to a single sample, we provide an in-depth evaluation of several occlusion methods in this work. All results for the remainder of this chapter are generated with the respective networks targeting k_2 (unless otherwise noted) using 100 training epochs with 48 000 traces for *ASCAD FIX* and 100 000 traces for *ASCAD VARIABLE*.

In Section 7.3.1, we introduce *n-occlusion* that hides several samples at once. We show that the occlusion technique can leverage the accumulated probabilities for multiple predictions, which closely reflects the underlying SCA problem, where single observations usually do not provide sufficient attack results. In a second step, we explore different occlusion techniques and the impact of the occlusion window size in Section 7.3.2. Additionally, attribution results for selected datasets and architectures are shown. Finally, in Section 7.3.3, we extend the method to higher-order occlusion where multiple non-consecutive parts of a trace are occluded, reflecting the nature of SCA attacks on masked implementations that require the combined leakage of multiple samples.

7.3.1 n-Occlusion

The n-occlusion attribution technique requires a trained network able to recover the correct key. In a first step, the key rank for different numbers of traces $N \in [1, \ldots, N_{attr}]$ is calculated from an attack set of N_{attr} traces. The number of traces N_0 required for the successful attack, i.e., such that $\mathrm{KR}(N_0) = 0$, determines a reference for the occlusion results. In a second step, n samples around a center sample $\dot{n}$ are occluded in all traces of the attack set[2]. The modified inputs are provided to the trained network, and the performance is evaluated similar to the reference evaluation. If the network is not able to reach key rank 0 anymore, the occluded samples can be considered important to the network. An increase in the number of traces to reach key rank 0 indicates that the occluded samples are important but can be compensated with other parts of the measurements – in Section 7.3.3 we introduce higher-order occlusion to detect combinations of important samples. Finally, the center $\dot{n}$ is shifted by $n \geq \Delta_n \geq 1$ samples and the procedure is repeated until the different parts of the traces have gradually been hidden from the network. This results in $M = \lfloor \frac{N_s - n}{\Delta_n} + 1 \rfloor$[3] different input ranges that can be occluded. A visualization of the method is shown in Fig. 7.2.

7.3.2 Exploring Different Occlusion Methods

With n-occlusion as defined in Section 7.3.1 there are two degrees of freedom[4]: the occlusion technique, i.e., how to replace occluded samples, and the occlusion window size n. We evaluate both parameters in this section and give recommendations suitable for the ASCAD databases. The occlusion technique requires a method to replace input samples that are hidden from the network as for an attribution of the pre-trained network we can not change its architecture, e.g., the input layer. In image

[2]Note that a perfect centering around $\dot{n}$ is only possible for odd values of n, and for even n we define the occlusion area left of $\dot{n}$ to include the additional sample.

[3]Note that for $\Delta_n > 1$ we ignore the remaining samples at the end of the trace that do not fit in a complete occlusion window of size n anymore.

[4]Δ_n does not impact the occlusion itself but only affects the accuracy of visualization.

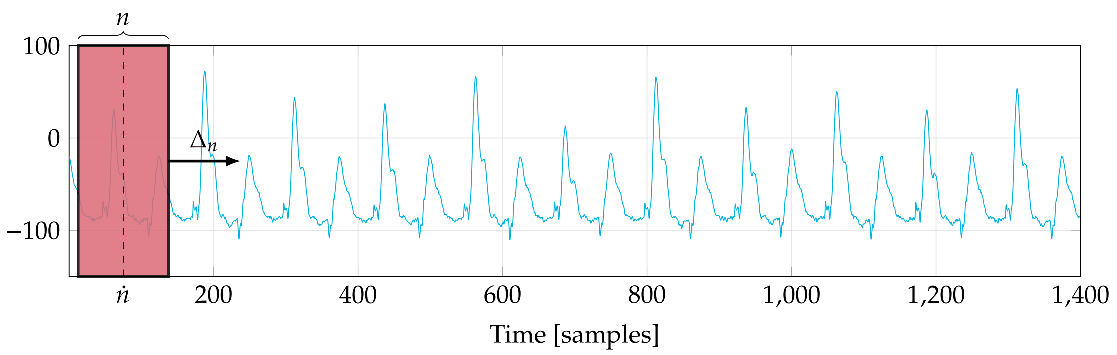


Figure 7.2 Visualization of the n-occlusion method.

processing, it is common to use gray squares [ZF14] since they represent the center of the RGB value range. An analogy to SCA measurements would be a substitution by zeros since these measurements are usually represented by signed integers. However, this is only one possible method, which is why we additionally investigate three other approaches for an occlusion interval $\boldsymbol{o} = [o_0, \dots, o_{n-1}]$ of size n:

1. Substitute by a constant value, either zeros $\boldsymbol{o}_0 = [0, \dots, 0]$ or the mean over all N_s samples of the N_{train} training traces $\boldsymbol{t}_i$ as $\boldsymbol{o}_{const.} = [\bar{t}, \dots, \bar{t}\,]$ with $\bar{t} := {}^{1}\!/\!_{(N_s \cdot N_{train})} \cdot \sum_{i=0}^{N_{train}-1} \sum_{j=0}^{N_s-1} \boldsymbol{t}_i\left[j\right]$.

2. Draw for each sample a new realization $\tilde{X}$ of a normal Gaussian distribution parametrized by the mean $\bar{t}$ and standard deviation σ of all samples in the N_{train} training traces resulting in $\boldsymbol{o}_{gauss} = [\tilde{X}_0, \dots, \tilde{X}_{n-1}]$ with $\tilde{X}_i \sim \mathcal{N}(\bar{t}, \sigma)$.

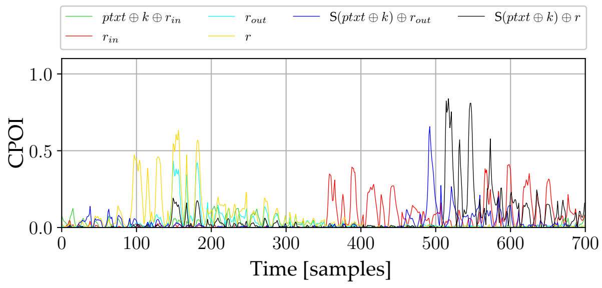
3. Preserve the approximate shape of the input traces by a substitution with the sample-wise mean over all training traces $\hat{\boldsymbol{t}}[j] := {}^{1}\!/\!_{N_{train}} \cdot \sum_{i=0}^{N_{train}-1} \boldsymbol{t}_i\left[j\right]$ resulting in $\boldsymbol{o}_{avg} = \left[\hat{\boldsymbol{t}}\left[\dot{n} - \frac{n-1}{2}\right], \dots, \hat{\boldsymbol{t}}\left[\dot{n} + \frac{n-1}{2}\right]\right]$ for odd values of n and $\boldsymbol{o}_{avg} = \left[\hat{\boldsymbol{t}}\left[\dot{n} - \frac{n}{2}\right], \dots, \hat{\boldsymbol{t}}\left[\dot{n} + \frac{n}{2} - 1\right]\right]$ for even values of n.

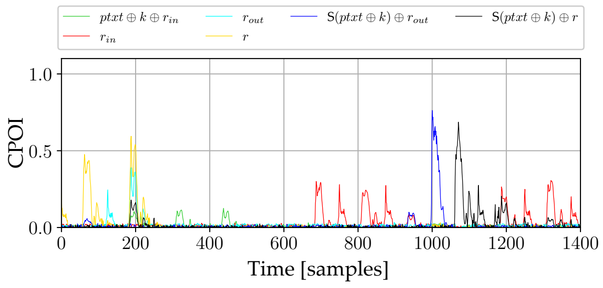
To select an appropriate occlusion method, we first define some desired properties by which we can evaluate the occlusion results afterwards. First, a good occlusion method should not significantly change attack results when occluding a region without necessary information for the network. Second, it should also induce the opposite, i.e., the key rank increases when a sample region with necessary information is occluded. Finally, the key rank should also increase the more information is occluded, i.e., with an increasing size of the occlusion window. We show the resulting key rank evolution for different occlusion methods for both ASCAD databases using CNN_{best} trained with the HW model[5] in Fig. 7.3. We use two different occlusion centers for comparison: The range around occlusion center $\dot{n} = 150$ (*ASCAD FIX*) and $\dot{n} = 175$ (*ASCAD VARIABLE*) contains several leakages according to the analysis in Figs. 7.3a and 7.3b, while the region around $\dot{n} = 380$ (*ASCAD FIX*) and $\dot{n} = 600$ (*ASCAD VARIABLE*) does not show any significant leakage[6]. Furthermore, we gradually increase the occlusion width n up to a full clock cycle (c.f. Table 5.1).

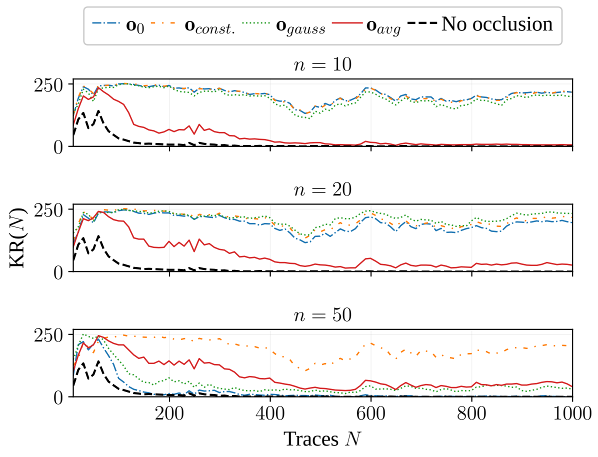
Influence of the Occlusion Technique Figure 7.3 shows that the chosen occlusion method has a significant impact on the attack results. For the occlusion of important leakage in Figs. 7.3c and 7.3d we expect the resulting key rank to increase with an increasing occlusion width since additional

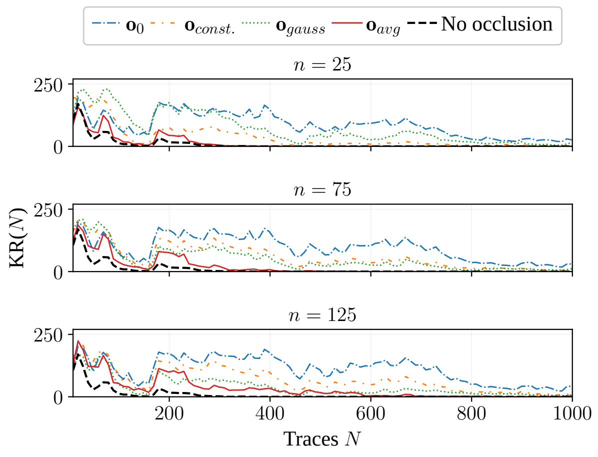
[5]We show evaluation results for the HW model for which the best combination function for a second-order attack is proven to be a combination of both the mask and the masked value [PRB09]. This reduces the possibility of additional leakage that the networks are able to learn, and therefore the attribution results are easier to interpret. However, we also evaluated training with the ID model, which shows the same characteristic but not as pronounced.

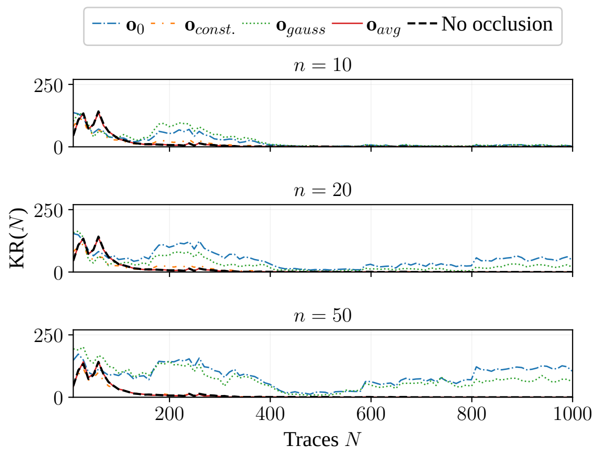
[6]Note that the evaluation shown in Figs. 7.3a and 7.3b uses the HW model instead of the ID model (Figs. 7.1a and 7.1b) and with the HW model the leakage of $ptxt \oplus k \oplus r_{in}$ is reduced significantly.


(a) CPOI (HW) for *ASCAD FIX*


(b) CPOI (HW) for *ASCAD VARIABLE*


(c) *ASCAD FIX*: $\dot{n}$ = 150


(d) *ASCAD VARIABLE*: $\dot{n}$ = 175


(e) *ASCAD FIX*: $\dot{n}$ = 380

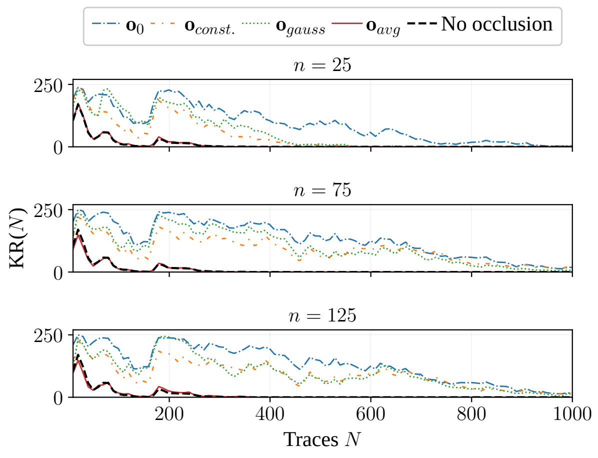
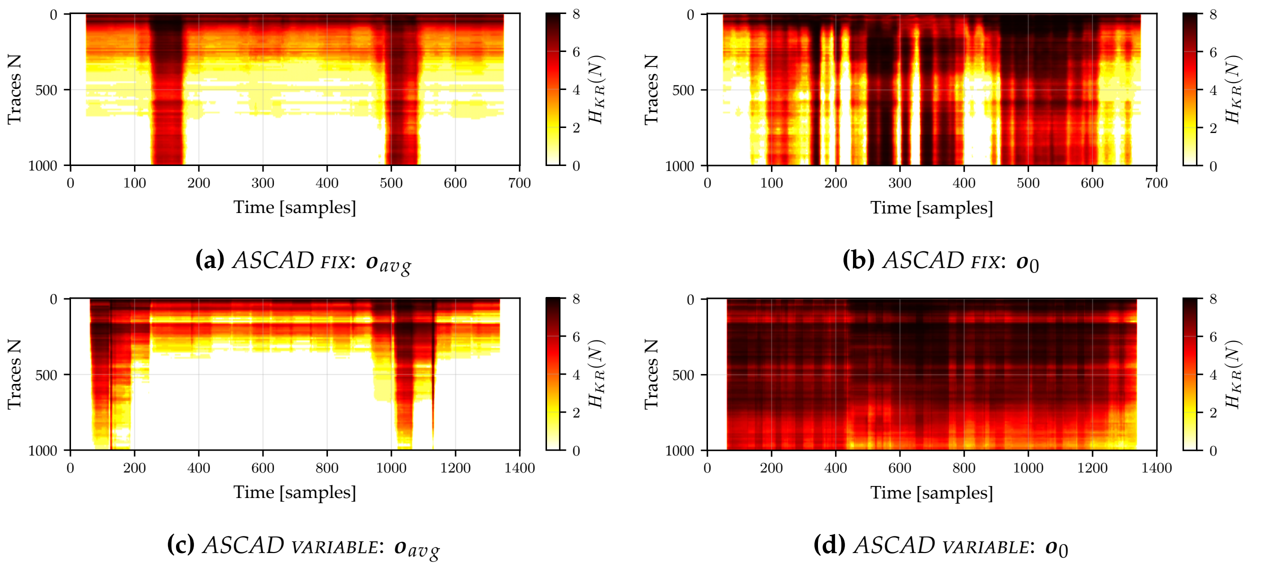

(f) *ASCAD VARIABLE*: $\dot{n}$ = 600

Figure 7.3 Key rank evolution using different occlusion methods for both ASCAD datasets.



(a) *ASCAD* FIX: $\boldsymbol{o}_{avg}$ **(b)** *ASCAD* FIX: $\boldsymbol{o}_0$

(c) *ASCAD* VARIABLE: $\boldsymbol{o}_{avg}$ **(d)** *ASCAD* VARIABLE: $\boldsymbol{o}_0$

Figure 7.4 Occlusion results for the different methods with $\Delta_n = 1$, $N_{attr} = 1000$, and an occlusion width of a whole clock cycle with $n = 50$ (*ASCAD* FIX) and $n = 125$ (*ASCAD* VARIABLE).

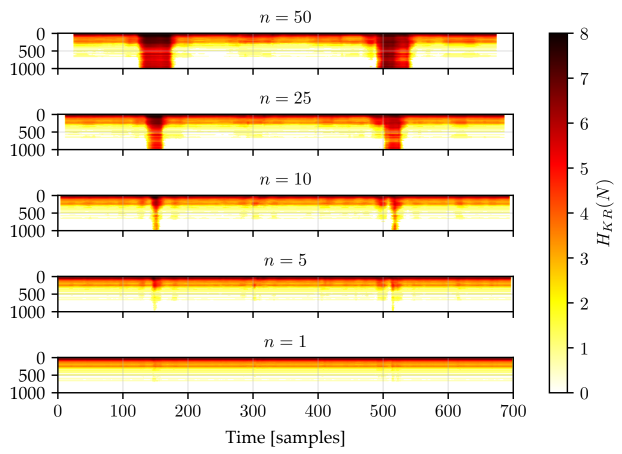
leakage is removed. This is only observable for the sample-wise mean occlusion method ($\boldsymbol{o}_{avg}$). The other occlusion techniques do not show deterministic and comprehensible changes while being very sensitive to the starting and ending points of the occlusion rather than the window size. For a region without important leakage, as shown in Figs. 7.3e and 7.3f, the occlusion should not significantly influence the attack result. A consistent behavior for both datasets is again only observable for the sample-wise mean occlusion method ($\boldsymbol{o}_{avg}$).

In order to emphasize the sensitivity of the different occlusion methods, we show the n-occlusion results for the sample-wise mean ($\boldsymbol{o}_{avg}$) and zero occlusion ($\boldsymbol{o}_0$) method in Fig. 7.4. Instead of limiting the analysis to particular samples, these waterfall plots depict the occlusion center $\dot{n}$ on the x-axis and the number of used attack traces on the y-axis. This allows to visualize the evolution of the logarithmic key rank, according to Eq. (5.4) as remaining entropy, for a fixed window n shifted by a chosen Δ_n with an increasing number of attack traces. For $\Delta_n = 1$, each pixel in a waterfall plot corresponds to the final key rank after occluding the n measurement samples around $\dot{n}$. Dark colors indicate a high key rank, i.e., the information required for a successful attack is occluded, while white represents a successful attack with a key rank of 0 for the given N_{attr} traces. For the comparison, we chose the zero occlusion method evaluated by Hettwer et al. [HGG20]. Since the underlying AES implementation is equivalent for both datasets, a similar behavior would be expected - except for differences in the measurement characteristics. This is only the case for the sample-wise mean occlusion shown in Figs. 7.4a and 7.4c, which highlights two regions of interest corresponding to the leakage of the masked S-box and the mask itself (c.f. the leakage evaluation in Figs. 7.3a and 7.3b) as expected for training with the HW model. Note that the ranges for the datasets differ slightly, as explained in Section 6.2.1, which is why the regions are not exactly aligned between both datasets. The results of the zero occlusion method differ between both datasets and do not lead to valuable attribution results.

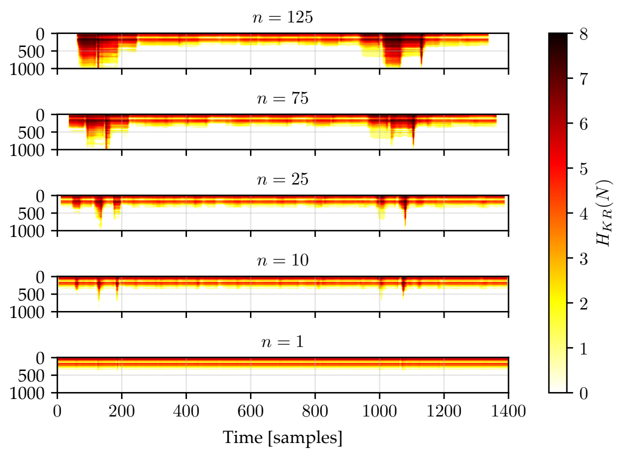
We conclude from our analysis that the sample-wise mean occlusion method ($\boldsymbol{o}_{avg}$) meets all criteria for an interpretable attribution technique and is therefore used in the remainder of this thesis.

Influence of the Occlusion Width n

After establishing a suitable occlusion method, we investigate the width n of the occlusion window. Intuitively, the larger n, the more samples are occluded, which increases the chance of removing leakage, which is important for the model's inference. On the other hand, decreasing n allows



(a) *ASCAD FIX*



(b) *ASCAD VARIABLE*

Figure 7.5 Occlusion results for different widths n of the occlusion window for both ASCAD databases.

to narrow down the sample ranges where the main leakage occurs, allowing for a more precise attribution of POIs. We analyze the impact of the window size for both ASCAD datasets for a training with HW labels and $\Delta_n = 1$ in Fig. 7.5.

The results show that the occlusion window can be decreased up to a minimum width for which the occlusion does not have an influence on the attack result anymore, i.e., the remaining leakage is sufficient to reach a key rank of zero. For *ASCAD FIX* (Fig. 7.5a) and *ASCAD VARIABLE* (Fig. 7.5b) a width of $n < 10$ does not change attack results significantly. The occlusion width in relation to the clock cycle is smaller for *ASCAD VARIABLE*, since in this dataset, the leakage is much more confined on a few samples, as it can be seen with the leakage analysis in Fig. 7.3b.

Our findings stress that the occlusion width n has to be adapted to fit the underlying data, and an approach like 1-occlusion [HGG20] is not suited to find relevant POIs in deep learning-based SCA. This is because leakage in side-channel measurements is usually spread over multiple samples of a clock cycle as a result of the low pass characteristics of side-channel measurement setups. For the remainder of this chapter, we show results for an occlusion width corresponding to the sample range of one clock cycle. The main leakage samples then correspond to the center of highlighted occlusion regions, as it can be seen in Fig. 7.5.

Experimental Results Using the established parameters for the occlusion method in the previous sections, we provide occlusion results for both databases and networks additionally using the ID model in Fig. 7.6. The method reliably identifies POI regions with leakage required by the CNNs for a successful attack. In comparison to gradient-based methods in Fig. 7.1, the attribution enables an easier interpretation of results compared to the actual leakage regions shown in Figs. 7.1a and 7.1b. All networks require the leakage of the masks, around sample 150 (*ASCAD FIX*) and 200 (*ASCAD VARIABLE*), and the masked value, around sample 525 (*ASCAD FIX*) and 1050 (*ASCAD VARIABLE*) for a successful attack. For *ASCAD FIX* and $\text{CNN}_{\text{small}}$ the network additionally deems the leakage around 100 and 580 as relevant, which corresponds to the additional leakage of these intermediates. Note that occlusion results for *ASCAD VARIABLE* and CNN_{best} in Fig. 7.6b are shown with an increased amount of $N_{attr} = 2000$ traces. In this scenario, the method identifies additional POIs around sample ranges 300 and 450, for which the network is able to compensate with other leakage given the additional traces. An explanation is that the network has a significantly higher amount of trainable parameters than $\text{CNN}_{\text{small}}$, and therefore it can learn additional combinations of leakages required for a successful attack. This could be an indicator of network overfitting to these additional leakages.

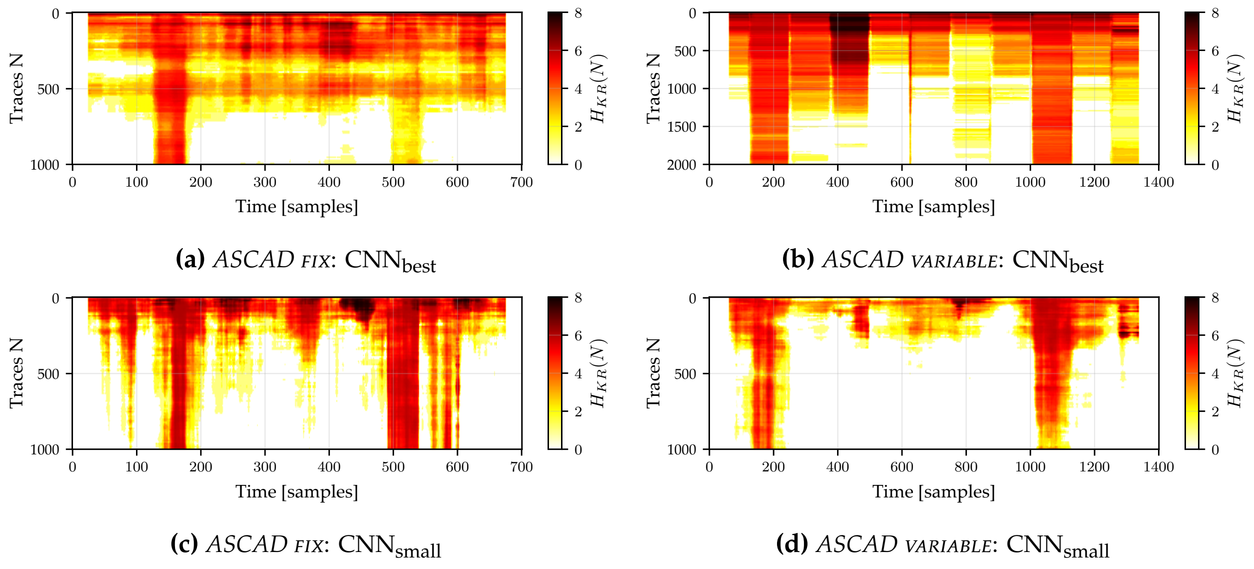

(a) *ASCAD FIX*: $\mathrm{CNN_{best}}$ **(b)** *ASCAD VARIABLE*: $\mathrm{CNN_{best}}$ **(c)** *ASCAD FIX*: $\mathrm{CNN_{small}}$ **(d)** *ASCAD VARIABLE*: $\mathrm{CNN_{small}}$

Figure 7.6 Occlusion results for a training with the ID model for both datasets and architectures.

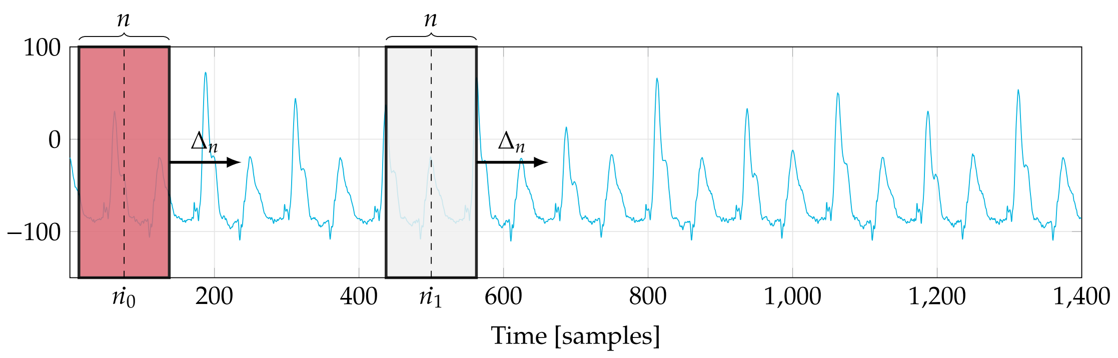
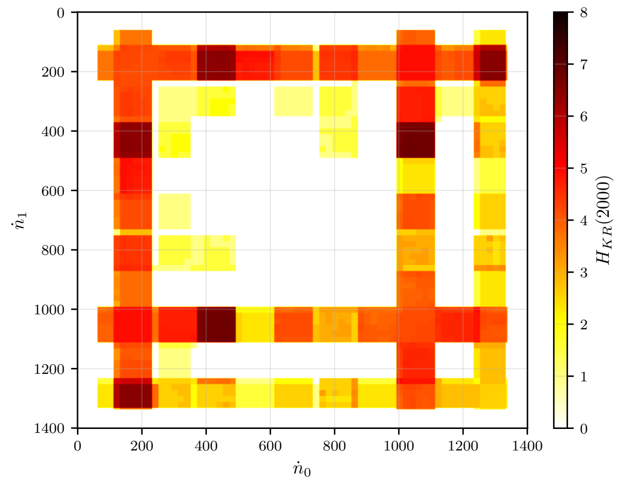


Figure 7.7 Visualization of the higher-order occlusion method with order $d = 2$.

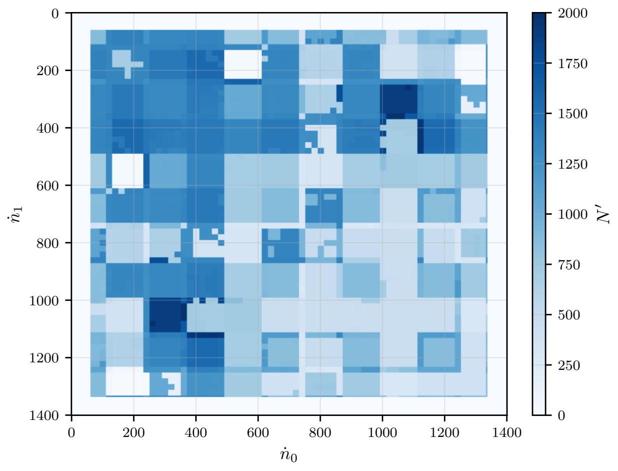
7.3.3 Higher-Order Occlusion: Hiding Multiple Parts at Once

We have shown in the previous sections that the n-occlusion method is able to identify sample ranges that have an impact on the attack results. Nevertheless, for higher-order attacks against a masked implementation, leakage of multiple samples corresponding to both shares of the masked intermediate has to be combined for a successful attack (c.f. Section 2.4). These shares are usually processed in different clock cycles for masked software implementations (c.f. Figs. 7.1a and 7.1b). We, therefore, introduce higher-order occlusion that hides multiple parts of the traces simultaneously to identify important sample range combinations. This also enables the identification of POIs in the presence of learned redundant leakage where the network is able to compensate occluded information with leakage from other samples. For a d-th order occlusion, d sample ranges around $\dot{n}_0, \ldots, \dot{n}_{d-1}$ are occluded and the impact on the resulting key rank is evaluated. We use the same size $n < N_s$ and a shift $\Delta_n \leq n$ for all occlusion windows. A visualization of the method for $d = 2$ is shown in Fig. 7.7.

In the following, we focus on second-order occlusion ($d = 2$), as it fits the first-order masked AES implementation of the ASCAD databases. In Fig. 7.8, we show second-order occlusion results for *ASCAD VARIABLE* and $\mathrm{CNN_{best}}$ trained with the ID model, corresponding to the first-order occlusion shown in Fig. 7.6b. For clarity, we show the entropy of the key rank after a fixed number of traces $N_{attr} = 2000$ in Fig. 7.8a instead of the waterfall visualization. Figure 7.8b depicts the first occurrence N' of the final key rank according to Eq. (5.5) and provides additional insights into the evolution of the key rank. While Fig. 7.8a could also be analyzed on its own to determine important input ranges and combinations thereof, Fig. 7.8b visualizes whether the result reaches a stable point ($N' \ll N_{attr}$,



(a) Final Entropy



(b) First Trace with Final Entropy

Figure 7.8 Second-order occlusion for *ASCAD* VARIABLE and CNN$_{\text{best}}$ using the ID model for $N_{attr} = 2000$ and $\Delta_n = 20$.

light colors) or the attack is still improving (N' close to N_{attr}, dark colors). Due to the computational complexity of higher-order occlusion, we present results for shifting $\dot{n}$ by $\Delta_n = 20$. This implies that for the visualization in Fig. 7.8, we show the same key rank for these 20 samples centered around the gradually shifted $\dot{n}$.

Figure 7.8a can be interpreted in the way that for each coordinate $(\dot{n}_0, \dot{n}_1)$ the resulting final entropy of the key rank is shown, given two simultaneous occlusions at the respective $\dot{n}$. After an occlusion of the leakage corresponding to the masks ($\dot{n}_0 = 200$) or the masked S-box ($\dot{n}_0 = 1050$), the network is no longer able to perform a successful attack regardless of the second occlusion center. This is consistent with the first-order occlusion results shown in Fig. 7.6b. However, it can be seen that if additionally the leakage corresponding to $ptxt \oplus k \oplus r_{in}$ at ($\dot{n}_0 = 200$, $\dot{n}_1 = 400$) or ($\dot{n}_0 = 400$, $\dot{n}_1 = 1050$) is occluded, the resulting key rank increases significantly. As a second observation, occluding the leakage corresponding to r_{in} ($\dot{n}_0 = 1300$) significantly increases the key rank. In this case, if the leakage of the masks is occluded in combination ($\dot{n}_0 = 1300$, $\dot{n}_1 = 200$), the network is not able to retrieve the correct secret key anymore. We conclude that in addition to a combination of both shares of the masked S-box, the network infers information from a combination of $ptxt \oplus k \oplus r_{in}$ with r_{in}[7]. If one leakage part of both combinations is not available to the networks, the resulting key rank is increased significantly.

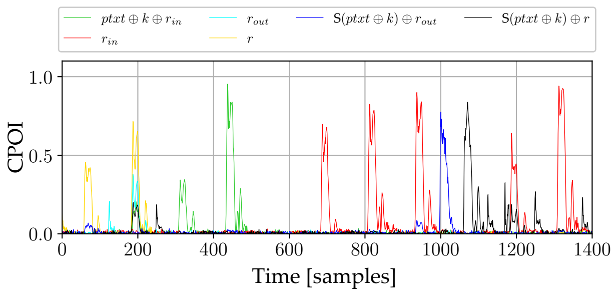
We conclude that higher-order occlusion can be a viable tool for leakage attribution in deep learning-based SCA to identify sample combinations for masked implementations.

7.4 Occlusion Evaluation of the ASCAD Databases

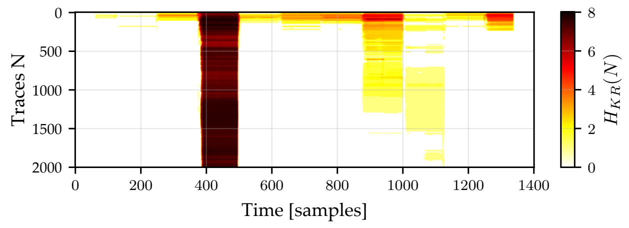
Having established the n-occlusion method and its higher-order variant, we use both methods to show that networks trained on other key bytes of the *ASCAD* VARIABLE database focus on different leakages than for the usually targeted byte two (k_2). We focus on the key bytes that were identified in Chapter 6 to exhibit special leakage characteristics. The results are shown using *ASCAD* VARIABLE and a training with CNN$_{\text{best}}$.

We start with an evaluation of a training on key byte k_3 for which the occlusion results are shown in Fig. 7.9.

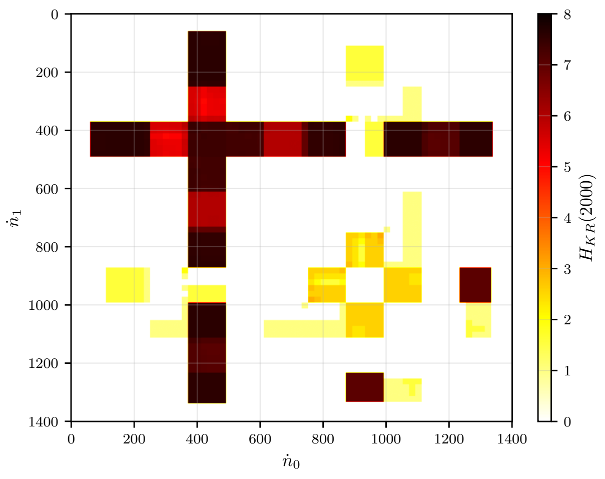
[7]This observation is also confirmed by the results presented in [YBP23].



(a) Leakage evaluation



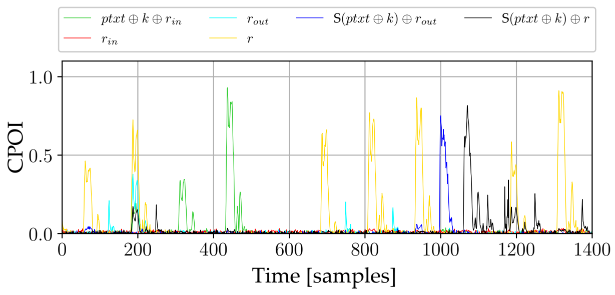
(b) First-order occlusion ($\Delta_n = 1$)



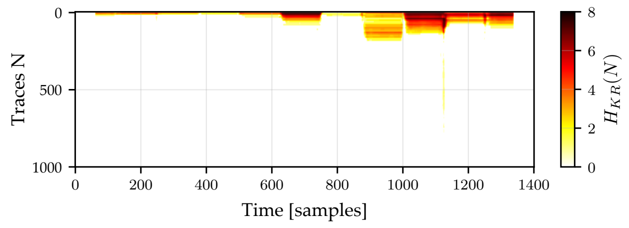
(c) Second-order occlusion ($\Delta_n = 20$)

Figure 7.9 Occlusion evaluation of a CNN$_{\text{best}}$ training for key byte k_3 of *ASCAD variable* using the ID model.

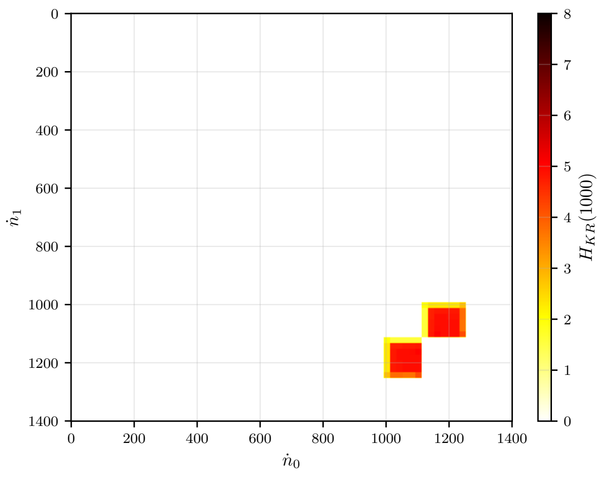
When comparing the first-order occlusion results for k_3 in Fig. 7.9b with the leakage evaluation, we find that the network requires the leakage of $ptxt \oplus k \oplus r_{in}$ ($\dot{n} = 450$) for a successful attack. Hence, it is able to combine this leakage with r_{in} to infer the correct class label. Occluding the masked S-box ($\dot{n} = 1050$) can mostly be compensated after 1500 traces. In the second-order occlusion Fig. 7.9c, it can be seen that the network has learned redundant leakage of r_{in}, since only if ($\dot{n}_0 = 950$, $\dot{n}_1 = 1300$) is occluded together the network is not able to reach key rank zero anymore. Please note that in contrast to k_2 (c.f. Fig. 7.1b), the leakage of r_{in} is increased significantly.



(a) Leakage evaluation



(b) First-order occlusion ($\Delta_n = 1$)



(c) Second-order occlusion ($\Delta_n = 20$)

Figure 7.10 Occlusion evaluation of a CNN$_{\text{best}}$ training for key byte k_5 of *ASCAD variable* using the ID model.

We additionally show occlusion results for a training on key byte k_5 in Fig. 7.10, which demonstrates the case of the network having learned enough redundant leakage such that a first-order occlusion is not sufficient to find important samples. The cause for this phenomenon is the additional and redundant leakage of the state mask r for k_5 as indicated in the leakage evaluation (Fig. 7.10a). This leads to the fact that first-order occlusion, shown in Fig. 7.10b, is not able to indicate important sample ranges. Only second-order occlusion, shown in Fig. 7.10c, reveals that a complete occlusion of both clock cycles containing leakage of the corresponding masked S-box value ($\dot{n}_0 = 1050$, $\dot{n}_1 = 1200$) decreases the attack performance. This indicates that the network has learned to use characteristic leakage of r of this key byte from more than two clock cycles.

7.5 Conclusion

In this chapter, we studied leakage explainability (attribution) techniques in the context of deep learning-based side-channel analysis. We introduced different types of occlusion methods, which provide a valuable addition to gradient-based approaches, whose results are often hard to interpret and subject to the exact epoch after which they are generated. While related work performing attribution for image classification indicates that the number of simultaneously occluded features and the replacement technique are important, this fact has not been considered in the context of side-channel analysis. We, therefore, introduced n-occlusion, where we occlude multiple samples at once and additionally analyze the evolution of the resulting key rank for an increasing number of attack traces. Our results show that replacing an entire clock cycle with the mean of each sample across all training traces is a promising approach. In order to identify leakage combinations required for an attack on masked implementations, we developed higher-order occlusion. Our reasoning is based on the analysis of two different network architectures for the established ASCAD databases. With the developed methods, we are able to show that the networks are able to actually exploit varying leakage characteristics that are observable for different key bytes of the targeted AES implementation.

8 Conclusion

This thesis provides multiple contributions in the field of SCA that can be summarized as the development and evaluation of attacks and countermeasures for two submissions of the NIST post-quantum cryptography competition as well as advances in the field of explainability for deep learning-based SCA.

First, one of the first works on countermeasures for software implementations of post-quantum cryptography was presented with an evaluation of a masking countermeasure for the NTRUEncrypt cryptosystem. Due to the maturity of the underlying mathematical problem, after a merger with a similar system, it has been a final candidate evaluated by NIST for standardization. Two masked implementations were presented, and their security was evaluated through first- and second-order side-channel evaluations. In order to achieve a low performance overhead, SIMD instructions of the target platform were used, and the possibility of a further hardening of the implementation through an additional low overhead shuffling countermeasure was demonstrated. The remainder of the first part of this thesis presented a comprehensive side-channel evaluation of the code-based HQC cryptosystem that is currently one of three remaining alternative candidates for standardization. This includes the first power side-channel chosen-ciphertext attack on the system as well as an updated version of the attack, which is required as the authors chose to increase the performance of the system with an updated error correction, rendering former attack strategies obsolete. The updated attack strategy is significantly improved and does not rely on simulations to verify its success as it is mathematically proven with high probability for the HQC parameter sets. Multiple attack vectors for the presented attack strategies were discussed and practical side-channel attack results showed the strength of developed attacks and therefore the necessity for the development of countermeasures. Finally, multiple countermeasures were evaluated, showing that the established codeword-masking countermeasure is not applicable. The developed countermeasure of inserting an additional random number of errors before the decoding was shown to be effective but comes with the downside of required changes to the implemented decoder, leading to the conclusion that the development of a complete masking scheme for HQC is required.

At the beginning of the second part of the thesis, a comprehensive evaluation of the ASCAD databases with classical SCA was performed, building the foundation for research on explainability methods for DL-based SCA. First, a leakage evaluation was performed that, after a careful examination of the underlying assembly implementation, revealed additional leakage contained in the chosen trace segments of the ASCAD authors and that leakage significantly differs between targeted key bytes. Additionally, multiple first-order leaks for both databases and univariate second-order leaks for *ASCAD FIX* were identified when using the selected trace segments. If the complete sample ranges contained in the databases are used, there are first-order leaks for most bytes of *ASCAD FIX* and all bytes of *ASCAD VARIABLE*. Further, it was concluded that an attack on *ASCAD FIX* corresponds to the best-case scenario from an attacker's point of view, and a more realistic attack scenario is given by *ASCAD VARIABLE*. Finally, it was shown with the use of two different architectures that DL-based SCA shows significant attack performance differences between bytes and that a network trained on one byte can not be automatically used for an attack on other bytes. This shows the possibility of leveraging attack results on different key bytes of ASCAD in addition to a comparison of results for different datasets in order to improve the robustness of results. As a last contribution, multiple advancements to the occlusion explainability method were presented. These include the introduction of n-occlusion, as it was identified that the occlusion of a single sample is not sufficient, and the development of higher-order occlusion, which is able to identify sample combinations that are used by a network to perform a successful attack. After determining suitable occlusion parameters, the developed higher-order occlusion method was used to show that neural network architectures actually utilize the different leakage characteristics identified in this thesis. Therefore, it can be concluded that the developed methods provide a valuable addition to established gradient-based explainability methods.

Summing up, the presented work in this thesis positively influenced the implementation security of multiple schemes in the NIST post-quantum cryptography competition. Primarily, the contributions for the HQC cryptosystem have a practical impact, as it is still a candidate for an alternative standard. The evaluation of ASCAD provides a foundation for future research on explainability for DL-based SCA using these databases. In addition, identified flaws in the databases must be taken into account when interpreting attack results, particularly when utilizing the full sample range.

Future Work In addition to the findings presented in this thesis, several directions for future work have been identified.

There are multiple hardware accelerators for NTRUEncrypt (e.g., [FSF$^+$19, FDNG19]) that should, in theory, be attackable using a similar attack as discussed in this thesis. Although these accelerators require significant resources, they may still be used in practice if a trade-off between execution time and resources is desired. Therefore, the side-channel evaluation of these accelerators and the development of countermeasures are required to securely use them in practice.

In the context of chosen-ciphertext attacks for the HQC cryptosystem, improvements and new attacks that lower the number of required oracle calls and therefore attack traces are likely. Attacks that decrease the amount of input that has to be set to a detectable value are beneficial from an attacker's point of view as these would decrease the efficiency by a parallel increase of the required resources of detection-based countermeasures and can ultimately render them ineffective. The strength of the side-channel requires the need for countermeasures and therefore the development of a complete masking scheme for the HQC cryptosystem. This includes the development of novel masking gadgets, especially for the decoder of the used error-correcting codes, and it should be the primary focus for future work on countermeasures. Another direction for future work is to evaluate if the presented attacks can be transferred to the other code-based cryptosystems Classic McEliece [ABC$^+$20] and BIKE [ABB$^+$20a] that are still under consideration for standardization. Finally, the development of a method to achieve the CCA-2 security of the algorithm without relying on a variant of the FO transformation and thus re-encryption could significantly reduce the effort and resources required for a protected implementation.

The presented results on DL-based SCA showed that an in-depth evaluation of used datasets through classical side-channel analysis is required for an interpretation of attack results. Therefore, a topic for future work is to perform this research on novel datasets or provide additional datasets with the respective evaluation. The possibilities for future work on the presented occlusion methods are manyfold. First, the applicability of the developed methods to other types of network architectures, such as MLPs, and other datasets could be investigated. Another direction for future work is to investigate the retraining of a model with omitted features and analyze the resulting behavior. Finally, future work might also explore inverted occlusions without retraining, i.e., only feeding small windows of actual data to the model instead of occluding certain regions.

A Proof of the Attack Strategy for HQC-RMRS

This chapter presents proofs of the lemmas and the theorem leading to the attack strategy on HQC-RMRS in Section 4.6.2. As noted in the introduction to Chapter 4, these proofs were developed by Lukas Holzbaur in [SHR$^+$22] but are shown for completeness in this thesis.

Proof of Lemma 1

Proof. The first case follows from observing that $f\hat{p}\check{p} = 0$ for these polynomials[1]. It is well-known that any codeword $p \in \mathcal{RM}(m)$, except the all-zero and the all-one word, i.e., any word with $\deg(p) = 1$, is of weight $d = 2^{m-1}$. Since $\deg(\hat{p}) = \deg(\check{p}) = 1$ and $\check{p} \notin \{\hat{p}, \hat{p}+1\}$ we have $\deg(\hat{p}+\check{p}) = 1$. Therefore, $\mathrm{HW}(\hat{p}+\check{p}) = d$ and we get

$$\begin{aligned}
\mathrm{HW}(\hat{p}+\check{p}) &= \mathrm{HW}(\hat{p}) + \mathrm{HW}(\check{p}) - 2\mathrm{HW}(\hat{p}\check{p}) \\
d &= 2d - 2\mathrm{HW}(\hat{p}\check{p}) \\
\frac{d}{2} &= \mathrm{HW}(\hat{p}\check{p}) \,.
\end{aligned}$$

The second case follows since we have $\mathrm{supp}(\hat{p}\check{p}) \subset \mathrm{supp}(f)$ for any $f \in \{1, \hat{p}, \check{p}, \hat{p}+\check{p}+1\}$. Now consider some $f \in \mathcal{RM}(m) \setminus \{0, \hat{p}+1, \check{p}+1, \hat{p}+\check{p}, 1, \hat{p}, \check{p}, \hat{p}+\check{p}+1\}$ and note that $\deg(f) = 1$. Observe that the supports of the polynomials $\{\hat{p}\check{p}, \hat{p}(\check{p}+1), (\hat{p}+1)\check{p}, (\hat{p}+1)(\check{p}+1)\}$ partition the 2^m codeword positions. Hence, by the pigeonhole principle, there exists some $\bar{p} \in \{\hat{p}(\check{p}+1), (\hat{p}+1)\check{p}, (\hat{p}+1)(\check{p}+1)\}$ with

$$\mathrm{HW}(\bar{p}f) \geq \left\lceil \frac{\mathrm{HW}(f) - \mathrm{HW}(\hat{p}\check{p}f)}{3} \right\rceil \geq \frac{d - \mathrm{HW}(\hat{p}\check{p}f)}{3} \,.$$

Further, it is easy to check that $\hat{p}\check{p} + \bar{p} \in \{\hat{p}, \check{p}, \hat{p}+\check{p}+1\}$, which implies $\deg(\hat{p}\check{p}+\bar{p}) = 1$ and $\mathrm{HW}(\hat{p}\check{p}+\bar{p}) = d$. Now, towards a contradiction, assume $\mathrm{HW}(f\hat{p}\check{p}) > \frac{d}{4}$. Then, we have

$$\begin{aligned}
d(\hat{p}\check{p}+\bar{p}, f) &= \mathrm{HW}(\hat{p}\check{p}+\bar{p}) + \mathrm{HW}(f) - 2\mathrm{HW}((\hat{p}\check{p}+\bar{p})f) \\
&= 2d - 2(\mathrm{HW}(\hat{p}\check{p}f) + \mathrm{HW}(\bar{p}f)) \\
&\leq 2\left(d - \left(\mathrm{HW}(\hat{p}\check{p}f) + \frac{d - \mathrm{HW}(\hat{p}\check{p}f)}{3}\right)\right) \\
&\leq 2\left(d - \frac{d + 2\mathrm{HW}(\hat{p}\check{p}f)}{3}\right) \\
&< 2\left(d - \frac{d + 2\frac{d}{4}}{3}\right) = d \,.
\end{aligned}$$

As both $\hat{p}\check{p}+\bar{p}$ and f are in $\mathcal{RM}(m)$, this can only be true if $\hat{p}\check{p}+\bar{p} = f$. However, we have $\hat{p}\check{p}+\bar{p} \in \{\hat{p}, \check{p}, \hat{p}+\check{p}+1\}$ and therefore, by definition of f, a contradiction. Now assume there exists an $f' \in \mathcal{RM}(m) \setminus \{0, \hat{p}+1, \check{p}+1, \hat{p}+\check{p}, 1, \hat{p}, \check{p}, \hat{p}+\check{p}+1\}$ with $\mathrm{HW}(f\hat{p}\check{p}) < \frac{d}{4}$ and note that this set is closed under inversions, i.e., also contains $f'+1$. Then, we have $\mathrm{HW}((f'+1)\hat{p}\check{p}) > \frac{d}{4}$, which cannot be true, as shown above. □

[1]Note that $f^2 = f$ in $\mathbb{F}_2[x]$, so $(\hat{p}+1)\hat{p}\check{p} = \hat{p}^2\check{p} + \hat{p}\check{p} = 2\hat{p}\check{p} = 0$.

Proof of Lemma 2

Proof. Denote $\mathcal{F} = \{\hat{p}, \check{p}, \hat{p} + \check{p} + 1\}$. First, note that the algorithm always returns a word r such that $\mathfrak{D}_0^e(r) = \mathsf{False}$. Clearly, this statement would only be false if $\mathfrak{D}_0^e(r) = \mathsf{True}$ for all steps in the for loops of Lines 2 and 7. To see that this cannot be the case, consider the $\frac{d}{4}$-th iteration in the for loop of Line 7. In this iteration we have $\mathrm{HW}(r) = \frac{3}{4}d$ and $\mathrm{supp}(r) \subset \hat{\mathcal{I}} = \mathrm{supp}(\hat{p})$, where $\hat{p} \in \mathcal{RM}(m)$ by definition. It follows that $r + e$ is in the unique decoding ball of $\hat{p}$, since

$$\begin{aligned} d(r+e, \hat{p}) &= \mathrm{HW}(\hat{p} + r + e) \\ &\leq \mathrm{HW}(\hat{p} + r) + \mathrm{HW}(e) \\ &= d - \mathrm{HW}(r) + \mathrm{HW}(e) < \frac{d}{2} \,. \end{aligned}$$

In this case, an ML decoder for the RM code would decide for $\hat{p}$, and it holds that $\mathfrak{D}_0^e(r) = \mathsf{False}$ and $\mathfrak{D}_{\mathcal{RM}}(r+e) = \hat{p} \in \{\hat{p}, \check{p}, \hat{p} + \check{p} + 1\}$. Note that this also implies $\mathrm{HW}(\hat{p}(\check{p}+1)r) \leq \frac{d}{4}$ for any returned word r. Now consider the case that Algorithm 13 terminates in the for loop of Line 2, i.e., for an r with $\mathrm{supp}(r) \subseteq (\hat{\mathcal{I}} \cap \check{\mathcal{I}})$. For this case, we show a statement that is slightly stronger than required, namely, we prove that for any $f \in \mathcal{RM}(m) \setminus (\mathcal{F} \cup \{0\})$ we have $d(r+e, 0) < d(r+e, f)$, which implies that f cannot be the outcome of an ML decoder[2]. To begin, observe that $\mathfrak{D}_{\mathcal{RM}}(r+e) \neq 1$ since $\mathrm{HW}(r+e) \leq \mathrm{HW}(r) + \mathrm{HW}(e) < \frac{n}{4} + \frac{d}{4}$ and therefore $\boldsymbol{d}(r+e, 0) = \mathrm{HW}(r+e) < n - \mathrm{HW}(r+e) = d(r+e, 1)$, so the ML decoder does not decode to the all-one word in this case. If $\mathrm{HW}(r) \leq \frac{d}{4}$, we get $\mathrm{HW}(r+e) < \frac{d}{2}$ and an ML decoder always decides for 0, i.e., $\mathfrak{D}_0^e = \mathsf{True}$, so we can assume that $\mathrm{HW}(r) > \frac{d}{4}$ when Algorithm 13 terminates. Denote $\bar{\mathcal{F}} = \mathcal{RM}(m) \setminus (\mathcal{F} \cup \{0, 1\})$. Now consider some $f \in \bar{\mathcal{F}}$ and note that $\mathrm{supp}(\hat{p}\check{p}) = \hat{\mathcal{I}} \cap \check{\mathcal{I}}$. Then, we have

$$\begin{aligned} d(r+e, f) &= \mathrm{HW}(f + r + e) \\ &\geq \mathrm{HW}(f + r) - \mathrm{HW}(e) \\ &= \mathrm{HW}(\hat{p}\check{p}(f + r)) + \mathrm{HW}((\hat{p}\check{p} + 1)(f + r)) - \mathrm{HW}(e) \\ &= \mathrm{HW}(\hat{p}\check{p}(f + r)) + \mathrm{HW}((\hat{p}\check{p} + 1)f) - \mathrm{HW}(e) \\ &\geq \mathrm{HW}(r) - \mathrm{HW}(\hat{p}\check{p}f) + \mathrm{HW}((\hat{p}\check{p} + 1)f) - \mathrm{HW}(e) \,. \end{aligned}$$

Since $f \notin (\mathcal{F} \cup \{0, 1\})$ Lemma 1 gives $\mathrm{HW}(\hat{p}\check{p}f) \leq \frac{d}{4}$, so $-\mathrm{HW}(\hat{p}\check{p}f) + \mathrm{HW}((\hat{p}\check{p}+1)f) \geq \frac{3}{4}d$. Therefore, we get $d(r+e, f) > \mathrm{HW}(r) + \frac{3}{4}d - \frac{1}{4}d = \mathrm{HW}(r) + \frac{1}{4}d$. On the other hand, the distance of $r+e$ to 0 is

$$\begin{aligned} d(r+e, 0) &= \mathrm{HW}(r + e) \\ &\leq \mathrm{HW}(r) + \mathrm{HW}(e) \\ &< \mathrm{HW}(r) + \frac{1}{4}d \,. \end{aligned}$$

Therefore, if Algorithm 13 terminates in the for loop of Line 2, the outcome of the ML decoder cannot be a word of $\bar{\mathcal{F}} \cup \{1\}$, which implies that $\mathfrak{D}_{\mathcal{RM}}(r) \in \mathcal{F}$. Now consider the case where Algorithm 13 terminates in the for loop of Line 7. Note that, by definition of the sets $\hat{\mathcal{I}}$ and $\check{\mathcal{I}}$, we have $\mathrm{supp}(r) \subset \mathrm{supp}(\hat{p})$ and, since the for loop of Line 2 is completed, it holds that $\mathrm{HW}(\hat{p}\check{p}r) = \frac{d}{2}$. To begin, observe that

$$\begin{aligned} d(r+e, \hat{p}) &= \mathrm{HW}(\hat{p} + r + e) \\ &\geq \mathrm{HW}(\hat{p} + r) + \mathrm{HW}(e) \\ &\overset{\text{(a)}}{=} d - \mathrm{HW}(r) + \mathrm{HW}(e) < \frac{5}{4}d - \mathrm{HW}(r) \,, \end{aligned} \tag{A.1}$$

[2]Note that this does *not* imply that the outcome is 0, since one of the words of $\mathcal{F}$ could still be closer to $r + e$ than 0.

where (a) holds because $\mathrm{supp}(r) \subset \mathrm{supp}(\hat{p})$ and $\mathrm{HW}(\hat{p}) = d$. It follows immediately from Lemma 1 that an $\mathcal{RM}(m)$ code can be partitioned by

$$\begin{aligned}\mathcal{RM}(m) = \{0\} \cup \{1\} \cup \{\hat{p}+1\} \cup \mathcal{F} \cup \{f \mid \mathrm{HW}(\hat{p}\check{p}f) = 0, \mathrm{HW}(\hat{p}(\check{p}+1)f) = \frac{d}{2}\} \\ \cup \{f \mid \mathrm{HW}(\hat{p}\check{p}f) = \frac{d}{4}, \mathrm{HW}(\hat{p}(\check{p}+1)f) = \frac{d}{4}\} \ .\end{aligned}$$

The statement holds if the distance to the words in all subsets except $\{0\}$ and $\mathcal{F}$ is larger than Eq. (A.1). We consider each subset separately:

- For $f = 1$ we have

$$\begin{aligned}d(r+e,f) &= \mathrm{HW}(f+r+e) \\ &\geq \mathrm{HW}(f) - \mathrm{HW}(r) - \mathrm{HW}(e) \\ &> 2d - \mathrm{HW}(r) - \frac{d}{4} \\ &> \frac{7}{4}d - \mathrm{HW}(r) > d(r+e,\hat{p}) \ .\end{aligned}$$

- For $f = \hat{p}+1$ we have

$$\begin{aligned}d(r+e,f) &= \mathrm{HW}(f+r+e) \\ &\geq \mathrm{HW}(f+r) - \mathrm{HW}(e) \\ &= \mathrm{HW}(\hat{p}(f+r)) + \mathrm{HW}((\hat{p}+1)(f+r)) - \mathrm{HW}(e) \\ &= \mathrm{HW}(\hat{p}r) + \mathrm{HW}((\hat{p}+1)f) - \mathrm{HW}(e) \\ &= \mathrm{HW}(r) + \mathrm{HW}(f) - \mathrm{HW}(e) \\ &= 2\mathrm{HW}(r) + d - \mathrm{HW}(r) - \mathrm{HW}(e) \\ &\overset{(a)}{\geq} 2d - \mathrm{HW}(r) - \mathrm{HW}(e) \\ &> \frac{7}{4}d - \mathrm{HW}(r) > d(r+e,\hat{p}) \ ,\end{aligned}$$

  where (a) holds because $\mathrm{HW}(r) \geq \mathrm{HW}(\hat{p}\check{p}r) = \frac{d}{2}$, as noted above.

- For any $f \in \mathcal{RM}(m)$ with $\mathrm{HW}(\hat{p}\check{p}f) = 0$ and $\mathrm{HW}(\hat{p}(\check{p}+1)f) \geq \frac{d}{4}$ we have

$$\begin{aligned}d(r+e,f) &= \mathrm{HW}(f+r+e) \\ &\geq \mathrm{HW}(f+r) - \mathrm{HW}(e) \\ &= \mathrm{HW}(\hat{p}\check{p}(f+r)) + \mathrm{HW}((\hat{p}\check{p}+1)(f+r)) - \mathrm{HW}(e) \\ &\geq \mathrm{HW}(\hat{p}\check{p}r) + \underbrace{\mathrm{HW}((\hat{p}\check{p}+1)f)}_{=\mathrm{HW}(f)=d} - \mathrm{HW}((\hat{p}\check{p}+1)r) - \mathrm{HW}(e) \\ &= d + \mathrm{HW}(\hat{p}\check{p}r) - \mathrm{HW}((\hat{p}\check{p}+1)r) - \mathrm{HW}(e) \\ &= d + \underbrace{2\mathrm{HW}(\hat{p}\check{p}r)}_{=d} - (\mathrm{HW}(\hat{p}\check{p}r) + \mathrm{HW}((\hat{p}\check{p}+1)r)) - \mathrm{HW}(e) \\ &= 2d - \mathrm{HW}(r) - \mathrm{HW}(e) \\ &> \frac{7}{4}d - \mathrm{HW}(r) > d(r+e,\hat{p}) \ .\end{aligned}$$

- For any $f \in \mathcal{RM}(m)$ with $\mathrm{HW}(\hat{p}\check{p}f) = \mathrm{HW}(\hat{p}(\check{p}+1)f) = \frac{d}{4}$ we have

$$\begin{aligned}
d(r+e,f) &= \mathrm{HW}(f+r+e) \\
&\geq \mathrm{HW}(f+r) - \mathrm{HW}(e) \\
&= \mathrm{HW}(\hat{p}\check{p}(f+r)) + \mathrm{HW}(\hat{p}(\check{p}+1)(f+r)) + \mathrm{HW}((p_1+1)(f+r)) \\
&\quad - \mathrm{HW}(e) \\
&= \mathrm{HW}(\hat{p}\check{p}(f+r)) + \mathrm{HW}(\hat{p}(\check{p}+1)f) - \mathrm{HW}(\hat{p}(\check{p}+1)r) \\
&\quad + \underbrace{\mathrm{HW}((p_1+1)f)}_{=\frac{d}{2}} - \mathrm{HW}(e) \\
&= \frac{d}{4} + \frac{d}{4} - \mathrm{HW}(\hat{p}(\check{p}+1)r) + \frac{d}{2} - \mathrm{HW}(e) \\
&= d + \mathrm{HW}(\hat{p}\check{p}r) - (\mathrm{HW}(\hat{p}\check{p}r) + \mathrm{HW}(\hat{p}(\check{p}+1)r)) - \mathrm{HW}(e) \\
&= \frac{3}{2}d - \mathrm{HW}(r) - \mathrm{HW}(e) \\
&> \frac{5}{4}d - \mathrm{HW}(r) > d(r+e,\hat{p}) \, .
\end{aligned}$$

We conclude that for any $f \in \mathcal{RM} \setminus (\mathcal{F} \cup \{0\})$ a word of $\mathcal{F}$ (specifically $\hat{p}$) is closer[3] to $r+e$ than f, and it follows that $\mathfrak{D}_{\mathcal{RM}}(r+e) \in \mathcal{F}$. Since the distance to the word of $\mathcal{F}$ was truly smaller in each of the discussed cases, i.e., not a tie, the decision is not the result of a tie in the distance with some other word $\mathcal{RM}(m) \setminus \mathcal{F}$. □

Proof of Lemma 3

Proof. Denote $\mathcal{F} = \{\hat{p}, \check{p}, \hat{p}+\check{p}+1\}$. By Lemma 2, we have $\mathfrak{D}_{\mathcal{RM}}(r+e) =: \tilde{p} \in \mathcal{F}$ for the word r returned at Line 4 of Algorithm 14. By definition of $\mathcal{F}$, this implies that $(\hat{\mathcal{I}} \cap \check{\mathcal{I}}) \subset \mathrm{supp}(\tilde{p})$, i.e., the positions $\hat{\mathcal{I}} \cap \check{\mathcal{I}}$ of $\tilde{p}$ are all one. Therefore, if a position in $\hat{\mathcal{I}} \cap \check{\mathcal{I}}$ of $r+e$ is changed from 0 to 1, the distance to $\tilde{p}$ *always* decreases by 1 and the ML decoder output does not change. On the other hand, if a position in $\hat{\mathcal{I}} \cap \check{\mathcal{I}}$ of $r+e$ is changed from 1 to 0, the distance to any polynomial of $\mathcal{F}$ *always* increases by 1, the distance to 0 decreases by 1, and the distance to any other word in $\mathcal{RM}(m) \setminus (\mathcal{F} \cup \{0\})$ decreases by at most 1. Hence, the ML decoding result changes from $\tilde{p}$ to 0 and the oracle returns True. □

Proof of Theorem 1

Proof. For the sake of readability and ease of notation, we focus on the case of multiplicity $s = 1$ in this proof. It is easy to verify that all statements also hold for $s > 1$ by essentially multiplying every weight/distance by s. Note that both Algorithms 13 and 14 are independent of s. Consider some choice of $\hat{\mathcal{I}}$ and $\check{\mathcal{I}}$ in Lines 2 and 3 of Algorithm 14. Note that there exist corresponding polynomials $\hat{p} \in \{p_1, p_1+1\}$ and $\check{p} \in \{p_2, p_2+1\}$ with $\mathrm{supp}(\hat{p}) = \hat{\mathcal{I}}$ and $\mathrm{supp}(\check{p}) = \hat{\mathcal{I}}$ and we have $\deg(\hat{p}) = \deg(\check{p}) = 1$ and $\check{p} \notin \{\hat{p}, \hat{p}+1\}$ for any such choice. The for loop in Line 6 of Algorithm 14 iterates over all positions of $\boldsymbol{r}$ in $\hat{\mathcal{I}} \cap \check{\mathcal{I}}$ and queries the oracle with this bit flipped. If this changes the oracle output to True, the corresponding bit is flipped in $\hat{\boldsymbol{e}}$, with the goal of obtaining $\hat{\boldsymbol{e}}_{\hat{\mathcal{I}} \cap \check{\mathcal{I}}} = \boldsymbol{e}_{\hat{\mathcal{I}} \cap \check{\mathcal{I}}}$ at the end of the loop. We consider the four different possible combinations of $\boldsymbol{e}_\xi$ and $\boldsymbol{r}_\xi$:

- $e_\xi = 0, r_\xi = 0$ or $e_\xi = 1, r_\xi = 1$: Flipping positions r_ξ corresponds to setting a 0 in $\boldsymbol{r}+\boldsymbol{e}$ to 1. By Lemma 3, this does not change the ML decoding result, i.e., the oracle still returns False. The bit $\hat{e}_\xi$ is not flipped, i.e., we have $\hat{e}_\xi = r_\xi$, and we correctly obtain $\hat{e}_\xi = e_\xi$.

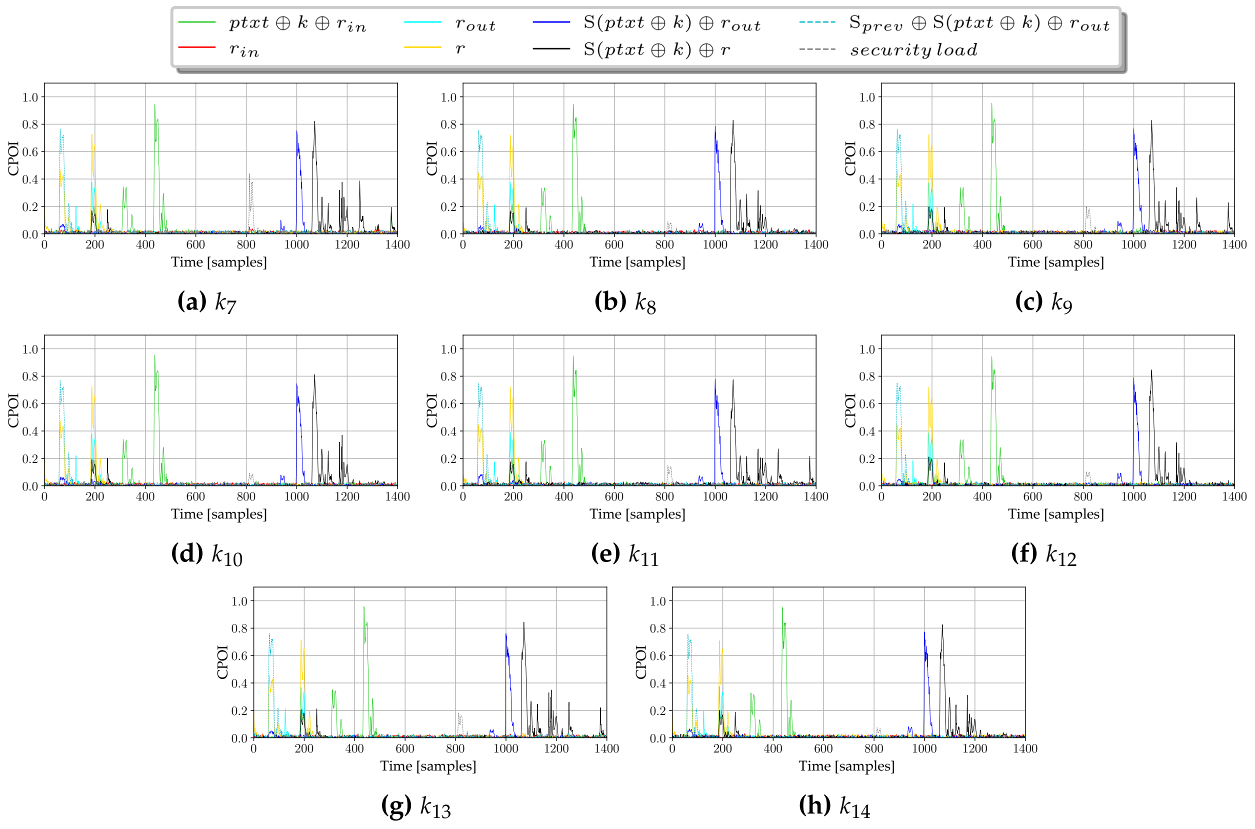
[3]Similarly to the previous case, this does not mean that the ML decoding result is necessarily $\hat{p}$, since the proof does not hold for $\check{p}$ and $\hat{p}+\check{p}+1$.

- $e_\xi = 0, r_\xi = 1$ or $e_\xi = 1, r_\xi = 0$: Flipping positions r_ξ corresponds to setting a 1 in $\boldsymbol{r} + \boldsymbol{e}$ to 0. By Lemma 3, this does change the ML decoding result to all-zero, i.e., the oracle now returns True. The bit $\hat{e}_\xi$ is flipped, i.e., we have $\hat{e}_\xi = r_\xi + 1$, and we correctly obtain $\hat{e}_\xi = e_\xi$.

We conclude that $\tilde{\boldsymbol{e}}_{\hat{\mathcal{I}} \cap \check{\mathcal{I}}} = \hat{\boldsymbol{e}}_{\hat{\mathcal{I}} \cap \check{\mathcal{I}}} = \boldsymbol{e}_{\hat{\mathcal{I}} \cap \check{\mathcal{I}}}$. This holds for any choice of $\hat{\mathcal{I}}$ and $\check{\mathcal{I}}$. The lemma statement follows from observing that the corresponding sets $\hat{\mathcal{I}} \cap \check{\mathcal{I}}$ partition the set of all positions $[0, q^m - 1]$. $\square$

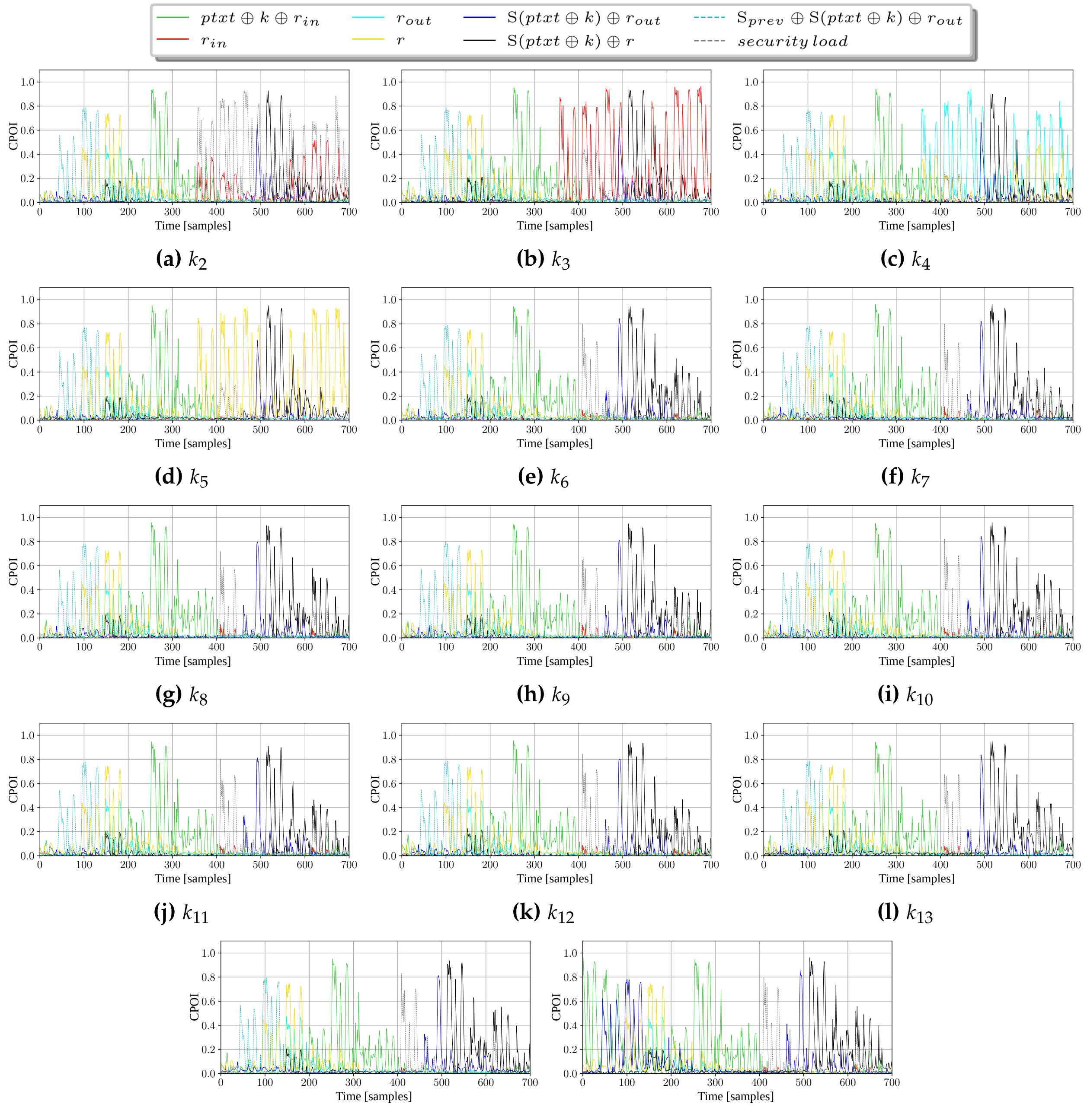
B CPOI Analysis of *ASCAD FIX* and *ASCAD VARIABLE*

For completeness, this chapter presents the leakage evaluation using the CPOI method for the remaining key bytes of *ASCAD VARIABLE* that are not shown in Fig. 6.2 as well as the results of all key bytes of *ASCAD FIX*. The results for *ASCAD VARIABLE* are shown in Fig. B.1 and for *ASCAD FIX* in Fig. B.2.



(a) k_7 **(b)** k_8 **(c)** k_9 **(d)** k_{10} **(e)** k_{11} **(f)** k_{12} **(g)** k_{13} **(h)** k_{14}

Figure B.1 CPOI analysis of the *ASCAD VARIABLE* dataset for the remaining key bytes not shown in Fig. 6.2.



(a) k_2 **(b)** k_3 **(c)** k_4

(d) k_5 **(e)** k_6 **(f)** k_7

(g) k_8 **(h)** k_9 **(i)** k_{10}

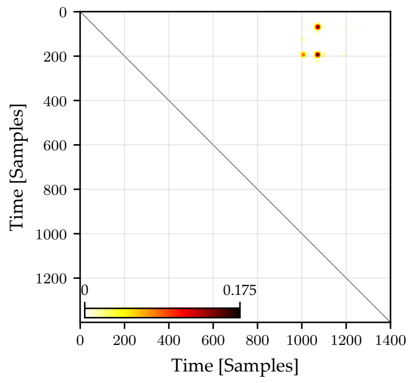
(j) k_{11} **(k)** k_{12} **(l)** k_{13}

(m) k_{14} **(n)** k_{15}

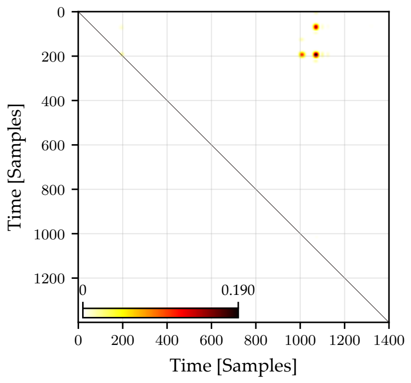
Figure B.2 CPOI analysis of the *ASCAD* FIX dataset.

C Second-Order Attack Results for *ASCAD VARIABLE*

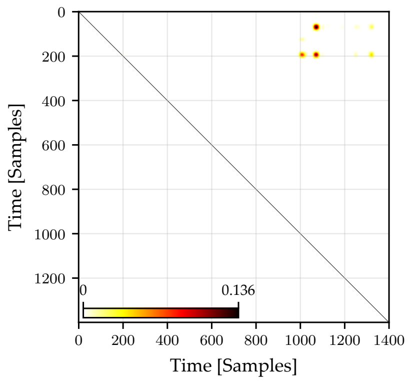
This chapter shows the second-order attack results for the remaining key bytes of *ASCAD* VARIABLE not shown in Fig. 6.4.



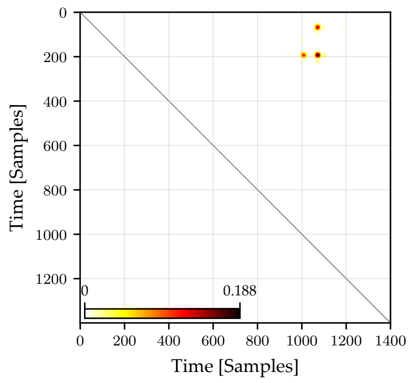
(a) k_7



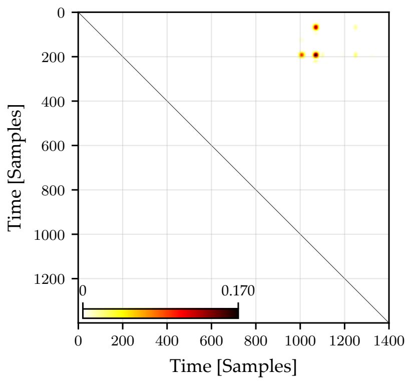
(b) k_8



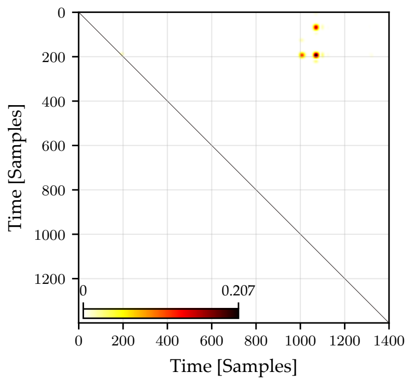
(c) k_9



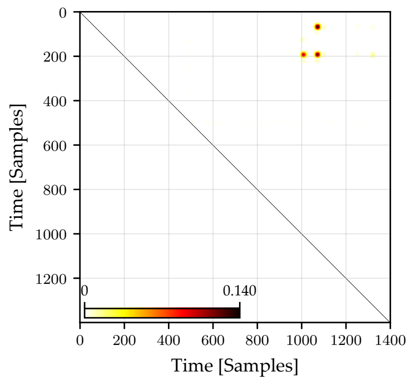
(d) k_{10}



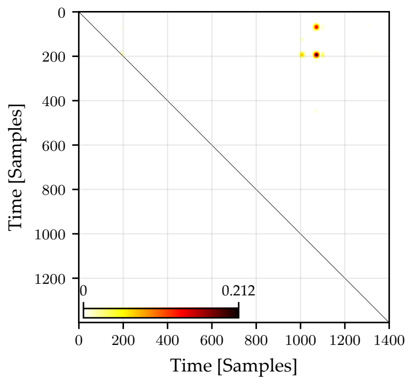
(e) k_{11}



(f) k_{12}



(g) k_{13}



(h) k_{14}

Figure C.1 Second-order attack results for the remaining key bytes of *ASCAD* VARIABLE not shown in Fig. 6.4.

Related Publications by the Author

[BFM+18] Konstantin Braun, Tim Fritzmann, Georg Maringer, Thomas Schamberger, and Johanna Sepúlveda. Secure and compact full NTRU hardware implementation. In *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 89–94, 2018.

[EST+22] Maximilian Egger, Thomas Schamberger, Lars Tebelmann, Florian Lippert, and Georg Sigl. A second look at the ASCAD databases. In Josep Balasch and Colin O'Flynn, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 75–99, Cham, 2022. Springer International Publishing.

[FSF+19] Tim Fritzmann, Thomas Schamberger, Christoph Frisch, Konstantin Braun, Georg Maringer, and Johanna Sepúlveda. Efficient hardware/software co-design for NTRU. In Nicola Bombieri, Graziano Pravadelli, Masahiro Fujita, Todd Austin, and Ricardo Reis, editors, *VLSI-SoC: Design and Engineering of Electronics Systems Based on New Computing Paradigms*, pages 257–280, Cham, 2019. Springer International Publishing.

[FVBBR+21] Tim Fritzmann, Michiel Van Beirendonck, Debapriya Basu Roy, Patrick Karl, Thomas Schamberger, Ingrid Verbauwhede, and Georg Sigl. Masked accelerators and instruction set extensions for post-quantum cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):414–460, Nov. 2021.

[GPK+21] Michael Gruber, Matthias Probst, Patrick Karl, Thomas Schamberger, Lars Tebelmann, Michael Tempelmeier, and Georg Sigl. DOMREP - an orthogonal countermeasure for arbitrary order side-channel and fault attack protection. *IEEE Transactions on Information Forensics and Security*, 16:4321–4335, 2021.

[HHP+21] Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine van Vredendaal. Chosen ciphertext k-trace attacks on masked CCA2 secure Kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):88–113, Aug. 2021.

[HPR+22] Anna-Lena Horlemann, Sven Puchinger, Julian Renner, Thomas Schamberger, and Antonia Wachter-Zeh. Information-set decoding with hints. In Antonia Wachter-Zeh, Hannes Bartz, and Gianluigi Liva, editors, *Code-Based Cryptography*, pages 60–83, Cham, 2022. Springer International Publishing.

[KSTS21] Alexander Kulow, Thomas Schamberger, Lars Tebelmann, and Georg Sigl. Finding the needle in the haystack: Metrics for best trace selection in unsupervised side-channel attacks on blinded RSA. *IEEE Transactions on Information Forensics and Security*, 16:3254–3268, 2021.

[RMB+23] Stefan Ritterhoff, Georg Maringer, Sebastian Bitzer, Violetta Weger, Patrick Karl, Thomas Schamberger, Jonas Schupp, and Antonia Wachter-Zeh. Fuleeca: A lee-based signature scheme. In Andre Esser and Paolo Santini, editors, *Code-Based Cryptography*, pages 56–83, Cham, 2023. Springer Nature Switzerland.

[SET23] Thomas Schamberger, Maximilian Egger, and Lars Tebelmann. Hide and Seek: Using occlusion techniques for side-channel leakage attribution in CNNs. In Jianying Zhou, Lejla Batina, Zengpeng Li, Jingqiang Lin, Eleonora Losiouk, Suryadipta Majumdar, Daisuke Mashima, Weizhi Meng, Stjepan Picek, Mohammad Ashiqur Rahman, Jun Shao, Masaki Shimaoka, Ezekiel Soremekun, Chunhua Su, Je Sen Teh, Aleksei Udovenko, Cong Wang, Leo Zhang, and Yury Zhauniarovich, editors, *Applied Cryptography and Network Security Workshops*, pages 139–158, Cham, 2023. Springer Nature Switzerland.

[SHR$^+$22] Thomas Schamberger, Lukas Holzbaur, Julian Renner, Antonia Wachter-Zeh, and Georg Sigl. A power side-channel attack on the Reed-Muller Reed-Solomon version of the HQC cryptosystem. In Jung Hee Cheon and Thomas Johansson, editors, *Post-Quantum Cryptography*, pages 327–352, Cham, 2022. Springer International Publishing.

[SMS19] Thomas Schamberger, Oliver Mischke, and Johanna Sepulveda. Practical evaluation of masking for NTRUEncrypt on ARM Cortex-M4. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 253–269, Cham, 2019. Springer International Publishing.

[SRSWZ21] Thomas Schamberger, Julian Renner, Georg Sigl, and Antonia Wachter-Zeh. A power side-channel attack on the CCA2-secure HQC KEM. In Pierre-Yvan Liardet and Nele Mentens, editors, *Smart Card Research and Advanced Applications*, pages 119–134, Cham, 2021. Springer International Publishing.

[USS$^+$20] Florian Unterstein, Marc Schink, Thomas Schamberger, Lars Tebelmann, Manuel Ilg, and Johann Heyszl. Retrofitting leakage resilient authenticated encryption to microcontrollers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):365–388, Aug. 2020.

Bibliography

[ABB$^+$20a] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Santosh Ghosh, Shay Gueron, Tim Güneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Jan Richter-Brockmann, Nicolas Sendrier, Jean-Pierre Tillich, Valentin Vasseur, and Gilles Zémor. Bit Flipping Key Encapsulation (BIKE) - Algorithm Specification. Submission to the NIST Post-Quantum Cryptography Standardization [Nat16], 2020. `https://bikesuite.org/`.

[ABB$^+$20b] Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Bas Westerbaan. SPHINCS+ - Algorithm Specification. Submission to the NIST Post-Quantum Cryptography Standardization [Nat16], 2020. `https://sphincs.org/`.

[ABC$^+$20] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, ngo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichett, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece - Algorithm Specification. Submission to the NIST Post-Quantum Cryptography Standardization [Nat16], 2020. `https://classic.mceliece.org`.

[ABD$^+$20] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber - Algorithm Specification. Submission to the NIST Post-Quantum Cryptography Standardization [Nat16], 2020. `https://pq-crystals.org/kyber/`.

[ABGV08] Ali C Atıcı, Lejla Batina, Benedikt Gierlichs, and Ingrid Verbauwhede. Power analysis on NTRU implementations for RFIDs: First results. In *The 4th Workshop on RFID Security–RFIDSec*, 2008.

[ACC$^+$20] Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Aaron Hutchinson, Amir Jalali, Koray Karabina, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik. Supersingular Isogeny Key Encapsulation (SIKE) - Algorithm Specification. Submission to the NIST Post-Quantum Cryptography Standardization [Nat16], 2020. `https://sike.org/`.

[ACÖG19a] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. DeepExplain: attribution methods for deep learning, 2019. `https://github.com/marcoancona/DeepExplain`.

[ACÖG19b] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. *Gradient-Based Attribution Methods*, pages 169–191. Springer International Publishing, Cham, 2019.

[AMBD$^+$18] Carlos Aguilar-Melchor, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Efficient encryption from random quasi-cyclic codes. *IEEE Transactions on Information Theory*, 64(5):3927–3943, 2018.

[APSQ06] C. Archambeau, E. Peeters, F. X. Standaert, and J. J. Quisquater. Template attacks in principal subspaces. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 1–14, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[AR21] Amund Askeland and Sondre Rønjom. A side-channel assisted attack on NTRU. Cryptology ePrint Archive, Report 2021/790, 2021. `https://ia.cr/2021/790`.

[BBD$^+$16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 116–129, New York, NY, USA, 2016. Association for Computing Machinery.

[BBF$^+$19] Hayo Baan, Sauvik Bhattacharya, Scott Fluhrer, Oscar Garcia-Morchon, Thijs Laarhoven, Rachel Player, Ronald Rietman, Markku-Juhani O. Saarinen, Ludo Tolhuizen, José Luis Torre-Arce, and Zhenfei Zhang. Round5 - Algorithm Specification. Submission to the NIST Post-Quantum Cryptography Standardization [Nat16], 2019. `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-2/submissions/Round5-Round2.zip`.

[BBM$^+$15] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10(7):1–46, 07 2015.

[BCE$^+$01] Daniel V. Bailey, Daniel Coffin, Adam Elbirt, Joseph H. Silverman, and Adam D. Woodbury. NTRU in constrained devices. In Çetin K. Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2001*, pages 262–272, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[BCH$^+$20] Shivam Bhasin, Anupam Chattopadhyay, Annelie Heuser, Dirmanto Jap, Stjepan Picek, and Ritu Ranjan Shrivastwa. Mind the portability: A warriors guide through realistic profiled side-channel analysis. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

[BCLvV17] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU Prime - Algorithm Specification. Submission to the NIST Post-Quantum Cryptography Standardization [Nat16], 2017. `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/NTRU_Prime.zip`.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Lecture Notes in Computer Science*, pages 16–29. Springer Berlin Heidelberg, 2004.

[BCS13] Daniel J. Bernstein, Tung Chou, and Peter Schwabe. McBits: Fast constant-time code-based cryptography. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013*, pages 250–272, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[BCS21] Olivier Bronchain, Gaëtan Cassiers, and François-Xavier Standaert. Give me 5 minutes: Attacking ASCAD with a single side-channel trace. Cryptology ePrint Archive, Paper 2021/817, 2021. `https://eprint.iacr.org/2021/817`.

[BDHD+19] Ciprian Băetu, F. Betül Durak, Loïs Huguenin-Dumittan, Abdullah Talayhan, and Serge Vaudenay. Misuse attacks on post-quantum cryptosystems. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 747–776, Cham, 2019. Springer International Publishing.

[BDK+20] Shi Bai, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium - Algorithm Specification. Submission to the NIST Post-Quantum Cryptography Standardization [Nat16], 2020. `https://pq-crystals.org/dilithium/`.

[BDMS22] Olivier Bronchain, François Durvaux, Loïc Masure, and François-Xavier Standaert. Efficient profiled side-channel analysis of masked implementations, extended. *IEEE Transactions on Information Forensics and Security*, 17:574–584, 2022.

[BGG+15] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-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*, pages 64–81, Cham, 2015. Springer International Publishing.

[BGP+10] Lejla Batina, Benedikt Gierlichs, Emmanuel Prouff, Matthieu Rivain, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Mutual information analysis: a comprehensive study. *Journal of Cryptology*, 24(2):269–291, October 2010.

[BIK+24] Elie Bursztein, Luca Invernizzi, Karel Král, Daniel Moghimi, Jean-Michel Picod, and Marina Zhang. Generalized power attacks against crypto hardware using long-range deep learning, 2024. `https://arxiv.org/abs/2306.07249`.

[BK] Cees-Bart Breunesse and Ilya Kizhvatov. Side-channel toolkit in Julia (Jlsca). `https://github.com/Riscure/Jlsca`.

[BLPR18] Ryad Benadjila, Victor Lomné, Emmanuel Prouff, and Thomas Roche. Secure AES128 encryption implementation for ATmega8515, 2018. `https://github.com/ANSSI-FR/secAES-ATmega8515/tree/master/src/Version1`.

[BPS+18] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Elenora Cagli, and Cécile Dumas. ASCAD (ANSSI SCA Database), 2018. `https://github.com/ANSSI-FR/ASCAD`.

[BPS+20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *J. Cryptogr. Eng.*, 10(2):163–188, 2020.

[CB23] Gaëtan Cassiers and Olivier Bronchain. SCALib: A side-channel analysis library. *Journal of Open Source Software*, 8(86):5196, 2023.

[CD23] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 423–447, Cham, 2023. Springer Nature Switzerland.

[CDH+19] Cong Chen, Oussama Danba, Jerey Hostein, Andreas Hülsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, and Zhenfei Zhang. NTRU - Algorithm Specification. Submission to the NIST Post-Quantum Cryptography Standardization [Nat16], 2019. `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-2/submissions/NTRU-Round2.zip`.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures. In Wieland Fischer and

Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 45–68, Cham, 2017. Springer International Publishing.

[CGTZ23] Jean-Sébastien Coron, François Gérard, Matthias Trannoy, and Rina Zeitoun. High-order masking of NTRU. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(2):180–211, Mar. 2023.

[CHWZ17] Cong Chen, Jeffrey Hoffstein, William Whyte, and Zhenfei Zhang. NTRUEncrypt - Algorithm Specification. Submission to the NIST Post-Quantum Cryptography Standardization [Nat16], 2017. `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/NTRUEncrypt.zip`.

[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*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[CK14] Omar Choudary and Markus G. Kuhn. Efficient template attacks. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications*, pages 253–270, Cham, 2014. Springer International Publishing.

[CK18] Marios O. Choudary and Markus G. Kuhn. Efficient, portable template attacks. *IEEE Transactions on Information Forensics and Security*, 13(2):490–501, 2018.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski, Çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[CZLG21] Pei Cao, Chi Zhang, Xiangjun Lu, and Dawu Gu. Cross-device profiled side-channel attack with unsupervised domain adaptation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):27–56, Aug. 2021.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In *Advances in Cryptology – EUROCRYPT 2014*, pages 423–440. Springer Berlin Heidelberg, 2014.

[DR20] Joan Daemen and Vincent Rijmen. *The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition*. Information Security and Cryptography. Springer, 2020.

[DR24] Loïc Demange and Mélissa Rossi. A provably masked implementation of BIKE key encapsulation mechanism. Cryptology ePrint Archive, Paper 2024/076, 2024. `https://eprint.iacr.org/2024/076`.

[DS16] François Durvaux and François-Xavier Standaert. From improved leakage detection to the detection of points of interests in leakage traces. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 240–262, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[Dwo15] Morris Dworkin. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD, 2015.

[EST$^+$22] Maximilian Egger, Thomas Schamberger, Lars Tebelmann, Florian Lippert, and Georg Sigl. A second look at the ASCAD databases. In Josep Balasch and Colin O'Flynn, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 75–99, Cham, 2022. Springer International Publishing.

[FDNG19] Farnoud Farahmand, Viet B. Dang, Duc Tri Nguyen, and Kris Gaj. Evaluating the potential for hardware acceleration of four NTRU-based key encapsulation mechanisms using software/hardware codesign. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography*, pages 23–43, Cham, 2019. Springer International Publishing.

[FGMDP$^+$18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):89–120, Aug. 2018.

[FHK$^+$20] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Porin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon - Algorithm Specification. Submission to the NIST Post-Quantum Cryptography Standardization [Nat16], 2020. `https://falcon-sign.info/`.

[FSF$^+$19] Tim Fritzmann, Thomas Schamberger, Christoph Frisch, Konstantin Braun, Georg Maringer, and Johanna Sepúlveda. Efficient hardware/software co-design for NTRU. In Nicola Bombieri, Graziano Pravadelli, Masahiro Fujita, Todd Austin, and Ricardo Reis, editors, *VLSI-SoC: Design and Engineering of Electronics Systems Based on New Computing Paradigms*, pages 257–280, Cham, 2019. Springer International Publishing.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GGJR11] Benjamin Jun Gilbert Goodwill, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, pages 115–136, 2011.

[GGP$^+$15] Cezary Glowacz, Vincent Grosso, Romain Poussier, Joachim Schüth, and François-Xavier Standaert. Simpler and more efficient rank estimation for side-channel security assessment. In Gregor Leander, editor, *Fast Software Encryption*, pages 117–129, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[GHJ$^+$22] Qian Guo, Clemens Hlauschek, Thomas Johansson, Norman Lahr, Alexander Nilsson, and Robin Leander Schröder. Don't reject this: Key-recovery timing attacks due to rejection-sampling in HQC and BIKE. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(3):223–263, Jun. 2022.

[GJ20] Qian Guo and Thomas Johansson. A new decryption failure attack against HQC. In *Advances in Cryptology – ASIACRYPT 2020*, pages 353–382. Springer International Publishing, 2020.

[GLG22] Guillaume Goy, Antoine Loiseau, and Philippe Gaborit. A new key recovery side-channel attack on HQC with chosen ciphertext. In Jung Hee Cheon and Thomas Johansson, editors, *Post-Quantum Cryptography*, pages 353–371, Cham, 2022. Springer International Publishing.

[GM10] Shuhong Gao and Todd Mateer. Additive fast fourier transforms over finite fields. *IEEE Transactions on Information Theory*, 56(12):6265–6272, 2010.

[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*, TIS '16, 2016.

[GP99] Louis Goubin and Jacques Patarin. DES and differential power analysis the "duplication" method. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems*, pages 158–172. Springer Berlin Heidelberg, 1999.

[HAS$^+$24] Suvadeep Hajra, Manaar Alam, Sayandeep Saha, Stjepan Picek, and Debdeep Mukhopadhyay. On the instability of softmax attention-based deep learning models in side-channel analysis. *IEEE Transactions on Information Forensics and Security*, 19:514–528, 2024.

[HCY19] Wei-Lun Huang, Jiun-Peng Chen, and Bo-Yin Yang. Power analysis on NTRU Prime. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):123–151, Nov. 2019.

[HDV20] Loïs Huguenin-Dumittan and Serge Vaudenay. Classical misuse attacks on NIST round 2 PQC. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *Applied Cryptography and Network Security*, pages 208–227, Cham, 2020. Springer International Publishing.

[HGG19] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Applications of machine learning techniques in side-channel attacks: a survey. *Journal of Cryptographic Engineering*, 10(2):135–162, 2019.

[HGG20] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Deep neural network attribution methods for leakage analysis and symmetric key recovery. In Kenneth G. Paterson and Douglas Stebila, editors, *Selected Areas in Cryptography – SAC 2019*, pages 645–666, Cham, 2020. Springer International Publishing.

[HGSSW03] Nick Howgrave-Graham, Joseph H. Silverman, Ari Singer, and William Whyte. NAEP: Provable security in the presence of decryption failures. Cryptology ePrint Archive, Paper 2003/172, 2003. `https://eprint.iacr.org/2003/172`.

[HGSW05] Nick Howgrave-Graham, Joseph H. Silverman, and William Whyte. Choosing parameter sets for NTRUEncrypt with NAEP and SVES-3. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, pages 118–135, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 341–371, Cham, 2017. Springer International Publishing.

[HHO20] Anh-Tuan Hoang, Neil Hanley, and Maire O'Neill. Plaintext: A missing feature for enhancing the power of deep learning in side-channel analysis? Breaking multiple layers of side-channel countermeasures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):49–85, Aug. 2020.

[HIM$^+$14] Johann Heyszl, Andreas Ibing, Stefan Mangard, Fabrizio De Santis, and Georg Sigl. Clustering algorithms for non-profiled single-execution attacks on exponentiations. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications*, pages 79–93, Cham, 2014. Springer International Publishing.

[HPP21] Julius Hermelink, Peter Pessl, and Thomas Pöppelmann. Fault-enabled chosen-ciphertext attacks on Kyber. In Avishek Adhikari, Ralf Küsters, and Bart Preneel, editors, *Progress in Cryptology – INDOCRYPT 2021*, pages 311–334, Cham, 2021. Springer International Publishing.

[HPR$^+$22] Anna-Lena Horlemann, Sven Puchinger, Julian Renner, Thomas Schamberger, and Antonia Wachter-Zeh. Information-set decoding with hints. In Antonia Wachter-Zeh, Hannes Bartz, and Gianluigi Liva, editors, *Code-Based Cryptography*, pages 60–83, Cham, 2022. Springer International Publishing.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe P. Buhler, editor, *Algorithmic Number Theory*, pages 267–288, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[HPS14] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *An Introduction to Mathematical Cryptography*. Springer New York, 2014.

[HPS$^+$17] Jeff Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, William Whyte, and Zhenfei Zhang. Choosing parameters for NTRUEncrypt. In Helena Handschuh, editor, *Topics in Cryptology – CT-RSA 2017*, pages 3–18, Cham, 2017. Springer International Publishing.

[HRSS17] Andreas Hülsing, Joost Rijneveld, John M. Schanck, and Peter Schwabe. NTRU-HRSS-KEM - Algorithm Specification. Submission to the NIST Post-Quantum Cryptography Standardization [Nat16], 2017. `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/NTRU_HRSS_KEM.zip`.

[HS00] Jeffrey Hoffstein and Joseph Silverman. Optimizations for NTRU. *Public-Key Cryptography and Computational Number Theory, De Gruyter Proceedings in Mathematics*, pages 77–88, 2000.

[IEE09] IEEE. IEEE standard specification for public key cryptographic techniques based on hard problems over lattices. *IEEE Std 1363.1-2008*, pages 1–81, 2009.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 463–481, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[JK95] L.L. Joiner and J.J. Komo. Decoding binary BCH codes. In *Proceedings IEEE Southeastcon '95. Visualize the Future*, pages 67–73, 1995.

[KAA21] Emre Karabulut, Erdem Alkim, and Aydin Aysu. Single-trace side-channel attacks on w-small polynomial sampling: With applications to NTRU, NTRU Prime, and CRYSTALS-DILITHIUM. In *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 35–45, 2021.

[Kai22] Tim Kaiser. Side-channel countermeasures for chosen-ciphertext attacks on HQC. Master's thesis, Technical University of Munich, October 2022.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA*, 2015. `http://arxiv.org/abs/1412.6980`.

[KFYF21] Kunihiro Kuroda, Yuta Fukuda, Kota Yoshida, and Takeshi Fujino. Practical aspects on non-profiled deep-learning side-channel attacks against AES software implementation with two types of masking countermeasures including RSM. In *Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security*, ASHES '21, page 29–40, New York, NY, USA, 2021. Association for Computing Machinery.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology — CRYPTO' 99. CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science (LNCS)*, pages 388–397. Springer, Berlin, Heidelberg, 1999.

[KLP$^+$22] Marina Krček, Huimin Li, Servio Paguada, Unai Rioja, Lichao Wu, Guilherme Perin, and Łukasz Chmielewski. Deep learning on side-channel analysis. In Lejla Batina, Thomas Bäck, Ileana Buhan, and Stjepan Picek, editors, *Security and Artificial Intelligence: A Crossdisciplinary Approach*, pages 48–71, Cham, 2022. Springer International Publishing.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[KRS19] Matthias J. Kannwischer, Joost Rijneveld, and Peter Schwabe. Faster multiplication in $\mathbb{Z}_{2^m}[x]$ on Cortex-M4 to speed up NIST PQC candidates. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 281–301, Cham, 2019. Springer International Publishing.

[KRSS] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. `https://github.com/mupq/pqm4`.

[KSTS21] Alexander Kulow, Thomas Schamberger, Lars Tebelmann, and Georg Sigl. Finding the needle in the haystack: Metrics for best trace selection in unsupervised side-channel attacks on blinded RSA. *IEEE Transactions on Information Forensics and Security*, 16:3254–3268, 2021.

[KUMH17] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[LBM11] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. Side channel attack: an approach based on machine learning. In *COSADE 2011 - Second InternationalWorkshop on Constructive Side-Channel Analysis and Secure Design*, 2011.

[LC04] Shu Lin and Daniel J. Costello. *Error Control Coding (Second Edition)*. Prentice-Hall, Inc., USA, 2004.

[LCGD18] Yann Le Corre, Johann Großschädl, and Daniel Dinu. Micro-architectural power simulator for leakage assessment of cryptographic software on ARM Cortex-M3 processors. In Junfeng Fan and Benedikt Gierlichs, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 82–98, Cham, 2018. Springer International Publishing.

[Lip20] Florian Lippert. Understanding deep learning for side-channel analysis. Master's thesis, Technical University of Munich, 2020.

[LLJ$^+$19] Xianhui Lu, Yamin Liu, Dingding Jia, Haiyang Xue, Jingnan He, Zhenfei Zhang, Zhe Liu, Hao Yang, Bao Li, and Kunpeng Wang. LAC - Algorithm Specification. Submission to the NIST Post-Quantum Cryptography Standardization [Nat16], 2019. `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-2/submissions/LAC-Round2.zip`.

[LSCH10] Mun-Kyu Lee, Jeong Eun Song, Dooho Choi, and Dong-Guk Han. Countermeasures against power analysis attacks for the NTRU public key cryptosystem. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E93.A(1):153–163, 2010.

[LZC$^+$21] Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Haining Lu. Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):235–274, Jul. 2021.

[MAB$^+$20] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jun Bos, Jean-Christophe Deneuville, Arnaud Dion, Philippe Gaborit, Jérôme Lacan, Edoardo Persichetti, Jean-Marc Robert, Pascal Véron, and Gilles Zémor. Hamming Quasi-Cyclic (HQC) - Algorithm Specification. Submission to the NIST Post-Quantum Cryptography Standardization [Nat16], 2020. `https://pqc-hqc.org/`.

[MDP19] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. Gradient visualization for general characterization in profiling attacks. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 145–167, Cham, 2019. Springer International Publishing.

[Mit97] Tom M. Mitchell. *Machine learning*. McGraw-Hill, New York, 1997.

[MM13] Amir Moradi and Oliver Mischke. On the simplicity of converting leakages from multivariate to univariate. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013*, pages 1–20, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Applied Cryptography Engineering*, pages 3–26, Cham, 2016. Springer International Publishing.

[MPW21] Ben Marshall, Dan Page, and James Webb. MIRACLE: MIcRo-ArChitectural Leakage Evaluation: A study of micro-architectural power leakage across many devices. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):175–220, Nov. 2021.

[MS77] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*, volume 16. Elsevier, 1977.

[MSS09] Amir Moradi, Mohammad Taghi Manzuri Shalmani, and Mahmoud Salmasizadeh. Dual-rail transition logic: A logic style for counteracting power analysis attacks. *Computers & Electrical Engineering*, 35(2):359–369, 2009.

[MSS13] Dominik Merli, Frederic Stumpf, and Georg Sigl. Protecting PUF error correction by codeword masking. Cryptology ePrint Archive, Paper 2013/334, 2013. `https://eprint.iacr.org/2013/334`.

[MWK$^+$22] Catinca Mujdei, Lennert Wouters, Angshuman Karmakar, Arthur Beckers, Jose Maria Bermudo Mera, and Ingrid Verbauwhede. Side-channel analysis of lattice-based post-quantum cryptography: Exploiting polynomial multiplication. *ACM Trans. Embed. Comput. Syst.*, nov 2022. Just Accepted.

[Nat16] National Institute of Standards and Technology - Computer Security Division. Post-Quantum Cryptography Standardization, 2016. `https://csrc.nist.gov/projects/post-quantum-cryptography`.

[Nat20] National Institute of Standards and Technology. NIST IR 8309: Status report on the second round of the NIST post-quantum cryptography standardization process, 2020. `https://doi.org/10.6028/NIST.IR.8309`.

[Nat22] National Institute of Standards and Technology. NIST IR 8413: Status report on the third round of the NIST post-quantum cryptography standardization process, 2022. `https://doi.org/10.6028/NIST.IR.8413-upd1`.

[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*, pages 529–545, 2006.

[PEC19] Guilherme Perin, Baris Ege, and Lukasz Chmielewski. Neural network model assessment for side-channel analysis. Cryptology ePrint Archive, Paper 2019/722, 2019. `https://eprint.iacr.org/2019/722`.

[Pet60] W. Peterson. Encoding and error-correction procedures for the Bose-Chaudhuri codes. *IRE Transactions on Information Theory*, 6(4):459–470, 1960.

[PGA$^+$23] Kostas Papagiannopoulos, Ognjen Glamočanin, Melissa Azouaoui, Dorian Ros, Francesco Regazzoni, and Mirjana Stojilović. The side-channel metrics cheat sheet. *ACM Comput. Surv.*, 55(10), feb 2023.

[PPHH] Thomas Poeppelmann, Peter Pessl, Daniel Heinz, and Julius Hermelink. Vorrichtung und Verfahren zum Entschlüsseln einer verschlüsselten Bitfolge. German Patent DE102021213560A1, 2023. `https://patents.google.com/patent/DE102021213560A1`.

[PPM$^+$23] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. SoK: Deep learning-based physical side-channel analysis. *ACM Comput. Surv.*, 55(11), feb 2023.

[PR07] Emmanuel Prouff and Matthieu Rivain. A generic method for secure SBox implementation. In *Information Security Applications*, pages 227–244. Springer Berlin Heidelberg, 2007.

[Pra62] E. Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

[PRB09] Emmanuel Prouff, Matthieu Rivain, and Regis Bevan. Statistical analysis of second order differential power analysis. *IEEE Transactions on Computers*, 58(6):799–811, 2009.

[PSB$^+$18] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cecile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. Cryptology ePrint Archive, Paper 2018/053, 2018. `https://eprint.iacr.org/2018/053`.

[PT20] Thales Bandiera Paiva and Routo Terada. A timing attack on the HQC encryption scheme. In Kenneth G. Paterson and Douglas Stebila, editors, *Selected Areas in Cryptography – SAC 2019*, pages 551–573, Cham, 2020. Springer International Publishing.

[PWP22a] Guilherme Perin, Lichao Wu, and Stjepan Picek. Exploring feature selection scenarios for deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):828–861, Aug. 2022.

[PWP22b] Guilherme Perin, Lichao Wu, and Stjepan Picek. I know what your layers did: Layer-wise explainability of deep learning side-channel analysis. Cryptology ePrint Archive, Paper 2022/1087, 2022. `https://eprint.iacr.org/2022/1087`.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In Isabelle Attali and Thomas Jensen, editors, *Smart Card Programming and Security*, pages 200–210, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[RBT18] Emmanuel Prouff Ryad Benadjila, Louiza Khati and Adrian Thillard. Hardened library for AES-128 encryption/decryption on ARM Cortex M4 achitecture, 2018. `https://github.com/ANSSI-FR/SecAESSTM32`.

[REB+21] Prasanna Ravi, Martianus Frederic Ezerman, Shivam Bhasin, Anupam Chattopadhyay, and Sujoy Sinha Roy. Will you cross the threshold for me? Generic side-channel assisted chosen-ciphertext attacks on NTRU-based KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):722–761, Nov. 2021.

[RLMI21] Thomas Roche, Victor Lomné, Camille Mutschler, and Laurent Imbert. A side journey to titan. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 231–248. USENIX Association, August 2021.

[Rot06] Ron Roth. *Introduction to Coding Theory*. Cambridge University Press, 2006.

[RPD09] Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-order masking and shuffling for software implementations of block ciphers. In *Lecture Notes in Computer Science*, pages 171–188. Springer Berlin Heidelberg, 2009.

[RSRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):307–335, Jun. 2020.

[RWPP21] Jorai Rijsdijk, Lichao Wu, Guilherme Perin, and Stjepan Picek. Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):677–707, Jul. 2021.

[SA08] François-Xavier Standaert and Cedric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, pages 411–425, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[Sch17] Thomas Schamberger. Towards secure post-quantum cryptography: Power analysis of NTRUEncrypt on ARM Cortex-M4. Master's thesis, Technical University of Munich, 2017.

[Sch18] John M. Schanck. A comparison of NTRU variants. Cryptology ePrint Archive, Report 2018/1174, 2018. `https://eprint.iacr.org/2018/1174`.

[Sen21] Nicolas Sendrier. Secure sampling of constant-weight words – application to BIKE. Cryptology ePrint Archive, Paper 2021/1631, 2021. `https://eprint.iacr.org/2021/1631`.

[SET23] Thomas Schamberger, Maximilian Egger, and Lars Tebelmann. Hide and Seek: Using occlusion techniques for side-channel leakage attribution in CNNs. In Jianying Zhou, Lejla Batina, Zengpeng Li, Jingqiang Lin, Eleonora Losiouk, Suryadipta Majumdar, Daisuke Mashima, Weizhi Meng, Stjepan Picek, Mohammad Ashiqur Rahman, Jun Shao, Masaki Shimaoka, Ezekiel Soremekun, Chunhua Su, Je Sen Teh, Aleksei Udovenko, Cong Wang, Leo Zhang, and Yury Zhauniarovich, editors, *Applied Cryptography and Network Security Workshops*, pages 139–158, Cham, 2023. Springer Nature Switzerland.

[SGSK16] Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. Not just a black box: Learning important features through propagating activation differences, 2016. `https://arxiv.org/abs/1605.01713`.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, oct 1997.

[SHR$^+$22] Thomas Schamberger, Lukas Holzbaur, Julian Renner, Antonia Wachter-Zeh, and Georg Sigl. A power side-channel attack on the Reed-Muller Reed-Solomon version of the HQC cryptosystem. In Jung Hee Cheon and Thomas Johansson, editors, *Post-Quantum Cryptography*, pages 327–352, Cham, 2022. Springer International Publishing.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *Cryptographic Hardware and Embedded Systems – CHES 2005*, pages 30–46. Springer Berlin Heidelberg, 2005.

[SM15] Tobias Schneider and Amir Moradi. Leakage assessment methodology. In *Cryptographic Hardware and Embedded Systems – CHES 2015*, page 495–513, Berlin, Heidelberg, 2015.

[Smi17] Leslie N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472, 2017.

[SMS19] Thomas Schamberger, Oliver Mischke, and Johanna Sepulveda. Practical evaluation of masking for NTRUEncrypt on ARM Cortex-M4. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 253–269, Cham, 2019. Springer International Publishing.

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

[SR16] Hermann Seuschek and Stefan Rass. Side-channel leakage models for RISC instruction set architectures from empirical data. *Microprocess. Microsystems*, 47:74–81, 2016.

[SR20] Jaime Sevilla and C. Jess Riedel. Forecasting timelines of quantum computing, 2020. `https://arxiv.org/abs/2009.05045`.

[SRSWZ21] Thomas Schamberger, Julian Renner, Georg Sigl, and Antonia Wachter-Zeh. A power side-channel attack on the CCA2-secure HQC KEM. In Pierre-Yvan Liardet and Nele Mentens, editors, *Smart Card Research and Advanced Applications*, pages 119–134, Cham, 2021. Springer International Publishing.

[Ste89] Jacques Stern. A method for finding codewords of small weight. In Gérard Cohen and Jacques Wolfmann, editors, *Coding Theory and Applications*, pages 106–113, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[SVCO$^+$10] François-Xavier Standaert, Nicolas Veyrat-Charvillon, Elisabeth Oswald, Benedikt Gierlichs, Marcel Medwed, Markus Kasper, and Stefan Mangard. The world is not enough: Another look on second-order DPA. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 112–129, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[SVZ14] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2014. `https://arxiv.org/abs/1312.6034`.

[SWM$^+$24] Travis L. Scholten, Carl J. Williams, Dustin Moody, Michele Mosca, William Hurley, William J. Zeng, Matthias Troyer, and Jay M. Gambetta. Assessing the benefits and risks of quantum computers, 2024. `https://arxiv.org/abs/2401.16317`.

[SWUH21] Marc Schink, Alexander Wagner, Florian Unterstein, and Johann Heyszl. Security and trust in open source security tokens. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):176–201, Jul. 2021.

[SZ15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA*, 2015.

[Tim19] Benjamin Timon. Non-profiled deep learning-based side-channel attacks with sensitivity analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):107–131, Feb. 2019.

[UXT$^+$21] Rei Ueno, Keita Xagawa, Yutaro Tanaka, Akira Ito, Junko Takahashi, and Naofumi Homma. Curse of re-encryption: A generic power/EM analysis on post-quantum KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):296–322, Nov. 2021.

[VCMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *Advances in Cryptology – ASIACRYPT 2012*, pages 740–757. Springer Berlin Heidelberg, 2012.

[vWWB11] Jasper G. J. van Woudenberg, Marc F. Witteman, and Bram Bakker. Improving differential power analysis by elastic alignment. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, pages 104–119, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[WAGP20] Lennert Wouters, Victor Arribas, Benedikt Gierlichs, and Bart Preneel. Revisiting a methodology for efficient CNN architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):147–168, Jun. 2020.

[WBB$^+$22] Guillaume Wafo-Tapa, Slim Bettaieb, Loïc Bidoux, Philippe Gaborit, and Etienne Marcatel. A practicable timing attack against HQC and its countermeasure. *Adv. Math. Commun.*, 16(3):621–642, 2022.

[WP20] Lichao Wu and Stjepan Picek. Remove some noise: On pre-processing of side-channel measurements with autoencoders. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):389–415, Aug. 2020.

[WWJ$^+$23] Lichao Wu, Yoo-Seung Won, Dirmanto Jap, Guilherme Perin, Shivam Bhasin, and Stjepan Picek. Ablation analysis for multi-device deep learning-based physical side-channel analysis. *IEEE Transactions on Dependable and Secure Computing*, pages 1–12, 2023.

[WWZ$^+$17] An Wang, Ce Wang, Xuexin Zheng, Weina Tian, Rixin Xu, and Guoshuang Zhang. Random key rotation: Side-channel countermeasure of NTRU cryptosystem for resource-limited devices. *Computers & Electrical Engineering*, 63:220–231, 2017.

[WZ21] Antonia Wachter-Zeh. Lecture notes for channel coding, 2021. `https://www.ce.cit.tum.de/en/lnt/teaching/lectures/channel-coding/`.

[XIU$^+$21a] Keita Xagawa, Akira Ito, Rei Ueno, Junko Takahashi, and Naofumi Homma. Fault-injection attacks against NIST's post-quantum cryptography round 3 KEM candidates.

In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 33–61, Cham, 2021. Springer International Publishing.

[XIU$^+$21b] Keita Xagawa, Akira Ito, Rei Ueno, Junko Takahashi, and Naofumi Homma. Fault-injection attacks against NIST's post-quantum cryptography round 3 KEM candidates. Cryptology ePrint Archive, Report 2021/840, 2021. `https://ia.cr/2021/840`.

[XPOZ23] Zhuang Xu, Owen Pemberton, David Oswald, and Zhiming Zheng. Reveal the invisible secret: Chosen-ciphertext side-channel attacks on NTRU. In *Smart Card Research and Advanced Applications*, pages 227–247. Springer International Publishing, 2023.

[YBBP23] Trevor Yap, Adrien Benamira, Shivam Bhasin, and Thomas Peyrin. Peek into the black-box: Interpretable neural network using sat equations in side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(2):24–53, Mar. 2023.

[YBP23] Trevor Yap, Shivam Bhasin, and Stjepan Picek. OccPoIs: Points of interest based on neural network's key recovery in side-channel analysis through occlusion. Cryptology ePrint Archive, Paper 2023/1055, 2023. `https://eprint.iacr.org/2023/1055`.

[YSPJ21] Honggang Yu, Haoqi Shan, Maximillian Panoff, and Yier Jin. Cross-device profiled side-channel attacks using meta-transfer learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 703–708, 2021.

[ZBHV19] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient CNN architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):1–36, Nov. 2019.

[ZCAW17] Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. In *International Conference on Learning Representations*, 2017.

[ZCD21] Xiaohan Zhang, Chi Cheng, and Ruoyu Ding. Small leaks sink a great ship: An evaluation of key reuse resilience of PQC third round finalist NTRU-HRSS. In Debin Gao, Qi Li, Xiaohong Guan, and Xiaofeng Liao, editors, *Information and Communications Security*, pages 283–300, Cham, 2021. Springer International Publishing.

[ZF14] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 818–833, Cham, 2014. Springer International Publishing.

[ZS20] Yuanyuan Zhou and François-Xavier Standaert. Deep learning mitigates but does not annihilate the need of aligned traces and a generalized ResNet model for side-channel attacks. *J. Cryptogr. Eng.*, 10(1):85–95, 2020.

[ZWW13] Xuexin Zheng, An Wang, and Wei Wei. First-order collision attack on protected NTRU cryptosystem. *Microprocess. Microsyst.*, 37(6–7):601–609, aug 2013.

Acronyms

ASCAD FIX	ASCAD fixed key
ASCAD VARIABLE	ASCAD variable key
ϵ-LRP	Layer-wise Relevance Backpropagation
AES	Advanced Encryption Standard
ANSSI	Agence nationale de la sécurité des systèmes d'information
ASCAD	ANSSI SCA Database
BCH	Bose-Chaudhuri-Hocquenghem
BIKE	Bit Flipping Key Encapsulation
CCA-2	Adaptive Chosen Ciphertext Attack
CMOS	Complementary Metal-Oxide-Semiconductor
CNN	Convolutional Neural Network
CPA	Correlation Power Analysis
CPOI	Correlation Point of Interest
DL	deep learning
DNN	Deep Neural Network
DOM	Domain-Oriented Masking
DPA	Differential Power Analysis
DUT	Device Under Test
ECC	Elliptic-Curve Cryptography
EM	electromagnetic
FO	Fujisaki-Okamoto
HD	Hamming distance
HQC	Hamming Quasi-Cyclic
HW	Hamming weight
ID	identity
IoT	Internet of Things
KEM	Key Encapsulation Mechanism
LDA	Linear Discriminant Analysis
LFSR	Linear Feedback Shift Register
MIA	Mutual Information Analysis
ML	maximum likelihood
ML	Machine Learning

MLP	Multi Layer Perceptron
NIST	National Institute of Standards and Technology
NTT	Number Theoretic Transform
PCA	Principal Component Analysis
PKE	Public Key Encryption Scheme
POI	point of interest
PRNG	Pseudo Random Number Generator
PUF	Physical Unclonable Function
QC-MDPC	Quasi-Cyclic Moderate Density Parity-Check
QCSD	Quasi-Cyclic Syndrome Decoding
ReLU	Rectified Linear Unit
RF	random forest
RM	Reed-Muller
RMSProp	Root Mean Square Propagation
RS	Reed-Solomon
RSA	Rivest-Shamir-Adleman
SCA	Side-Channel Analysis
SeLU	Scaled Exponential Linear Unit
SIMD	single instruction, multiple data
SNR	Signal-to-Noise Ratio
SPA	Simple Power Analysis
SVM	Support Vector Machine

Credits

Figs. 4.11, 4.14, 4.21 and 4.22 and Table 4.9 are reproduced with permission from Springer Nature from Schamberger/Holzbaur/Renner/Wachter-Zeh/Sigl: *A Power Side-Channel Attack on the Reed-Muller Reed-Solomon Version of the HQC Cryptosystem* published in *Post-Quantum Cryptography (PQCrypto)*, 2022 [SHR+22].

Figs. 6.1 to 6.7 and Tables 6.1 to 6.4 are reproduced with permission from Springer Nature from Egger/Schamberger/Tebelmann/Lippert/Sigl: *A Second Look at the ASCAD Databases* published in *Constructive Side-Channel Analysis and Secure Design (COSADE)*, 2022 [EST+22].

Figs. 7.1 to 7.10 are reproduced with permission from Springer Nature from Schamberger/Egger/Tebelmann: *Hide and Seek: Using Occlusion Techniques for Side-Channel Leakage Attribution in CNNs* published in *Applied Cryptography and Network Security Workshop (ACNS)*, 2023 [SET23].