Robert Primas BSc

Side-Channel Attacks on Efficient Lattice-Based Encryption

MASTER'S THESIS

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Stefan Mangard

Institute of Applied Information Processing and Communications (IAIK)

Second Supervisor
Dipl.-Ing. Peter Pessl Bsc

Graz, May 2017

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date

Signature

Acknowledgements

I would first like to thank my thesis advisors Dipl.-Ing. Peter Pessl and Univ.-Prof. Dipl.-Ing. Dr.techn. Stefan Mangard of the Institute of Applied Information Processing and Communications (IAIK) at Graz University of Technology. Whenever I ran into a trouble spot or had a question about my research or writing I could count on their continuous support. They consistently allowed this thesis to be my own work, but steered me in the right the direction whenever they thought I needed it.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Abstract

Asymmetric cryptography is currently facing the future threat that is quantum computing. A quantum algorithm, originally presented by Peter Shor, is capable of breaking current asymmetric cryptographic schemes in sub-exponential time. However, a powerful quantum computer is required for its efficient execution. Even though it appears that the construction of practical quantum computers will not happen within the next 20 years, the continuous progress is of concern to the cryptographic community.

To counteract the threat of quantum computing, the development of quantum computer secure asymmetric cryptographic schemes has become a hot research topic in recent years. Currently, there already exist many such schemes that are mainly based on Lattice-based Cryptography, Code-based Cryptography, and Supersingular Elliptic Curves. Especially lattices are promising since they allow the construction of efficient cryptographic schemes. While many schemes offer proper theoretical security claims, there is still a lack of research on their implementation security.

In this thesis, we present a first step towards implementation security of quantum computer secure asymmetric cryptographic schemes. We show a template-based single-trace side-channel attack on a microprocessor implementation of an asymmetric lattice-based encryption scheme. The focus of our side-channel attack is on the Number Theoretic Transform operation that is incorporated by many lattice-based cryptographic schemes for efficiency. Hence, similar variations of the presented attack may be applicable to attacking other lattice-based schemes. We combine our side-channel analysis with algebraic attacks. The Belief Propagation algorithm is used to marginalize leakage information from multiple leakage points, thereby reducing the noise level. Finally, we present a key recovery algorithm that can recover the full private key, given enough correctly determined intermediate values in the Number Theoretic Transform operation. We also perform our attack on a masked implementation that cannot be attacked via ordinary first-order differential power analysis or template attacks. In our evaluation, the success rate of our attack on the masked implementation is 1. To ensure generality and reproducibility we also evaluate our attack for simulated leakage with variable noise levels. Here, we achieve high success rates as long as the noise in the simulated leakage is not stronger than in the real leakage.

Keywords: Belief Propagation Algorithm, Implementation Security, Lattice-based Cryptography, Learning With Errors Problem, Number Theoretic Transform, Post-Quantum Cryptography, Side-Channel Attack, Single-Trace Attack

Kurzfassung

Die Asymmetrische Kryptographie steht momentan vor einem zukünftigen Problem, den Quantencomputern. Ein Quantenalgorithmus, ursprünglich beschrieben von Peter Shor, kann aktuelle asymmetrische kryptographische Verfahren in subexponentieller Zeit brechen. Jedoch ist ein leistungsstarker Quantencomputer für dessen effiziente Ausführung notwendig. Obwohl es derzeit danach aussieht, als ob die Entwicklung von leistungsstarken Quantencomputern nicht innerhalb der nächsten 20 Jahre passieren wird, ist der kontinuierliche Fortschritt ein Grund zur Sorge für die kryptographische Forschungsgemeinschaft.

Um dem Problem der Quantencomputer entgegenzuwirken ist die Entwicklung von quantencomputerresistentenen asymmetrischen kryptographischen Verfahren in den letzten Jahren zu einem regen Forschungsgebiet geworden. Es gibt schon viele solcher Verfahren die überwiegend auf gitterbasierter Kryptographie, codebasierter Kryptographie und supersingulären elliptischen Kurven basieren. Besonders vielversprechend sind die mathematischen Gitter, da sie die Konstruktion von effizienten kryptographischen Verfahren erlauben. Während viele dieser Verfahren den theoretischen Sicherheitsanforderungen entsprechen ist die Forschung an deren Implementierungssicherheit noch nicht sehr weit fortgeschritten.

In dieser Arbeit präsentieren wir einen ersten Schritt in Richtung Implementierungssicherheit für asymmetrische quantencomputerresistente kryptographische Verfahren. Wir zeigen eine auf Templates und Single-Traces basierende Seitenkanalattacke auf eine Mikroprozessor-Implementierung eines asymmetrischen gitterbasierten Verschlüsselungsverfahren. Der Fokus unserer Seitenkanalattacke liegt auf der Number Theoretic Transform, welche von vielen gitterbasierten kryptographischen Verfahren aus Effizienzgründen verwendet wird. Wir kombinieren unsere Seitenkanalattacke mit algebraischen Attacken. Der Belief Propagation Algorithmus wird benutzt um Seitenkanalinformationen aus verschiedenen Angriffspunken zu kombinieren und somit Rauschen zu reduzieren. Schlussendlich präsentieren wir einen Key-Recovery Algorithmus der den kompletten privaten Schlüssel finden kann, wenn genug Zwischenwerte aus der Number Theoretic Transform bekannt sind. Wir führen unsere Attacke auch auf eine maskierte Implementierunge aus, die nicht durch gewöhnliche Differential Power Analysis oder Template-Attacken der ersten Ordnung attackiert werden kann. In unserer Evaluierung erreicht unsere Attacke eine Erfolgsquote von 1. Um Generalität und Reproduzierbarkeit zu gewährleisten evaluieren wir unsere Attacke auch für simulierte Seitenkanalinformation mit variablem Rauschen. Hier erreichen wir hohe Erfolgsquoten solange das Rauschen in der simulierten Seitenkanalinformation nicht stärker als in der echten Seitenkanalinformation ist.

Stichwörter: Implementierungssicherheit, Gitterbasierte Kryptographie, Quantencomputerresistente Kryptographie, Seitenkanalattacke

Contents

List of Figures

Chapter 1

Introduction

Many currently used asymmetric cryptographic schemes are based on ideas that date back at least 30 years. While RSA encryption and the Diffie-Hellman key exchange were proposed in the mid 70's, Elliptic Curve Cryptography was originally proposed in the mid 80's, yet not widely used until 2005.

In the year 1994, Peter Williston Shor announced the ground breaking discovery of a quantum algorithm that could break most of the currently used asymmetric cryptographic schemes in sub-exponential time [73]. His algorithm however, comes with the catch that a sufficiently large practical quantum computer is required in order to allow efficient execution. Since the concept of quantum computers was mainly of purely theoretical interest in the mid 90's, his algorithm was not considered an immediate threat to asymmetric cryptography for a long time.

At the same time the research on quantum computers sees a slow but steady progress. Up until recently, there already exist several noteworthy achievements regarding the construction of practical quantum computers [22, 58, 64]. The presented constructions are still limited in many ways and hence cannot be used in meaningful applications yet. However, they do show that quantum computers are indeed possible to construct and not just a theoretic concept. Still, it is unclear when or even if the construction of quantum computers capable of performing Shor's algorithm can become reality.

Nevertheless, the continuous progress in the construction of practical quantum computers started to raise concern about the long-term security of existing cryptographic schemes in the research community. In the year 2016, these concerns were eventually backed up by national institutions like the NSA [56]. In their publication, they promote the transition from asymmetric cryptographic schemes like RSA and ECC to quantum computer resistant alternatives like Lattice-based Cryptography or Code-based Cryptography. While symmetric primitives are not affected by Shor's algorithm, there exists another quantum algorithm called "Grovers Search Algorithm" that can perform a trivial key search in $\mathcal{O}(2^{\frac{n}{2}})$ time [33]. The implication of this algorithm on symmetric primitives is however limited since a simple doubling of key size restores their original security claims. For asymmetric schemes, no such simple mitigation against Shor's algorithm is known to date. Hence, the search for quantum computer resistant asymmetric cryptographic schemes became a hot research topic.

There already exist various proposals for quantum computer secure asymmetric cryptographic schemes like signatures [35], public-key encryption [49], and many more [5, 27, 36, 10]. Some of those proposals are over 30 years old but only got little attention since they are comparably inefficient and quantum computer resistance was not valued back then.

While there are no official standards yet, some of the proposals are already tested in practice, e.g. in TLS handshakes by Google [11]. To date the most promising candidates are based on Lattice-based Cryptography, Code-based Cryptography or Supersingular Elliptic Curve Cryptography.

There already exist many publications regarding the theoretical security of these new cryptographic schemes. However, in terms of implementation security, many schemes lack sufficient analysis. Most published implementation proposals so far mainly consider runtime efficiency, memory efficiency, or space efficiency in case of hardware implementations. Having said that, a more profound security analysis of the implementation is usually missing.

In this thesis, we want to provide a first step in this important but yet still vastly unexplored topic. We show an implementation attack on one of the candidates for future quantum computer secure asymmetric encryption schemes. For this purpose we have chosen to attack an asymmetric encryption scheme which is based on Lattice-based Cryptography, more precisely the Ring-Learning With Errors Problem. This encryption scheme was originally proposed by Lyubashevsky et al. [49] and features low runtime, RSA-like key sizes, and limited provable security. The scheme is currently one of the most promising candidates for future quantum computer secure asymmetric encryption schemes.

More precisely, we perform a single-trace side-channel attack on a microprocessor implementation of the Number Theoretic Transform operation, as a part of decryption operation. The Number Theoretic Transform is one of the main building block of virtually all efficient lattice-based cryptographic schemes. As a consequence, our analysis is of value to a multitude of other cryptographic schemes that incorporate a Number Theoretic Transform. We also consider implementations that feature dedicated side-channel countermeasures like masking. A masked implementation cannot be attacked via ordinary first-order DPA or template attacks. However, an attack that solely relies on single-trace side-channel information can be used to circumvent such a countermeasure. Ultimately, we show that single-trace attacks are indeed possible for lattice-based cryptographic schemes.

Organization of the thesis

We first cover related work in Chapters 2-5. Our attack is then described in Chapter 6. We discuss our results in Chapters 7-8.

In Chapter 2, we give an overview of Lattice-based Cryptography. We explain the mathematical concept of lattices and how they can be used to build cryptographic schemes. Since the main goal of this thesis is an attack on a lattice-based encryption scheme, we explain the construction of such an encryption scheme in more detail. We also give a short discussion on the current state of lattice-based cryptographic schemes.

We attack the microprocessor implementation using side-channel information. In Chapter 3, we present implementation attacks, mention different types of implementation attacks, and explain how they can be used to attack HW/SW implementations of cryptographic schemes. We also mention common countermeasures against implementation attacks and how they can be circumvented.

In Chapter 4, we describe algebraic side-channel attacks. These attacks attempt to efficiently combine leakage information from multiple leakage points, thereby lowering the noise level of the side-channel measurements. In this thesis we use a method called “Soft Analytical Side-Channel Attacks” for this very purpose.

Soft analytical side-channel attacks use an algorithm called Belief Propagation. Given a graph like representation of a function, this algorithm can be used to calculate marginal distributions of the function's intermediate values. A comprehensive description of the Belief Propagation algorithm is given in Chapter 5.

In Chapter 6, we describe our main contribution of this thesis, i.e. an attack on a RLWE decryption operation. The attack description is split up into three steps, an initial side-channel attack step, a subsequent post processing step incorporating the Belief Propagation algorithm, and a final key recovery step. We also state some intermediate results of the individual steps.

The results of our attack are then stated in Chapter 7. Apart from showing an attack on a real microprocessor implementation, we also show an attack evaluation for more generic simulated side-channel information.

In Chapter 8, we conclude our work. We state the implications of our attack for existing proposals of encryption schemes and mention possible countermeasures. Finally, we suggest possible future work based on our presented attack.

Chapter 2

Lattice-based Cryptography

This chapter covers the most important aspects of Lattice-based Cryptography and gives a description of an efficient lattice-based asymmetric encryption scheme. The presented encryption scheme is one of the candidates for future post-quantum asymmetric encryption schemes and is based on a mathematical structure called *Lattice*. Lattices have proven to be quite useful in the field of cryptography since they allow to build a variety of cryptographic schemes featuring many beneficial properties that currently used cryptographic schemes usually do not provide.

Section 2.1 formally introduces the concept of lattices. Section 2.2 gives an overview of the state-of-the-art in the field of Lattice-based Cryptography. The security of all lattice-based cryptographic schemes relies on the hardness of certain problems that are defined on lattices. In Section 2.3, two problems are presented that are directly based on lattices. While those two problems are not directly used as a basis for building cryptographic schemes, there exist related problems that can be used for building cryptographic schemes. Two such commonly used lattice related problems are presented in Section 2.4. Many cryptographic schemes based on regular lattices have the drawback of high runtime or large key size requirements. In order to get around this problem, so-called *ideal* lattices have been proposed. Ideal lattices feature some additional structure that can be used to build more efficient, yet still secure, cryptographic schemes. Section 2.5 explains the concept of ideal lattices and shows how a problem related to ideal lattices is used to build the efficient encryption scheme which is attacked in this thesis. Since many lattice-based cryptographic schemes require a source of samples drawn from a discrete Gaussian distribution, Section 2.6 gives an overview of the state-of-the-art in the design of discrete Gaussian samplers.

2.1 Lattices

Definition 2.1.1. A lattice $\mathcal{L}$ defined by a vector basis $\mathbf{A}$ is a set of points in n-dimensional space with a periodic structure. Every point in the lattice can be described by an integer combination of the n linearly independent basis vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n$ that fully determine the lattice:

$$\mathcal{L}(\mathbf{A}) = \mathcal{L}\begin{pmatrix} \vdots & \vdots & \vdots \\ \mathbf{a}_1 & \cdots & \mathbf{a}_n \\ \vdots & \vdots & \vdots \end{pmatrix} = \left\{ \sum_{i=1}^{n} x_i \mathbf{a}_i : x_i \in \mathbb{Z} \right\}$$

In Figure 2.1, an example of a two dimensional lattice is shown. It is easy to see that any point in the lattice can be reached by a combination of either the black or red vector basis. In fact, there is an infinite number of possible vector basis for any one lattice. Of particular interest for cryptographic applications are so-called *q-ary* lattices. A q-ary lattice, for some integer q, is a lattice that contains a vector $\mathbf{x}$ if and only if $\mathbf{x}$ mod q is also in the lattice. Unless stated otherwise, all lattices in this thesis are assumed to be q-ary.

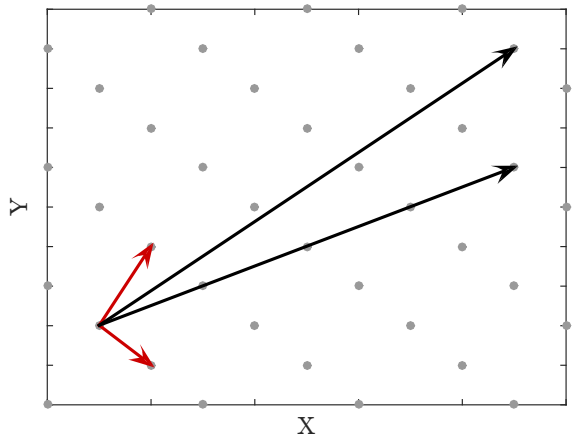


Figure 2.1: A 2D lattice with two possible vector basis (**red** and **black**).

2.2 Lattice-based Cryptography

Lattice-based Cryptography is a term for cryptographic constructions based on problems in lattices. The first break-through in this field was accomplished by Ajtai in 1996 when he introduced the first public-key encryption scheme based on a lattice problem [3]. In fact, in the field of number theory, lattices have been studied for much longer. Mathematicians like Lagrange [42] and Gauss [38] have already discovered the hardness of certain lattice-based problems in the late 18^{th} century. Yet, no proposals for lattice-based cryptographic schemes were made until 1996.

Some of the great promises of Lattice-based Cryptography are the resistance against quantum computers, provable security, as well as worst case hardness. Quantum computer resistance is a favorable property as there exists currently no widely used quantum computer resistant asymmetric cryptographic scheme. While quantum computers are only a minor threat for current symmetric cryptographic schemes, it was shown that there exist quantum computer algorithms that can break RSA or ECC-based cryptographic schemes in polynomial time [73]. It is not known yet whether or when quantum computers can be constructed in such a way that they become a real threat to current asymmetric cryptographic schemes. Nevertheless, quantum computer resistance is a property worth mentioning since there is continuous progress in the construction of more efficient and larger quantum computers [22, 58, 64]. Especially the quantum computer resistance in combination with increasing practicality lead to an increased research activity in Lattice-based Cryptography in the past ten years.

Many currently used asymmetric cryptographic schemes are based on the average-case difficulty of some problem. This means that for a given problem, like factorization,

there exist instances of the problem that are harder/easier than others. One example of a simple factorization problem instance is the factorization of even numbers. Average-case hardness is not ideal because the security of a cryptographic scheme depends on the actual parameterization of the underlying problem. Worst-case hardness, on the other hand, ensures that every instance of a problem is equally hard. In other words, given an algorithm that solves any one problem instance efficiently, the algorithm is guaranteed to also solve any other problem instance efficiently. Therefore, the parameterization of the problem does not affect the security of the cryptographic scheme as long as all requirements of the encryption schemes are met.

There already exist several proposals of lattice-based cryptographic schemes covering many different cryptographic applications like hash functions [48], signatures [35], public-key encryption [49], and many more. All lattice-based cryptographic schemes are built on top of one of several lattice-related problems which are assumed to be hard, even for quantum computers. Lattice-related problems are not directly defined on lattices but can be reduced to real lattice problems. In the next section, problems are presented that are defined directly on lattices. Section 2.4 will then present lattice-related problems than can be used to build cryptographic schemes.

2.3 Lattice Problems

This section presents two problems that play a major role in terms of provable security in Lattice-based Cryptography. Almost every lattice-based cryptographic construction is based on the assumption that one of the following problems is hard. While there exists both a search version and a decision version for each of the presented problems, in this section only the corresponding search variants are presented.

Notation. We use lower-case bold symbols to indicate that a variable $\mathbf{s}$ is a vector or polynomial. Upper-case bold symbols indicate that a variable $\mathbf{A}$ is a matrix. We further denote a dot product of the two vectors $\mathbf{s}, \mathbf{t}$ by $\langle \mathbf{s}, \mathbf{t} \rangle$.

2.3.1 Shortest Vector Problem

Definition 2.3.1. Given a lattice $\mathcal{L}$ defined by n linearly independent, uniformly random basis vectors $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_n]$, the Shortest Vector Problem (SVP) asks to find a vector $\mathbf{x} \in \mathcal{L}$ with length equal to the true shortest vector in $\mathcal{L}$:

$$\|\mathbf{x}\| = \lambda(\mathcal{L}),$$

where $\lambda(\mathcal{L})$ denotes the length of the true shortest vector:

$$\lambda(\mathcal{L}) = \min_{\mathbf{v} \in \mathcal{L}/\{0\}} \|\mathbf{v}\|$$

Usually, this problem does not require the solver to find the exact solution but a solution vector $\mathbf{x} \in \mathcal{L}$ with length of at most some polynomial approximation factor γ of the true shortest vector:

$$\|\mathbf{x}\| < \gamma\lambda(\mathcal{L})$$

This γ-approximation version of SVP is often denoted as SVP_γ. A visual interpretation for SVP in a two dimensional lattice is shown in Figure 2.2.

SVP is rather easy in the case when the lengths of the given $\mathbf{a}_1, \ldots, \mathbf{a}_n$ are already close to the true shortest vector. Since $\mathbf{a}_1, \ldots, \mathbf{a}_n$ are chosen uniformly at random, the probability of an easy problem instance is negligible for lattices of a reasonable dimension. In fact, Ajtai has shown that SVP is NP-hard [3]. The best known results for solving SVP are based on the LLL-algorithm [43] and HKZ-basis reductions [55]. However, none of those approaches can be used to solve SVP in sub-exponential time.

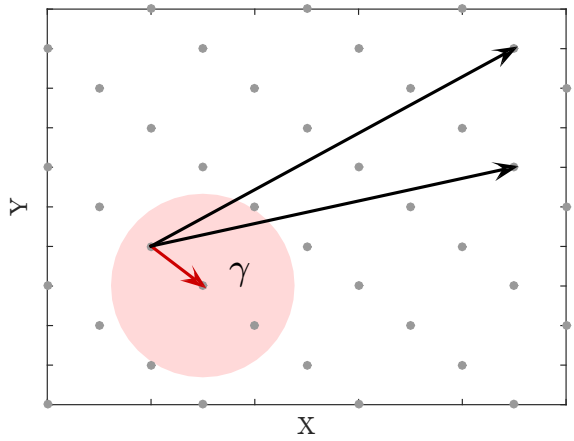


Figure 2.2: SVP: Given a vector base (**black**), find the shortest vector (**red**) or a γ-approximation (SVP_γ).

2.3.2 Closest Vector Problem

Definition 2.3.2. Given a lattice $\mathcal{L}$ defined by n linearly independent, uniformly random basis vectors $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_n]$ and a challenge vector $\mathbf{y}$, the Closest Vector Problem (CVP) asks to find a lattice point with minimal distance from the challenge vector:

$$\underset{x}{\operatorname{argmin}} \|\mathbf{Ax} - \mathbf{y}\|$$

Similarly to SVP, there also exists a γ-approximation version CVP_γ, where it is sufficient to find a solution vector $\mathbf{x}$, such that the distance from the challenge vector is at most a polynomial factor γ:

$$\|\mathbf{Ax} - \mathbf{y}\| < \gamma$$

A visual interpretation of CVP is presented in Figure 2.3.

There is a strong correspondence between CVP_γ and SVP_γ as there exist reductions in both directions [21]. While the used norm in SVP and CVP is usually the euclidean norm, any other norm can be used as well.

2.4 Lattice Related Problems

This section presents two problems that are used as a basis for many lattice-based cryptographic schemes. Both problems offer a reduction to a hard lattice problem. For the second problem we also state the corresponding decision problem as it is used in many cryptographic schemes.

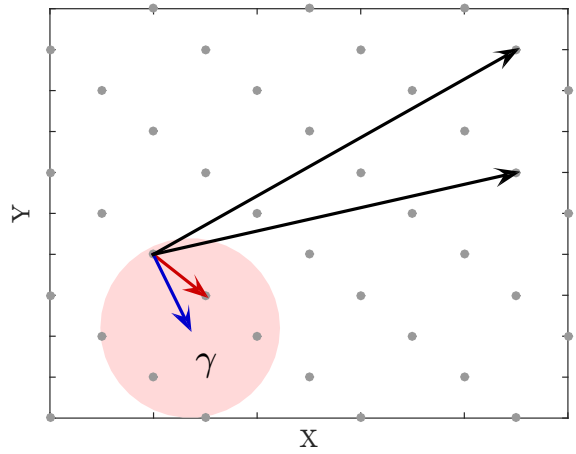


Figure 2.3: CVP: Given a vector base (**black**) and a challenge vector (**blue**), find the closest vector (**red**) or a γ-approximation (CVP_γ).

2.4.1 Shortest Integer Solution Problem

The Shortest Integer Solution Problem (SIS) was first proposed by Ajtai et al. [3] in 1996. It is primarily used for the construction of lattice-based one-way functions.

Definition 2.4.1. Given a lattice $\mathcal{L}$ defined by n linearly independent, uniformly random basis vectors $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_n]$, a constraint parameter β and a modulus q, SIS asks to find a non-zero vector $\mathbf{x}$, such that a subset of $\mathbf{A}$ sums up to zero:

$$\mathbf{A}\mathbf{x} = 0 \mod q$$

In order to rule out trivial solutions, the following constraints have to be met as well:

$$\|\mathbf{x}\| \leq \beta < q$$

Without the requirement $\|\mathbf{x}\| \leq \beta$, SIS can be solved efficiently by using Gaussian Elimination. The requirement $\beta < q$ ensures that the trivial solution $x = (q, 0, \ldots, 0)$ is not possible.

Ajtai has shown that solving SIS is secure in the average case if SVP_γ is hard in the worst case scenario [3]. Currently, there are already several proposals for using SIS to build one-way functions [3], collision resistant hash functions [48], or signature schemes [35].

2.4.2 Learning with Errors Problem

Definition 2.4.2. Given a lattice $\mathcal{L}$ defined by n linearly independent and uniformly random basis vectors $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_n]$, a discrete Gaussian error distribution $\mathcal{X}$, a modulus q and an arbitrary number j of LWE-tuples with the following structure:

$$\begin{aligned}
(\ \mathbf{a}_{i_1}\ ,\ b_1\ ) &= (\ \mathbf{a}_{i_1}\ ,\ \langle \mathbf{a}_{i_1}, \mathbf{s} \rangle + e_1\ ) \mod q \\
&\vdots \\
(\ \mathbf{a}_{i_j}\ ,\ b_j\ ) &= (\ \mathbf{a}_{i_j}\ ,\ \langle \mathbf{a}_{i_j}, \mathbf{s} \rangle + e_j\ ) \mod q,
\end{aligned}$$

where $i_x \in [1, \ldots, n]$, $e_x \in \mathbb{Z}_q$ is sampled from $\mathcal{X}$ and $\mathbf{s} \in \mathbb{Z}_q^n$ is uniformly random, the search variant of the Learning with Errors Problem (LWE) requires the solver to output

$\mathbf{s}$. The coefficient-wise operation $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i$ can also be written as matrix-vector multiplication for n pseudo random bits $\mathbf{b} = [b_1, \ldots, b_n]$:

$$\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e},$$

Beside the search variant of LWE, there also exists a decision variant. While Regev et al. has shown that both variants are equivalent [66], only the decision variant is currently used for proving the security of cryptographic schemes. The decision variant of LWE requires the solver to distinguish with non-negligible advantage between LWE-tuples and tuples $(\mathbf{a}_i \ , \ b')$ where b' is distributed uniformly random:

$$\begin{aligned} (\mathbf{a}_{i_1} \ , \ b'_1) \ \text{ vs. } \ (\mathbf{a}_{i_1} \ , \ b_1) \ &= \ (\mathbf{a}_{i_1} \ , \ \langle \mathbf{a}_{i_1}, \mathbf{s} \rangle + e_1) \mod q \\ &\vdots \\ (\mathbf{a}_{i_j} \ , \ b'_j) \ \text{ vs. } \ (\mathbf{a}_{i_j} \ , \ b_j) \ &= \ (\mathbf{a}_{i_j} \ , \ \langle \mathbf{a}_{i_j}, \mathbf{s} \rangle + e_j) \mod q \end{aligned}$$

The assumption here is that even though the $b_1, \ldots, b_j$ are not distributed uniformly random given $\mathbf{a}_{i_1}, \ldots, \mathbf{a}_{i_j}$, an attacker is not able to distinguish them from truly uniformly random values $b'_1, \ldots, b'_j$.

The LWE problem turned out to be a quite versatile basis for cryptographic applications. Besides public-key encryption schemes there are proposals for oblivious transfer protocols [62], identity-based encryption [78, 30, 14, 2], leakage-resilient encryption [4, 7, 25], and fully homomorphic encryption [29]

The hardness assumption of LWE is based on results from Regev et al. [66] and Peikert et al. [60]. They show that there exist reductions to a variant of SVP_γ, which itself is proven to be a hard lattice problem.

One of the main drawbacks of LWE-based encryption schemes is that their runtime as well as key size requirements are substantially higher than what we are used to from ECC or RSA based schemes. In practice, standard LWE encryption schemes would require key sizes of up to one megabit which is impractical for everyday usage such as in TLS-handshakes. The large key sizes are a result of the space needed to store the lattice basis themselves. The high runtime is the result of time consuming vector operations that have to be performed for every bit of the cipher text.

2.5 Efficient Implementations

This section presents ideal lattices as a way to significantly improve the practicality of lattice-based cryptographic schemes. Section 2.5.1 explains the concept of ideal lattices and mentions the differences to general lattices. One of the advantages of using ideal lattices is that we can replace time consuming vector operations in e.g. LWE, by more efficient polynomial multiplications. For this purpose an FFT-like operation called *Number Theoretic Transform* (NTT) is commonly used. A description of the NTT is given in Section 2.5.2. In Section 2.5.3 an adapted version of the LWE problem is defined that is based on ideal lattices. A description of an efficient encryption scheme based on ideal lattices and efficient polynomial multiplication is then provided in Section 2.5.4.

Notation. We use $\mathbf{x} * \mathbf{y}$ to denote a point-wise multiplication and $\mathbf{x} \cdot \mathbf{y}$ to denote a polynomial multiplication. A superscript tilde symbol indicates that a variable $\widetilde{\mathbf{x}}$ is the NTT transformed of $\mathbf{x}$.

2.5.1 Ideal Lattices

Ideal lattices differ from regular lattices in that they contain additional structure. While the vectors of regular lattices are chosen uniformly at random, vectors of ideal lattices are *nega-cyclic* shifted versions of each other. In this respect, a nega-cyclic shift means an element-wise rotation of a vector with a subsequent negation of the first element. In order to better illustrate the difference between regular and ideal lattices, Figure 2.4 shows examples for both lattice types. Ideal lattices have the favorable property that the whole lattice is completely defined by one vector. In other words, the space complexity of storing a lattice basis can be reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$.

Besides reduced memory consumption, ideal lattices also allow us to use faster arithmetic. This is due to the fact that ideal lattices can be interpreted as ideals in a specific finite ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(f)$. The polynomial f is hereby required to be monic, irreducible, and to have a norm of specific size. f is usually set to $x^n + 1$ for performance reasons. n has to be a power of two and the prime q is chosen such that $q \equiv 1 \bmod 2n$. Given an ideal lattice, time consuming matrix multiplication operations can be replaced by more efficient polynomial multiplications. More precisely, the matrix multiplication $\mathbf{As}$, as used in LWE, can be replaced by the polynomial multiplication $\mathbf{a} \cdot \mathbf{s}$ in $\mathcal{R}_q$. While such a polynomial multiplication in a finite ring can still be time consuming due to the required reduction step, a more efficient implementation is possible due to the NTT operation. A detailed description of the NTT algorithm is given in the next section.

1	5	9	8
2	2	0	7
5	7	1	4
4	8	3	6

(a) Regular lattice basis with uniformly random vectors (e.g. LWE)

1	-4	-5	-2
2	1	-4	-5
5	2	1	-4
4	5	2	1

(b) Ideal lattice basis with nega-cyclic vectors (e.g. RLWE)

Figure 2.4: Regular and ideal lattices

2.5.2 Number Theoretic Transform

The NTT is an algorithm that is at the core of many efficient lattice-based cryptographic schemes. It basically allows us to perform efficient polynomial multiplication, similar to the Fast Fourier Transform (FFT), yet in a specific finite ring. We recall that the n-point FFT can be used to efficiently evaluate an n-degree polynomial in the n-th root of unity and therefore perform the polynomial multiplication $\mathbf{c} = \mathbf{a} \cdot \mathbf{b}$ by calculating

$$\mathbf{c} = \mathrm{IFFT}_{\omega_n}(\ \mathrm{FFT}_{\omega_n}(\ \mathbf{a}\ )\ *\ \mathrm{FFT}_{\omega_n}(\mathbf{b})\ )$$

in time $\mathcal{O}(n \log n)$. Contrary to the FFT, in the NTT the roots of unity are taken from a finite ring. Consequently, the polynomial multiplication is performed in a finite ring

$\mathcal{S}_q = \mathbb{Z}_q[x]/[\cdot]$. By simply replacing the complex roots in an FFT by the primitive n-th roots of unity in $\mathbb{Z}_q$ we can perform efficient polynomial multiplication in the ring $\mathcal{S}_q = \mathbb{Z}_q[x]/(x^n - 1)$. A primitive n-th root of unity for n, $q \geq 2$, is a solution x to the equation:

$$x^n \equiv 1 \mod q,$$

where n is additionally the smallest such exponent for a choice of x. Thus we can calculate $\mathbf{c} = \mathbf{a} \cdot \mathbf{b}$ in $\mathcal{S}_q$ like the following:

$$\mathbf{c} = \mathrm{INTT}_{\omega_n}(\ \mathrm{NTT}_{\omega_n}(\ \mathbf{a}\ )\ *\ \mathrm{NTT}_{\omega_n}(\mathbf{b})\ )$$

The adaption of the NTT from the ring $\mathcal{S}_q$ to the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$, which is used in almost every lattice-based cryptographic scheme, can be accomplished by exploiting relation between their primitive roots. In fact, by performing an initial scaling of the input polynomial by the $2n$-th primitive root of unity ω_{2n} as well as a scaling of the the output by inverse exponents of ω_{2n} we can perform polynomial multiplication in $\mathcal{R}_q$:

$$a'_i \ \text{ of } \ \mathbf{a}' = a_i \cdot \omega_{2n}^i$$
$$b'_i \ \text{ of } \ \mathbf{b}' = b_i \cdot \omega_{2n}^i$$

$$\mathbf{c}' = \mathrm{INTT}_{\omega_{2n}}(\,\mathrm{NTT}_{\omega_{2n}}(\,\mathbf{a}'\,)\ *\ \mathrm{NTT}_{\omega_{2n}}(\,\mathbf{b}'\,)\,)$$
$$c_i \ \text{ of } \ \mathbf{c} = c'_i \cdot n^{-1}\omega_{2n}^{-i}$$

An iterative description of the NTT, taken from [17], is given in Algorithm 1. The factor ω, as used in line 7 is called *twiddle factor*. The general structure of the NTT is similar to the FFT. The same optimizations that apply for the FFT, such as in the Cooley-Tukey algorithm, can be used in the NTT. Hence, an NTT operation can also be implemented by using a butterfly as illustrated in Figure 2.5. In order to build an NTT for polynomial with more coefficients, a butterfly network with recursive structure can be used. Such a butterfly network is illustrated in Figure 2.6.

The inverse transformation INTT is almost identical to the forward transformation. The only implementation difference is that the n-th ($2n$-th) primitive root of unity is replaced by its inverse.

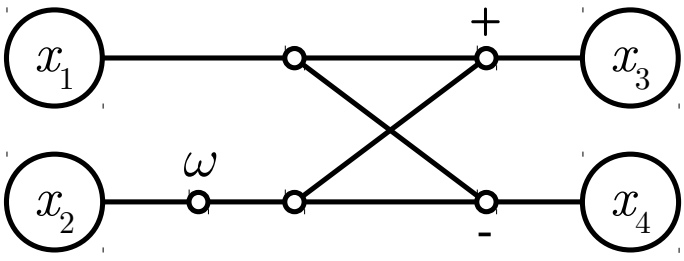


Figure 2.5: A single NTT butterfly

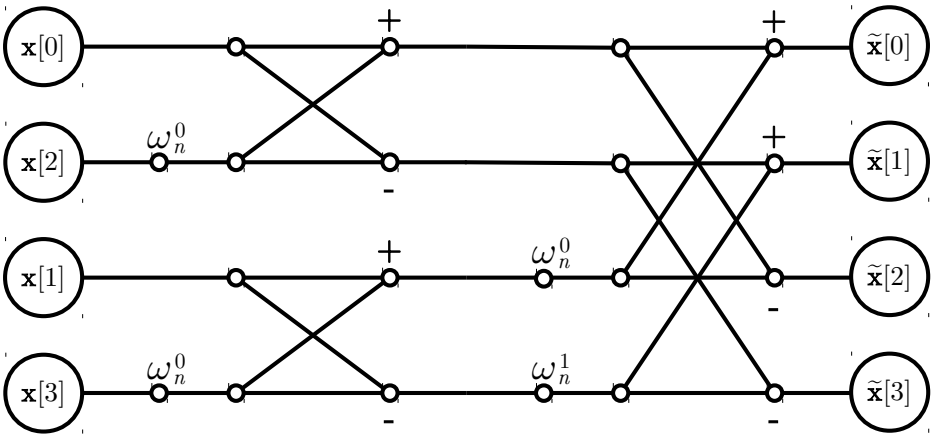


Figure 2.6: A 4-coefficient NTT

Algorithm 1: Iterative $n-$coefficient NTT

Input:

$\mathbf{x}$	Polynomial $\in \mathbb{Z}_q^n$
ω_n	$n-$th primitive root of unity $\in \mathbb{Z}_q$

Output:

$\widetilde{\mathbf{x}}$	Polynomial $\in \mathbb{Z}_q^n = \text{NTT}(\mathbf{x})$

$\widetilde{\mathbf{x}} \leftarrow \text{BitReverse}(\mathbf{x})$
for $m = 2$ to n by $m = 2m$ **do**
  $\omega_m \leftarrow \omega_n^{n/m}$
  $\omega \leftarrow 1$
  for $j = 0$ to $\frac{m}{2} - 1$ **do**
    for $k = 0$ to $n - 1$ by m **do**
      $t \leftarrow \omega \cdot \widetilde{\mathbf{x}}[k + j + \frac{m}{2}]$
      $u \leftarrow \widetilde{\mathbf{x}}[k + j]$
      $\widetilde{\mathbf{x}}[k + j] \leftarrow u + t$
      $\widetilde{\mathbf{x}}[k + j + \frac{m}{2}] \leftarrow u - t$
    end for
    $\omega \leftarrow \omega \cdot \omega_m$
  end for
end for

2.5.3 Ring-Learning with Errors Problem

Definition 2.5.1. Given an n-dimensional ideal lattice $\mathcal{L}$ defined by nega-cylic shifts of a basis vector $\mathbf{a}$, a discrete Gaussian error distribution $\mathcal{X}$, a ring $R_q = \mathbb{Z}_q[x]/(f)$, a modulus q, and RLWE-tuples with the following structure:

$$(\mathbf{a} , \mathbf{b}) = (\mathbf{a} , \mathbf{a} \cdot \mathbf{s} + \mathbf{e}) \in R_q,$$

where $\mathbf{e} \in \mathbb{Z}_q^n$ is sampled from $\mathcal{X}$ and $\mathbf{s} \in \mathbb{Z}_q^n$ is uniformly random, the search variant of the Ring-Learning With Errors Problem (RLWE) requires the solver to output $\mathbf{s}$. Note

that all basis vectors of $\mathcal{L}$ are completely defined by $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{Z}_q^n$. The remaining basis vectors are implicitly given by $\mathbf{a}_i = (a_i, \ldots, a_n, -a_1, \ldots, -a_{i-1})$.

Similarly to LWE, there also exists a decision variant of RLWE that requires the solver to distinguish with non-negligible advantage between RLWE-tuples and tuples $(\mathbf{a} , \mathbf{b}')$ where $\mathbf{b}' \in \mathbb{Z}_q^n$ is uniformly random:

$$(\mathbf{a} , \mathbf{b}') \text{ vs. } (\mathbf{a} , \mathbf{b}) = (\mathbf{a} , \mathbf{a} \cdot \mathbf{s} + \mathbf{e}) \in R_q$$

In standard LWE, one LWE-sample is generated by $b'_x = \langle \mathbf{a}_x, \mathbf{s} \rangle + e_x$. This is quite inefficient since time consuming matrix operations have to be performed for every sample. Contrary to that, in RLWE, n samples can be generated in one run by calculating $\mathbf{b} = \mathbf{a} \cdot \mathbf{s} + \mathbf{e} \in R_q$. The multiplication $\mathbf{a} \cdot \mathbf{s}$ is no longer a dot product but a polynomial multiplication in a finite ring. The runtime of the multiplication, and therefore also for the encryption, can be reduced to $\mathcal{O}(n \log n)$ by using the NTT operation:

$$\begin{gathered} \widetilde{\mathbf{a}} = \mathrm{NTT}(\mathbf{a}) , \quad \widetilde{\mathbf{s}} = \mathrm{NTT}(\mathbf{s}) , \quad \widetilde{\mathbf{e}} = \mathrm{NTT}(\mathbf{e}) \\ \widetilde{\mathbf{b}} = \widetilde{\mathbf{a}} * \widetilde{\mathbf{s}} + \widetilde{\mathbf{e}} \\ \mathbf{b} = \mathrm{INTT}(\widetilde{\mathbf{b}}) \end{gathered}$$

Usually, variables are kept in the NTT domain during and after encryption. As a consequence, less of the expensive (I)NTT invocations are needed, especially during decryption.

Even though RLWE is a huge step towards the construction of practical lattice-based cryptographic schemes, it comes with one caveat. While LWE-based schemes feature security proofs that ensure NP-hardness, there exists no such classical reduction of RLWE to an NP-hard problem at the present time. The only currently known way to reduce RLWE to an NP-hard problem is using a so-called *quantum* reduction [49]. Quantum reductions require a quantum computer in order to be efficient, i.e., run in sub-exponential time. Finding a way to reduce RLWE to an NP-hard problem using a classical reduction is still an open research topic. Figure 2.7 shows an overview of the current state of provable security for LWE and RLWE cryptographic schemes.

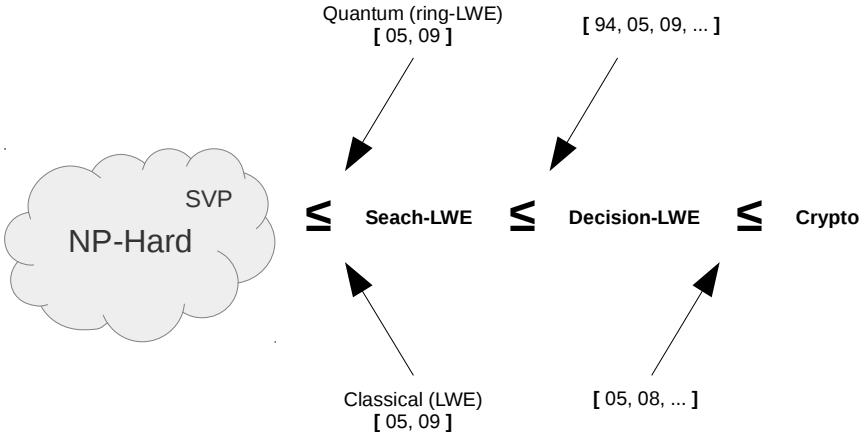


Figure 2.7: RLWE reduction chain. The numbers in the brackets denote the year of the discovery of a reduction or the significant improvement over previous reductions.

2.5.4 Ring-Learning With Errors Encryption Scheme

This section gives a detailed description of an RLWE-based encryption scheme. The encryption scheme was proposed by from Lyubashevsky et al. [49], our notation is based on Roy et al. [71].

The scheme is parameterized by the tuple $\mathcal{U}, \mathcal{X}, \sigma, q$ as well as an encoder/decoder-function. $\mathcal{U}$ and $\mathcal{X}$ are probability distributions with $\mathcal{U}$ being a uniform distribution and $\mathcal{X}$ being a discrete Gaussian distribution with zero mean and standard deviation σ. All calculations are performed modulo q. The encoder-function maps a binary input vector $\mathbf{m}$ of size n into $\bar{\mathbf{m}} \in R_q$ by performing an element-wise multiplication with $\frac{q}{2}$. Similarly, the decoder-function maps an input $\bar{\mathbf{m}} \in R_q$ to a binary vector $\mathbf{m}$ by decoding values in the interval $(-\frac{q}{4}, \frac{q}{4}\ ]$ to '0' and otherwise to '1'. The encryption scheme additionally requires an n-dimensional ideal lattice that is defined by the polynomial $\mathbf{a} \in R_q$. $\mathbf{a}$ is sampled from $\mathcal{U}$ and can be publicly shared.

- **KeyGen($\mathbf{a}$):** Sample the polynomials $\mathbf{r}_1 \in R_q$ from $\mathcal{X}$ and $\mathbf{r}_2 \in \{0,1\}^n$. Calculate $\mathbf{p} = \mathbf{r}_1 - \mathbf{a} \cdot \mathbf{r}_2$. Perform an NTT transformation of the three polynomials $\mathbf{a}, \mathbf{p}$ and $\mathbf{r}_2$ to get $\tilde{\mathbf{a}}, \tilde{\mathbf{p}}$ and $\tilde{\mathbf{r}}_2$.

  The public key is $(\tilde{\mathbf{a}}, \tilde{\mathbf{p}})$.

  The private key is $\tilde{\mathbf{r}}_2$.

  All keys are stored in their NTT transformed version to reduce the amount of NTT transformations during encryption and decryption. $\mathbf{r}_1$ is no longer needed after the key generation is done.

- **Encrypt($\mathbf{m}, \tilde{\mathbf{a}}, \tilde{\mathbf{p}}$):** A message $\mathbf{m}$ is encoded to $\bar{\mathbf{m}}$ by using the encoder-function. Three polynomials $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3 \in \mathbb{R}^n$ are sampled from $\mathcal{X}$. The cipher text $(\tilde{\mathbf{c}}_1, \tilde{\mathbf{c}}_2)$ is computed as:

$$
\begin{aligned}
\tilde{\mathbf{e}}_1 &\leftarrow \mathrm{NTT}(\mathbf{e}_1) \\
\tilde{\mathbf{e}}_2 &\leftarrow \mathrm{NTT}(\mathbf{e}_2) \\
\widetilde{\mathbf{c}}_1 &\leftarrow \widetilde{\mathbf{a}} * \widetilde{\mathbf{e}}_1 + \widetilde{\mathbf{e}}_2 \\
\widetilde{\mathbf{c}}_2 &\leftarrow \widetilde{\mathbf{p}} * \widetilde{\mathbf{e}}_1 + \mathrm{NTT}(\mathbf{e}_3 + \bar{\mathbf{m}})
\end{aligned}
$$

- **Decrypt($\bar{\mathbf{m}}, \tilde{\mathbf{r}}_2$):** Compute $\mathbf{m}' = \mathrm{INTT}(\tilde{\mathbf{c}}_1 * \tilde{\mathbf{r}}_2 + \tilde{\mathbf{c}}_2)$ and recover original message $\mathbf{m}$ from $\mathbf{m}'$ by using the decoder-function.

The parameter values used in this thesis are $q = 7681$, $\sigma = 20.39$ and $n = 256$. They were proposed by Göttert et al. [32] for the application in hardware designs. The approximate security level of an RLWE problem with said parameterization is about 128 bit. There exist many more proposals for RLWE encryption schemes and various security levels. Table 2.1 contains a list of several RLWE parameter proposals.

It is easy to see that the decryption operation is a lot faster than the encryption operation. In fact, since the only runtime relevant operations (excluding the discrete Gaussian sampling) are the NTT transformations, the decryption operation runtime is only about $\frac{1}{3}$ of the encryption operation runtime. When compared to the runtime of ECC encryption schemes it turns out that RLWE encryption/decryption operations can be faster than their ECC counterparts [71].

Authors	n	q	σ	Public Key Size	Security Level
Lindner et al. [45]	128	2053	1.07	1536 bits	$\ll 128$ bit
	192	4093	1.41	2304 bits	< 128 bit
	256	4093	1.32	3072 bits	≈ 128 bit
	320	4093	1.27	3840 bits	> 128 bit
Göttert et al. [32]	256	7681	1.80	3328 bits	≈ 128 bit
	512	12289	1.93	7168 bits	≈ 256 bit
Micciancio et al. [52]	136	2003	2.07	1496 bits	≥ 128 bit
	214	16381	1.17	2996 bits	≥ 128 bit

Table 2.1: Various parameterization proposals for RLWE-based encryption schemes

The correctness of the RLWE encryption scheme can be shown by simple substitution. The decrypted, yet still encoded, $\mathbf{m}'$ does not fully correspond to the encoded plain text $\bar{\mathbf{m}}$ because of the way how the discrete Gaussian noise is used during encryption. However, the parameters of $\mathcal{X}$ are chosen such that the decoding function has a negligible probability of decoding errors when recovering $\mathbf{m}$ from $\mathbf{m}'$:

$$\begin{aligned}
\mathbf{m}' &= \mathbf{c}_1 \cdot \mathbf{r}_2 + \mathbf{c}_2 \\
&= (\mathbf{a} \cdot \mathbf{e}_1 + \mathbf{e}_2) \cdot \mathbf{r}_2 + \mathbf{b} \cdot \mathbf{e}_1 + \mathbf{e}_3 + \bar{\mathbf{m}} \\
&= (\mathbf{a} \cdot \mathbf{e}_1 + \mathbf{e}_2) \cdot \mathbf{r}_2 + (\mathbf{e_t} - \mathbf{a} \cdot \mathbf{r}_2) \cdot \mathbf{e}_1 + \mathbf{e}_3 + \bar{\mathbf{m}} \\
&\approx (\mathbf{a} \cdot \mathbf{e}_1 + \underbrace{\cancel{\mathbf{e}_2}}_{\text{small}}) \cdot \mathbf{r}_2 + (\underbrace{\cancel{\mathbf{r}_1}}_{\text{small}} - \mathbf{a} \cdot \mathbf{r}_2) \cdot \mathbf{e}_1 + \underbrace{\cancel{\mathbf{e}_3}}_{\text{small}} + \bar{\mathbf{m}} \\
&\approx \mathbf{a} \cdot \mathbf{e}_1 \cdot \mathbf{r}_2 - \mathbf{a} \cdot \mathbf{e}_1 \cdot \mathbf{r}_2 + \bar{\mathbf{m}} \\
&\approx \boldsymbol{\epsilon} + \bar{\mathbf{m}}
\end{aligned}$$

2.6 Discrete Gaussian Samplers

Lattice-based cryptographic schemes often require samples from a discrete Gaussian distribution. In the case of an encryption scheme based on LWE, discrete Gaussian distributed errors terms are necessary for the generation of a pseudo random cipher text. This section is meant to give an overview of the current state in the design of discrete Gaussian samplers. First, a discrete Gaussian distribution is formally defined in Section 2.6.1. In Section 2.6.2, the three most important properties for practical samplers are stated. Finally, in Section 2.6.3, a comparison of state-of-the-art sampler designs is shown.

2.6.1 Discrete Gaussian Distribution

The continuous Gaussian distribution is parameterized by standard deviation $\sigma > 0$, mean $\mu \in \mathbb{R}$, and is defined as follows:

Definition 2.6.1. Let X be a random variable on $\mathbb{R}$, then for $x \in \mathbb{R}$ we have:

$$\Pr(X = x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

The discrete version of the Gaussian distribution over $\mathbb{Z}$ with mean 0 and standard deviation $\sigma > 0$ is then defined as follows:

Definition 2.6.2. Let X be a random variable on $\mathbb{Z}$ then:

$$\Pr(X = x) = \frac{1}{S} e^{-x^2/2\sigma^2},$$

where S is a normalization factor and is approximately $\sigma\sqrt{2\pi}$.

2.6.2 Properties of Gaussian Samplers

The three most important properties are:

Precision. As mentioned previously, the security of certain lattice-based cryptographic schemes is based on reductions to known hard problems. However, some of those reduction steps are only valid under the assumption that true discrete Gaussian distributions are used [26]. In practice, true discrete Gaussian samplers are intractable as they would require infinite arithmetic precision [75]. A common approach is to circumvent this problem by using samplers with negligible statistical distance to the true distribution. The analysis of sampler parameterizations in recent literature [44, 45, 46] indicates that a statistical distance of 2^{-90} to the true distribution can be considered as sufficiently precise. It should be noted though that some authors claim that already half of the aforementioned precision is sufficient for almost all applications [72].

Efficiency. New cryptographic schemes need to compete with existing schemes in terms of runtime. In the case of hardware implementations, also area requirements need to be considered. In many lattice-based cryptographic schemes the only runtime relevant operations are polynomial multiplications and the creation of error polynomials using discrete Gaussian sampling. As a matter of fact, most of the runtime is usually used by the sampler [75]. The development of efficient discrete Gaussian samplers is thus an important step towards the removal of potential bottlenecks in lattice-based cryptographic schemes. In recent years, several proposals to increase the performance of discrete Gaussian samplers were made [61, 65, 70]. In the next section, a performance comparison of FPGA based designs is shown.

Security. Last but not least, since discrete Gaussian samplers are a core building block of lattice-based cryptographic schemes, their implementation needs to be secure as well. Currently, many proposals for discrete Gaussian samplers focus primarily on precision and efficiency which make them susceptible to the leakage of side-channel information [37]. This information leakage may then be used compromise the whole cryptographic scheme [12]. There exist proposals for side-channel resistant discrete Gaussian samplers [70]. However, it was shown that the proposed side-channel countermeasures may not be effective enough to prevent practical attacks as of now [63].

2.6.3 Comparison of state-of-the-art Implementations

In Table 2.6.3, a comparison of four state-of-the-art implementations of discrete Gaussian samplers is shown. All implementations feature sufficient precision for application in

lattice-based encryption schemes. Signature schemes require a larger σ, thus more precision and are not meant to be implemented using on of these samplers. To allow a fair comparison all designs are FPGA implementations.

Authors	Algorithm	Design Goal	Slices	Clocks/Sample
Pöppelmann et al.[65]	Bernoulli	Area	37	144
Roy et al. [70]	Knuth-Yao	Performance	35	$\approx 2.5^*$
	Knuth-Yao	Area	30	17
	Knuth-Yao	SCA Resistance	52	$\approx 1.6^*$

Table 2.2: Comparison of discrete Gaussian sampler implementations. Values indicated by * are averaged over multiple successive executions.

When looking at the bare numbers, designs based on the Knuth-Yao algorithm seem to have taken the lead at the present time. And indeed, while the Bernoulli-based design was able to outperform older designs based on Knuth-Yao [75], recent improvements by Roy et al.[70] result in a significant performance gain for their designs. Of particular interest for practical applications is the implementation that features side-channel resistance. Implementations of the Knuth-Yao algorithm leak a tremendous amount of timing information. Roy et al. attempt to overcome this problem by introducing random shuffling. This countermeasure comes at the price of a significantly increased area consumption.

Chapter 3

Side-Channel Attacks

This chapter discusses side-channel attacks as a method for the exploitation of information that is leaked by hardware and software implementations during normal operation.

Section 3.1 explains the basic principle of side-channel attacks and mentions a variety of different types of side-channel attacks. Of particular interest in this thesis are side-channel attacks based on timing and power consumption. Section 3.2 explains how differences in runtime can be used to learn about processed data. In Section 3.3, multiple ways of exploiting a device's power consumption are presented. The most common side-channel countermeasure strategies are discussed in Section 3.4.

3.1 Overview

The key idea behind side-channel attacks is that all implementations of cryptographic algorithms leak some kind of information about the processed data through side-channels. In the past, side-channels such as power consumption, timing, cache access patterns, electromagnetic radiation, acoustic waves etc. have been exploited. Side-channel leakage may contain information about a secret that is hidden inside the attacked device. Usually, the hidden secret corresponds to the secret key which is used by the implemented cryptographic algorithm.

Back in 1996, Kocher et al. presented the first side-channel attack based on a timing side-channel in OpenSSL's RSA implementation [39]. Following Kocher's pioneering work, many authors presented various types of side-channel attacks on various implementations like Timing Attacks [39, 24], Cache Attacks [9, 34], Acoustic Attacks [28], Power Analysis Attacks [40] and Electromagnetic Attacks [47]. Based on their invasiveness, side-channel attacks can be separated into the following groups:

Invasive Attacks. A side-channel attack is said to be invasive if any modifications can be made to the attacked device. As a first step, invasive attacks usually consist of a depackaging of the attacked device. By doing so, protective metal layers are removed and the circuity of the attacked device becomes more accessible to the attacker. The next step then usually consists of an ordinary side-channel attack, thus the exploitation of leakage information. In some cases the circuitry of the attacked device is modified as well. For this purpose tools like laser beams or probing stations are necessary. In general, invasive attacks are rather rare as they are cost intensive and require special equipment. The circuit modifications are also error prone and multiple attack devices may be necessary until

the intended attack can be performed. Examples of invasive side-channel attacks are [41, 6]

Semi-Invasive Attacks. Semi-invasive side-channel attacks typically also start with a depackaging of the attacked device. However, no additional modifications to the device's circuitry are performed here. The main goal in semi-invasive attacks is to infer information about data that is located or processed by the device. One of the difficulties is the exact localization of the target data in the circuitry. Only then side-channel information can be gathered with most accuracy for later analysis. Semi-invasive attacks are very powerful as they combine almost all benefits of invasive attacks at reduced costs. Compared to invasive attacks, semi-invasive attacks are easier to perform as no error prone circuit modifications are made. Example of semi-invasive side-channel attack types are:

- Acoustic Attacks (with depackaging)
- Electromagnetic Attacks (with depackaging)
- Power Analysis Attacks (with depackaging)

Non-Invasive Attacks. A side-channel attack is said to be non-invasive of no modifications to the attacked device are performed at all. The only attack vectors are the accessible device interfaces and no traces of the attack are left behind. The impact of non-invasive attacks is higher than the impact of invasive attacks as they are comparably easy to perform and usually require only inexpensive equipment. There is a large variety of side-channel attack types using different forms of non-invasive side-channel information such as:

- Acoustic Attacks
- Cache Attacks
- Electromagnetic Attacks
- Power Analysis Attacks
- Timing Attacks

The next two sections cover the necessary background for the side-channel attacks which are used in this thesis. While the used variant of a timing attack is rather simple, a more sophisticated variant of a power analysis attack is used and therefore explained in more detail.

3.2 Timing Attacks

Timing attacks exploit the fact that the runtime of algorithms often depends on the actual processed data. The main influence factors of algorithmic runtime are hereby:

- Performance Optimizations
- Branching and Conditional Statements
- Processor Instructions
- RAM and Cache Hits

Timing attacks are usually known plain text/cipher text attacks. Given remote or direct access to a device under attack, a timing attack looks at how long it takes a device to perform certain operations on a known input and uses statistical analysis to partly or fully recover the cryptographic key which is used by the attacked device. The knowledge of the actual input is important for the attacker to distinguish whether a timing difference is caused by the key or not.

As demonstrated in Figure 3.1, timing information can be extracted from power traces. There, the power consumptions of three modular multiplications are shown. The three traces correspond to three input pairs of differing size. It is easy to see that the performed operation took the least time for the operands corresponding to the red trace, followed by the operands corresponding to the green and blue traces. This is due to the fact that processors can finish the execution for certain instructions early if the size of operands is small. Such a basic observation can already be used by an attacker to make some assumptions on the used operators. These operators may then allow further assumptions on the used cryptographic key.

While there exist way more sophisticated timing attacks than the one presented here [24, 20, 13], no further descriptions are given in this thesis as only very basic timing attacks will be used in the presented attack.

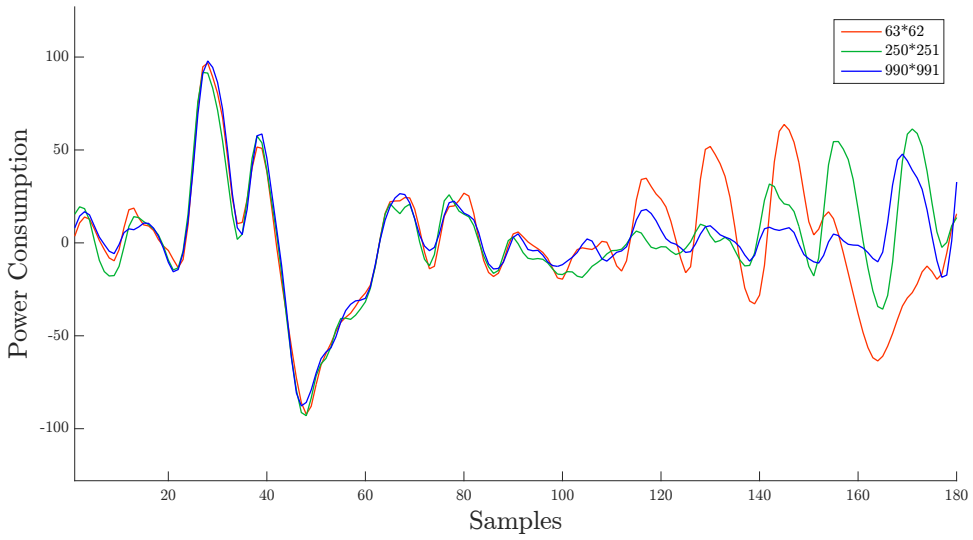


Figure 3.1: Timing differences for modular multiplications on an ARM-Cortex-M4F.

3.3 Power Analysis Attacks

Power analysis is a variation of side-channel attacks in which the power consumption of a device is studied. The attack is based on the observation that CMOS logic, nowadays used in almost every integrated circuit, has a somewhat predictable power profile. While CMOS devices have a very low static power consumption, the only significant power draw occurs during the event of a state transition [1]. Even though such a power profile is favorable for the implementation of low power integrated circuits, in terms of security it is not ideal.

Similarly to timing attacks, power analysis attacks are often known plain text/cipher text attacks. Yet, in the case of power analysis attacks, direct access to the attacked

device and a suitable measurement setup are mandatory. The main goal in power analysis attacks is to recover data which is processed by certain instructions on the attacked device. This data may then be used by the attacker to infer information about a cryptographic key which is used by the attacked device.

In the following, three variants of power analysis attacks are presented. In this thesis we are using the third variant to approximately identify the operators of modular multiplication operations which occur in an inverse NTT operation. As a consequence, only the third variant is described in more detail.

3.3.1 Simple Power Analysis

The Simple Power Analysis (SPA) is the most basic version of a side-channel attack based on power consumption information. It can be accomplished by a direct interpretation of the current flow caused by the attacked device.

Figure 3.2 shows the power consumption corresponding to two MOV instructions on an ARM-Cortex-M4F. One instruction moves the byte 0x00, the other one moves the byte 0xFF. Both operations are repeated ten times and the recorded power traces are averaged for better illustration. It is easy to see that those two operations can be distinguished reliably by observing the difference in power consumption. The same technique can be used to distinguish byte values other than 0x00 and 0xFF, yet the difference between the recorded power consumption will be smaller and the distinction less reliable

SPA becomes more difficult the more noise is present in the measured power traces. Especially, if devices are implemented with power analysis attacks in mind, the dependency between processed data and power consumption can be reduced up to a point where SPA becomes impractical. In such a case more sophisticated attack methods may be used as presented in the following two sections.

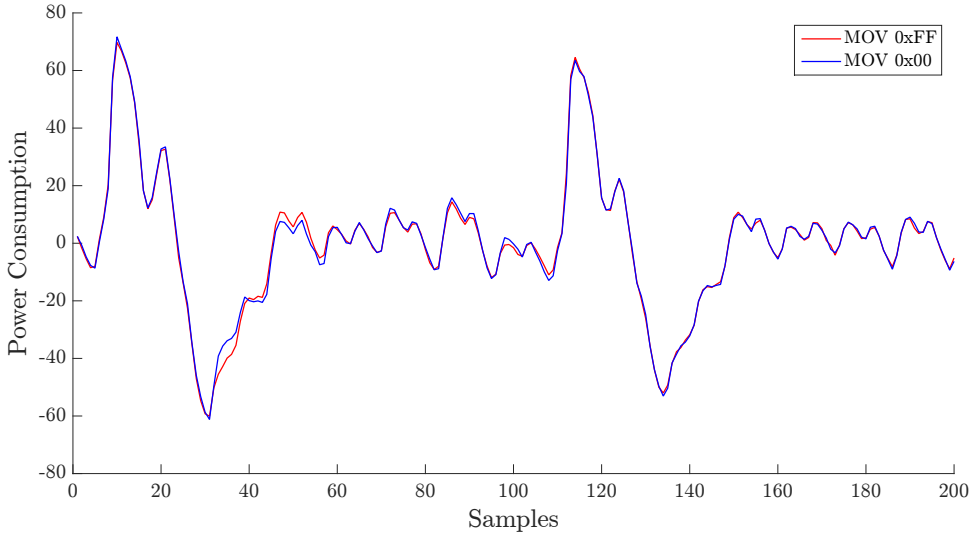


Figure 3.2: Simple Power Analysis.

3.3.2 Hypothesis Testing - Differential Power Analysis

The Differential Power Analysis (DPA) is a more sophisticated variant of a power analysis attack. It was first proposed by Kocher et al. in 1999 and turned out to be a quite versatile

and effective attack against a lot of cryptographic devices [40]. In contrast to the SPA, a DPA requires more than one recorded power trace of the same operation while using different known inputs and a constant key. These traces are then statistically analyzed by the means of hypothesis testing.

In general, hypothesis tests are used to determine whether or not there is enough evidence in a data set to infer if one out of many hypothesis holds or needs to be rejected. In the context of DPA, the data set corresponds to set of recorded power traces and the hypothesis correspond to the predicted power consumptions of the attacked operation for every possible processed data value. The DPA attack then consists of generating hypothesis for every possible processed data value followed by the evaluation and selection of the hypothesis that fits best to the given data set. The data value corresponding to the best fitting hypothesis is the most likely processed data value.

To generate hypothesis in a DPA, a so-called *power model* is needed. A power model is function that is used to approximately predict the power consumption of the attacked operation depending on our current hypothesis, i.e., the assumed processed data value. A power model could be any function of the attacked operation and its processed data theoretically. In practice however, there are a few standard power models, one of which usually works best in every occasion. Two very frequently used power models are the Hamming weight model and the Hamming distance model.

Hamming Weight Model. The Hamming weight power model assumes that the power consumption of an operation is related to the number of binary ones in the operation input x. This power model is commonly used for modelling the power consumption of MOV instructions in software implementations:

$$\mathrm{HW}(x) = \mathrm{CountBinaryOnes}(x)$$

Hamming Distance Model. The Hamming distance power model assumes that the power consumption of an operation is related to the Hamming weight of the XOR difference between operation input x and output y. This power model is commonly used to describe the power consumption of various hardware instructions as CMOS circuits are expected to have higher power consumption the more state transitions occur:

$$\mathrm{HD}(x, y) = \mathrm{HW}(x \oplus y)$$

A hypothesis evaluation function tells us which of our hypothesis fits best to a given data set. In DPA a hypothesis evaluation corresponding to one guessed key value is usually performed for multiple known operation inputs. For each of those inputs a hypothesis (i.e. power consumption prediction) is created based on the used power model. These hypothesis are then evaluated using real power measurement data corresponding to the inputs used in the hypothesis generation. The two most commonly used hypothesis evaluation functions are Difference of Mean and the Pearson correlation coefficient.

Difference of Mean. The Difference of Mean (DoM) function is a rather simple method for hypothesis evaluation. Based on the predicted power consumptions the measured power traces are split up into the two groups G_{low} and G_{high}. G_{low} contains power traces with a low predicted power consumption and G_{high} contains power traces with a high predicted power consumption. The DoM can then be calculated as:

$$S_{\mathrm{DoM}} = \frac{1}{n} \sum G_{\mathrm{high}} - \frac{1}{n} \sum G_{\mathrm{low}}$$

A score for the correctness of the hypothesis S_{DoM} is evaluated by calculating the difference between the averages of both groups G_{low} and G_{high}. A high positive difference indicates an accurate classification of power traces. Hence, a promising candidate for the correct key hypothesis has been found. A low difference indicates a wrong classification caused by a wrong prediction of power consumption. An incorrect key assumption during hypothesis generation is likely.

Pearson correlation coefficient. The Pearson correlation coefficient is a more sophisticated hypothesis evaluation function that additionally considers the variance in the power traces. It can be evaluated as:

$$S_{\text{corr}} = \frac{cov(X,Y)}{\sqrt{var(X)}\sqrt{var(Y)}} ,$$

where X corresponds to the predicted power consumption of our hypothesis and Y corresponds to the observed power consumption. The resulting correlation coefficient S_{corr} indicates how well our hypothesis fits to the observed data. The value range of S_{corr} is: $-1 \leq r_{\text{corr}} \leq 1$. Values close to 1 indicate a perfect fit of our hypothesis. Values close to 0 indicate no fit of our hypothesis. Values close to -1 indicate an inverse perfect fit of our hypothesis.

The biggest advantage of DPA over an SPA is that the attacker does not need precise knowledge about the actual implementation of the attacked device. Knowing the exact time of the attacked operation is not necessary since the hypothesis evaluation can be repeated over every instance of time in a larger time frame. Additionally, since a hypothesis is evaluated over multiple traces, a DPA is usually more resistant to noise and therefore yields better results.

3.3.3 Template Attacks

Template Attacks (TA), first proposed by Chari et al. [15], are often referred to as the most powerful type of side-channel attacks based on power analysis. This is due to the fact that TA are able to exploit all measurable dependencies between the data which is processed by the operation and its respective power consumption. In the following we give a description of TA based on the notation by Mangard et al. [51, Chapter 5-6].

Similar to DPA, TA are also based on hypothesis testing, yet they are also so-called *profiled attacks*. A profiled attack consist of two phases, an initial analysis phase and a later attacking phase. In the analysis phase, an attacker is assumed to gain full control over the attacked device. He may set arbitrary inputs to precisely analyze the leakage distribution of the attacked device in every possible scenario. Later, in the attacking phase, the attacker is given a set of power traces corresponding to unknown processed data and the attacker is required to recover it. In practice, the leakage analysis is only performed on one or very few specific operations that are executed by the device and assumed are to leak a sufficient amount of information. In the context of TA, the analysis phase and attacking phase are called template building phase and template matching phase, respectively.

Template building phase

The template building phase is performed before the actual attack takes place and requires physical access to the attacked device or an identical copy of it. The goal of the template

building phase is to find highly accurate characterizations, i.e. so-called *templates*, for the dependencies between any processed data value of the attacked operation and its respective power consumption. To do so, for every of the m possible processed data values $d \in \mathcal{D}$ of the attacked operation a set of n power traces $\mathbf{T}_d$ is recorded. Each set of power traces $\mathbf{T}_d$ is then characterized by a multivariate normal distribution. A multivariate normal distribution is the extension of a normal distribution to higher dimensions and is described by a mean vector $\mathbf{m}$ and a covariance matrix $\mathbf{C}$:

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n \cdot \det(\mathbf{C})}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{m})^{\mathrm{T}}\mathbf{C}^{-1}(\mathbf{x}-\mathbf{m})\right)$$

A template h_d for one processed data value d consists of $\mathbf{m}$ and $\mathbf{C}$. In this respect, $\mathbf{m}$ contains the averaged power consumption of $\mathbf{T}_d$ and $\mathbf{C}$ contains covariance information between all traces in $\mathbf{T}_d$. Given a set $[\mathbf{t}_1, \ldots, \mathbf{t}_n] \in \mathbf{T}_d$ of recorded power traces for one d, $\mathbf{m}$ and $\mathbf{C}$ can be estimated by:

$$\mathbf{m} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{t}_i, \tag{3.1}$$

$$\mathbf{C} = \begin{pmatrix} \mathrm{cov}(\mathbf{t}_1, \mathbf{t}_1) & \cdots & \mathrm{cov}(\mathbf{t}_1, \mathbf{t}_n) \\ \vdots & \ddots & \vdots \\ \mathrm{cov}(\mathbf{t}_n, \mathbf{t}_1) & \cdots & \mathrm{cov}(\mathbf{t}_n, \mathbf{t}_n) \end{pmatrix}, \tag{3.2}$$

where $\mathrm{cov}(\mathbf{t}_a, \mathbf{t}_b)$ denotes the covariance between vectors $\mathbf{t}_a$ and $\mathbf{t}_b$.

The result of the template building phase is a set of templates that characterize the power consumption of the attacked operation for every possible processed data value in $\mathcal{D}$.

Template matching phase

The template matching phase corresponds to the attacking phase of a profiled attack. The goal of the attacker is to determine the data value which is processed by the attacked operation, solely based on measured power traces and the leakage analysis obtained from the profiling phase. In the context of TA this is accomplished by matching the observed power trace to the previously calculated templates. Given a power trace $\mathbf{t}$ corresponding to an unknown processed data value and a template $h_d = (\mathbf{m}, \mathbf{C})$ for any of the possible data values $d \in \mathcal{D}$, we can evaluate the probability density function of the multivariate normal distribution and calculate the probability:

$$\mathrm{p}(\mathbf{t}; (\mathbf{m}, \mathbf{C})) = \frac{\exp\left(-\frac{1}{2} \cdot (\mathbf{t}-\mathbf{m})^{\mathrm{T}}\right) \cdot \mathbf{C}^{-1} \cdot (\mathbf{t}-\mathbf{m})\big)}{\sqrt{(2\pi)^{\mathrm{T}} \cdot \det(\mathbf{C})}} \tag{3.3}$$

By repeating this step for every possible template, we can determine the most likely data value that corresponds to the template with the highest probability. Thus, we follow the maximum likelihood decision rule:

$$\mathrm{p}(\mathbf{t}; h_{d_l}) > \mathrm{p}(\mathbf{t}; h_{d_j}) \; \forall l \neq j \tag{3.4}$$

In some applications one is not only interested in the template of maximum likelihood but instead in the probability distribution over all templates. The probability $\mathrm{p}(d_j|\mathbf{t_i})$ of data value d_j given an observed power trace $\mathbf{t_i}$ can be calculated by using Bayes' theorem as follows:

$$\mathrm{p}(d_j|\mathbf{t}_i) = \frac{\mathrm{p}(\mathbf{t}_i|d_j) \cdot \mathrm{p}(d_j)}{\sum_{l=1}^{m} \left(\mathrm{p}(\mathbf{t}_i|d_l) \cdot \mathrm{p}(d_l)\right)}, \tag{3.5}$$

where m denotes the number of possible data values and the prior probability $\mathrm{p}(d_x)$ is set to $(1/n)$ since its probability distribution is usually uniform.

The extension of 3.5 from one given trace $\mathbf{t}$ to multiple traces $\mathbf{T}$ can be calculated as follows:

$$\mathrm{p}(d_j|\mathbf{T}) = \frac{\left(\prod_{i=1}^{n} \mathrm{p}(\mathbf{t}_i|d_j)\right) \cdot \mathrm{p}(d_j)}{\sum_{l=1}^{m} \left(\left(\prod_{i=1}^{n} \mathrm{p}(\mathbf{t}_i|d_j)\right) \cdot \mathrm{p}(d_l)\right)}, \tag{3.6}$$

where n denotes the number of traces in $\mathbf{T}$.

When using 3.5 or 3.6 the result of a TA is a probability distribution over $\mathcal{D}$ that assigns a probability to every possible outcome $d \in \mathcal{D}$. In the remaining part of this section we mention optimizations that increase effectiveness and efficiency of TA when applied in practice.

Optimizations

When using the methods described before to perform a TA, one might come across a few problems that arise frequently in practice. Therefore, in this section we briefly discuss them.

Trace Size. The runtime complexity of TA heavily depends on the number of samples in each trace $\mathbf{t}$. In the matching phase the evaluation of $\mathrm{p}(\mathbf{t}; (\mathbf{m}, \mathbf{C}))$ in 3.3 involves the inversion of the covariance matrix $\mathbf{C}$ of size $\mathrm{len}(\mathbf{t})^2$. Since matrix inversion itself has a complexity $> \mathcal{O}(n^2)$, runtime quickly becomes an issue as the number of samples in $\mathbf{t}$ increases. To overcome this problem a technique called *Trace Compression* is used. Trace compression is based on the fact that not all samples of a trace contain an equal amount of information. Reducing a trace to a subset of samples, so-called *Points of Interest* (POI), that contain the most information and therefore improve the runtime of a TA significantly while not affecting the overall attack performance. In this thesis a *T-Test* was used for the selection of POIs. The application of T-Tests for the selection of POIs was first proposed by Gierlichs et al. [31] and has proven to be more accurate than previously used methods. Given sets of traces $\mathbf{T}_i$ for different processed data values, with mean $\mathbf{m}_i$, variance σ_i^2 and sample size n_i, the sum of pairwise t-differences can be calculated as:

$$\sum_{i,j=1}^{K} \left(\frac{\mathbf{m}_i - \mathbf{m}_j}{\sqrt{\frac{\sigma_i^2}{n_i} + \frac{\sigma_j^2}{n_j}}} \right)^2 \text{ for } i > j \tag{3.7}$$

The result of 3.7 is a vector containing a *t-score* for each sample position. A sample position with a high t-score indicates at the presence of high variability between the sets. In other words, at said sample position $\mathbf{T}_i$ can be distinguished with higher confidence. We can now use this information by removing all samples that do not have a sufficiently high t-score. By doing so we can significantly reduce the trace size while not affecting the TA performance.

Figure 3.3 illustrates the outcome of a T-Test evaluated for 25 sets of operands for a modular multiplication operation. The squared sum is omitted in the illustration, the single summands are illustrated instead. It is easy to see that out of 250 samples only a few show a significant variance. As a result, all recorded traces can be reduced to the 5 samples corresponding to the POIs of the T-Test analysis.

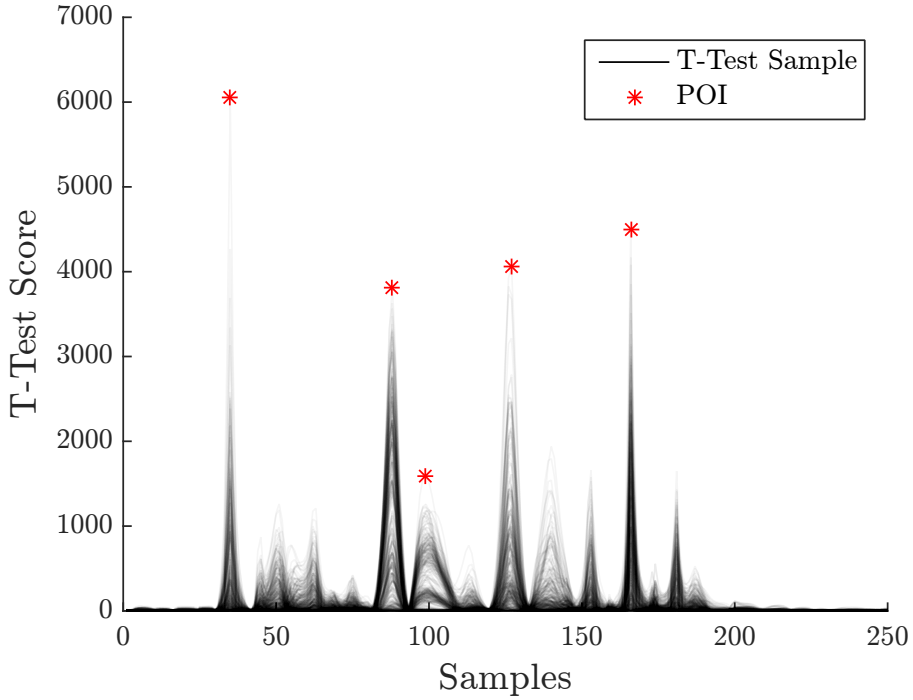


Figure 3.3: Point of interest calculation after T-Test

Finite Arithmetic Precision. Another problem that might arise in Equation 3.3 is the occurrence of values that cannot be expressed using ordinary 32 or 64-bit floating point data types. The usage of the logarithm applied to 3.3 can help is such situations. An additional beneficial side effect of the logarithm is the removal of the expensive exponentiation:

$$\ln \mathrm{p}(\mathbf{t}; (\mathbf{m}, \mathbf{C})) = -\frac{1}{2}(\ln((2 \cdot \pi)^{N_{IP}} \cdot \det(\mathbf{C})) + (\mathbf{t} - \mathbf{m})' \cdot \mathbf{C}^{-1} \cdot (\mathbf{t} - \mathbf{m}))$$

When replacing probabilities by their logarithm our decision rule for finding the template with maximum likelihood needs to be replaced by the minimum of absolute logarithmic probability:

$$| \ln \mathrm{p}(\mathbf{t}; h_{d_l}) | < | \ln \mathrm{p}(\mathbf{t}; h_{d_j}) | \; \forall l \neq j$$

3.4 Countermeasures

After the presentation of the first side-channel attacks in 1996 [39], the design of side-channel attack resistant implementations became a new field of research. In this section, we discuss the most important countermeasures that have proven to significantly increase the difficulty of performing side-channel attacks. Section 3.4.1 mentions countermeasures that, if implemented correctly, can prevent timing attacks. In Section 3.4.2 we mention two common countermeasures against power analysis attacks. The application of one of these power analysis countermeasures for an RLWE encryption scheme is explained in more detail.

3.4.1 Timing Countermeasures

As mentioned earlier, timing attacks exploit differences in runtime that may be caused by performance optimizations, branching and conditional statements, processor instructions, RAM and cache hits. The obvious solution to runtime variance is the usage of constant-time algorithms. This involves:

- Removal of conditional branches based on secret information
- Disabling caches (for SW)
- Usage of constant time algorithms/instructions

In some occasions, also masking or blinding schemes can be used to tackle timing attacks. While they do not ensure constant runtime, they eliminate the dependency between input data and secret key. Hence, the observed timing differences do not contain any exploitable information. Many of these proposals can be implemented easily without having a negative side effect. Some of them however, like disabling CPU caches, come at a cost that usually supersedes the obtained security goals. In those cases, performance/security trade-offs are necessary.

3.4.2 Power Analysis Countermeasures

In this section, we will have a look at two popular side-channel countermeasures called *Hiding* and *Masking*. The goal of both countermeasures is to remove the dependency between the power consumption of a device and its processed data. In practice, a combination of multiple countermeasures is often implemented since none of these countermeasures alone can fully prevent power analysis attacks. Finally, we present a masking scheme that can be applied to the RLWE-based encryption scheme that is attacked in this thesis.

Hiding

The hiding countermeasure attempts to eliminate the correlation between processed data, performed operations, and their resulting power consumption. This can be achieved by either ensuring constant power consumption of all operations or by ensuring perfect noise in the device's power consumption. While both approaches cannot be implemented to an extend where e.g. performing a DPA becomes impossible, they can still be used to significantly increase the difficulty of performing a DPA. We now discuss a couple of implementation approaches for software and hardware applications.

One popular hiding countermeasure is called *Shuffling*. Shuffling attempts to reduce the correlation between processed data and power consumption by randomizing the execution sequence of certain operations for a given algorithm. By doing so, an attacker cannot reliably distinguish power traces of the attacked operation from power traces of irrelevant operations. The performance of DPA and TA decreases since hypothesis are partially evaluated on wrong data sets. One example application of shuffling is the SubBytes lookup operation in AES. There, 16 successive table lookups are performed that are suspect to leak secret data. Since the table lookups are independent of each other, their execution sequence can be randomized easily. Shuffling alone cannot prevent power analysis attacks, yet it can help to significantly reduce the correlation between processed data and observed power consumption. Since the implementation of shuffling is rather easy, both in software and hardware, it is used in many real world cryptographic implementations.

Another effective method for reducing the leakage via the power side-channel is simultaneous execution. By executing multiple operations simultaneously, such as e.g. in multi-threaded applications, the resulting overall power consumption is averaged over all simultaneously executed operations. Hence, the power leakage of every one operation is reduced. If multithreading is not available on the target device, the usage of SIMD statements may be a reasonable alternative. In many hardware designs simultaneous execution is naturally occurring and only limited by the data path width. As a consequence, hardware implementations usually leak less information via the power side-channel than software implementations. However, the usage of spatial probing can be used to increase the effectiveness of power analysis attacks especially for hardware implementations.

A rather expensive hiding implementation for hardware designs is the usage of filters or noise generators. In both cases the goal is to eliminate the correlation between processed data and their resulting power consumption. Filters are typically placed between the power supply and the computation circuit. They consist of capacitors, constant current sources, and other power regulation circuitry. On the other hand, noise generators are usually based on random number generators and perform random operations that cause high variability in the devices overall power consumption. While noise generators might effectively reduce the leakage of power information, there might still exist other side-channels that remain unaffected by this countermeasure. The EM side-channel for example can leak power information of a small portion of a chip behind a noise generator. This EM leakage may then be mitigated by spreading multiple noise generators over the whole chip.

Masking

A masking scheme for a given operation defines a way to combine its input/output with random values such that all intermediate values are concealed without causing any differences on the operation output. The random values are called *mask* (input) and *inverse-mask* (output), generated by the implementation itself and not visible to an attacker. The combination of masks with operation input/output is usually accomplished using simple arithmetic operations like XOR and modular addition/multiplication. In general, a masking scheme for an operation with input x and output y is implemented as follows:

1. Generate a uniformly random sample r and set:

$$r = x' \quad \text{and} \quad x'' = r \oplus x \quad (\text{or} \quad x'' = x - r \mod q), \tag{3.8}$$

where x', x'' denote the masked inputs such that:

$$x = x' \oplus x'' \quad (\text{or} \quad x = x' + x'' \mod q) \tag{3.9}$$

2. Calculate the masked operation for both uniformly random shares x' and x'' and obtain the masked outputs y' and y''.

3. Combine y' and y'' as:

$$y = y' \oplus y'' \quad (\text{or} \quad y = y' + y'' \mod q) \tag{3.10}$$

to obtain the unmasked operation output y.

While a masking scheme is rather easy to implement for linear operations, the same cannot be said for non-linear operations. The simple relations between masks and inputs/outputs in 3.9 - 3.10 do not apply for non-linear operations. Thus more sophisticated masking schemes are needed in such cases.

One example application of a masking scheme for non-linear operations is the SubBytes operation in AES when implemented via table lookups. Table lookups are a common side-channel attack target since the required MOV operations usually leak a lot of side-channel information due to implementation specifics of the memory bus. One approach for building a masked SubBytes operation is the usage of special masked lookup tables for every possible mask value. These masked lookup table can be chosen such that 3.9 - 3.10 hold. However, This approach comes at the cost of increased runtime and memory requirement.

Masking has proven to be a quite effective countermeasure against DPA attacks. By using masked intermediate values, the attacker cannot reliably create key hypothesis for masked operations since the processed data values also depend on an unknown mask values.

Masking for Lattice-Based Encryption

We now give a more detailed description of a masking scheme proposed by Reparaz et al. [69], that can be applied to the RLWE encryption scheme that is attacked in this thesis. More precisely, the presented masking scheme allows us to mask the input of the INTT operation used during decryption.

The RLWE decryption of the cipher text $(\widetilde{\mathbf{c}}_1, \widetilde{\mathbf{c}}_2)$ and the private key $\widetilde{\mathbf{r}}_2$ is defined as follows:

$$\mathbf{m}^* = \text{INTT}(\ \widetilde{\mathbf{c}}_1 * \widetilde{\mathbf{r}}_2 + \widetilde{\mathbf{c}}_2\ )$$
$$\mathbf{m} = \text{Decode}(\ \mathbf{m}^*\ )$$

where $\mathbf{m}^*$ corresponds to an encoded version of the plain text $\mathbf{m}$. The masking scheme works by splitting the private key $\widetilde{\mathbf{r}}_2$ into the two shares $\widetilde{\mathbf{r}}_2'$, $\widetilde{\mathbf{r}}_2''$ such that:

$$\widetilde{\mathbf{r}}_2 = \widetilde{\mathbf{r}}_2' + \widetilde{\mathbf{r}}_2'' \mod q$$

The decryption input is then processed by the INTT implementation twice, once for each private key share:

$$\mathbf{m}^{*\prime} = \text{INTT}(\ \widetilde{\mathbf{c}}_1 * \widetilde{\mathbf{r}}_2' + \widetilde{\mathbf{c}}_2\ )\ , \quad \mathbf{m}^{*\prime\prime} = \text{INTT}(\ \widetilde{\mathbf{c}}_1 * \widetilde{\mathbf{r}}_2''\ ),$$

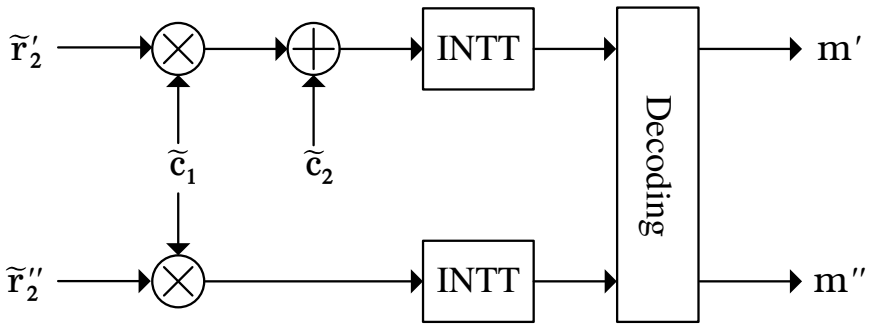


Figure 3.4: Basic masking scheme for an RLWE-based decryption

and a special masked decoder is used for both plain text shares:

$$\mathbf{m}' = \text{MaskedDecode}(\ \mathbf{m}^{*\prime}\ )\ , \quad \mathbf{m}'' = \text{MaskedDecode}(\ \mathbf{m}^{*\prime\prime}\ )$$

The original plain text can be recovered by:

$$\mathbf{m} = \mathbf{m}' \oplus \mathbf{m}''$$

By using this masking scheme the INTT input is no longer only dependent on cipher text and key, but also on a uniformly random masking value. Thus, known cipher text attacks are not applicable anymore. First-order DPA is not possible since we cannot correctly create key hypothesis for multiple inputs. First-order TA is also affected as they are now limited to single-trace template matching. An illustration of the described masking scheme can be found in Figure 3.4.

Chapter 4

Soft Analytical Side-Channel Attacks

This chapter explains Soft Analytical Side Channel Attacks (SASCA), first proposed by Veyrat-Charvillon et al. [77], as an intuitive way to combine traditional power analysis attacks with algebraic methods.

SASCA is inspired by previous work on algebraic side-channel attacks (ASCA). The goal of ASCA is similar to SASCA, i.e. improving the performance of a side-channel attack (SCA) using algebraic methods, yet their approaches are fundamentally different. In ASCA, leakage information of a previously executed SCA is used to improve the performance of attacks based on algebraic cryptanalysis (ACA). Section 4.1 gives a short introduction into ACA and mentions common limitations that ASCA attempts to remove. In Section 4.2, the concept of ASCA is explained. While ASCA turn out to be quite efficient on simulated leakage data, it is often not robust enough to allow attacks on real implementations. This is exactly what SASCA aim to do. A description of SASCA is given in Section 4.3.

4.1 Algebraic Cryptanalysis

The attack target in ACA is the cryptographic algorithm itself as opposed to just one implementation of an algorithm in SCA. As a consequence, successful ACA attacks are more severe since all implementations of the attacked algorithms are affected.

In ACA, the attacked cryptographic algorithm is represented as a system of equations. These equations usually contain known input and output variables $\mathrm{P}_{1\ldots p}$, $\mathrm{C}_{1\ldots c}$ as well as unknown key variables $\mathrm{K}_{1\ldots k}$. One trivial approach to setup an ACA attack on any cipher is to express any bit of the cipher text as a function of the plain text and the secret key:

$$\begin{aligned} \mathrm{C}_1 &= f_1(\mathrm{P}_1, \ldots, \mathrm{P}_p, \mathrm{K}_1, \ldots, \mathrm{K}_k) \\ &\vdots \\ \mathrm{C}_c &= f_c(\mathrm{P}_1, \ldots, \mathrm{P}_p, \mathrm{K}_1, \ldots, \mathrm{K}_k) \end{aligned}$$

If an attacker is able to solve such a system of equations for the unknown key variables $\mathrm{K}_{1\ldots k}$, the attack is successful and the cryptographic algorithm is considered to be broken. Obviously, solving such a system of equations is hard for any secure cipher since they are designed with ACA in mind. While diffusion layers ensure that every bit of the cipher

text depends on every bit of plain text and key, additional non-linear operations ensure high-degree monomials that cannot be accurately described by linear equations. In AES, for example, the AddRoundKey operation introduces dependencies between plain text and the secret key, diffusion is caused by the MixColumns and ShiftRows, operation and the SubBytes operation ensures non-linearity.

Usually, attacks solely based on ACA are not effective enough for breaking up-to-date cryptographic schemes. Nevertheless, ACA is still considered an important tool for analysis purposes. In order to improve attacks based on ACA, combinations with other attack methods have been proposed. One of these combinations, ASCA, is explained in the next section.

4.2 Algebraic Side Channel Attacks

The term "Algebraic Side Channel Attack" (ASCA) refers to an attack that combines traditional power analysis attacks with methods from ACA. The attack was first proposed by Renauld et al. in 2010 [67], after Courtois et al. showed the potential of ACA for over-determined equation systems in 2002 [19].

The main idea behind ASCA is that leakage information, gathered from an SCA, can be used to narrow down variable domains or to increase the number of equations in the equation system of an ACA. Since ASCA requires leakage information, it is not as generic as ACA and therefore only applies to implementations of a cryptographic scheme that have a similar leakage profile to the attacked implementation. ASCA usually consists of two phases, a short online phase and a long offline phase.

Online Phase

In the online phase, leakage information is collected via standard power analysis methods like SPA, DPA, or TA. In contrast to them however, the goal in the ASCA online phase is somewhat different. Traditional power analysis attacks follow a divide and conquer principle. They do not attack the whole secret key at once but parts of the key in a repeating manner. Hence, they can only attack operations at the beginning or the end of an algorithm because otherwise the diffusion layers would cause to many dependencies on key bits. In ASCA, however, the used equation system describes the whole cryptographic algorithm or at least a large portion of it, which is why side-channel information from the whole algorithm is useful.

Offline Phase

The offline phase then basically corresponds to an ACA that incorporates the information gathered from the online phase. This phase is usually more time consuming since it consists of performing runtime-intensive algorithms that try to find solutions for the constructed system of equations. The most common techniques to solve such systems of equations are based on SAT solvers [8, 18]. Since SAT solvers cannot directly work with the probability distributions obtained from the online phase some adaptions are necessary. In most proposals probability distributions are discretized and sieved, leaving left only a few of the values with highest probability for each intermediate variable. These are then given to, e.g. a SAT solver to determine a subset that does not contradict each other,

hence a valid assignment of values to all intermediate variables in the algorithm (including the secret key).

One of the main benefits of ASCA over traditional power analysis attacks is that they can exploit more leakage information since many more operations can be attacked. As a result, the number of required power traces for the attack to work may be as little as just one. Such single trace attacks are especially powerful since they are resistant to side-channel countermeasures like masking. Experiments on simulated leakages, e.g. low noise Hamming weight information, have shown that ASCA can indeed achieve significantly better results than attacks purely based on ACA [67]. It has to be noted that ASCA seems to be prone to noise. If noise is too high, then SAT solvers quickly fail in finding solutions for the constructed system of equations. Hence, their performance on real leakage data is currently rather poor. If a single trace does not contain enough information for a successful attack, multiple traces can be averaged to reduce noise. On top of that, even though ASCA could exploit leakage information from the whole algorithm, in practice only a subset of them is actually used to further reduce the runtime of equation solving.

4.3 Soft Analytical Side Channel Attacks

SASCA is a new approach for combining traditional SCA with algebraic methods. In contrast to ASCA, where the cryptographic algorithm is described by a system of equations, the idea here is to work directly on the posterior distributions of intermediate values obtained in the offline phase. Given the posterior probability distribution:

$$\mathbf{p}_i = \Pr(\ X_i = x_i \mid l_i\ )$$

for each observed intermediate variable x_i, obtained from a leakage l_i, a graphical model describing the whole algorithm is constructed. This graphical model expresses intermediate values x_i as nodes which are connected by constraints that model the dependency between the intermediate values. Additional constraints are then added for each observed x_i representing the leakage information $\mathbf{p}_i$. Unobserved variables are not directly connected to leakage information. Hence, their values need to be inferred by the information about their neighbours.

Once such a graphical model is constructed, the Belief Propagation algorithm (BP) is used to calculate the marginal distribution of each x_i, given the leakage information of all other intermediate values $x_j, \forall j \neq i$. If the attack is successful, the resulting probability distributions for each x_i assign a distinct value with probability 1, while all others have probability 0. A detailed description of marginalization in graphical models as well as the BP algorithm is given in Chapter 5.

Experiments by Veyrat-Charvillon et al. [77] on an AES implementation with simulated leakage data have shown that SASCA performs well even when noisy simulated leakage data is used. While the attack may require multiple traces, given leakage information with a low signal-to-noise ratio (SNR), the traces are not required to correspond to the same inputs. This is a more realistic scenario compared to ASCA where multiple traces can only be used if they correspond to the same input.

Compared to results from ASCA, SASCA has two main advantages. First of all, the runtime and memory complexity is greatly reduced compared to e.g. SAT-solver based ASCA for algorithms whose operations can simply be expressed in a graphical model. The runtime of the algorithm used in the offline phase is mainly determined

by the implementation of update rules that model the dependency between the x_i. In the attack on AES these dependencies can be implemented efficiently as AddRoundKey, SubBytes, and MixColumns describe simple relations between 2 or 3 variables. Second, given a sufficient number of traces SASCA can deal with any SNR which makes it a promising candidate for attacks on real applications.

Chapter 5

Marginalization in Graphical Networks

This chapter presents an algorithm called *Belief Propagation* (BP), which is used for the efficient calculation of marginal distributions for a variety of functions. The problem of marginalization is usually an expensive task and defined in Section 5.1. However, for certain functions the cost of computing marginals can be reduced significantly. The presented algorithm requires to represent a function by a so-called *factor graph*. Section 5.2 explains what factor graphs are and how they can be constructed. Once a factor graph is constructed and it is acyclic, the BP algorithm can be used as described in Section 5.3. In the case of a cyclic factor graph, an adapted version of the BP algorithm may be used. The adapted algorithm is called *loopy*-BP and is presented in Section 5.4 alongside with possible limitations. Our description and notation is largely based on that of MacKay et al. [50, Chapter 26]

5.1 Marginalization Problem

Definition 5.1.1. Given a function P^* of a set of N variables $\mathbf{x} \equiv \{x_n\}_{n=1}^N$ that is defined by the product of M of its factors:

$$P^*(\mathbf{x}) = \prod_{m=1}^{M} f_m(\mathbf{x}_m),$$

where each of the factors $f_m(\mathbf{x}_m)$ is a function of a subset $\mathbf{x}_m$ of the variables that make up $\mathbf{x}$, the problem of marginalization is to compute the marginal function Z_n of any variable x_n:

$$Z_n(x_n) = \sum_{\{x_{n'}\}, n' \neq n} P^*(\mathbf{x})$$

Usually, one is also interested in the calculation of normalized marginals:

$$P_n(x_n) = \frac{1}{Z}\, Z_n(x_n),$$

where the normalizing constant Z is defined by:

$$Z = \sum_{\mathbf{x}} \prod_{m=1}^{M} f_m(\mathbf{x})$$

The runtime of the summation in the marginalization step is believed to grow exponentially with the number of variables N. However, for certain functions, their factorization can be exploited in such a way that the calculation of their marginals becomes feasible, even if they contain many variables.

5.2 Factor Graphs

A factor graph is a representation of a function's factorization as a bipartite graph. It is used to model dependencies between the variables of a function and will allow us to perform the BP algorithm at a later step. Since a factor graph is an extension to a Markov Random Field, a description of Markov Random Fields is given first.

A Markov Random fields represents a probability distribution over variables $x_1, \ldots, x_n$ as an undirected graph. Each of the random variables fulfills the Markov property and is represented by a node in the graph. Figure 5.2 shows an example of a Markov random field. It consists of four variable nodes $x_1, \ldots, x_4$ and their corresponding connections that are modeled by undirected edges. Markov random fields cannot be used directly in

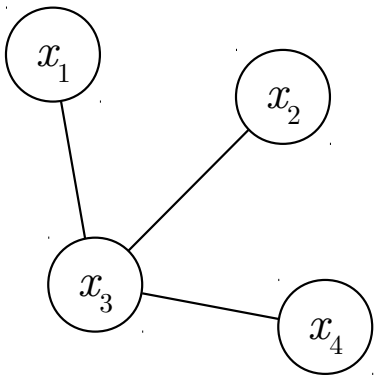


Figure 5.1: Markov Random Field

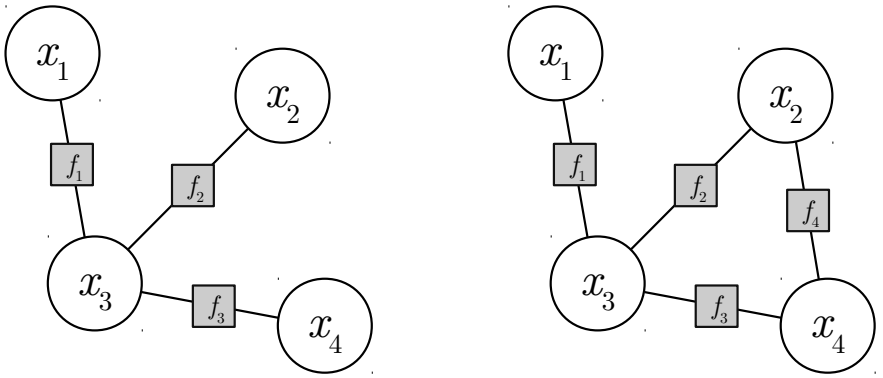
the BP algorithm because they do not contain information about the actual dependency between a pair of connected variable nodes. Fortunately, the conversion to a factor graph is possible for any Markov random field according to the Hammersley-Clifford theorem [16].

Factor graphs extend the concept of Markov Random Fields by introducing a second type of nodes. Besides variable nodes they also contain factor nodes that are used to model the dependency between pairs of connected variable nodes. A factor node is only connected to variable nodes and vice versa. Given a function F and its factorization:

$$F(x_1, x_2, x_3, x_4) = f_1(x_1, x_2) f_2(x_1, x_3) f_3(x_2, x_3) f_4(x_3, x_4),$$

it is easy to find the corresponding factor graph as depicted in Figure 5.2a. It can be constructed by connecting a factor node to all variable nodes that are used in the corresponding factor. While the factor nodes f_i can be any function of the two corresponding variable nodes, they are usually modeled by a table containing the variable's joint distribution. This generic way of expressing dependencies via joint distribution tables can come at the cost of high runtime/memory requirements though. If the variable domain is large or the number of factor node neighbours is high, other ways of describing variable dependencies have to be found. In the attack presented in Chapter 6, we are dealing

with a factor graph whose variable domain is $\{0, \ldots, 7680\}$. The joint distribution table approach cannot be used there anymore.



(a) Acyclic factor graph. Belief propagation **will** work.

(b) Cyclic factor graph. Belief propagation **might** work.

Figure 5.2: Acyclic factor graph (left) and cyclic factor graph (right)

Once the factor graph is constructed and it is acyclic, the BP algorithm can be used to calculate the exact marginals of all variables in the network. However, if the resulting factor graph does contain cycles, as shown in Figure 5.2b, the outcome of the algorithm is not clear. In Section 5.3 the standard BP algorithm is explained under the assumption that the used factor graph is acyclic. Section 5.4 explains how an adapted version of the BP algorithm can be used on cyclic factor graphs and mentions possible limitations.

5.3 Belief Propagation

The BP algorithm, also called "Sum-Product" algorithm, is a network based message passing algorithm for the efficient computation of unobserved node marginals conditioned on observed nodes. It was first proposed by Judea Pearl [59] and is mainly used in the field of artificial intelligence and information theory. While there exist multiple variants of the BP algorithm for different kinds of networks, a factor graph based description is presented in this section. The algorithm works by exchanging messages between variable nodes and factor nodes iteratively. In this respect, a message is the belief of a variable's probability distribution. Each iteration consists of two steps. In step one, variable nodes send messages to factor nodes. In step two, factor nodes send messages to variable nodes. After a couple of iterations, the variable node marginals can be calculated in a final step. These three steps are described in the following:

From variable to factor: A variable node x_n sends a message to a connected factor node f_m by multiplying all messages x_n gets from its neighbour factor nodes except f_m. By doing so, x_i is telling f_m about what other factors-nodes belief about x_n's probability distribution:

$$\mathfrak{q}_{n\to m}(x_n) \;=\; \prod_{m'\in\mathcal{M}(n)\setminus\{m\}} \mathfrak{r}_{m'\to n}(x_n),$$

$\mathcal{M}(n)$ is the set of all factors in which n participates
$\mathfrak{q}_{n\to m}$ denotes messages from variable nodes to factor nodes

$\mathfrak{r}_{m\to n}$ denotes messages from factor nodes to variable nodes
For all leaf variable nodes n: $\mathfrak{q}_{n\to m}$ is set to a uniform distribution

From factor to variable: Similarly, a factor node f_m sends a message to a connected variable node x_n by multiplying all messages f_m gets from its neighbour variable nodes except x_n, multiplying the result with its own joint distribution table and finally summing out all variables except x_n:

$$\mathfrak{r}_{m\to n}(x_n) \;=\; \sum_{x_m\setminus n}\left(f_m(\mathbf{x}_m)\prod_{n'\in\mathcal{N}(m)\setminus m}\mathfrak{q}_{n'\to m}(x_n)\right),$$

$\mathcal{N}(m)$ denotes the set of variables the m-th factor depends on
For all leaf factor nodes m: $\mathfrak{r}_{m\to n}$ is set to $f_m(x_n)$

Final Step: The final marginal function $Z_n(x_n)$ of a variable node can be calculated by multiplying all messages from the connected factor nodes:

$$Z_n(x_n) = \prod_{m\in\mathcal{M}(n)}\mathfrak{r}_{m\to n}(x_n)$$

The normalized marginals can be obtained easily by dividing the marginal function by the sum of the marginal functions:

$$P_n(x_n) = \frac{1}{Z}Z_n(x_n),$$

where the normalization constant Z is computed as:

$$Z = \sum_{x_n} Z_n(x_n)$$

Since messages are passed only between neighbouring nodes, multiple message passing steps are necessary until a node has observed beliefs of all other nodes in the network. The actual number of necessary iterations depends on the structure of the used factor graph. If the factor graph is acyclic, the following scheduling scheme can be used to calculate exact marginals in as little as two iterations:

1. Select one variable node as the root node.
2. Propagate beliefs inwards, starting at leaf nodes.
3. Propagate beliefs outwards, starting from the root node.
4. Calculate marginals for variable nodes.

In conclusion, the BP algorithm is a rather efficient method for calculating marginals in acyclic factor graphs. Even though the runtime is still exponential in the worst case, many real world problems can be solved efficiently. We now explain how the algorithm can be extended to general networks and mentions possible limitations.

5.4 Loopy-Belief Propagation

The BP algorithm was originally designed for usage in acyclic networks. However, in many problem instances cycles in factor graph are unavoidable. The factor graph corresponding to an NTT operation, presented in Chapter 6, is one of those examples. This lead to the adaption of the original algorithm to the loopy-BP algorithm. The loopy-BP algorithm uses the same message passing methods as the standard algorithm but a different initialization and scheduling:

1. Initialize all variable nodes to a uniform distribution.
2. Send messages from variable nodes to connected factor nodes.
3. Send messages from factor nodes to connected variable nodes.
4. Go to step 2, repeat until variable node distributions have converged (if possible).
5. Calculate marginals for variable nodes.

Curiously, using this adapted version of the original algorithm works for many problem instances based on cyclic factor graphs [54]. The resulting marginals may be not exact but usually they are good approximations of the true marginals. Nevertheless, it has to be noted that convergence is not guaranteed here. It is not completely understood under which conditions a loopy-BP will converge or not. Part of the reason why the algorithm may not converge on cyclic factor graphs is that variables do influence their own distributions which can result in positive feedback loops. Mooij et al. [53] states a few sufficient conditions for the convergence of the loopy-BP algorithm to a unique fixed point. Apart from the fact that convergence may not happen, also the number of necessary iterations is not predictable. As a rule of thumb the longest path in the factor graph can be used as a first reference value. In practice, however, the number of iterations may be a multiple or a fraction of that value.

Chapter 6

Attack on an RLWE-based Encryption Scheme

This chapter contains a detailed description of the main goal of this thesis, i.e. an attack on a microprocessor implementation of an RLWE-based encryption scheme. In Section 6.1, we give an overview of our attack procedure and goals. The attack procedure itself is then split up into 3 steps as follows:

In Section 6.2, a side-channel attack on the attacked encryption scheme is described. Based on the obtained leakage information from the first step, the attack results are improved by the application of the BP algorithm as shown in Section 6.3. Since the outcome of the BP algorithm still does not allow us to perform a full private key recovery, we perform a lattice reduction in order to efficiently recover the remaining unknown private key coefficients. This last step is explained in Section 6.4.

6.1 Attack Overview

The main goal of this thesis is an attack on a microprocessor implementation of an encryption scheme that is based on the RLWE problem (Section 2.5.3). Since the encryption scheme is asymmetric, the attack is targeting only the decryption operation in which the private key is involved. The decryption of a cipher text $(\widetilde{\mathbf{c}}_1, \widetilde{\mathbf{c}}_2)$ under secret key $\widetilde{\mathbf{r}}_2$, and a shared modulus q is defined as follows:

$$\mathbf{m}^* = \text{Decrypt}(\ \widetilde{\mathbf{c}}_1, \widetilde{\mathbf{c}}_2, \widetilde{\mathbf{r}}_2\ ) = \text{INTT}_q(\ \widetilde{\mathbf{c}}_1 * \widetilde{\mathbf{r}}_2 + \widetilde{\mathbf{c}}_2\ ), \tag{6.1}$$

where $\mathbf{m}^*$ corresponds to an encoded version the original plain text and INTT denotes the inverse NTT operation. All calculations are modulo q.

A successful attack on a device that performs the operations from 6.1 yields the unknown private key coefficients of $\widetilde{\mathbf{r}}_2$. As with all implementation attacks, we start by performing a side-channel analysis on certain operations that are suspect to leak information about $\widetilde{\mathbf{r}}_2$. One of the obvious choices for a side-channel attack are the coefficient-wise operations $\widetilde{\mathbf{c}}_1 * \widetilde{\mathbf{r}}_2 + \widetilde{\mathbf{c}}_2$. Since an attacker may have full control over the inputs $\widetilde{\mathbf{c}}_1$ and $\widetilde{\mathbf{c}}_2$, a prediction of these intermediate values is a reasonable goal. However, side-channel countermeasures can be implemented for these operations quite easily as well. In fact, there already exist proposals that suggest the application of masking and/or shuffling for such obvious side-channel targets [57]. Hence, we decided not to include these operations

into our side-channel analysis and instead focus on operations that are harder or more expensive to protect.

Currently, no implementation proposal for RLWE-based encryption schemes features side-channel countermeasures inside the INTT operation. However, a masking of the INTT input seems to be an obvious design choice due to the linearity of the (I)NTT operation. One such masking scheme was proposed by Reparaz et al. [69, 68] and conceals all intermediate variables inside the INTT operation (Section 3.4.2).

Given such a masking scheme, the application of side-channel analysis becomes more difficult. Since the INTT input no longer only depends on the decryption input and the private key, the exploitation of multiple power traces e.g. by means of first-order DPA becomes infeasible. Single-trace attacks are however still possible. A successful attack on the INTT operation yields the INTT input coefficients as well as the values of all butterfly intermediates. If the whole INTT input is known to the attacker, private key recovery is simple. In the case of an unmasked implementation we can express the INTT input $\widetilde{\mathbf{x}}$ as:

$$\widetilde{\mathbf{x}} = \widetilde{\mathbf{c}}_1 * \widetilde{\mathbf{r}}_2 + \widetilde{\mathbf{c}}_2,$$

where $\widetilde{\mathbf{c}}_1$ and $\widetilde{\mathbf{c}}_2$ are public. This equation can simply be rewritten to:

$$\widetilde{\mathbf{r}}_2 = \widetilde{\mathbf{x}} - \widetilde{\mathbf{c}}_2 * \widetilde{\mathbf{c}}_1^{-1},$$

thus leaking the NTT transformed of the private key $\mathbf{r}_2$. For masked implementations we need to successfully attack both INTT operations corresponding to each private key share. Again, we express the INTT inputs $\widetilde{\mathbf{x}}', \widetilde{\mathbf{x}}''$ corresponding to both private key shares as:

$$\begin{aligned} \widetilde{\mathbf{x}}' &= \widetilde{\mathbf{c}}_1 * \widetilde{\mathbf{r}}_2' + \widetilde{\mathbf{c}}_2 \\ \widetilde{\mathbf{x}}'' &= \widetilde{\mathbf{c}}_1 * \widetilde{\mathbf{r}}_2'', \end{aligned}$$

and calculate the private key shares by:

$$\begin{aligned} \widetilde{\mathbf{r}}_2' &= \widetilde{\mathbf{c}}_1^{-1} * \widetilde{\mathbf{x}}' - \widetilde{\mathbf{c}}_2 \\ \widetilde{\mathbf{r}}_2'' &= \widetilde{\mathbf{c}}_1^{-1} * \widetilde{\mathbf{x}}'', \end{aligned}$$

which gives us the the unmasked key $\widetilde{\mathbf{r}}_2 = \widetilde{\mathbf{r}}_2' + \widetilde{\mathbf{r}}_2'' \mod q$.

In the following, we show a single-trace attack that is able to recover the full private key of an RLWE encryption scheme that uses the (I)NTT for runtime efficiency. While we show the attack outcome for an RLWE encryption scheme, the attack may also be applicable to any lattice-based cryptographic scheme that performs an (I)NTT operation on sensitive data.

We now give an overview of the 3 steps that are involved in our attack:

Step 1. The first step our attack consists of a traditional side-channel analysis that is performed on modular additions, subtractions, and multiplications that occur in an INTT butterfly, i.e. during the decryption operation. Since we also want to deal with masked implementations we limit the number of traces in the attack phase of our side-channel analysis to one per decryption. Clearly, the information obtained about each INTT intermediates will have a high noise level and a key recovery is not possible yet.

Step 2. In step 2 we want to make use of the large amount of potentially leaking modular operations inside the INTT butterfly and their comparably simple algebraic connections. We follow the idea of SASCA and show the application of the BP algorithm in a graphical model representing the INTT butterfly. This approach is comparable with the offline phase in a SASCA. The outcome of the algorithm are approximate marginal distributions of all intermediate values of an INTT operation. If the resulting marginal distributions indicate distinct values for all intermediates (including the input), a key recovery can be performed as described above. If the BP algorithm can only determine a subset of all intermediates with high confidence, a more sophisticated key recovery is necessary.

Step 3. In the last step of our attack, we show an algorithm that, given enough correct intermediates, can recover all coefficients of the private key with significantly less computational cost compared to the trivial brute force approach. As we will see later, the application of this algorithm improves the practicality of our attack as we are usually not able to correctly determine all intermediates after step 2.

6.2 Attack Step 1: Side-Channel Attacks on an INTT Butterfly Network

In this section, we describe how we perform side-channel attacks on the modular operations inside an INTT butterfly. Section 6.2.1 explains the measurement setup that is used for the measurement of traces. Section 6.2.2 describes the software implementation that is used by the attacked microprocessor. In Section 6.2.4, we describe the kinds of side-channel attacks we use for each of the attacked operations. Finally, in Sections 6.2.5 and 6.2.6, we present the first results of our side-channel analysis, both for real and simulated leakage data.

6.2.1 Measurement Setup

Our side-channel analysis is performed on a Texas Instruments MSP432 (ARM Cortex-M4F) microcontroller on a MSP432P401R LaunchPad development board. The same microcontroller was already used by several other authors for the evaluation of lattice-based cryptographic implementations [23, 57, 68]. Our microcontroller was clocked at its maximum possible frequency of 48 MHz.

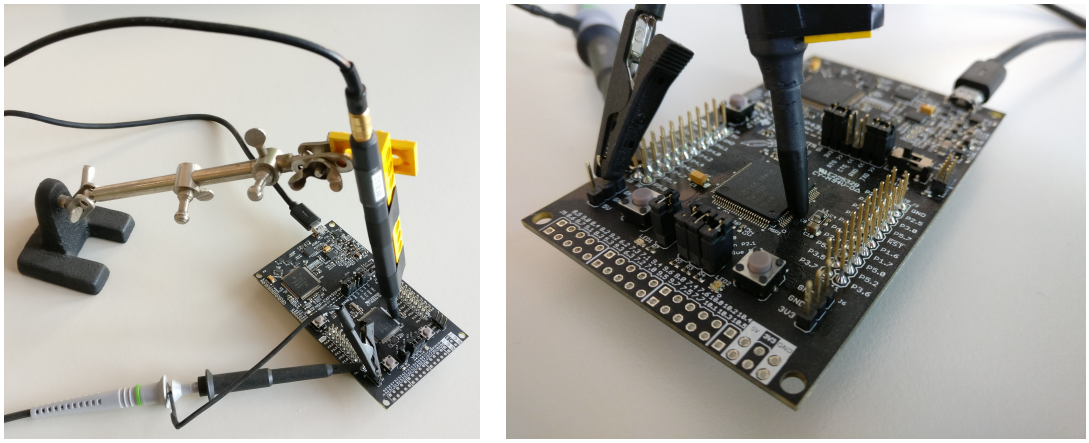
Leakage is measured via the EM side-channel. We use a Langer RF-B 3-2 near-field probe placed in proximity to the external core-voltage regulation circuitry. Hence, we expect the recorded traces to be similar to power consumption leakage. A PicoScope 6404 oscilloscope was used for the recording of the traces. Each trace corresponds to the execution of one modular operation. We setup a trigger to ensure alignment of the recorded traces. The sampling rate is set to 500 MS/s, the number of recorded samples per trace is 250.

After the measurement phase we applied a band-pass filter from the MATLAB DSP System Toolbox to rule out unwanted signal components. A comprehensive list of the used oscilloscope/filter settings is given in Table 6.1. Figure 6.1 shows the measurement setup.

Oscilloscope Settings	
Sampling Rate	500 MS/s
Samples / Trace	250

MATLAB Filter Settings	
Filter Order	20
Cutoff Frequency 1	6e6
Cutoff Frequency 2	48e6
Sample Rate	500e6
Design Method	window

Table 6.1: Used settings for measurements.



(a) Evaluation board with EM probe and trigger probe.

(b) Close-up onto the placement of the EM probe.

Figure 6.1: Pictures of the measurement setup

6.2.2 Microprocessor implementation

We now describe the microprocessor implementation of the attacked RLWE encryption scheme. As mentioned before, we focus our side-channel analysis solely on the operations that occur inside the INTT operation. Hence, we also limit the description of our microprocessor source code to the three modular operations: addition, subtraction, and multiplication. The source code is based on the efficient assembler implementation of the RLWE encryption scheme by deClercq et al. [23]. For easier evaluation we only implemented the leaking operations as they are easy to identify and separate.

Modular Multiplication

The modular multiplication is implemented using constant time `SMULBB` instructions, the subsequent reduction is implemented via trivial division:

$$ab \bmod q \quad = \quad ab - \left\lfloor \frac{ab}{q} \right\rfloor q$$

This leaks limited timing information depending on the used operands. On our microprocessor, the division takes between 2 and 12 cycles. All parameters and intermediate variables remain in CPU registers throughout the computations. We do not use leakage coming from explicit `LOAD` and `STORE` instructions. To ensure separation, we place `NOP` instructions before/after the execution of the multiplication.

```
LOADW %[op_1]
LOADW %[op_2]
LOADW %[modulus] #7681
NOP
:
NOP
SMULBB %[product_1], %[op_1], %[op_2]
UDIV %[quotient], %[product_1], %[modulus]
SMULBB %[product_2], %[quotient], %[modulus]
SUB %[result], %[product_1], %[product_2]
NOP
:
```

Modular Addition

For the modular addition implementation:

$$a + b \bmod q = \begin{cases} a + b - q & a + b \geq q \\ a + b & a + b < q \end{cases}$$

we use a constant time implementation based on the conditional ARM `IT` instruction. If the sum of the operands does not require a subsequent reduction step, i.e. exceed q, additional `NOP` instructions are executed instead to ensure constant runtime.

```
LOADW %[op_1]
LOADW %[op_2]
LOADW %[modulus] #7681
NOP
:
NOP
ADD %[sum], %[op_1], %[op_2]
CMP %[sum], %[modulus]
IT PL
SUBPL %[sum], %[sum], %[modulus]
NOP
:
```

Modular Subtraction

The implementation of the modular subtraction:

$$a - b \bmod q = \begin{cases} a - b + q & a < b \\ a - b & a \geq b \end{cases}$$

features constant runtime as well and is implemented similarly to the addition. If the difference is < 0 then the modulus q is added.

```
LOADW %[op_1]
LOADW %[op_2]
LOADW %[modulus] #7681
NOP
:
NOP
SUBS %[diff], %[op_1], %[op_2]
IT MI
ADDMI %[diff], %[diff], %[modulus]
NOP
:
```

By using these three operations and Algorithm 1, we can construct a software implementation of an INTT butterfly for arbitrarily large input vectors.

6.2.3 Side-Channel Attack on Real Leakage

In this section, we give a detailed description of our side-channel analysis on the INTT butterfly network. The goal of our analysis is to gather information on intermediate values that occur during an INTT operation. Throughout the rest of this thesis we will use the term *intermediates* to denote the inputs of all butterflys that make up our INTT butterfly network.

As mentioned earlier and shown in Figure 2.6, an INTT butterfly network consists of the three modular operations: addition, subtraction, and multiplication. We perform a template attack (TA) (Section 3.3.3) on all three operations, yet with different goals in mind. For the modular multiplication operation, we want to gain information about the unknown intermediate that is multiplied with a known twiddle factor. In case of the modular addition/subtraction operation, we use a TA in order to detect whether or not a reduction is performed during execution.

We simplify our template building phase by isolating the 3 modular operations. For each operation, we record power traces for all possible input/output combinations. An attack on the full INTT butterfly is then simulated by assuming leakage information corresponding to the real input/output data of every operation.

We additionally perform a simple SPA-based timing attack on the modular multiplication operation as seen in Figure 3.1. The timing information is used to reduce the number of possible input combinations for each modular multiplication. Hence, the performance of the corresponding TA is improved.

In the following a detailed description of our performed side-channel attacks on each of the 3 modular operations is given.

SPA on Modular Multiplication

Our implementation of the modular multiplication is leaking timing information due to the implementation of the reduction step by trivial division. The hardware divider on our microprocessor has an operand depending runtime. We observed that these timing differences depend on the bit size of the non-reduced product. After closer inspection, we

Product Bit-Size	Sample Offset	Timing Class
< 12	0	1
< 16	25	2
< 20	38	3
< 24	50	4
≥ 24	63	5

Table 6.2: Timing differences for modular multiplication

Timing Class	Possible Intermediates
1	$0 \ldots 1$
2	$2 \ldots 19$
3	$20 \ldots 309$
4	$310 \ldots 4959$
5	$4960 \ldots 7680$

Table 6.3: Exploiting the runtime of a modular multiplication with $\omega = 3383$.

were able to identify a total of 5 different timing classes which are shown in Table 6.2. This timing information can be obtained by e.g. a visual inspection of a trace. Depending on the sample offset of the first significant power draw (Figure 3.1) we are able deduce the bit-size of the non-reduced product of a known twiddle factor and an unknown intermediate.

With this timing information we can, given the known twiddle factor ω, narrow down the possible values for the other factor, i.e. the used intermediate. Let us assume we are looking at a modular multiplication operation with $\omega = 3383$. If we can determine the runtime of the multiplication, we can narrow down possible values of the used intermediate as shown in Table 6.3. In the following, we denote the timing classes of a modular multiplication with $\mathcal{C}_{\omega,1}, \ldots, \mathcal{C}_{\omega,5}$. Hereby, each $\mathcal{C}_{\omega,x}$ contains the possible input combinations for a modular multiplication with ω and timing class x.

In our attack, we use a simple thresholding approach for determining the timing class of a modular multiplication. We thoroughly tested our approach for over 10000 random modular multiplications and achieved a correct classification rate of 1.

TA on Modular Multiplication

In each modular multiplication, an unknown intermediate is multiplied with a known twiddle factor ω^x, i.e. the x^{th} power of the used primitive root of unity. The domain size of an intermediate is determined by the modulus q, in our case 7681. The domain size of ω is determined by the size of the input vector. An $n-$coefficient INTT requires the n/2 different twiddle factors $\omega^1, \ldots, \omega^{n/2}$. Hence, the total number of input combinations for a modular multiplication in a 256-coefficient INTT is $7681 \times 128 = 983168$.

In the template building phase we recorded 100 power traces for each input combination of the modular multiplication operation. Thus, in total we have recorded about 100 million

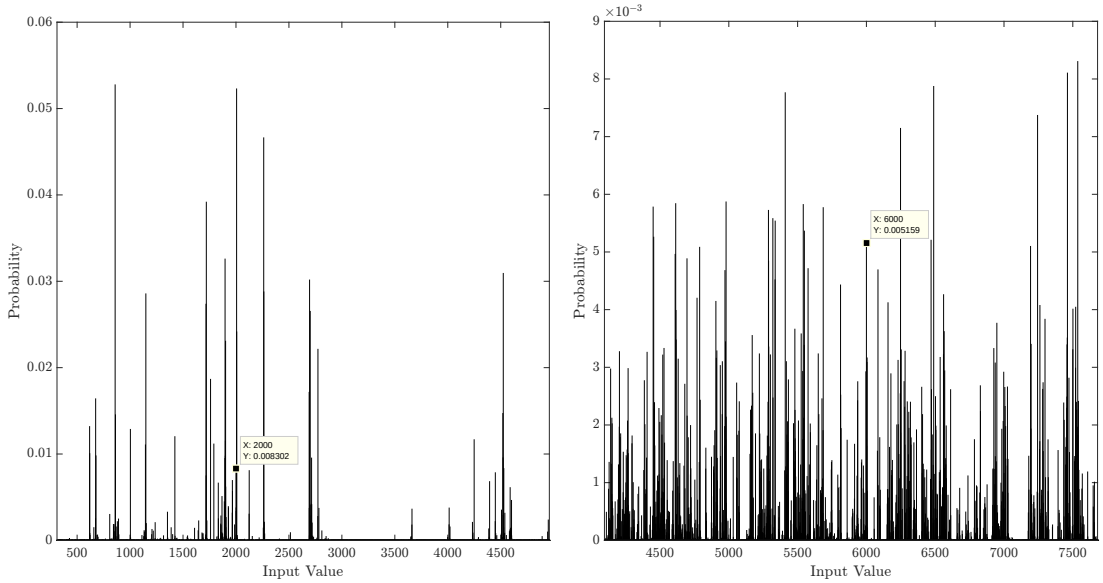
traces. This took about two days with our measurement setup. While storing such an amount of power traces may sound difficult, in practice the complete raw measurement data requires about 20GB of storage. Since the duration of a modular multiplication is rather short, each power trace only consists of 250 sample points.

After we completed the measurement phase, we applied trace compression on our measurement data. We identified points of interest (POI) with a T-Test (Section 3.3.3). On average, we identified about 6 POIs out of the 250 samples of each trace. These POIs feature a high variability between all input combinations corresponding to one known twiddle factor. By keeping only the calculated POIs of each set of power traces we could reduce the storage requirement to 2.4% of its original amount.

With this compressed measurement data we calculate a template $h_{\omega,d}$ for each known ω and each of its possible input combinations $d \in \mathcal{D}_\omega$. We evaluate the performance as follows. Out of the 100 traces per input combination we choose 99 traces at random in the template building phase. The one remaining trace is then used in the template matching phase. For every one ω and one of its timing classes $\mathcal{C}_{\omega,x}$ we calculate the probabilities (Section 3.3.3):

$$\mathrm{p}(t_{d'}|h_{\omega,d}) \quad \forall\ d \in \mathcal{C}_{\omega,x}, \tag{6.2}$$

where $t_{d'}$ corresponds to one recorded trace for the modular multiplication $\omega d' \bmod q$. Hence, we obtain a probability distribution over $\mathcal{C}_{\omega,x}$ for all ω. Figure 6.2 shows two such obtained probability distributions with varying informative value.



(a) input = 2000, ω = 3383, timing class = 4, q = 7681, entropy = 6.4-bit

(b) input = 6000, ω = 1, timing class = 2, q = 7681, entropy = 9.2-bit

Figure 6.2: Single-trace TA on two modular multiplications. We calculate the resulting probability distribution like in 6.2 and indicate the real input and its estimated probability in each plot.

TA on Modular Addition/Subtraction

The goal of our TA on modular addition/subtraction operations is to determine whether or not a reduction step is executed. As mentioned earlier, the implementation of the modular addition/subtraction is constant time. A simple thresholding approach based on timing is therefore not possible here. Nevertheless, we can build templates that can distinguish the two cases based on the difference in their power consumption. Hence, we exploit the fact that `NOP` instructions have a different power profile compared to `ADD`/`SUBS` instructions.

In our evaluation of this binary TA, the occurrence of a subsequent reduction step in a modular addition/subtraction can be determined with a success rate of > 0.99. Therefore we will assume correct knowledge about performed reductions for addition/subtraction operations in our butterfly network.

6.2.4 Side-Channel Attack on Simulated Leakage

Besides our side-channel analysis on a real device we repeat our analysis based on simulated leakage data to give more generic results and allow for easier reproducibility. For this purpose we use a simple leakage model, namely the noisy Hamming Weight model. In this leakage model the attacker gets two samples of the form:

$$l = (\ \mathrm{HW}(x_i) + \mathcal{N}(0,\sigma)\ ,\ \mathrm{HW}(x_i\omega_n^j \bmod q) + \mathcal{N}(0,\sigma)\ ), \tag{6.3}$$

where HW denotes the Hamming weight function, x_i denotes the used intermediate in the butterfly , ω_n^j denotes the corresponding twiddle factor, and $\mathcal{N}(0,\sigma)$ denotes a sample from a normal distribution with zero mean and standard deviation σ. We then perform a 2-variate template matching on this data. In the simulated attack, besides reduction information in the modular addition/subtraction, we now use probability distributions obtained from the TA on simulated leakage data instead of real leakage data. We then repeat this procedure for σ in range $[0, 0.1, \ldots, 1]$ to evaluate performance of our attack for varying noise levels.

6.2.5 Results - Real Leakage

We now present the results of our TA on the modular multiplication based on real leakage from our microprocessor implementation. As we will see, the information contained in the obtained probability distributions varies greatly and depends on the inputs of the modular multiplication operation. In Figure 6.4, we can see the probabilities obtained from a single-trace TA on the modular multiplication operation with $\omega = 3383$. To allow a simpler illustration, we only look at the timing classes 2 and 3. The plots show the performance of our TA by color-coding the obtained probability distributions. Each line represents a TA evaluation based on a single-trace template matching. A correct classification is accomplished if the value on the diagonal has the highest value in each row.

To give a better sense of how much information is contained within our obtained probability distributions, we will use the entropy metric as proposed in [76]. With entropy we can measure the information contained in a probability distribution. Given a probability

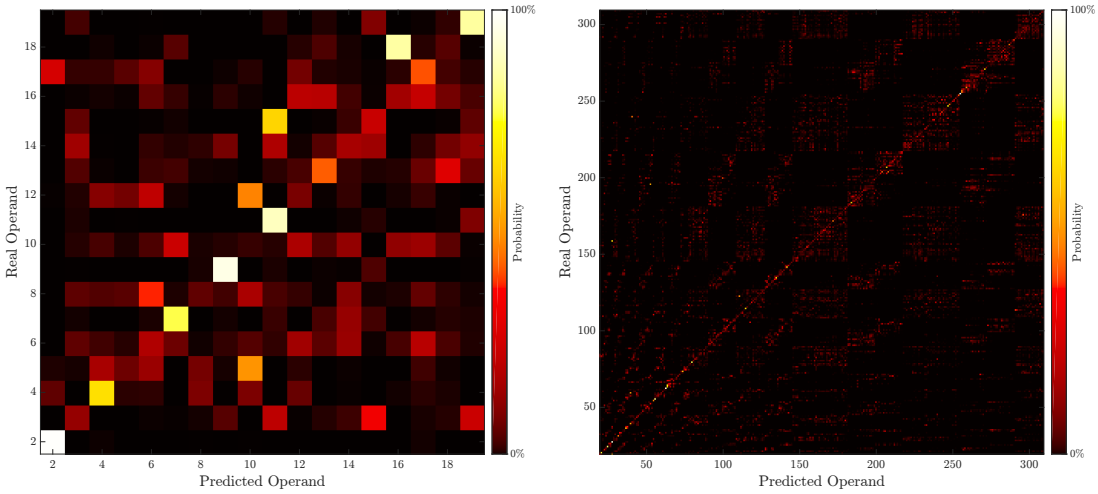
distribution X, the entropy H(X) is defined as:

$$\mathrm{H(X)} = -\sum_{i=1}^{q} \mathrm{p}(x_i) \log_2 \mathrm{p}(x_i), \tag{6.4}$$

where x_i denotes one of the q possible outcomes of a given probability distribution. Highest entropy is achieved with a uniform distribution (maximum uncertainty), while zero entropy is achieved if a distribution has one outcome with a probability of 1 (no uncertainty). The maximum achievable entropy for a distribution with q outcomes is $\log_2(q)$ bit, in our case $\log_2(7681) \approx 12.9$ bit. Since we can already eliminate possible intermediates solely based on timing information, the maximum entropy varies depending on the actual inputs of the modular multiplication. For example, in case of $\omega = 3383$ and timing class 2, the maximum entropy already is as small as 4.2 bit. However, for $\omega = 3383$ and timing class 4 the maximum entropy is still 12.1 bit.

On average, we have observed an entropy of about 7 bit for the probability distributions obtained from template attacks across all input combinations. While probability distributions corresponding to multiplications with large operands tend to have below-average entropy, multiplications with small operands tend be responsible for probability distributions with above-average entropy. One exceptional case in which our template attack performs very poorly is when $\omega = 1$. The average entropy is about 10.5 bit. This is unfortunate since the most frequent value of ω in an INTT butterfly is in fact 1.

Clearly, after performing a TA on the modular multiplication the obtained information is not precise enough to allow a reliable determination of the intermediates. On average the correct classification rate is below 10%. In order to improve these results, in the next section, we feed our obtained probability distributions into a graphical model and attempt to calculate marginal distributions of each intermediate based on the probability distributions of all other intermediates.



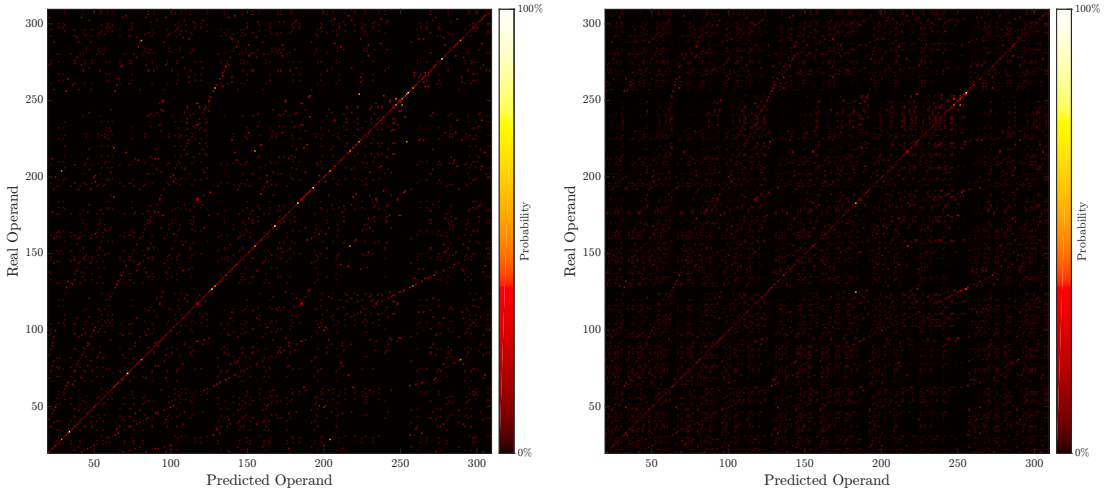
(a) TA on real leakage: Modular multiplication with $\omega = 3383$, timing class = 2.

(b) TA on real leakage: Modular multiplication with $\omega = 3383$, timing class = 3.

Figure 6.3: Classification results of single-trace template attack on real leakage.

6.2.6 Results - Simulated Leakage

The TA outcome based on simulated leakage data varies depending on the noise level in the leakage model. In Figure 6.4 we have illustrated our TA evaluation based on single-trace simulated leakage for the same setting as in Figure 6.3b. A visual comparison confirms that our simulated leakage data is indeed a good approximation of the real leakage data. For $\sigma = 0$, the average entropy of the obtained probability distributions is about as high as in case of real leakage data. For $\sigma = 0.5$, the simulated leakage data already contains significantly more noise, the average entropy is about 9 bit. It should be noted though that we do not use timing information from division in the TA evaluation for simulated leakage. The restriction of the possible intermediates in Figure 6.3b is therefore solely to allow an easier comparison. For the attack evaluation on simulated leakage data the domain of each intermediate is always $[0, \ldots, 7680]$.



(a) TA on simulated leakage: Modular multiplication with $\omega = 3383$, $\sigma = 0$, timing class = 3.

(b) TA on simulated leakage: Modular multiplication with $\omega = 3383$, $\sigma = 0.5$, timing class = 3.

Figure 6.4: Classification results of single-trace template attack on simulated leakage.

6.3 Attack Step 2: Belief Propagation in an NTT Butterfly Network

In this section, we show how we apply the BP algorithm in order to improve the information gain for each intermediate after our side-channel analysis. The BP algorithm requires a graphical model of the target function. Thus, we have to construct a factor graph of the NTT. The way we construct our factor graph is described in Section 6.3.1.

We perform the loopy BP algorithm to calculate approximate marginal distributions for all intermediates in our NTT network. Unfortunately, in our case the runtime of a standard BP implementation is rather high. Therefore, we identified time consuming operations and propose optimizations that significantly reduce algorithmic runtime in Section 6.3.2. In Section 6.3.4, we show the outcome of the BP algorithm. We additionally propose adaptions for the constructed factor graph to further improve results.

6.3.1 Factor Graph Construction

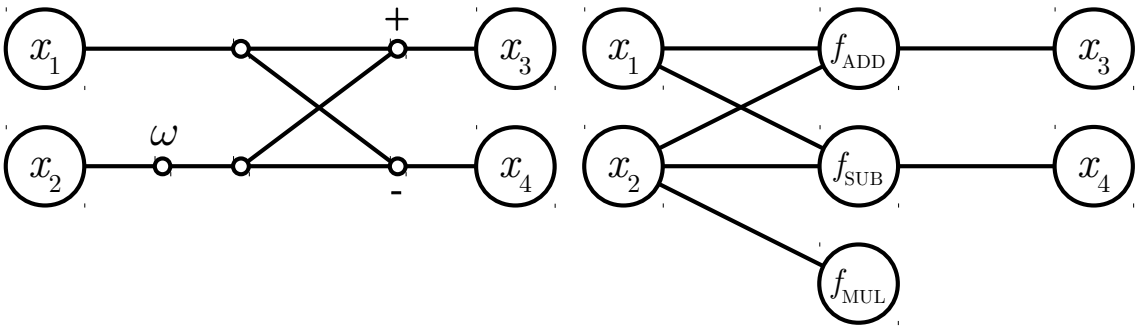
A factor graph (Section 5.2) is a graphical model that represents the factorization of a function, in our case the INTT operation. The graph consists of two type of nodes, variable nodes and factor nodes. Variable nodes represent input, output, as well as intermediate variables of the described function. Factor nodes are used to describe the dependencies between those variable nodes. Since factor graphs are required to be bipartite, variable nodes are only connected to factor nodes and vice versa. In the following, we describe how we can represent the butterfly network corresponding to an INTT operation as a factor graph that is made up of variable nodes and factor nodes.

Variable Nodes

A variable node represents the current probability distribution of one intermediate in our INTT butterfly network. At any time, it stores one belief from every connected factor node. A belief is an estimated probability distribution of the variable node from the view of the connected factor node. The domain $\mathcal{D}$ of a variable node is determined by the modulus q. In our case, $q = 7681$ and the variable domain is therefore $[\, 0, \ldots, 7680 \,]$. Given the beliefs of all connected factor nodes, the current approximate marginal distribution of a variable node can be calculated by computing the product over all current beliefs.

Factor Nodes

Factor nodes model the dependency between variable nodes. Given a set of beliefs from all connected variable nodes, a factor node updates the beliefs corresponding to the represented dependency. Clearly, the implementation of factor nodes differs for every different type of dependency between variable nodes. In the factor graph representing an INTT butterfly network, variable nodes are connected via modular addition, modular subtraction and modular multiplication. Therefore we use the 3 types of factor nodes: f_{ADD}, f_{SUB} and f_{MUL} as seen in Figure 6.5. We now explain their implementation.



(a) 2-coefficient butterfly (b) Corresponding factor graph

Figure 6.5: Butterfly network (left) and our corresponding factor graph (right)

Factor Node - Modular Multiplication

The implementation of f_{MUL} is comparably simple since its only purpose is to add side-channel information of the modular multiplication operation into the factor graph. In

every iteration of the BP algorithm this node sends a belief to the connected variable node that consists of the probability vector obtained from the TA.

$$f_{\mathrm{MUL}}(x_{i_2}) = \Pr(\ x_2 = x_{i_2} \mid l\ )$$

Factor Node - Modular Addition

A factor node of type f_{ADD} is used to model the dependency between variable nodes that are connected via a modular addition operation. In Figure 6.5, the value of x_3 is determined by the modular addition of x_1 and $x_2\omega$. Hence, given the distributions of x_1, x_2 and x_3, f_{ADD} is supposed to increase the probability of likely value combinations while reducing the probability of unlikely or even impossible value combinations. Such an operation can be modeled by a joint distribution table. We can assign the probability (0 or 1) to each value combination of x_1, x_2 and x_3, depending on whether or not they can occur in a modular addition. We start by modelling a modular addition as follows:

$$f_{\mathrm{ADD}}(x_{i_1}, x_{i_2}, x_{i_3}) = \begin{cases} 1 \text{ if } x_{i_1} + x_{i_2}\omega \equiv x_{i_3} \bmod q \\ 0 \text{ otherwise} \end{cases}$$

We obtain a joint distribution table that models the dependency of the two summands and their sum for a modular addition. Given the beliefs of x_1, x_2 and x_3, an update for x_1 can be calculated by building the product over the beliefs from x_2 and x_3, multiplying the result with the joint distribution table and finally summing out all variables other than x_1. Updates for x_2 and x_3 can be calculated in a similar way.

Note that we assume knowledge about the occurrence of a reduction in every modular addition/subtraction. We include this information by splitting up our modular addition model into the two cases with/without reduction. For modular addition with reduction a combination x_1, x_2, x_3 is only possible if additionally $x_{i_1} + (x_{i_2}\omega \bmod q) \geq q$, i.e. a reduction is performed:

$$f_{\mathrm{ADD_{Red}}}(x_{i_1}, x_{i_2}, x_{i_3}) = \begin{cases} 1 \text{ if } x_{i_1} + x_{i_2}\omega \equiv x_{i_3} \bmod q \text{ and } x_{i_1} + (x_{i_2}\omega \bmod q) \geq q \\ 0 \text{ otherwise} \end{cases}$$

Similarly, for a modular addition without reduction we additionally require that no reduction is performed:

$$f_{\mathrm{ADD_{NoRed}}}(x_{i_1}, x_{i_2}, x_{i_3}) = \begin{cases} 1 \text{ if } x_{i_1} + x_{i_2}\omega \equiv x_{i_3} \bmod q \text{ and } x_{i_1} + (x_{i_2}\omega \bmod q) < q \\ 0 \text{ otherwise} \end{cases}$$

Factor Node - Modular Subtraction

The implementation of f_{SUB} is very similar to f_{ADD}. Instead of x_3, the variable x_4 is now used as the result of the operation. The entries of a joint distribution table representing a modular subtraction is given by:

$$f_{\mathrm{SUB}}(x_{i_1}, x_{i_2}, x_{i_4}) = \begin{cases} 1 \text{ if } x_{i_1} - x_{i_2}\omega \equiv x_{i_4} \bmod q \\ 0 \text{ otherwise} \end{cases}$$

Again, we introduce side-channel information by splitting up our model in the two cases with/without reduction. For modular subtraction with reduction we additionally require

that $x_{i_1} - (x_{i_2}\omega \bmod q) < 0$, i.e. a reduction is performed:

$$f_{\mathrm{SUB_{Red}}}(x_{i_1}, x_{i_2}, x_{i_4}) = \begin{cases} 1 \text{ if } x_{i_1} - x_{i_2}\omega \equiv x_{i_4} \bmod q \text{ and } x_{i_1} - (x_{i_2}\omega \bmod q) < 0 \\ 0 \text{ otherwise} \end{cases}$$

For modular subtraction without reduction we require that no reduction is performed:

$$f_{\mathrm{SUB_{NoRed}}}(x_{i_1}, x_{i_2}, x_{i_4}) = \begin{cases} 1 \text{ if } x_{i_1} - x_{i_2}\omega \equiv x_{i_4} \bmod q \text{ and } x_{i_1} - (x_{i_2}\omega \bmod q) \geq 0 \\ 0 \text{ otherwise} \end{cases}$$

Additional Leakage

Using a factor graph representation allows us to add additional side-channel information at virtually every point if available. Any kind of prior probability can simply be added to the corresponding variable node as another factor node with degree 1 (see f_{MUL}). One possible location for additional side-channel information are the modular addition/subtraction operations, as we only use information about the occurrence of a reduction. Also the additional leakage of `LOAD` and `STORE` instructions may be included since they are a typical side-channel attack target.

6.3.2 Belief Propagation Runtime Analysis

We now give an estimation for the runtime of the BP algorithm based on the factor graph which is described in the previous section. As we will see, the standard implementation of the algorithm has such a high runtime that additional optimizations are necessary to ensure practicality of the presented attack. These optimizations are presented in the following section.

The expected runtime of the BP algorithm is characterized by the number of iterations, the number of variable nodes, the domain of the variable nodes, the number of factor nodes, and the degree of the factor nodes.

Each iteration of the BP algorithm involves the invocation of the update rules $\mathfrak{q}$ (variable nodes to factor nodes) as well as $\mathfrak{r}$ (factor nodes to variable nodes) for all nodes in the factor graph. The number of necessary iterations is usually small, in our case ≤ 25, and therefore does not have a significant impact on the asymptotic runtime.

Update rule $\mathfrak{q}$ of a variable node x_i with degree $\deg(x_i)$ and its neighbors $f_{i_1}, \ldots, f_{i_{\deg(x_i)}}$ consists of a multiplication of $\deg(x_i) - 1$ incoming beliefs. The runtime complexity of this step is comparably low as the number of needed multiplications is linear in $\deg(x_i)$.

The step $\mathfrak{r}$ of a factor node f_i with degree $\deg(f_i)$ and its neighbors $x_{i_1}, \ldots, x_{i_{\deg(f_i)}}$ with domain $\mathcal{D}$ consists of addition and multiplication operations performed on a joint distribution table of size $|\mathcal{D}|^{\deg(f_i)}$. The runtime and memory complexity of $\mathfrak{r}$ grows exponentially in the degree of f_i. Therefore, the factor node with highest degree usually determines the overall complexity of the algorithm.

In our scenario, i.e. a factor graph corresponding to a 256-coefficient INTT operation, the highest degree of a factor node is 3 and the domain size $|\mathcal{D}|$ of every variable node is 7681. The resulting size of a joint distribution table, as used in step $\mathfrak{r}$, is $7681^3 \approx 2^{38}$. Taking additionally into consideration that the number of factor nodes with degree 3 is 2048, the runtime complexity of performing one iteration of the BP algorithm can be estimated as $\approx 2^{49}$. Even though such a runtime can be considered as still feasible, it would drastically reduce the practicality of our attack. Hence, we tried to find a more efficient way to compute $\mathfrak{r}$ for our degree 3 factor nodes.

6.3.3 Belief Propagation Performance Improvements

We now present an optimization that allows us to compute the update rule $\mathfrak{r}$ for all factor nodes of degree 3 in our factor graph with a drastically reduced runtime. By using this optimization, we can perform one iteration of the BP algorithm for our whole factor graph in about one minute on a single core of an Intel Core i7-5600U notebook grade CPU and negligible memory. Our factor nodes with degree 3 are of type f_{ADD} and f_{SUB}. They are used to model modular additions and a modular subtractions which occur in an INTT operation.

The key idea behind our optimization is the fact that update rules for distributions of input/output values of modular additions/subtractions can be efficiently expressed in matrix vector notation. Let us consider the modular addition:

$$a + b = c \mod q,$$

and additionally the vectors $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ that contain the current probability distribution of each respective variable. Such a probability distribution assigns a probability to each of the possible outcomes:

$$a_i = \Pr(a = i), \quad b_i = \Pr(b = i), \quad c_i = \Pr(c = i)$$

f_{ADD} updates the probability distribution of each variable based on the distribution of the other two variables. Hence, the updated distribution $\mathbf{c}^*$ depends on $\mathbf{a}$, $\mathbf{b}$. One entry c_k^* can be computed as the sum over all a_i, b_j with $i + j \equiv k \mod q$. This update rule can be written in matrix-vector notation as follows:

$$\begin{bmatrix} a_0 & a_{q-1} & \cdots & a_2 & a_1 \\ a_1 & a_0 & a_{q-1} & & a_2 \\ \vdots & a_1 & a_0 & \ddots & \vdots \\ a_{q-2} & & \ddots & \ddots & a_{q-1} \\ a_{q-1} & a_{q-2} & \cdots & a_1 & a_0 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{q-2} \\ b_{q-1} \end{bmatrix} = \begin{bmatrix} c_0^* \\ c_1^* \\ \vdots \\ c_{q-2}^* \\ c_{q-1}^* \end{bmatrix},$$

where the left matrix is a circulant matrix obtained by circularly shifting $\mathbf{a}$. This equation can be written as the circular convolution:

$$\mathbf{a} \star \mathbf{b} = \mathbf{c}^*$$

The circular convolution theorem can be used to transform the cyclic convolution into the component-wise multiplication:

$$\mathrm{FFT}_q(\mathbf{a} \star \mathbf{b}) = \mathrm{FFT}_q(\mathbf{a}) * \mathrm{FFT}_q(\mathbf{b}) = \mathrm{FFT}_q(\mathbf{c}^*),$$

and the updated probabilities for $\mathbf{c}^*$ are given by:

$$\mathbf{c}^* = \mathrm{IFFT}_q(\, \mathrm{FFT}_q(\mathbf{a}) * \mathrm{FFT}_q(\mathbf{b})\,)$$

The update rules for $\mathbf{a}^*$ and $\mathbf{b}^*$ can be obtained similarly by additionally using complex conjugations:

$$\mathbf{a}^* = \mathrm{IFFT}_q(\,\mathrm{FFT}_q(\mathbf{c}) * \mathrm{CONJ}(\mathrm{FFT}_q(\mathbf{b}))\,)$$
$$\mathbf{b}^* = \mathrm{IFFT}_q(\,\mathrm{FFT}_q(\mathbf{c}) * \mathrm{CONJ}(\mathrm{FFT}_q(\mathbf{a}))\,)$$

Note that our implementation of f_{ADD} also incorporates binary side-channel information about the execution of a reduction step. We can model the impact of a reduction step by replacing the q-coefficient FFTs by $2q$-coefficient FFTs plus a final selection of the upper or lower half of the IFFT output.

A summary of our efficient f_{ADD} implementation can be found in Algorithm 2. For f_{SUB} only minor modifications to the algorithm are necessary. By using Algorithm 2 the runtime cost of performing update rule $\mathfrak{r}$ for our degree 3 factor nodes is reduced to $\mathcal{O}(q \log q)$ since now the only runtime relevant operations are FFT's.

Algorithm 2: Efficient BP for Modular Addition

Input:

$\mathbf{a}, \mathbf{b}, \mathbf{c}$	Probability distributions of summands and result
Reduction	True if a reduction step was executed

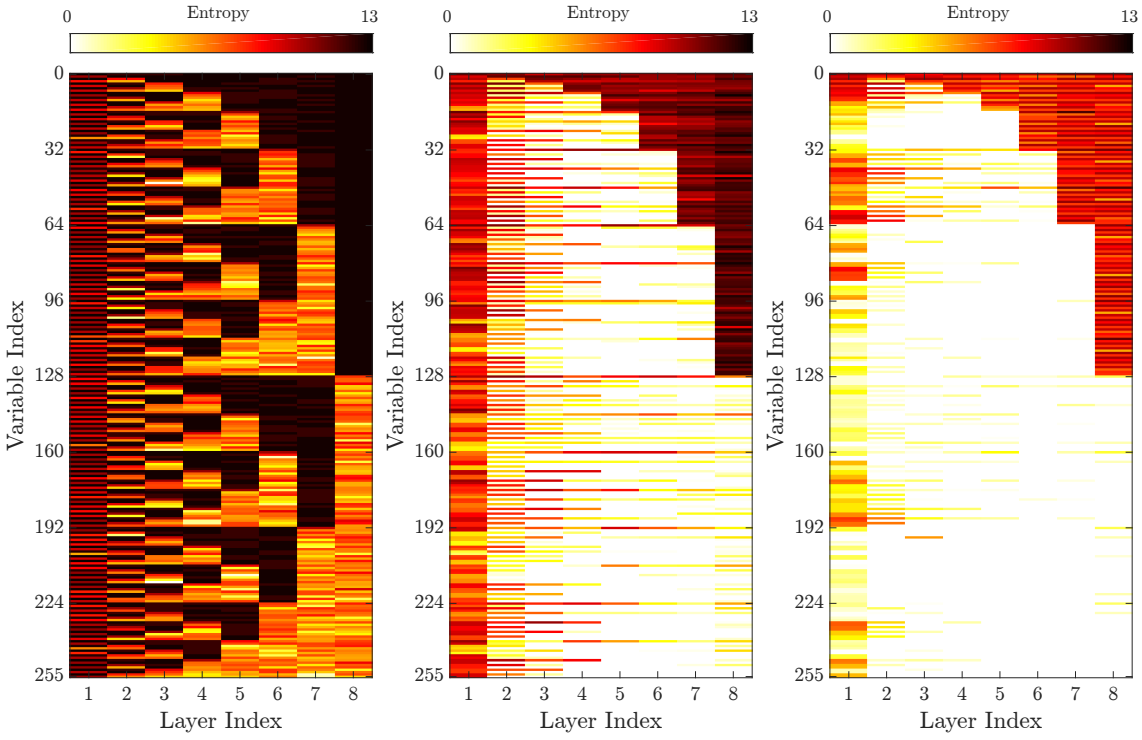
Output:

$\mathbf{a}^*, \mathbf{b}^*, \mathbf{c}^*$	Updated probability distributions of summands and result

1: $\widetilde{\mathbf{a}} = \mathrm{FFT}_{2q}(\mathbf{a})$
2: $\widetilde{\mathbf{b}} = \mathrm{FFT}_{2q}(\mathbf{b})$
3: $\widetilde{\mathbf{c}} = \mathrm{FFT}_{2q}(\mathbf{c})$
4: $\mathbf{t}_a = \mathrm{IFFT}_{2q}(\mathrm{CONJ}(\widetilde{\mathbf{b}}) * \widetilde{\mathbf{c}})$
5: $\mathbf{t}_b = \mathrm{IFFT}_{2q}(\mathrm{CONJ}(\widetilde{\mathbf{a}}) * \widetilde{\mathbf{c}})$
6: $\mathbf{t}_c = \mathrm{IFFT}_{2q}(\widetilde{\mathbf{a}} * \widetilde{\mathbf{b}})$
7: **if** Reduction **then**
8: $\quad \mathbf{a}^* = \mathbf{t}_a\,[\,q \;...\; 2q-1\,]$
9: $\quad \mathbf{b}^* = \mathbf{t}_b\,[\,q \;...\; 2q-1\,]$
10: $\quad \mathbf{c}^* = \mathbf{t}_c\,[\,q \;...\; 2q-1\,]$
11: **else**
12: $\quad \mathbf{a}^* = \mathbf{t}_a\,[\,0 \;...\; q-1\,]$
13: $\quad \mathbf{b}^* = \mathbf{t}_b\,[\,0 \;...\; q-1\,]$
14: $\quad \mathbf{c}^* = \mathbf{t}_c\,[\,0 \;...\; q-1\,]$
15: **end if**

6.3.4 Applying the BP algorithm

With a factor graph that represents an INTT operation and probability distributions for the intermediates obtained from our TA, we applied the BP algorithm to calculate approximate marginal distributions for all variable nodes. In our first attempt we performed the BP algorithm on the whole factor graph. While we were able to narrow down the possible values of most variable nodes, the results were not good enough to allow a subsequent key recovery.



(a) 1 Iteration (b) 8 Iterations (c) 25+ Iterations

Figure 6.6: Application of BP on the full factor graph. We show the entropy of each variable node after set number of iterations. No convergence is achieved after 25+ iterations.

Figure 6.6 shows the outcome of the BP algorithm on the whole factor graph. Each cell represents the entropy of one variable node in our factor graph. We color-code the entropy from zero (white) to 13 (black). Hence, white cells indicate that one distinct value has been determined and black cells indicate maximum uncertainty. After the first iteration, we see the introduction of prior probabilities into our factor graph. In the following iterations, the BP algorithm attempts to calculate marginal distributions for every variable node. After 25 iterations the overall entropy of all variable nodes is not decreasing anymore. In many areas of the factor graph the estimated probability distributions are oscillating between two or more states in consecutive iterations.

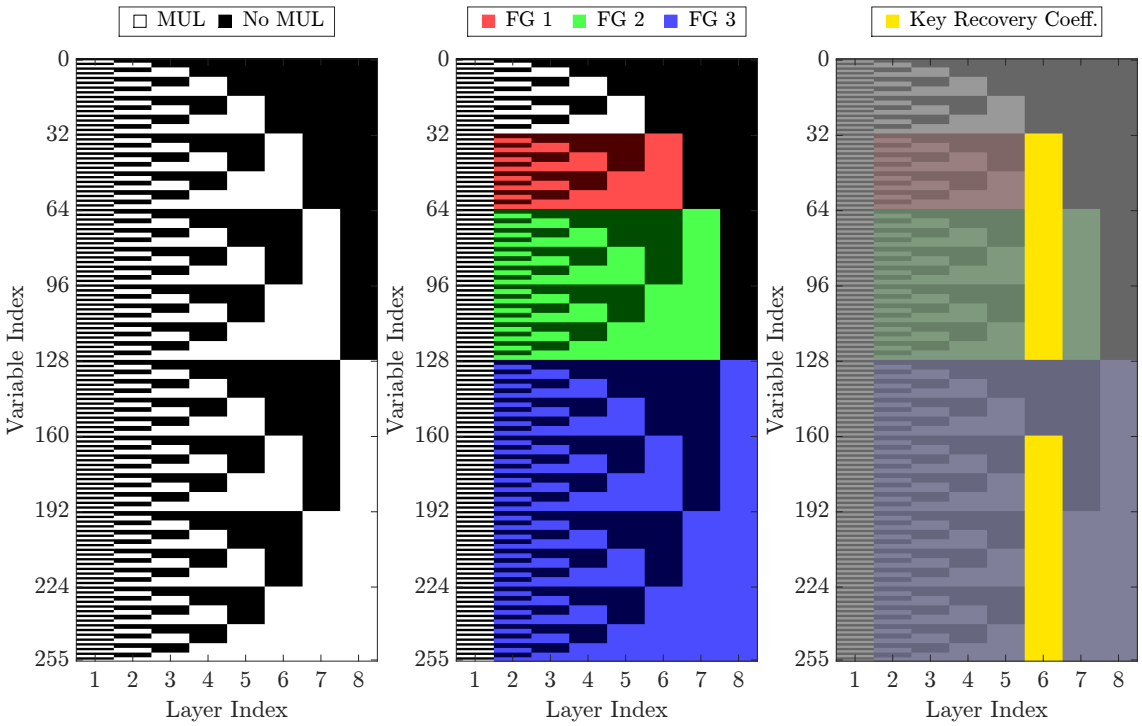
We identified two major reasons why the BP algorithm fails to calculate distinct marginal distributions when run on the whole factor graph.

Problem 1: Uneven availability of side-channel information

The first problem is the uneven distribution of side-channel information in our factor graph. Figure 6.7a illustrates the location of side-channel information, i.e. modular multiplications in our factor graph. Each cell represents one butterfly containing 2 variable nodes and 2-3 factor nodes. Black cells contain the factors $f_{\mathrm{ADD}}, f_{\mathrm{SUB}}$, white cells additionally contain f_{MUL}. It is easy to see that while many multiplications are executed in the bottom-right corner of the graph, no multiplications are executed in the top-right corner. The BP algorithm struggles with calculating marginal distributions especially in areas where little side-channel information is available.

Problem 2: Varying outcome of the TA

The results of our TA on modular multiplications vary in informativeness depending on the used operands. The probability vectors obtained from multiplications with large operands yield better information than the probability vectors obtained from attacking multiplications with small operands. One of the operands, the twiddle factor, is set to 1 for all multiplications in the first layer. As a consequence, no reduction is executed, the value of operands is rather low, and the resulting probability vectors contain little useful information.



(a) Locations of multiplications in our factor graph

(b) The 3 factor graphs we use for the key recovery attack

(c) The coefficients which are used in the key recovery algorithm

Figure 6.7: Based on the location of side-channel information in 6.7a, we decided to divide our factor graph in three independent parts as shown in 6.7b. We then use the coefficients highlighted in 6.7c for key recovery.

To circumvent these two problems, we separated the whole factor graph into the 3 smaller and independent factor graphs FG1, FG2, and FG3. This is depicted in Figure 6.6b. Applying the BP algorithm independently on these factor graphs gives significantly better results. The rather noisy side-channel information from layer 1 is neglected and the overall ratio between observed and unobserved variables is higher. Even though FG1, FG2, and FG3 do not contain as many variables as the whole factor graph, they still contain enough variables for a full key recovery. We have chosen to use variables from layer 6 for key recovery, since layer 6 is the last layer of FG1 and variables in later layers are usually recovered with higher confidence by the BP algorithm. As we will see in the next section, knowing 192 out of the possible 256 coefficients in layer 6 allows us to perform a full private

key recovery in about one minute. In our attack, we use the variables 32 ... 127 and 160 ... 255 in layer 6 for key recovery as shown in Figure 6.6c

6.4 Attack Step 3: Private Key Recovery

The application of the BP algorithm can significantly improve the results of the previously executed TA by combining all available leakage information of the attacked operation. Yet, a simple key recovery is still not possible since there is still too much uncertainty in certain parts of our factor graph. Of high interest to an attacker is the input of the INTT, i.e. the marginal distributions of variable nodes in layer 1. Given the input of the INTT operation $\widetilde{\mathbf{x}}$ that corresponds to:

$$\widetilde{\mathbf{x}} = \widetilde{\mathbf{c}}_1 * \widetilde{\mathbf{r}}_2 + \widetilde{\mathbf{c}}_2, \tag{6.5}$$

the private key $\widetilde{\mathbf{r}}_2$ can be calculated by:

$$\widetilde{\mathbf{r}}_2 = (\widetilde{\mathbf{x}} - \widetilde{\mathbf{c}}_2) * \widetilde{\mathbf{c}}_1^{-1} \tag{6.6}$$

In our case, a more sophisticated key recovery approach is necessary since the INTT input cannot be determined with high enough confidence, as shown in Figure 6.6c. However, if we divide our factor graph in the 3 subgraphs FG1, FG2, and FG3, as described in the previous section, we can determine many (in our example 192) intermediate variables in layer 6 with high confidence. One immediate consequence of knowing intermediates in layer 6 is that also some values in layer 1, the INTT input are known. Given our choice of 192 intermediates in layer 6, 160 intermediates in layer 1 can be determined by a partial inversion of the INTT for the indices $[32 \dots 63, 64 \dots 127, 192 \dots 255]$. This is the a result of the recursive structure of butterfly networks. Hence, by using 6.6 we can calculate 160 coefficients of $\widetilde{\mathbf{r}}_2$, leaving only 96 coefficients left to be determined. A simple brute force attack is still infeasible though since the value domain of the 96 each unknown coefficients is 7681 and the resulting search space is $7681^{96} \approx 2^{1239}$.

A more efficient approach to recover the remaining key coefficients is shown in a pending submission to the CHES 2017 conference. There, we show in a collaboration with Peter Pessl that there exists an efficient way to recover all private key coefficients of a partially known private key in an RLWE encryption scheme. First, we generate linear equations in the private key depending on the available intermediates. The final key recovery is then performed by means of a lattice reduction.

6.4.1 Generating Linear Equations in the Key

We now describe how information about intermediates in the INTT butterfly network can be combined with the public key of a corresponding RLWE decryption operation. We start by building equations in the private key $\widetilde{\mathbf{r}}_2$ that depend on intermediates that can be determined with high confidence. Recall that we can rewrite polynomial multiplication in the ring $\mathcal{R}_q$ in matrix-vector notation. Hence, we can express the INTT output $\mathbf{m}^\star$ as:

$$\mathbf{m}^\star \;=\; \mathbf{c}_1 * \mathbf{r}_2 + \mathbf{c}_2 \;=\; \mathbf{C}_1\mathbf{r}_2 + \mathbf{c}_2, \tag{6.7}$$

where $\mathbf{C}_1$ corresponds to a nega-cyclic matrix obtained by repeated nega-cyclic shifts of $\mathbf{c}_1$. A nega-cyclic shift means an element-wise rotation of a vector with a subsequent negation of the first element.

Since all operations inside the (I)NTT are linear, we can transform our equation system such that it describes any of its intermediates. In our case, we want to describe the known intermediates in layer 6. Therefore, going back from the INTT output, we need to partially revert the last 3 layers of the butterfly network until we reach the known intermediates, i.e. the inputs of layer 6. The partial NTT reversal can be computed simply by:

$$\begin{aligned} x_1 &= \frac{x_3 + x_4}{2} \mod q \\ x_2 &= \frac{x_3 - x_4}{2\omega} \mod q, \end{aligned}$$

where $x_1, \ldots, x_4$ and ω denote the input/output and twiddle factor corresponding to a 2-coefficient butterfly as illustrated in Figure 6.5. As a result, we build an equation system:

$$\mathbf{C}_1'\mathbf{r}_2 + \mathbf{c}_2' \;=\; \mathbf{x}, \tag{6.8}$$

where $\mathbf{x}$ contains, the 196 known intermediates from layer 6 and $\mathbf{C}_1'$, $\mathbf{c}_2'$ contain the transformed coefficients.

6.4.2 Key Recovery using Lattice Reduction

Once we have a linear equation system that relates $\mathbf{r}_2$ to our known intermediates we can define a Closest Vectors Problem (CVP) (Section 2.3.2) by additionally embedding the public information $(\mathbf{a}, \mathbf{p})$. $\mathbf{a}$ defines our ideal lattice and challenge vector $\mathbf{p}$ is defined as:

$$\mathbf{p} = \mathbf{r}_1 - \mathbf{a}\mathbf{r}_2, \tag{6.9}$$

where $\mathbf{r}_1$ is a Gaussian error term used only during key generation. In CVP the solver is required to find a vector that is sufficiently close to the challenge vector. The parameters for our RLWE-based encryption system are chosen such that solving this CVP problem is infeasible. One possible solution is $-\mathbf{a}\mathbf{r}_2$ since $\mathbf{r}_1$ is small by definition.

We can use the linear equations from the previous section to reduce the complexity of the corresponding CVP. More precisely, we can reduce the dimension of the given lattice up to a point where solving CVP is feasible.

First we substitute the 192 equations from 6.8 into 6.9 to get:

$$\mathbf{p}' = \mathbf{r}_1 - \mathbf{A}'\mathbf{r}_2'$$

The number of columns in $\mathbf{A}'$ is hereby reduced to 64, as is the lattice dimension of the corresponding CVP. Solving CVP for this lattice is feasible. We can solve CVP by adding the challenge vector $\mathbf{p}'$ to our lattice $(\mathbf{A}'||\mathbf{p}')$, followed by a subsequent search for an unusually short vector $\mathbf{r}_1$ (or $-\mathbf{r}_1$). For this purpose we use the BKZ implementation from Shoups NTL library [74]. Once a candidate solution for $\mathbf{r}_1$ (or -$\mathbf{r}_1$) is found we can calculate all coefficients of $\mathbf{r}_2$ by solving the linear system from 6.9. The correctness of $\mathbf{r}_2$ is ensured if its distribution follows the error distribution used during key generation. That is, either a binary uniformly random distribution or a Gaussian distribution with zero mean and small standard deviation.

The runtime cost of the BKZ basis reduction algorithm, when applied to our reduced CVP, is comparably low. On average, the single thread runtime is about 45 seconds on a Intel Xeon E5-2699v4 CPU. We also validated the correctness of our key recovery by repeating it 1000 times for 192 correct layer 6 intermediates. The key recovery was always successful.

Chapter 7

Results

We now present the results of our attack on a microprocessor implementation of the RLWE-based asymmetric encryption scheme. The encryption scheme was proposed by Lyubashevsky et al. [49], our microprocessor implementation is based on an efficient software implementation by deClercq et al. [23]. The attack was evaluated both for real leakage data as well as simulated leakage data. We also consider a masked implementation of the INTT operation, used in virtually all efficient RLWE-based cryptographic schemes, as proposed by Reparaz et al. [69].

In Section 7.1, we show the performance of our attack on a microprocessor implementation running on an ARM-Cortex-M4F. Here, leakage data is obtained by an EM-based side-channel analysis. In Section 7.2, we repeat the attack, yet we use simulated leakage data based on a noisy Hamming weight leakage model to give more generic results.

7.1 Results for Real Leakage

In step 1 of our attack based on real leakage data, we performed a side-channel analysis on a microprocessor implementation (Section 6.2). We used an ordinary single-trace TA on the modular multiplication operation to approximately recover intermediate values within the INTT operation.

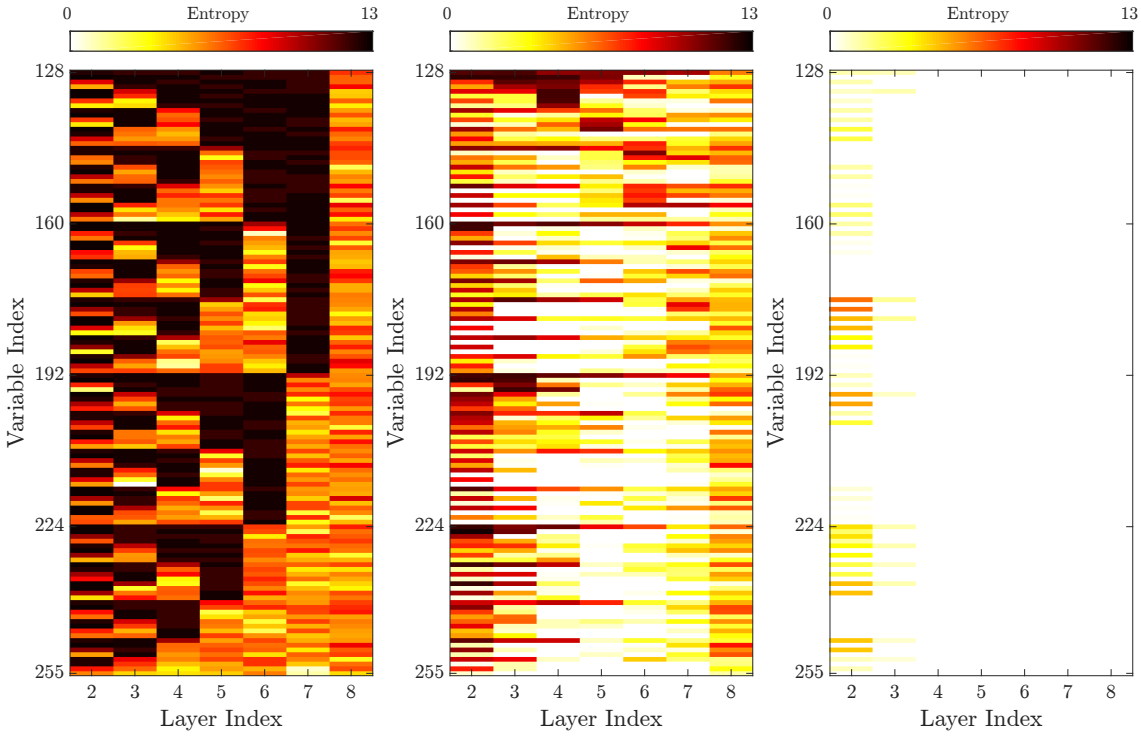
To simplify our attack setup we performed the TA on an isolated modular multiplication operation. For each possible input combination we recorded 100 traces. We then simulated one decryption by calculating all INTT intermediates for a known cipher text and key. In the attack phase, we have chosen 1 out of the 100 traces for every occurring input combination of the modular multiplication for template matching. The remaining traces were used in the corresponding template building phases.

We also performed a binary TA on the modular addition/subtraction operation to detect whether or not a reduction was executed. Since we achieved a high success rate for the detection of reductions, we assumed this knowledge in the rest of our attack.

In step 2, we then embed the obtained leakage information into a factor graph representation of our attacked INTT operation and apply the BP algorithm (Section 6.3). The outcome of applying the BP algorithm on the whole factor graph is not satisfactory. Hence, we divide our factor graph into 3 independent subgraphs to improve the performance of the BP algorithm. After a successful application of the BP algorithm on all subgraphs, we can determine a large part of the intermediates in layer 6 of our factor graph with high confidence.

An example of an outcome of applying the BP algorithm on the subgraph FG3 is illustrated in Figure 7.1. We color-code the entropy (6.4) of each intermediate. Black nodes indicate maximum uncertainty while white nodes indicate that the value is determined. After one iteration, the side-channel information from step 1 is introduced into the factor graph. After about 20 iterations, all intermediates in the right half of the subgraph are determined. The results for FG1 and FG2 are similar. The outcome of step 2 is the reliable recovery of 192 intermediates [32 ... 63, 64 ... 127, 160 ... 255] in layer 6 of the attacked INTT operation. We have repeated this step many times for differing decryption operations. In our evaluation, the success rate of recovering the 192 layer-6 intermediates is 1.

Based on the results of step 2, we then perform a full key recovery based on lattice reduction in step 3 (Section 6.4). Given the 192 correct intermediates from step 2 we setup 192 linear equations in the private key. We use these equations to reduce the lattice of the corresponding CVP problem to 64. Such a problem instance is considered feasible. We use the BKZ algorithm to recover all coefficients of the private key. The success rate of step 3 is also 1. Therefore, we conclude that the success probability of our attack in this scenario is also 1.



(a) 1 Iteration (b) 4 Iterations (c) 20 Iterations

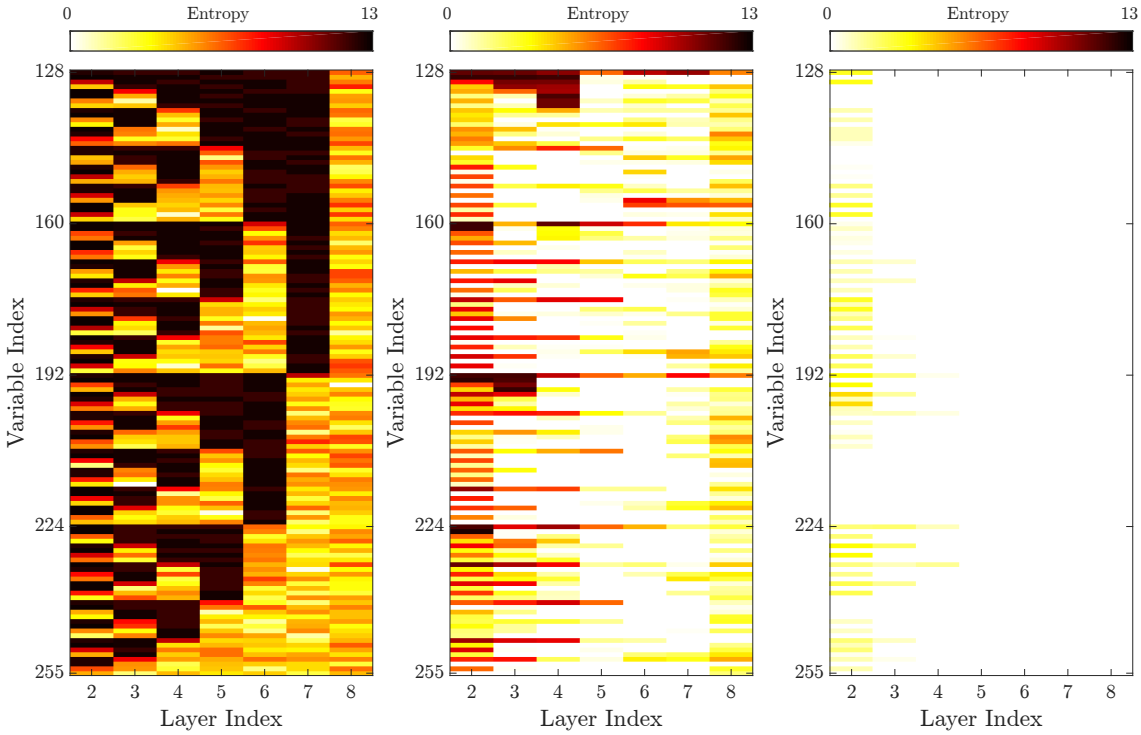
Figure 7.1: Application of BP on the subgraph FG3. We show the entropy of each intermediate after a set number of iterations. Intermediates in e.g. layer 6 are determined with high confidence. The results for FG1 and FG2 are similar.

7.2 Results for Simulated Leakage

In our attack evaluation for simulated leakage data we perform the same attack steps 1-3, yet in step 1 we instead use simulated leakage data for our TA on the modular multiplication operation. The simulated leakage data is derived from a noisy Hamming weight leakage model and is described in Section 6.2. We have repeated the attack for the varying noise parameter σ in the range of $[0, 0.1, \ldots, 1]$. Figure 7.2 shows the outcome of applying the BP algorithm on FG3 with $\sigma = 0.2$ (also compare to Figure 7.1). In the shown scenario, all intermediates in the right half of the factor graph are determined correctly, yet compared to real leakage data more iterations were necessary here.

An overview of the results is shown in Figure 7.3. We show the attack success rate and average entropy of the prior probabilities. In contrast to the attack on real leakage data, the success probability is not 1 anymore. Even in the case that $\sigma = 0$, the recovered values of intermediates in layer 6 contains a few incorrect entries with a probability of about 0.05. In such a case, key recovery in step 3 fails as well. Nevertheless, for $\sigma \leq 0.4$ the success rate of our attack is above 0.9.

In the case of an unmasked implementation the success rate can be increased by repetition. More precisely, we can perform multiple side-channel measurements for each operation, average the obtained traces, perform our TA and continue with attack step 2 as usual. Thus, noise can be decreased arbitrarily. Albeit of requiring multiple traces, our attack is not single-trace anymore and cannot be used for masked implementations. When attacking a masked implementation we need to successfully perform the steps 1-2 of our attack twice in a row. The obtained shared intermediates can then be combined and a normal key recovery is performed according to attack step 3. As expected, the success rate on masked implementations is about the square of the success rate in the unmasked case.



(a) 1 Iteration (b) 4 Iterations (c) 25 Iterations

Figure 7.2: Application of BP on the subgraph FG3. We show the entropy of each intermediate after a set number of iterations. The results for FG1 and FG2 are similar.

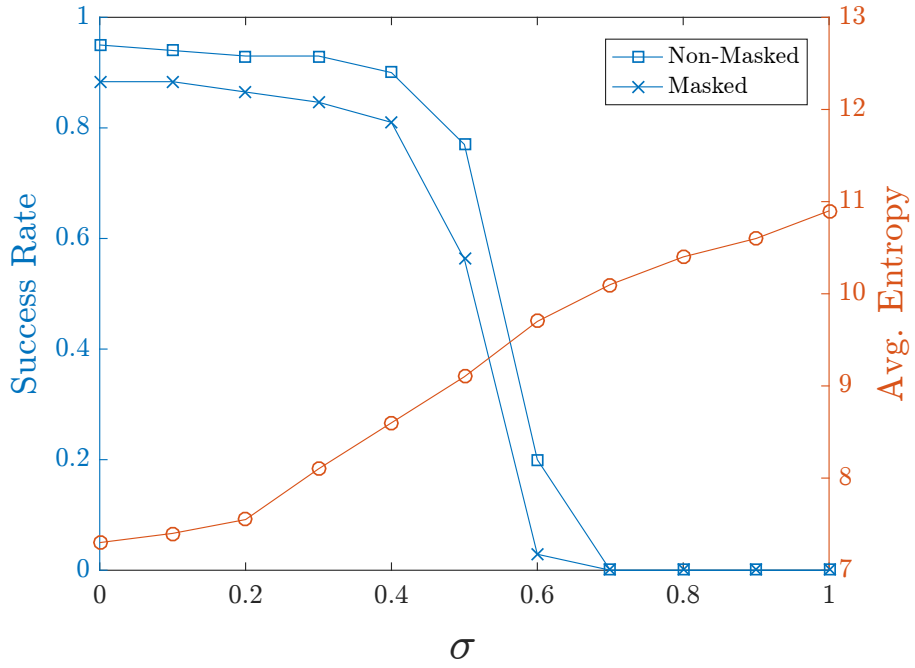


Figure 7.3: Success probability of key recovery for varying σ

Chapter 8

Conclusions

We now summarize the results of our attack, mention possible future work and give implementation recommendations that could prevent our attack.

Section 8.1 explains the implications of our attack on currently existing implementation proposals. In Section 8.2, we give suggestions that can significantly increase the required effort of performing our attack. Finally, we want to give an outlook on possible future work based on our attack in Section 8.3.

8.1 Implications of our Attack

We have shown the first single-trace side-channel attack that targets an RLWE-based encryption scheme. Since we only use leakage information from within the INTT operation, a similar variation of our attack may be applicable to many other lattice-based cryptographic schemes.

We have also considered a masked implementation, proposed by Reparaz et al. [69], that conceals input/output as well as all intermediate values of the INTT, in our attack evaluation. As shown in Chapter 7, we can break such a masked INTT implementation with high success rates, both for real and simulated leakage data. Since our attack can handle rather noisy side-channel information from a singe-trace template matching, we can calculate the shares of the intermediates, combine them, perform a key recovery, and obtain the unmasked private key.

Based on the ideas of Soft-Analytical Side-Channel Attacks (SASCA) (Section 4.3), we built a factor graph representation of the attacked INTT operation. We then found an efficient way of performing the BP algorithm to calculate marginal distributions of intermediate variables. We were not able to correctly determine all intermediate variables after the marginalization step. However, we presented an efficient key recovery algorithm that can, given enough correctly determined INTT intermediate values, recover the full private key.

Since we have performed an implementation attack, the theoretical security claims of RLWE-based encryption schemes remain unaffected. Yet, we have shown that practical implementations need to feature sufficient side-channel countermeasures. A masking of the INTT operation alone is not sufficient. Also, shuffling of coefficient-wise multiplications before/after the INTT operation, as proposed by Oder et al. [57] does not protect against our attack since our attack is focused solely on leakage information from within the INTT operation. In the next section, we present a few countermeasures that do have an impact

on our attack.

8.2 Suggested Countermeasures

Based on our attack results we now propose countermeasures, some of which should be considered in practical implementations of cryptographic schemes that incorporate an (I)NTT. While all countermeasures are fairly easy to implement they can significantly affect the performance/efficiency of the corresponding implementation.

Constant Time Operations

In our attack, the main targets during power analysis are the modular multiplication operations. While the multiplication instruction is executed in constant time, the reduction is implemented via trivial division using hardware dividers and does leak timing information. We use this timing information to perform an initial classification of the unknown intermediates into 1 out of 5 possible value ranges. Thereby, we increase the performance of the subsequent TA by reducing the value domain of each attacked intermediate. One obvious countermeasure is the usage of constant time modular multiplications as suggested by Oder et al. [57]. Even though this countermeasure does not prevent our attack in principle, it will increase the value domain of each intermediate. Thus, the noise in the prior probabilities in attack step 2.

Shuffling inside the INTT

As mentioned earlier, there exist implementation proposals for RLWE encryption that incorporate shuffling. Yet, to the best of our knowledge, nobody has proposed shuffling inside the INTT operation. The implementation of shuffling inside an INTT operation is straight forward since the butterflys in each layer are independent. A shuffled INTT would have severe consequences on the power analysis in attack step 1, since a reliable assignment of traces to corresponding butterflys is not possible anymore. However, the cost of shuffling is rather high, since random execution sequences should be generated for every layer.

Adaption of the RLWE Parameterization

The choice of the RLWE encryption parameterization can be adapted such that single-trace side-channel analysis becomes more difficult. The size of the modulus q has a large impact on our side-channel analysis performance, since the entropy of every INTT intermediate is upper-bounded by $\log_2(q)$. The usage of an RLWE parameterization that requires a larger q may therefore increase the noise in the leakage information up to a point where the BP algorithm cannot calculate marginal distributions reliably anymore.

One example of an RLWE parameterization that uses a comparably large q was proposed by Micciancio et al. [52] (Table 2.1). In one of their proposals they suggest the RLWE parameterization:

$$n = 214\ , \quad q = 16381\ , \quad \sigma = 1.17,$$

which would result in a security level of $\geq$ 128-bit and a 2996-bit public key (excluding the lattice). Thus, the maximum entropy per intermediate in our attack would increase from 12.9 to 14.0.

8.3 Future Work

In this section, we present a few ideas on possible future work based on our presented attack.

Exact Marginalization for General Graphs

One limitation of the BP algorithm, when applied on a cyclic factor graph, is that the obtained marginals are only approximations of the real marginal distribution of each variable node. Solely in case of an acyclic factor graphs the BP algorithm is guaranteed to deliver exact marginals. However, the so-called Junction Tree algorithm may be used here instead.

The Junction Tree algorithm is closely related to the BP algorithm, yet it allows us to calculate marginal distributions in general graphs. The algorithm works on a junction tree representation of a given function that eliminates cycles by clustering them into single nodes. The transformation of our factor graph into a junction tree as well as the application and evaluation of the junction tree algorithm is left for future work.

Evaluation for Different RLWE Parameterizations

Our attack evaluation is based on a hardware friendly RLWE parameterization from Göttert et al. [32]:

$$n = 256 \ , \quad q = 7681 \ , \quad \sigma = 1.80$$

However, authors have also proposed many more secure parameterizations as shown in Table 2.1. Since our attack can be scaled easily, a performance evaluation might be interesting for parameterizations with a particularly small modulus [52]:

$$n = 136 \ , \quad q = 2003 \ , \quad \sigma = 2.07$$

or comparably high lattice dimensions and thus large factor graphs [32]:

$$n = 512 \ , \quad q = 12289 \ , \quad \sigma = 1.93$$

We expect that the choice of the modulus q has the biggest impact on the performance of our attack. A large q implies a larger value domain for each intermediate and thus a larger entropy in the side-channel measurements. The number of required measurements in the template building phase also increases from ≈ 100 to e.g. ≈ 157 million for $q = 12289$. The runtime of our attack is most influenced by the lattice dimension n. By doubling the lattice dimension, compared to our RLWE parameterization, the number of intermediates in the NTT butterfly networks increases from 2304 (256×9) to 5120 (512×10). The BP algorithm needs to executed on factor graphs with more than twice the size. Still, we expect the runtime of our attack to be rather low. Another implication of doubling n is that the value domain of ω increases from 128 to 256. This results in an increased amount of required measurements in the template building phase of ≈ 314 million for $n = 512$ and $q = 12289$.

The choice of σ is only relevant for step 3 of our attack. However, we do not expect a slightly higher σ to have a significant impact on the performance of our key recovery algorithm.

Evaluation for Hardware Implementations

Our attack was performed on a microprocessor implementation of an efficient RLWE encryption scheme. However, there also exist efficient RLWE encryption implementations on FPGAs, such as proposed by Roy et al [71]. Their hardware implementations are heavily optimized, either for high throughput or small area. Hence, their implementation of the INTT operation is quite different to the implementation that was used in this thesis. An evaluation of our attack on such a hardware implementation with different types of leakage information is left for further work.

Appendix A

Definitions

A.1 Abbreviations

ACA	Algebraic Cryptanalysis
BP	Belief Propagation
AES	Advanced Encryption Standard
ASCA	Algebraic Side-Channel Attack
BKZ	Block Korkine Zolotarev Algorithm
CMOS	Complementary metal-oxide-semiconductor
CVP	Closest Vectors Problem
DPA	Differential Power Analysis
ECC	Elliptic Curve Cryptography
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
INTT	Inverse Number Theoretic Transform
LWE	Learning With Errors Problem
NTT	Number Theoretic Transform
RSA	Rivest-Shamir-Adleman Cryptosystem
RLWE	Ring-Learning With Errors Problem
SASCA	Soft Analytical Side-Channel Attack
SCA	Side-Channel Attack
SPA	Simple Power Analysis
SIS	Shortest Integer Solution Problem
SNR	Signal-to-Noise Ratio
SVP	Shortest Vectors Problem
TA	Template Attack

Bibliography

[1] Cmos, the ideal logic family. Online, 1983. https://www.fairchildsemi.com/application-notes/AN/AN-77.pdf.

[2] S. Agrawal, D. Boneh, and X. Boyen. Efficient lattice (H)IBE in the standard model. In *Proc. of Eurocrypt'10*, volume 6110 of *LNCS*, pages 553–572, 2010.

[3] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 99–108, New York, NY, USA, 1996. ACM.

[4] A. Akavia, S. Goldwasser, and V. Vaikuntanathan. *Simultaneous Hardcore Bits and Cryptography against Memory Attacks*, pages 474–495. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[5] E. Alkim, L. Ducas, T. Pppelmann, and P. Schwabe. Post-quantum key exchange - a new hope. Cryptology ePrint Archive, Report 2015/1092, 2015. http://eprint.iacr.org/2015/1092.

[6] R. J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 2001.

[7] B. Applebaum, D. Cash, C. Peikert, and A. Sahai. *Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems*, pages 595–618. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[8] G. V. Bard, N. T. Courtois, and C. Jefferson. Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over gf(2) via sat-solvers. Cryptology ePrint Archive, Report 2007/024, 2007. http://eprint.iacr.org/2007/024.

[9] D. J. Bernstein. Cache-timing attacks on aes. Technical report, 2005.

[10] D. J. Bernstein, T. Chou, and P. Schwabe. *McBits: Fast Constant-Time Code-Based Cryptography*, pages 250–272. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[11] M. Braithwaite. Experimenting with post-quantum cryptography. Google Security Blog, 2016. https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html.

[12] L. G. Bruinderink, A. Hlsing, T. Lange, and Y. Yarom. Flush, gauss, and reload – a cache attack on the bliss lattice-based signature scheme. Cryptology ePrint Archive, Report 2016/300, 2016. http://eprint.iacr.org/2016/300.

[13] D. Brumley and D. Boneh. Remote timing attacks are practical. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM'03, pages 1–1, Berkeley, CA, USA, 2003. USENIX Association.

[14] D. Cash, D. Hofheinz, E. Kiltz, and C. Peikert. *Bonsai Trees, or How to Delegate a Lattice Basis*, pages 523–552. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[15] S. Chari, J. R. Rao, and P. Rohatgi. *Template Attacks*, pages 13–28. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[16] S. Cheung. Proof of Hammersley-Clifford Theorem. Feb. 2008.

[17] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[18] N. T. Courtois and G. V. Bard. *Algebraic Cryptanalysis of the Data Encryption Standard*, pages 152–169. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[19] N. T. Courtois and J. Pieprzyk. *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations*, pages 267–287. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[20] S. A. Crosby, D. S. Wallach, and R. H. Riedi. Opportunities and limits of remote timing attacks. *ACM Trans. Inf. Syst. Secur.*, 12(3):17:1–17:29, Jan. 2009.

[21] S. G. Daniele Micciancio. *Complexity of Lattice Problems: a cryptographic perspective*, volume 671 of *The Kluwer International in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, Massachusetts, Mar. 2002.

[22] N. S. Dattani and N. Bryans. Quantum factorization of 56153 with only 4 qubits. *CoRR*, abs/1411.6758, 2014.

[23] R. de Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede. Efficient software implementation of ring-lwe encryption. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, pages 339–344, San Jose, CA, USA, 2015. EDA Consortium.

[24] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems. *A Practical Implementation of the Timing Attack*, pages 167–182. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[25] Y. Dodis, S. Goldwasser, Y. Tauman Kalai, C. Peikert, and V. Vaikuntanathan. *Public-Key Encryption Schemes with Auxiliary Inputs*, pages 361–381. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[26] L. Ducas and P. Q. Nguyen. *Faster Gaussian Lattice Sampling Using Lazy Floating-Point Arithmetic*, pages 415–432. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[27] S. D. Galbraith, C. Petit, B. Shani, and Y. B. Ti. On the security of supersingular isogeny cryptosystems. In *ASIACRYPT (1)*, pages 63–91. Springer, 2016.

[28] D. Genkin, A. Shamir, and E. Tromer. *RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis*, pages 444–461. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[29] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[30] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 197–206, New York, NY, USA, 2008. ACM.

[31] B. Gierlichs, K. Lemke-Rust, and C. Paar. Templates vs. stochastic methods. In L. Goubin and M. Matsui, editors, *CHES 2006, 8th International*, volume 4249 of *LNCS*, pages 15–29. Springer, 2006.

[32] N. Göttert, T. Feller, M. Schneider, J. Buchmann, and S. Huss. *On the Design of Hardware Building Blocks for Modern Lattice-Based Encryption Schemes*, pages 512–529. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[33] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, 1996. ACM.

[34] D. Gruss, C. Maurice, and K. Wagner. Flush+flush: A stealthier last-level cache attack. *CoRR*, abs/1511.04594, 2015.

[35] T. Güneysu, V. Lyubashevsky, and T. Pöppelmann. *Practical Lattice-Based Cryptography: A Signature Scheme for Embedded Systems*, pages 530–547. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[36] J. Hoffstein, J. Pipher, and J. H. Silverman. *NTRU: A ring-based public key cryptosystem*, pages 267–288. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[37] J. Howe, A. Khalid, C. Rafferty, F. Regazzoni, and M. O'Neill. On practical discrete gaussian samplers for lattice-based cryptography. *IEEE Transactions on Computers*, PP(99):1–1, 2016.

[38] H. B. Keller and J. R. Swenson. Experiments on the lattice problem of gauss. *Mathematics of Computation*, 17(83):223–230, 1963.

[39] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, pages 104–113, 1996.

[40] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 388–397, 1999.

[41] O. Kömmerling and M. G. Kuhn. Design principles for tamper-resistant smartcard processors. In *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, WOST'99, pages 2–2, Berkeley, CA, USA, 1999. USENIX Association.

[42] J. L. Lagrange. Recherches darithmetique. nouveaux memoires de lacademie de berlin. 1773.

[43] L. A. L. L. Lenstra, H.W. jr. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[44] T. Lepoint and M. Naehrig. *A Comparison of the Homomorphic Encryption Schemes FV and YASHE*, pages 318–335. Springer International Publishing, Cham, 2014.

[45] R. Lindner and C. Peikert. Better key sizes (and attacks) for lwe-based encryption. In *Proceedings of the 11th International Conference on Topics in Cryptology: CT-RSA 2011*, CT-RSA'11, pages 319–339, Berlin, Heidelberg, 2011. Springer-Verlag.

[46] M. Liu and P. Q. Nguyen. *Solving BDD by Enumeration: An Update*, pages 293–309. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[47] J. Longo, E. D. Mulder, D. Page, and M. Tunstall. Soc it to em: electromagnetic side-channel attacks on a complex system-on-chip. Cryptology ePrint Archive, Report 2015/561, 2015. `http://eprint.iacr.org/2015/561`.

[48] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. *SWIFFT: A Modest Proposal for FFT Hashing*, pages 54–72. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[49] V. Lyubashevsky, C. Peikert, and O. Regev. *On Ideal Lattices and Learning with Errors over Rings*, pages 1–23. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[50] D. J. C. MacKay. *Information theory, inference, and learning algorithms*. Cambridge University Press, 2003.

[51] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[52] D. Micciancio and O. Regev. Lattice-based cryptography, 2008.

[53] J. M. Mooij and H. J. Kappen. Sufficient Conditions for Convergence of the SumProduct Algorithm. *IEEE Transactions on Information Theory*, 53(12):4422–4437, Dec. 2007.

[54] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. *CoRR*, abs/1301.6725, 2013.

[55] P. Q. Nguyen and D. Stehle. Low-dimensional lattice basis reduction revisited. *ACM Trans. Algorithms*, 5(4):46:1–46:48, Nov. 2009.

[56] NSA/IAD. CNSA Suite and Quantum Computing FAQ, January 2016. `https://www.iad.gov/iad/library/ia-guidance/ia-solutions-for-classified/algorithm-guidance/cnsa-suite-and-quantum-computing-faq.cfm`.

[57] T. Oder, T. Schneider, T. Pöppelmann, and T. Güneysu. Practical cca2-secure and masked ring-lwe implementation. *IACR Cryptology ePrint Archive*, 2016:1109, 2016.

[58] P. J. J. O'Malley, R. Babbush, I. D. Kivlichan, J. Romero, J. R. McClean, R. Barends, J. Kelly, P. Roushan, A. Tranter, N. Ding, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, A. G. Fowler, E. Jeffrey, E. Lucero, A. Megrant, J. Y. Mutus, M. Neeley, C. Neill, C. Quintana, D. Sank, A. Vainsencher, J. Wenner, T. C. White, P. V. Coveney, P. J. Love, H. Neven, A. Aspuru-Guzik, and J. M. Martinis. Scalable quantum simulation of molecular energies. *Phys. Rev. X*, 6:031007, Jul 2016.

[59] J. Pearl. Reverend Bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the American Association of Artificial Intelligence National Conference on AI*, pages 133–136, Pittsburgh, PA, 1982.

[60] C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem: Extended abstract. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 333–342, New York, NY, USA, 2009. ACM.

[61] C. Peikert. *An Efficient and Parallel Gaussian Sampler for Lattices*, pages 80–97. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[62] C. Peikert, V. Vaikuntanathan, and B. Waters. *A Framework for Efficient and Composable Oblivious Transfer*, pages 554–571. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[63] P. Pessl. Analyzing the shuffling side-channel countermeasure for lattice-based signatures. Cryptology ePrint Archive, Report 2017/033, 2017. `http://eprint.iacr.org/2017/033`.

[64] W. Pfaff, B. Hensen, H. Bernien, S. B. van Dam, M. S. Blok, T. H. Taminiau, M. J. Tiggelman, R. N. Schouten, M. Markham, D. J. Twitchen, and R. Hanson. Unconditional quantum teleportation between distant solid-state quantum bits. *Science*, 2014.

[65] T. Pöppelmann and T. Gneysu. Area optimization of lightweight lattice-based encryption on reconfigurable hardware. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2796–2799, June 2014.

[66] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 84–93, New York, NY, USA, 2005. ACM.

[67] M. Renauld and F.-X. Standaert. *Algebraic Side-Channel Attacks*, pages 393–410. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[68] O. Reparaz, S. S. Roy, R. de Clercq, F. Vercauteren, and I. Verbauwhede. Masking ring-lwe. *Journal of Cryptographic Engineering*, 6(2):139–153, 2016.

[69] O. Reparaz, S. Sinha Roy, F. Vercauteren, and I. Verbauwhede. *A Masked Ring-LWE Implementation*, pages 683–702. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[70] S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede. Compact and side channel secure discrete gaussian sampling. Cryptology ePrint Archive, Report 2014/591, 2014. `http://eprint.iacr.org/2014/591`.

[71] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede. *Compact Ring-LWE Cryptoprocessor*, pages 371–391. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[72] M.-J. O. Saarinen. Gaussian sampling precision in lattice cryptography. Cryptology ePrint Archive, Report 2015/953, 2015. `http://eprint.iacr.org/2015/953`.

[73] P. W. Shor. Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Sci. Statist. Comput.*, 26:1484, 1997.

[74] V. Shoup. NTL: A library for doing number theory. `http://www.shoup.net/ntl`, 2003.

[75] S. Sinha Roy, F. Vercauteren, and I. Verbauwhede. *High Precision Discrete Gaussian Sampling on FPGAs*, pages 383–401. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[76] F.-X. Standaert, T. G. Malkin, and M. Yung. *A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks*, pages 443–461. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[77] N. Veyrat-Charvillon, B. Gérard, and F.-X. Standaert. *Soft Analytical Side-Channel Attacks*, pages 282–296. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[78] J. Wang and J. Bi. Lattice-based identity-based broadcast encryption scheme. *IACR Cryptology ePrint Archive*, 2010:288, 2010.