Cryptanalysis and Secure Implementation of Modern Cryptographic Algorithms

Abdel Alim K. Farag

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy (Electrical and Computer Engineering) at
Concordia University
Montreal, Quebec, Canada

July 2012

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: **Abdel Alim K. Farag**

Entitled: **Cryptanalysis and Secure Implementation of Modern Cryptographic Algorithms**

and submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY (Electrical & Computer Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Abstract

Cryptanalysis and Secure Implementation of Modern Cryptographic Algorithms

Abdel Alim K. Farag, PhD

Concordia University, 2012

Cryptanalytic attacks can be divided into two classes: pure mathematical attacks and Side Channel Attacks (SCAs). Pure mathematical attacks are traditional cryptanalytic techniques that rely on known or chosen input-output pairs of the cryptographic function and exploit the inner structure of the cipher to reveal the secret key information. On the other hand, in SCAs, it is assumed that attackers have some access to the cryptographic device and can gain some information from its physical implementation.

Cold-boot attack is a SCA which exploits the data remanence property of Random Access Memory (RAM) to retrieve its content which remains readable shortly after its power has been removed. Fault analysis is another example of SCAs in which the attacker is assumed to be able to induce faults in the cryptographic device and observe the faulty output. Then, by careful inspection of faulty outputs, the attacker recovers the secret information, such as secret inner state or secret key. Scan-based Design-For-Test (DFT) is a widely deployed technique for testing hardware chips. Scan-based SCAs exploit the information obtained by analyzing the scanned data in order to retrieve secret information from cryptographic hardware devices that are designed with this testability feature.

In the first part of this work, we investigate the use of an off-the-shelf SAT solver, CryptoMinSat, to improve the key recovery of the Advance Encryption Standard (AES-128) key schedules from its corresponding decayed memory images which can be obtained using cold-boot attacks.

We also present a fault analysis on both NTRUEncrypt and NTRUSign cryptosystems. For this specific original instantiation of the NTRU encryption system with parameters (N, p, q), our attack succeeds with probability $\approx 1 - \frac{1}{p}$ and when the number of faulted coefficients is upper bounded by t, it requires $O((pN)^t)$ polynomial inversions in $\mathbb{Z}/p\mathbb{Z}[x]/(x^N - 1)$. We also investigate several techniques to strengthen hardware implementations of NTRUEncrypt against this class of attacks. For NTRUSign with parameters (N, $q = p^l$, $\mathcal{B}$, *standard*, $\mathcal{N}$), when the attacker is able to skip the norm-bound signature checking step, our attack needs one fault to succeed with probability $\approx 1 - \frac{1}{p}$ and requires $O((qN)^t)$ steps when the number of faulted polynomial coefficients is upper bounded by t. The attack is also applicable to NTRUSign utilizing the *transpose* NTRU lattice but it requires double the number of fault injections. Different countermeasures against the proposed attack are also investigated.

Furthermore, we present a scan-based SCA on NTRUEncrypt hardware implementations that employ scan-based DFT techniques. Our attack determines the scan chain structure of the polynomial multiplication circuits used in the decryption algorithm which allows the cryptanalyst to efficiently retrieve the secret key.

Several key agreement schemes based on matrices were recently proposed. For example, Álvarez *et al.* proposed a scheme in which the secret key is obtained by multiplying powers of block upper triangular matrices whose elements are defined over $\mathbb{Z}_p$. Climent *et al.* identified the elements of the endomorphisms ring $End(\mathbb{Z}_p \times \mathbb{Z}_{p^2})$ with elements in a set, E_p, of matrices of size 2×2, whose elements in the first row belong to $\mathbb{Z}_p$ and the elements in the second row belong to $\mathbb{Z}_{p^2}$. Keith Salvin presented a key exchange protocol using matrices in the general linear group, $GL(r, \mathbb{Z}_n)$, where n is the product of two distinct large primes. The system is fully specified in the US patent number 7346162 issued in 2008. In the second part of this work, we present mathematical cryptanalytic attacks against these three schemes and show that they can be easily broken for all practical choices of their security parameters.

Acknowledgments

I would like to thank my supervisor, Dr. Amr Youssef, for his patient guidance, and encouragement. I have been extremely lucky to have a supervisor who cared so much about me and my work, and who responded to my questions and queries so promptly.

I also would like to express my deepest gratitude for the constant support, understanding and love that I received from my mother, my father, my wife, and my family during the past years.

Completing this work would have been all the more difficult were it not for the support and friendship provided by my colleagues in the Crypto Lab at CIISE. I am indebted to them for their help. I also would like to thank Tadeusz Obuchowicz for his help and support in the area of hardware implementations using VHDL.

Finally, I am very grateful for the financial support of the Mission Sector in the Egyptian Ministry of Higher Education.

ABDEL ALIM K. FARAG

To my family for their love and support

Table of Contents

List of Figures

List of Tables

List of Acronyms

AES	Advanced Encryption Standard
ANF	Algebraic Normal Form
BDD	Binary Decision Diagrams
BP	Bit Packing
CBA	Constant Based Attack
CNF	Conjunctive Normal Form
CSP	Conjugacy Search Problem
CPU	Central Processing Unit
CRT	Chinese Remainder Theorem
CUDA	Compute Unified Device Architecture
DEMA	Differential Electromagnetic Analysis
DES	Data Encryption Standard
DFT	Design-For-Test
DLP	Discrete Logarithm Problem
DPA	Differential Power Analysis
DRAM	Dynamic Random Access Memory
DSA	Digital Signature Algorithm
DSCA	Differential Side Channel Attacks
EDC	Error Detecting Code
EESS	Efficient Embedded Security Standards

EMA	Electromagnetic Analysis
FHDA	Fixed Hamming Distance based Attack
FPGA	Field Programmable Gate Array
FSMD	Finite State Machine Data path
GCHQ	Government Communications Headquarters
GF	Galois Field
GGH	Goldreich-Goldwasser-Halevi cryptosystem
GL	General Linear Group
GM	Global Memory
GPU	Graphics Processing Unit
ISE	Integrated Software Environment
JTAG	Joint Test Action Group
LFSR	Linear Feedback Shift Register
LUT	Look Up Table
MI5	Military Intelligence, Section 5
MUX	Multiplexer
NC	Normal Coefficients Representation
NIST	National Institute of Standards & Technology
NP	Normal Polynomial Form
NSA	National Security Agency
NTRUEncrypt	NTRU Encryption Algorithm
NTRUSign	NTRU Digital Signature Scheme
PF	Product Form
RAM	Random Access Memory
RBT	Redundancy Based Technique
RFID	Radio Frequency Identification
ROM	Read Only Memory

RSA	Rivest-Shamir-Adleman cryptosystem
SAT	Satisfiability Problem
SCAs	Side Channel Attacks
SEMA	Simple Electromagnetic Analysis
SH	Shared Memory
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction Multiple Thread
SM	Streaming Multiprocessor
SP	Streaming Processor
SPA	Simple Power Analysis
SRAM	Static Random Access Memory
SSCA	Simple Side Channel Attacks
TCK	Test Clock
TDI	Test Data In
TDO	Test Data Out
TMS	Test Mode Select
TRST	Test Reset
VHSIC	Very High Speed Integrated Circuits
VHDL	VHSIC Hardware Description Language

Chapter 1

Introduction

1.1 Motivation

In the academic cryptanalysis literatures, a “break” of a cryptosystem is defined quite conservatively: if the effort required by the attacker is less than the effort required by naïve exhaustive search, then the cryptosystem is said to be broken. This conservative definition, while can be arguably justified, led to many published mathematical attacks that require impractical amounts of time, memory, or known/chosen plaintext-ciphertext pairs. On the other hand, in many practical scenarios, the attackers may have access to the cryptographic device. In such scenarios, SCAs allow the attackers to break these cryptosystems using a relatively small amount of computational resources and a small number of known/chosen plaintext-ciphertext pairs.

The basic idea behind SCAs is not new. For example, Wright [190] reported an early application of acoustic cryptanalysis. During his employment with GCHQ in 1965, the British intelligence agency (MI5) tried to break a cipher used by the Egyptian embassy in London. In order to reduce the computational resources required to break the cipher, Wright suggested placing a spy microphone near the rotor-cipher machine used by the Egyptians to detect the click-sound produced by the machine. By listening to the clicks of the rotors as cipher clerks reset them each morning, MI5 successfully deduced the core position of some of the machines rotors which reduced the computation effort needed to break the cipher, and MI5 was able to spy on the embassys communication for years. Another example is given in a recently declassified National Security Agency (NSA) document [172] which reveals that in 1943, an engineer with Bell Telephone observed decipherable spikes on an oscilloscope associated with the decrypted output of a certain encrypting teletype.

Currently, the wide spread of unprotected software or hardware cryptographic implementations offers various possibilities for side channel attacks. Such attacks are practical; they do not even require expensive equipments. For example, Skorobogatov and Anderson [10] carried out fault analysis attacks using a flashgun bought second-hand from a camera store for $30 and with an $8 laser pointer. They also developed relatively simple techniques to set or reset any individual bit of SRAM in a microcontroller. Thus, unless suitable effective countermeasures are taken, vulnerability to this class of attacks poses a big problem for the industry, especially with the wide spread of smart cards, RFIDs, and embedded devices.

In the first part of this thesis, we investigate the application of several SCAs against both AES [59] and NTRU [84]. In particular, we study the use of an off-the-shelf SAT solver, CryptoMinSat, to improve the key recovery of the AES-128 key schedules from its corresponding decayed memory images resulting from applying the cold-boot attack. We present a fault analysis on NTRUEncrypt and NTRUSign and propose several hardware countermeasures. Proof of concept FPGA implementations for NTRUEncrypt are also presented. Furthermore, we present a scan-based SCA on NTRUEncrypt hardware implementations that employ scan-based DFT techniques.

Public-key cryptography [124] provides key exchange mechanisms in which secret keys can be exchanged between users over insecure communication channels. These key exchange mechanisms are usually based on number theory problems such as the discrete logarithm problem (DLP) [57], integer factorization [158] and elliptic curve DLP [26]. However, such systems require a large number of arithmetic operations, which makes them hard to implement in most resource constrained applications. To overcome this problem, key exchange protocols based on efficient matrix algebra have been proposed. For example, Álvarez *et al.* [5] proposed a scheme in which the secret key is obtained by multiplying powers of block upper triangular matrices whose elements are defined over $\mathbb{Z}_p$. Climent *et al.* [40] identified the elements of the endomorphisms ring $End(\mathbb{Z}_p \times \mathbb{Z}_{p^2})$ with elements in a set, E_p, of matrices of size 2×2, whose elements in the first row belong to $\mathbb{Z}_p$ and elements in the second row belong to $\mathbb{Z}_{p^2}$. Keith Salvin [175] presented a key exchange protocol using matrices in the general linear group, $GL(r, \mathbb{Z}_n)$, where n is the product of two distinct large primes. The system is fully specified in the US patent number 7346162 issued in 2008. In the second part of this work, we present mathematical cryptanalytic attacks against these three schemes and show that they can be efficiently broken for all practical choices of their security parameters.

1.2 Thesis contributions

In this thesis, we investigate the application of several side channel attacks on different ciphers. We also propose some countermeasures against these types of attacks. Some implementations of theses countermeasures are presented. We also present cryptanalysis of some recently proposed key exchange schemes. The contribution of this thesis can be summarized as follows:

1. We investigate the use of an off-the-shelf SAT solver, CryptoMinSat, to improve the key recovery of the AES-128 key schedules from its corresponding decayed memory images. By exploiting the asymmetric decay of the memory images and the redundancy of key material inherent in the AES key schedule, rectifying the faults in the corrupted memory images of the AES-128 key schedule is formulated as a Boolean satisfiability problem which can be solved efficiently for relatively large decay factors. Our experimental results show that this approach improves upon the previously known results.

2. We present a fault analysis attack on NTRUEncrypt. The fault model in which we analyze

the cipher is the one in which the attacker is assumed to be able to fault a small number of coefficients of the polynomial input to (or output from) the second step of the decryption process but cannot control the exact location of injected faults. For this specific original instantiation of the NTRU encryption system with parameters (N, p, q), our attack succeeds with probability $\approx 1 - \frac{1}{p}$ and when the number of faulted coefficients is upper bounded by t, it requires $O((pN)^t)$ polynomial inversions in $\mathbb{Z}/p\mathbb{Z}[x]/(x^N - 1)$. We also investigate several techniques to strengthen hardware implementations of NTRUEncrypt against this class of attacks. In particular, by utilizing the algebraic structure of the cipher, we propose several countermeasures based on error detection checksum codes, and spatial/temporal redundancies. The error detection capabilities of these countermeasures, as well as their impact on the decryption throughput and area, are also presented.

3. We present a fault analysis attack on NTRUSign. The utilized fault model is the one in which the attacker is assumed to be able to fault a small number of coefficients in a specific polynomial during the signing process but cannot control the exact location of the injected transient faults. For NTRUsign with parameters (N, $q = p^l$, $\mathcal{B}$, *standard*, $\mathcal{N}$), when the attacker is able to skip the norm-bound signature checking step, our attack needs one fault, succeeds with probability $\approx 1 - \frac{1}{p}$ and requires $O((qN)^t)$ computation steps when the number of faulted polynomial coefficients is upper bounded by t. The attack is also applicable to NTRUSign utilizing the *transpose* NTRU lattice but it requires double the number of fault injections. Different countermeasures against the proposed attack are investigated.

4. We present a scan-based SCA on NTRUEncrypt hardware implementations that employ scan-based Design-For-Test (DFT) techniques. Our attack determines the scan chain structure of the polynomial multiplication circuits used in the decryption algorithm which allows the cryptanalyst to efficiently retrieve the secret key.

5. We show that breaking the key exchange scheme proposed by Álvarez *et al.* [8] with

security parameters (r, s, p) is equivalent to solving a set of $3(r + s)^2$ linear equations with $2(r + s)^2$ unknowns in $\mathbb{Z}_p$, which renders this system insecure for all the suggested practical choices of the security parameters.

6. Climent *et al.* [40] identified the elements of the endomorphisms ring $End(\mathbb{Z}_p \times \mathbb{Z}_{p^2})$ with elements in a set, E_p, of matrices of size 2×2, whose elements in the first row belong to $\mathbb{Z}_p$ and the elements in the second row belong to $\mathbb{Z}_{p^2}$. By taking advantage of matrix arithmetic, they proposed a key exchange protocol using polynomial functions over E_p defined by polynomials in $\mathbb{Z}[X]$. We show that this key exchange protocol is insecure; it can be broken by solving a set of 10 consistent homogeneous linear equations in 8 unknowns over $\mathbb{Z}_{p^2}$.

7. Keith Salvin [175] presented a key exchange protocol using matrices in the general linear group, $GL(r, \mathbb{Z}_n)$, where n is the product of two distinct large primes. The system is fully specified in the US patent number 7346162 issued in 2008. In the patent claims, it is argued that the best way to break this system is to factor n. Furthermore, for efficiency reasons, it is suggested to use $r = 2$. We show that this cryptosystem can be easily broken by solving a set of consistent homogeneous r^2 linear equations in $2r$ unknowns over $\mathbb{Z}_n$.

The contributions made throughout this work have been published in [99–109].

1.3 Outline of the thesis

The rest of the thesis is organized as follows. In chapter 2, we provide a brief overview of side channel attacks. A classification of these attacks, examples, and proposed countermeasures are also presented. We also briefly review the Boolean satisfiability (SAT) problem. In chapter 3, we show how to use an off-the-shelf SAT solver, CryptoMinSat, to improve the key recovery of the AES-128 key schedules from its corresponding decayed memory images. Chapter 4 presents our fault analysis attacks against NTRUEncrypt and NTRUSign cryptosystems.

Different countermeasures are also presented. In chapter 5, we describe our proposed scan-based side channel attack on the NTRUEncrypt cryptosystem. The simulation results and their analysis are also provided. Chapter 6 provides cryptanalysis of the three key exchange schemes proposed by Álvarez *et al.* [8], Climent *et al.* [40], and Keith Salvin [175]. Our conclusions and some suggestions for future research directions are given in chapter 7. An architecture that offers different area-speed trade-off for NTRUEncrypt is presented in Appendix A together with a proof of concept FPGA implementation results. Results on implementing NTRUEncrypt on the NVIDIA GTX275 GPU, using the CUDA framework are presented in Appendix B.

Chapter 2

Background and Literature Review

2.1 Introduction

The classical focus of cryptography has been communication security. Consequently, traditional cryptanalytic techniques focused on information flowing over the communication channel rather than the endpoint cryptographic devices. This view has changed dramatically with the introduction of SCAs in the open literatures. Cryptography is now widely deployed in many devices ranging from pay TV units, cell phones, prepaid cards and smart cards. Because these cryptographic devices are easily obtainable, attackers can study the internal structure of the hardware to learn specific details about their implementations. Knowledge of the implementation may then be used to carry out attacks on the system without directly attacking the mathematics of the algorithms. In other words, even when totally secure algorithms and protocols are employed, attackers might still be able to learn valuable secret information due to the specific software or hardware implementation of the system. Figure 2.1 shows a modern view of the cryptographic model including different side channel information that can be obtained from the endpoint devices.

In this chapter we provide a brief overview of side channel attacks. A classification of SCAs is given in the next section. Examples of these attacks and proposed countermeasures are presented in section 2.3 and 2.4, respectively. Finally, in section 2.5 we briefly review the Boolean satisfiability (SAT) problem.

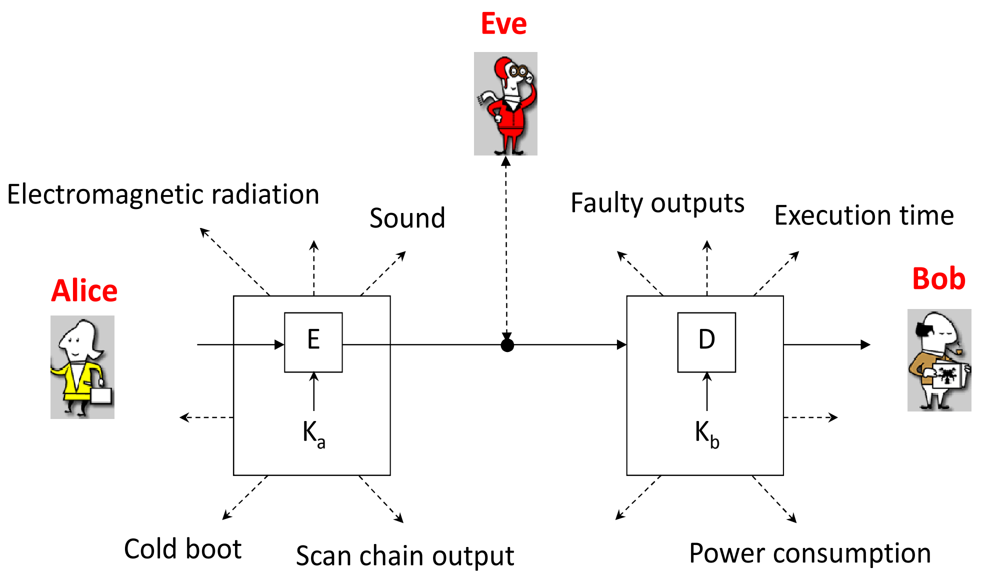


Figure 2.1: The cryptographic model including examples for side-channels

2.2 Classifications of side channel attacks

SCAs can be classified according to (i) the attackers capabilities to control the computation process, (ii) the way of accessing the cryptographic module and (iii) the method used in the analysis process [196].

2.2.1 Controls over the computation process

SCAs can be divided based on the control over the computation process by attackers into two main categories: passive attacks and active attacks. Passive attacks refer to those that do not interfere with the operation of the target system, i.e., the attacker is assumed to be able to collect some information about the operation of the target system without disturbing its behavior. In contrast, in active attacks, the adversary can affect the behavior of the target system, while the attacked system may or may not be able to detect such influence.

2.2.2 Ways of accessing the module

Anderson *et al.* [10] classified side channel attacks into three classes: invasive attacks, semi-invasive attacks and non-invasive attacks.

A. **Invasive attacks**

Invasive attacks require a direct physical access to the internal components of the cryptographic modules. A classic example of this class of attacks is when attackers may get access to the inner layer of the cryptographic module and place a probing needle on the data bus to record and later analyze the data transferred. Several defensive measures are usually implemented in hardware to counter such invasive attacks effectively. For example, some cryptographic modules with higher security level reset all their memories when tampering is detected [60].

B. **Semi-invasive attacks**

The notion of semi-invasive attack was first introduced by Skorobogatov and Anderson [174]. This type of attacks involves access to the device, but without physical access or damage to the layer that contains the cryptographic module. For example, in fault-induced attacks, the attacker may use a laser beam to ionize the device in order to change some of its memory contents and consequently change the output of the device.

C. **Non-invasive attacks**

Non-invasive attacks require close examination or manipulation of the device's behavior. This class of attacks exploits information that is unintentionally leaked. A classical example of such attacks is timing analysis which measures the time consumed by a device to perform an operation and analyze it to deduce the secret keys. One of the main characteristics of a non-invasive attack is that it is completely undetectable. For example, a tamper resistant smart card cannot figure out that its running time is currently being measured. Furthermore, contrary to invasive attacks that require individual processing of each attacked device, non-invasive attacks are often low-cost to deploy on a large scale. Therefore, non-invasive attacks present more threat for embedded devices and smart cards.

2.2.3 Methods used in the analysis process

Based on the methods used in the analysis process of the sampled data, SCA attacks are divided into Simple Side Channel Attack (SSCA) and Differential Side Channel Attack (DSCA). SSCA exploits the side channel output primarily depending on the performed operations. Typically, a single trace is used in an SSCA analysis, and therefore the secret key can be directly read from the side channel trace. If the SSCA is not successful due to noise in the measurements, DSCA can be used. DSCA exploits the correlation between the processed data and the side channel output. Since this correlation is often very small, statistical methods are used to exploit it efficiently.

2.3 Examples of side channel attacks

Side channel attacks exploit the information leaked by the physical characteristics of the cryptographic modules during execution of the algorithm. This extra information can be extracted from timing, power consumption or electromagnetic radiation characteristics. Other forms of side channel information can also be available as a result of hardware or software failures, changes in frequency or temperature, and computational errors.

2.3.1 Timing attack

The majority of optimized implementations of cryptographic algorithms execute the computations in a non-constant time. If these operations involve secret parameters, these timing variations can leak some information that can provide enough knowledge of the implementation. A careful statistical analysis sometimes could even lead to the total recovery of the secret parameters. Timing attacks were first introduced in 1996 by Kocher [115] who demonstrated the feasibility of these attacks against the RSA system. Later, Schindler [160] presented timing attacks on implementations of RSA exponentiation that employ the Chinese Remainder Theorem (CRT). Afterwards, several experimental results on various cryptographic algorithms have been reported (e.g., [52, 78]).

In order to illustrate the idea of the timing attack, consider an RSA cryptosystem where the encryption operation involves the computation of $R = y^x \bmod n$ where n is the public modulus, and y is the message being encrypted. The goal of the attacker is to find the secret key x.

Algorithm 1 shows the modular exponentiation algorithm which is typically used to computes $R = y^x \bmod n$, where x is w bits long. Kocher [115] shows that an attacker who can record the message received and the time taken to respond to each y, will be able to predict the operations of the algorithm. In particular, the timing attack described by Kocher allows someone who knows the exponent bits $0 \cdots (b-1)$ to find bit b. This is because when the first b bits are known, the attacker can compute the first b iterations of the for loop to find the value of s_b. In the next iteration, the first unknown bit will be tested. If it is set, $R_b = (s_b \cdot y) \bmod n$ will be computed, else this operation will be skipped. If the total modular exponentiation time for the iteration is ever fast when $R_b = (s_b \cdot y) \bmod n$ is slow then bit b must be zero. Conversely, if the modular exponentiation time is slow, then the bit must be set. The same set of timing measurement can be used to determine the rest of the exponentiation bits.

Algorithm 1 Modular Exponentiation Algorithm [115]

```
1: Let $s_0 = 1$.
2: **for** $k = 0$ to $w - 1$ **do**
3:     **if** (bit $k$ of $x$ is 1) **then**
4:         Let $R_k = (s_k \cdot y) \bmod n$
5:     **else**
6:         Let $R_k = s_k$
7:     **end if**
8:     Let $s_{k+1} = R_k^2 \bmod n$
9: **end for**
10: Return $(R_{w-1})$
```

A special class of timing attacks is cache based attacks which involve monitoring the movement of data into and out of the cache. Cache profiles can be used to recover the secret key information of a cryptographic algorithm. Such attacks usually consist of a collection phase that provides the attacker with profiles of execution, and an analysis phase which recovers the secret information. Cache based attacks can be categorized as trace driven attacks [152] or time driven attacks [185, 186]. Trace driven attacks rely on the ability of the attacker to capture a profile of cache activity that results from running the algorithm. That is, in order to perform a successful attack, the adversary needs a cache trace which shows cache hits or misses for every memory access. Time driven attacks rely on the fact that the execution time is mainly affected by memory accesses since cache hits result in a lower execution time and cache misses result in a comparably higher execution time.

2.3.2 Fault attacks

If the attacker is able to induce a fault or error during the operation of the cryptographic module, then the faulty behavior may provide the attacker with valuable side channel information that can greatly increase the cipher's vulnerability to cryptanalysis. Fault cryptanalysis presents practical and effective attacks against cryptographic hardware devices such as smart cards. The threat posed by these attacks was first demonstrated in the open literatures by Boneh *et al.* [29, 30] who showed that it is easy to extract the private RSA decryption/signing keys by inducing errors in the CRT implementation and observing the faulty output. Since then, many cryptographic algorithms have been broken by using such types of attacks. In [25], Biham and Shamir presented fault analysis attacks on the DES symmetric-key encryption scheme. Shamir *et al.* [79] developed general fault analysis techniques which can be used to attack the standard constructions of stream ciphers based on LFSRs.

A special type of fault analysis can be performed when the adversary is able to manipulate the program counter to skip instructions of an algorithm. In these fault models, we assume that the fault injection allows the attacker to skip the targeted instruction with a specific probability, i.e., we assume that the fault injection may also cause other instructions to be skipped or other variables to be changed randomly. Thus, an adversary must be able to check, whether the fault injection was successful or not. In practice, these fault attacks can be launched by introducing power glitch or spikes or by clocking the module with a clock rate outside the allowed range. Various experimental results confirming the above ideas were verified (e.g., [161]).

2.3.3 Power analysis attacks

In addition to the computation time and the faulty behavior, the power consumption of a cryptographic device can also leak useful information about the running operations and their involved secret parameters. Unlike the timing and the fault analysis, power analysis attacks are applicable only to hardware implementations. These attacks proved to be very effective in attacking smart cards and other embedded systems. Generally, power analysis attacks can be categorized into Simple and Differential Power Analysis (referred to as SPA and DPA, respectively). In SPA attacks, the goal is utilize the measured power traces in order to guess which particular instruction is being carried out at a specific time as well as the input and output values of this instruction. For that, the adversary needs to know the exact structure of the implementation in order to apply such an attack. In contrast, DPA attacks do not require knowledge of the implementation details and instead exploit statistical methods in the analysis process. In general, DPA is one of the most powerful SCAs which can be applied using relatively little resources [116, 150]. Experimental results with power analysis attacks on smart cards were reported in [3, 62, 126].

2.3.4 Electromagnetic attacks

Similar to any electrical device, the components of cryptographic devices usually generate electromagnetic radiations as part of their execution processes. By analyzing these emanations, the attacker can deduce the relationship between it and the underlying computation and data. This may allow the attacker to extract some useful information about the secret parameters of the cryptographic algorithm. Electromagnetic Analysis (EMA) can also be divided into two main classes: simple Electromagnetic Analysis (SEMA) and Differential Electromagnetic Analysis (DEMA). Experimental results on electromagnetic analysis attacks on cryptographic devices such as smart cards and comparisons to power analysis attacks were presented in [61, 156].

2.3.5 Cold-boot attacks

A cold-boot attack [72] is a SCA that exploits the fact that data loss of a non-powered random access memory can be retarded by cooling it down. In 2002, Skorobogatov [173] performed experiments to study the temperature dependency of data retention time in static RAM devices. The reported experimental results indicate that many chips may preserve data for relatively long periods of time at temperatures above $-20°$C which contradicted the common wisdom that was widely believed at that time. The temperature at which 80% of the data remained for one minute varied widely between devices. While some devices required cooling to at least $-50°$C, others, surprisingly, retained data for this period at room temperature. Memory retention time also varied between devices of the same type from the same manufacturer, most likely, because controlling data retention time is not a part of the chip manufacturing quality process [173].

Thus, one way to launch a cold-boot attack is to remove the memory module, after cooling it, from the target system and immediately plugging it in another system under the adversary's control. This system is then booted to access the memory. Another possible approach to execute the attack is to cold-boot the target machine by cycling its power OFF and then ON without letting it shut down properly. Then a lightweight operating system is instantly booted where the content of targeted memory is dumped to a file. Further analysis can then be performed against the information that is retrieved from memory in order to find sensitive information such as cryptographic keys or passwords.

2.3.6 Scan-based attacks

Scan-based Design-for-Test [34] is a widely deployed technique for testing hardware chips. Using this approach, all flip-flops in the design under test are connected to a scan chain where their states can be scanned out during the testing phase. Scan-based SCAs exploit the information obtained by analyzing the scanned data in order to retrieve secret information from cryptographic hardware devices that are designed with this testability feature.

While scan-based DFT improves the quality of testing, it also opens a powerful side channel against hardware implementations of cryptographic devices that utilize this technique. Despite the fact that the internal structure of the scan chain is usually not known to attackers, exploiting the information obtained from analyzing the scanned data allows cryptanalysts to ascertain this structure and retrieve the secret key information from cryptographic hardware devices implementing various cryptographic algorithms

2.3.7 Acoustic attacks

Most research on side channel attack focused on timing, power consumption, and fault features. However, one of the oldest eavesdropping methods, namely acoustic emanations, has received little attention. Recently, Shamir *et al.* [166] have presented introductory proof-of-concept that confirm the correlation between the sound of a PC and its computation. Although we may consider the technique used by P. Wright in 1965 as one of the primitive acoustic attacks [190], this is a relatively new field of research, and much research is needed to confirm its effectiveness against modern cryptographic algorithms and devices.

2.4 Countermeasures against side channel attacks

SCAs primarily focus on the implementation of the cryptographic algorithm. Several software and hardware countermeasures were proposed to combat side channel attacks. Some of these countermeasures make algorithmic changes to the cryptographic primitives so that attacks are provably inefficient on the obtained implementation. Other countermeasures decorrelate the output traces on individual runs. In what follows, we briefly explain how specific countermeasure can be applied in order to thwart different forms of SCAs.

2.4.1 Preventing timing attacks

The most obvious way to prevent timing attacks is to force all operations to consume the same amount of time. When this is not possible, another approach is to make timing measurements extremely inaccurate, e.g., by introducing random timing shifts and wait states, or by inserting dummy instructions. Random delays added to the processing time do increase the number of ciphertexts required by the cryptanalyst, but attackers can compensate for this by collecting more measurements. The number of samples required increases roughly as the square of the timing noise [115]. In [115], Kocher describes a methodology for preventing timing attack based on the techniques used for blinding signature. His approach is to choose a random pair $(v_i,\ v_f\ )$ such that $v_f^{-1} = v_i^x \bmod n$. Before the modular exponentiation operation, the input message should be multiplied with $v_f \bmod n$, and afterwards the result is corrected by again multiplying it with $v_f \bmod n$. The $(v_i,\ v_f\ )$ pair should not be reused since they themselves might be compromised by timing attacks. An efficient solution to this problem is to update v_i and v_f before each modular exponentiation step. If $(v_i,\ v_f\ )$ is secret, then the attacker has no useful knowledge about the input to the modular exponentiation.

Wang *et al.* [188] propose two approaches to overcome cache based SCAs. The first approach is to use a partition based approach to eliminate cache interference. The second is based on randomizing cache interference in such a way to guarantee zero information leakage. The authors have also presented a new security aware cache design which consists of Partition Locked cache (PLcache) and Random Permutation cache (RPcache).

2.4.2 Preventing power analysis attacks

Mitigation power analysis attacks can be achieved by modifying the design of the hardware device in such away that its power consumption becomes random or to make the device consume an equal amount of power for all operations and for all processed data values. The data masking technique, which can be introduced at the software or hardware level, is another widely used countermeasure against power analysis attacks [38, 70]. The basic idea of most masking approaches is to randomize the intermediate values that are processed in the computation of the algorithm operation. In case of AES, the SubByte operation is the only nonlinear operation in the AES algorithm. This operation consists of two sub-operations: the inverse operation and the affine operation. Hence, one should focus on the masking of the inverse operation [148]. The idea of masking the inverse operation is to input the value $A \oplus M$ to the algorithm in such a way that we obtain the output value $A^{-1} \oplus M$, where A is the intermediate data, and M is the mask. There are two types of masking: multiplicative masking and additive masking (e.g., see [4, 69, 150, 182, 195]). Techniques to mitigate higher order power analysis are investigated in e.g., [63, 71, 94, 112, 155, 157, 162].

2.4.3 Preventing fault attacks

Given the relative ease of injecting faults into cryptographic devices, proper countermeasures must be taken in order to keep theses devices secure. Typically, such countermeasures aim to detect any transient or permanent faults that occur in the cryptosystem and immediately suppress the resulting faulted outputs or reset all the output bits to zeros (or any other arbitrary value) in order to prevent the attacker from observing the output of faulty devices and hence mitigate the susceptibility of the system to these attacks. Numerous approaches to fault detection techniques for ciphers have been proposed. These approaches can be divided into two classes: Error Detecting Codes (EDCs) and Redundancy-Based Techniques (RBT) [20, 117].

2.4.4 Preventing cold-boot attacks

As mentioned above, cold-boot attacks exploit DRAM remanence to acquire memory images from which keys and other sensitive data can be extracted. In order to mitigate the threat of this attack, one should apply some transformations to the key as it is stored in memory in order to make it more difficult to reconstruct in the case of errors [35]. One should also prevent keys from being paged to disks by clearing memory at boot time. Other suggested countermeasures include: (i) physically protecting the memory. For example, DRAM modules could be locked in place inside the machine, (ii) suspending the system safely by powering off the machine completely when it is not in use, (iii) encrypting in the disk controller. This approach differs from typical disk encryption systems in that encryption and decryption are done by the disk controller rather than by software in the main CPU, and that the main encryption keys are stored in the disk controller rather than in DRAM. One example for AES implementations that is secure against this class of attacks is Tresor [136] which is a kernel patch for Linux based operating systems which loads and manipulates key related data directly in the microprocessor and its registers.

2.5 The SAT problem

The Boolean Satisfiability (SAT) problem [44] is defined as follows: given a Boolean formula, check whether an assignment of Boolean values to the propositional variables in the formula exists such that the formula evaluates to true. If such an assignment exists, the formula is said to be satisfiable; otherwise, it is unsatisfiable. For a formula with m variables, there are 2^m possible assignments. The Conjunctive Normal Form (CNF) is the most frequently used form for representing Boolean formulas. In CNF, the variables of the formula appear in literals (e.g., x) or their negation (e.g., $\overline{x}$). Literals are grouped into clauses, which represent a disjunction (logical *OR*) of the literals they contain. A single literal can appear in any number of clauses. The conjunction (logical *AND*) of all clauses represents a formula. For example, the CNF formula $(x_1) \wedge (\overline{x}_2 \vee \overline{x}_3) \wedge (x_1 \vee x_3)$ contains three clauses: x_1, $\overline{x}_2 \vee \overline{x}_3$ and $x_1 \vee x_3$. Two literals in these clauses are positive (x_1, x_3) and two are negative ($\overline{x}_2$, $\overline{x}_3$). For a variable assignment to satisfy a CNF formula, it must satisfy each of its clauses. For example, if x_1 is true and x_2 is false, then all three clauses are satisfied, regardless of the value of x_3.

Given a set of equations of a system which we want solve, the higher degree are handled by noting that, for example, the logical expression

$$(x_1 \vee \overline{T})(x_2 \vee \overline{T})(x_3 \vee \overline{T})(x_4 \vee \overline{T})(T \vee \overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4) \tag{2.1}$$

is tautologically equivalent to $T \Leftrightarrow (x_1 \wedge x_2 \wedge x_3 \wedge x_4)$, or the $GF(2)$ equation $T = x_1x_2x_3x_4$. Similar expressions exist for higher order terms. Thus, the system of equations obtained in this step can be linearlized by introducing new variables.

Chapter 3

Applications of SAT Solvers to AES Key Recovery from Decayed Key Schedule Images

3.1 Introduction

A cold-boot attack [72] is a SCA that exploits the fact that data loss of a non-powered RAM can be deployed by cooling it down. In 2002, Skorobogatov [173] performed experiments to study the temperature dependency of data retention time in static RAM devices. The reported results indicated that many chips may preserve data for relatively long periods of time at temperatures above -20°C which contradicted the common wisdom that was widely believed at that time. The temperature at which 80% of the data remained for one minute varied widely between devices. While some devices required cooling to at least -50°C, others, surprisingly, retained data for this period at room temperature. Memory retention time also varied between devices of the same type from the same manufacturer, most likely, because controlling data retention time is not a part of the chip manufacturing quality process.

Thus, one way to launch a cold-boot attack is to remove the memory module, after cooling it, from the target system and immediately plug it in another system under the adversary's control. This system is then booted to access the memory. Another possible approach to execute the attack is to cold-boot the target machine by cycling its power OFF and then ON without letting it shut down properly. Then a lightweight operating system is, instantly, booted where the content of targeted memory is dumped to a file. Further analysis can then be performed against the information that is retrieved from memory in order to find sensitive information such as cryptographic keys or passwords.

Because of the nature of cold-boot attacks, it is realistic to assume that only a corrupted image of the contents of memory will be available to the attacker, i.e., a fraction of the memory bits will be flipped. Halderman et *al.* [72] observed that, within a specific memory region, the decay is overwhelmingly asymmetric, i.e., either $0 \rightarrow 1$ or $1 \rightarrow 0$. When trying to retrieve cryptographic keys, the decay direction for a region can be determined by comparing the number of 0s and 1s since, in an uncorrupted key, the expected number of 0s and 1s should approximately be equal.

Given this asymmetric decay, rectifying these faults can be achieved by further exploiting the redundancy of key material inherent in many widely used cryptographic algorithms. For example, in [75], Heninger *et al.* showed that an RSA private key with small public exponent can be efficiently recovered given a 0.27 fraction of its bits at random. In [72], Halderman *et al.* have developed a recovery algorithm for the 128-bit version of the Advanced Encryption Standard (AES-128) that recovers keys from 30% decayed AES-128 Key Schedule images in less than 20 minutes about half the time. Tsow [184] further improved upon this proof of concept and presented a heuristic algorithm that solved all cases at 50% decay or less in under half a second. At 60% decay, Tsow recovered the worst case in 35.500 seconds while solving the average case in 0.174 seconds. At the extended decay rate of 70%, recovery time averages grew to over 6 minutes with the median time at about five seconds. Nearly half of the 17.4 day run was consumed by solving the worst case of the test suite; the second slowest case was over six times faster.

It should be noted, however, that the algorithm developed by Tsow is complex and is certainly uneasy to develop and fine tune. On the other hand, the relations that have to be satisfied between the different subround key bits in the AES key schedule can be easily formulated as a Boolean SAT problem which lends itself naturally to SAT solvers. In this chapter, we explore the use of CryptoMiniSAT SAT solver [176] on the above problem, i.e., to recover AES-128 keys from its decayed key schedule images.

3.2 The SAT problem and its applications to cryptanalysis

While the SAT problem is NP-complete [44], efficient heuristics exist that can solve many real-life SAT formulations. Furthermore, the wide range of target applications of SAT have motivated advances in SAT solving techniques that have been incorporated into freely-available SAT software tools (e.g., [18, 55, 67, 77, 132]. Also see the international SAT competitions web page [19].)

Given the versatility and effectiveness of SAT solving techniques, the use of SAT solvers in cryptanalysis has recently attracted the attention of many cryptanalysts. In the area of cryptanalysis of block ciphers, Courtois *et al.* [46] presented an attack on the KeeLoq block cipher. They showed that when about 2^{32} known plaintexts are available, KeeLoq becomes very week and for 30% of all keys, the full key can be recovered with complexity of 2^{28} KeeLoq encryptions. Erickson *et al.* [58] used the Gröbner basis attacks [32] on SMS4 equation system over GF(2) and GF(2^8) and used the SAT solver over the GF(2) model. They implement their design in Gröbner basis using a Magma tool and in SAT solver using the MiniSAT tool. In [45], 6 rounds of DES are attacked with only a single plaintext-ciphertext pair.

SAT solvers have also been applied to the cryptanalysis of stream ciphers. Eibach *et al.* [56] presented experimental results over a slightly modified version of Trivium (Bivium) using a SAT solver, an exhaustive search, a BDD based attack, a graph theoretic approach, and Gröbner basis. The results indicate that the initial state of the cipher is recovered and the use of

SAT solver is faster than the other attacks. The full key of Hitage2 stream cipher is recovered in a few hours when using MiniSat 2.0 [47]. In [48], the full 48-bit key of the MiFare Crypto-1 algorithm was recovered in 200 seconds on a PC, given one known Initialization Vector (IV) (from one single encryption).

Mironov and Zhang [129] described some initial results on using SAT solvers to automate certain components in cryptanalysis of hash functions of the MD and SHA families. They generated full collisions for MD4 and MD5. De *et al.* [51] presented heuristics for solving inversion problems for functions that satisfy certain statistical properties similar to those of random functions. They demonstrate that this technique can be used to solve the hard case of inverting a popular secure hash function and inverted MD4 up to 2 rounds and 7 steps in less than 8 hours.

In this work, we use the CryptoMiniSat [176], which is an extension of MinSat (a state-of-the-art DPLL-based [50] SAT solver), refined to understand the XOR operation, which is common in cryptography, besides functions in CNF that is native to many SAT solvers.

3.3 Structure of the AES-128 key schedule

In this section, we briefly review the relevant details of the AES-128 key schedule [49, 59]. Bytes of initial key are denoted by $K^0_{i,j}$, where $0 \le i, j \le 3$ stand for the row index and column index, respectively in the standard AES state matrix representation. Figure 3.1 shows the AES-128 key schedule.

These 16 initial key bytes are bijectively mapped to 10 additional round-keys denoted by $K^{r+1}_{i,j}$, where $0 \le r \le 9$ stands for the number of the subkeys (rounds). The r^{th} key schedule round consists of the following transformations

$$\begin{aligned}
K^{r+1}_{0,0} &\leftarrow S(K^r_{1,3}) \oplus K^r_{0,0} \oplus \text{ Rcon}(r+1) \\
K^{r+1}_{i,0} &\leftarrow S(K^r_{(i+1) \text{ mod } 4,3}) \oplus K^r_{i,0},\ 1 \le i \le 3 \\
K^{r+1}_{i,j} &\leftarrow K^{r+1}_{i,j-1} \oplus K^r_{i,j},\ 0 \le i \le 3, 1 \le j \le 3,
\end{aligned} \tag{3.1}$$

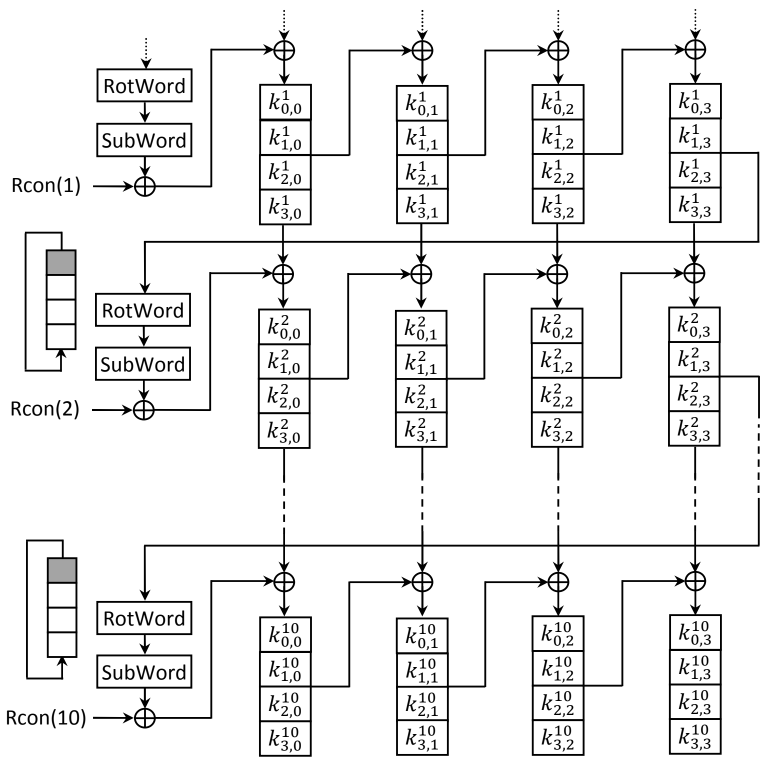


Figure 3.1: The key schedule of the AES-128

where Rcon$(\cdot)$ is a round-dependent constant and $S(\cdot)$ denotes the s-box (SubBytes) operation which is performed, on each byte of the state, by first taking the multiplicative inverse in $GF(2^8)$ using the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$ and then applying an affine transformation over $GF(2)$.

Similar to any Boolean function, each one of the eight coordinate of the s-box, $S_l, l = 1, \cdots, 8$, has a unique representation as a polynomial over $GF(2)$, called the Algebraic Normal Form (ANF) which is obtained by summing up distinct products terms of $x_1, x_2, \ldots, x_n$, and can be written as

$$\begin{aligned} S_l(x_1, \ldots, x_n) = \;& a_0 \bigoplus_{i=1}^{n} a_i x_i \bigoplus_{1 \le i < j \le n} a_{ij} x_i x_j \\ & \bigoplus \cdots \bigoplus a_{123\ldots n} x_1 x_2 \ldots x_n, \end{aligned}$$

where $a_0, a_i, \ldots, a_{123\ldots n} \in GF(2)$.

3.4 Formulating the AES key schedule as a SAT problem

The AES key-schedule, described in the previous section, is the primary source of redundancy utilized to rectify faults in the corrupted memory images of the AES-128 key schedule. The conversion from the key schedule to the Boolean SAT problem proceeds as follows.

First, the system of equations of the AES-128 key schedule are generated according to the pseudocode in Equation (3.1). In each round, the s-box equations in lines 1, 2 of Equation (3.1) are represented in their ANF (e.g., Figure 3.2 shows the ANF of the first coordinate function of the s-box). Then, the terms of quadratic and higher degrees are handled by using the formula in (2.1).

$$\begin{aligned}
S_1 \; = \; & x_1 \oplus x_3 \oplus x_4 \oplus x_6 \oplus x_1x_3 \oplus x_1x_7 \oplus x_1x_8 \oplus x_2x_4 \oplus x_2x_6 \oplus x_2x_8 \oplus x_3x_5 \oplus x_4x_6 \oplus x_6x_7 \\
& \oplus x_6x_8 \oplus x_1x_2x_4 \oplus x_1x_2x_6 \oplus x_1x_2x_8 \oplus x_1x_3x_4 \oplus x_1x_3x_5 \oplus x_1x_4x_7 \oplus x_1x_5x_8 \oplus x_1x_6x_8 \oplus \\
& x_2x_3x_4 \oplus x_2x_3x_6 \oplus x_2x_3x_8 \oplus x_2x_4x_6 \oplus x_2x_5x_7 \oplus x_2x_5x_8 \oplus x_2x_6x_7 \oplus x_2x_7x_8 \oplus x_3x_4x_5 \oplus \\
& x_3x_4x_8 \oplus x_3x_5x_6 \oplus x_3x_5x_7 \oplus x_3x_5x_8 \oplus x_3x_6x_8 \oplus x_3x_7x_8 \oplus x_4x_7x_8 \oplus x_5x_6x_7 \oplus x_5x_6x_8 \oplus \\
& x_1x_2x_3x_4 \oplus x_1x_2x_3x_6 \oplus x_1x_2x_3x_7 \oplus x_1x_2x_4x_6 \oplus x_1x_2x_4x_7 \oplus x_1x_2x_4x_8 \oplus x_1x_2x_5x_7 \oplus \\
& x_1x_2x_5x_8 \oplus x_1x_3x_4x_7 \oplus x_1x_3x_5x_8 \oplus x_1x_4x_5x_6 \oplus x_1x_4x_5x_7 \oplus x_1x_4x_5x_8 \oplus x_1x_4x_6x_8 \oplus \\
& x_1x_4x_7x_8 \oplus x_1x_5x_6x_7 \oplus x_1x_5x_6x_8 \oplus x_1x_6x_7x_8 \oplus x_2x_3x_4x_5 \oplus x_2x_3x_5x_8 \oplus x_2x_4x_5x_8 \oplus \\
& x_2x_4x_6x_7 \oplus x_2x_4x_6x_8 \oplus x_2x_5x_6x_8 \oplus x_2x_5x_7x_8 \oplus x_3x_4x_6x_7 \oplus x_3x_5x_6x_7 \oplus x_3x_5x_6x_8 \oplus \\
& x_3x_6x_7x_8 \oplus x_4x_5x_6x_7 \oplus x_4x_5x_7x_8 \oplus x_4x_6x_7x_8 \oplus x_5x_6x_7x_8 \oplus x_1x_2x_3x_4x_5 \oplus x_1x_2x_3x_4x_7 \oplus \\
& x_1x_2x_3x_5x_7 \oplus x_1x_2x_3x_5x_8 \oplus x_1x_2x_3x_6x_7 \oplus x_1x_2x_3x_7x_8 \oplus x_1x_2x_4x_5x_8 \oplus x_1x_2x_4x_6x_7 \oplus \\
& x_1x_2x_6x_7x_8 \oplus x_1x_3x_4x_7x_8 \oplus x_1x_3x_6x_7x_8 \oplus x_1x_4x_5x_6x_8 \oplus x_1x_4x_5x_7x_8 \oplus x_2x_3x_4x_5x_6 \oplus \\
& x_2x_3x_4x_5x_8 \oplus x_2x_3x_4x_6x_7 \oplus x_2x_3x_4x_7x_8 \oplus x_2x_3x_5x_6x_7 \oplus x_2x_4x_5x_6x_8 \oplus x_2x_4x_6x_7x_8 \oplus \\
& x_2x_5x_6x_7x_8 \oplus x_3x_4x_5x_6x_8 \oplus x_3x_4x_5x_7x_8 \oplus x_4x_5x_6x_7x_8 \oplus x_1x_2x_3x_4x_5x_7 \oplus x_1x_2x_3x_4x_6x_8 \\
& \oplus x_1x_2x_3x_5x_6x_8 \oplus x_1x_2x_3x_5x_7x_8 \oplus x_1x_2x_4x_5x_6x_8 \oplus x_1x_2x_4x_5x_7x_8 \oplus x_1x_2x_5x_6x_7x_8 \oplus \\
& x_1x_3x_4x_5x_6x_8 \oplus x_1x_3x_4x_5x_7x_8 \oplus x_1x_3x_4x_6x_7x_8 \oplus x_2x_3x_4x_5x_6x_8 \oplus x_1x_2x_3x_4x_5x_6x_8 \oplus \\
& x_1x_2x_3x_4x_5x_7x_8
\end{aligned}$$

Figure 3.2: Algebraic normal form of the first coordinate function of the AES s-box

Thus, the system of equations obtained in this step can be linearlized by introducing new variables as illustrated by the following toy example.

Example 3.1 *Suppose we would like to find the Boolean variable assignment that satisfies the following formula*

$$x_0 \oplus x_1x_2 \oplus x_0x_1x_2 = 0$$

Using the approach illustrated in (2.1), we introduce two linearization variables, $T_0 = x_1x_2$ and $T_1 = x_0x_1x_2$. Thus we have

$$\begin{aligned}
&x_0 \oplus T_0 \oplus T_1 = 0,\\
&(\overline{T}_0 \vee x_1) \wedge (\overline{T}_0 \vee x_2) \wedge (T_0 \vee \overline{x}_1 \vee \overline{x}_2) = 1,\\
&(\overline{T}_1 \vee x_0) \wedge (\overline{T}_1 \vee x_1) \wedge (\overline{T}_1 \vee x_2) \wedge\\
&(T_1 \vee \overline{x}_0 \vee \overline{x}_1 \vee \overline{x}_2) = 1.
\end{aligned} \tag{3.2}$$

Since the CryptoMinSAT expects only positive clauses and the CNF form does not have any constants, we need to overcome the problem that the first line in Equation (3.2) corresponds to a negative, i.e., false, clause. Adding the clause consisting of a dummy variable, d, or equivalently ($d \wedge d \cdots \wedge d$) would require the variable d to be true in any satisfying solution, since all clauses must be true in any satisfying solution. In other words, the variable d will serve the place of the constant 1. Therefore, the above formula can be reexpressed as

$$\begin{aligned}
&d = 1,\\
&x_0 \oplus T_0 \oplus T_1 \oplus d = 1,\\
&(\overline{T}_0 \vee x_1) \wedge (\overline{T}_0 \vee x_2) \wedge (T_0 \vee \overline{x}_1 \vee \overline{x}_2) = 1,\\
&(\overline{T}_1 \vee x_0) \wedge (\overline{T}_1 \vee x_1) \wedge (\overline{T}_1 \vee x_2) \wedge\\
&(T_1 \vee \overline{x}_0 \vee \overline{x}_1 \vee \overline{x}_2) = 1.
\end{aligned}$$

Table 3.1 shows the CryptoMiniSat input corresponding to Example 3.1.

In our AES-128 key schedule system, each s-box can be represented by 8 XOR equations corresponding to its 8 Boolean coordinates; and 246 CNF equations corresponding to 246 linearization variables. The total number of clauses corresponding to these CNF equations is equal to 1254. Since, each round in the AES-128 key schedule involves four s-box look-up operations and 96 linear XOR equations (line 3 in Equation (3.1)), then the total number of clauses (including XOR clauses) in each round is equal to $4 \times (1254 + 8) + 96 = 5144$. Thus, for the complete 10 rounds key schedule, we have 10×5144 clauses $+1$ dummy variable to present the

Table 3.1: CryptoMinSAT input corresponding to Example 3.1

```
c Lines starting with ‘c’ are comments
c The first line in the SAT file is in the form:
c ‘p cnf # variables # clauses’
c Each line should end with ‘0’
c Lines starting with ‘x’ denote XOR equations
c True variables are denoted by numbers
c False variables are denoted by negating these numbers
c In this example, $d \rightarrow 1$, $x_0 \rightarrow 2$ (consequently $\overline{x}_0 \rightarrow -2$)
c $x_1 \rightarrow 3$, $x_2 \rightarrow 4$, $T_0 \rightarrow 5$, $T_1 \rightarrow 6$
p cnf 6 9
1   0
x   2   5   6   1   0
-5  3   0
-5  4   0
5   -3  -4  0
-6  2   0
-6  3   0
-6  4   0
6   -2  -3  -4  0
```

constant 1.

3.5 Experimental results

Similar to the previous work in [72,184], throughout our experimental results, we assume an asymmetric decay model where bits overwhelmingly decay to their ground state rather than their charged state. Using this model, only the bits that remain in their charged state will be useful to the cryptanalyst since one cannot be sure about the original values of the 0 bits, i.e., whether they were originally 0s or decayed 1s. Let β denote the fraction of decayed bits. If the percentage of 0s and 1s in the original key schedule bits is p_z and $1 - p_z$, respectively, then the fraction, f, of key bits that can be assumed to be known by examining the decayed memory of the AES key schedule is given by

$$f = 1 - (p_z + \beta \times (1 - p_z)) = (1 - p_z) \times (1 - \beta).$$

Since in an uncorrupted AES key schedule, we expect the number of 0s and 1s to be approximately equal, i.e., $p_z \approx 1/2$, then we have $f \approx (1 - \beta)/2$.

Table 3.2 shows a comparison between our work and the results reported in [184] which recover the AES-128 key schedule from its decayed memory images for a decay factor up to 70%. In this table, for our work, all statistics were generated using 10,000 runs for each decay factor.

It should be noted that the performance in [184] was evaluated on Dell Precision Workstation 7400 running a 3.4 GHz quad-core Xeon processor with 4 GB of RAM. The performance of our approach was evaluated on a slightly less powerful machine: Dell Precision 370 Workstation running a 3.0 GHz Intel Pentium 4 CPU with 1 GB of RAM.

While the algorithm in [184] runs slightly faster for small values of $\beta \leq 50\%$, it is clear that our proposed SAT approach outperforms the algorithm in [184] for large values of β. In [184] with decay factor 70%, the recovery time for the worst case was more than 8.5 days. The average time was 5 minutes and median time was about 5 seconds. In our work, the worst case for the recovery time was obtained in less than 3.5 minutes and 7968 cases were recovered in less than one second. The average and median recovery times are 1.2, 0.36 seconds respectively. It should also be noted that for small values of β, the difference in the run time statistics between the two approaches is practically not very significant because the run time is usually very short.

Table 3.3 shows the time statistics corresponding to decay factors between $72\% - 80\%$. For such a large value of the decay factors, the median is a better indication of the performance of the algorithm than the average since some cases may take a relatively long time while the majority of the cases take a short time. At 72% and 74% the results are promising. The 10,000 cases were solved in an average of 22.3 and 62.4 seconds, respectively. At 72% decay factor, 92% of the cases were recovered in less than 10 seconds, while a similar percentage were

Table 3.2: Run-time statistics for decay factors 30%, 40%, 50%, 60%, and 70%.

	β	30%	40%	50%	60%	70%
This work	Min	0.046	0.046	0.062	0.062	0.078
	Max	0.593	0.140	0.187	0.593	207.171
	Avg.	0.064	0.066	0.074	0.102	1.233
	St.Dev	0.009	0.007	0.008	0.028	4.899
	Med.	0.062	0.062	0.078	0.093	0.359
[184]	Min	0.000[a]	0.000	0.000	0.000	0.000
	Max	0.015	0.015	0.078	2.094	737,266.687
	Avg.	0.009	0.009	0.014	0.174	300.897
	St.Dev	0.007	0.008	0.015	0.772	10,677.913
	Med.	0.015	0.015	0.015	0.031	4.938

a. The value 0.000 means that it less than 1/64

recovered in less than one minute with 74% decay factor. Due to the extended time for key recovery with decay factors 76%, 78%, and 80%, less cases were examined. For $\beta = 80\%$, the CryptoMiniSat search did not terminate for 10 days in the 8^{th} case. Furthermore, in one of the successfully terminated 7 cases, the key returned by the SAT solver was different from the original key. The original key could have been easily found by re-running the SAT solver again after adding a clause to exclude the found key.

Table 3.3: Run-time statistics for decay factors 72%, 74%, 76%, 78%, and 80%.

β	72%	74%	76%	78%	80%
Min	0.078	0.109	0.156	0.625	38.65^b
Max	109794	126772	84819	19987	5523.8^b
Avg.	22.271	62.404	799.327	6958.473	1901.95^b
St.Dev	1155.83	1373.14	5423.73	25880.70	2119.58^b
Med.	0.812	2.656	14.578	173.508	685.046^b
# Tests	10000	10000	1000	100	7

b. These statistics excludes the 8^{th} case where the search did not terminate for 10 days

3.6 Conclusion

In this chapter, we modeled the problem of key recovery of the AES-128 key schedules from its corresponding decayed memory images as a Boolean SAT problem and solved it using the CryptoMiniSat solver. Our experimental results confirm the versatility of our proposed approach which allows us to efficiently recover the AES-128 key schedules for large decay factors. The method presented in this work can be extended in a straightforward way to AES-192, AES-256 and other ciphers with key schedules that can be presented as a set of Boolean equations and, hence, lend themselves naturally to SAT solvers.

Chapter 4

Fault Attacks Against the NTRU Cryptosystems and their Countermeasures

4.1 Introduction

The NTRU encryption algorithm, also known as NTRUEncrypt, is a parameterized family of lattice-based public key cryptosystems [84, 87]. Both the encryption and decryption operations in NTRU are based on simple polynomial multiplication which makes it very fast compared to other alternatives such as RSA, and elliptic-curve-based systems [97]. Recently, the NTRU system has been accepted to the IEEE P1363 standards under the specifications for lattice-based public-key cryptography. In the past few years, the security of NTRUEncrypt was analyzed by many researchers (e.g., [95, 119, 142, 171]). One challenging aspect in cryptanalyzing the NTRU encryption algorithm is its large number of variants and many possible instantiations.

NTRUSign [42, 80, 81, 87] is a parameterized family of lattice-based public key digital signature schemes that is currently under consideration for standardization by the IEEE P1363 working group. It was proposed after the original NTRU Signature Scheme (NSS) [86] was broken [64, 65]. Similar to the GGH cryptosystem [68], the security of NTRUSign is related to the hardness of approximating the Closest Vector Problem (CVP) in a lattice. However, NTRUSign uses a compact lattice, referred to as the NTRU lattices, instead of the GGH lattices [87]. This NTRU lattice has the property that a private $2n$-dimensional basis for the lattice can be described with 2 vectors, each with n coefficients, and a public basis can be described with a single n-dimensional vector. This enables public keys to be represented in $O(n \log n)$ space, rather than $O(n^2)$ as is the case with GGH signature schemes. Furthermore, operations take $O(n^2)$ time, as opposed to $O(n^3)$ for elliptic curve based cryptosystems and RSA private key operations. In April, 2011, Security Innovation, the company that developed the NTRU cryptosystems, announced that its NTRU cryptosystem is approved by the Accredited Standards Committee X9 as a new encryption standard to protect data for financial transactions.

During the past few years, several attempts to analyze the security of NTRUSign were presented. Min *et al.* [128] showed that the original version of NTRUSign signature scheme [81] which was proposed at CT-RSA'03 is not secure in terms of being strongly existential forgeable, i.e., it is malleable. The proposed attack allows an adversary to forge new signatures for a message of her choice, given a signature for this message. This forgery requires a specific polynomial with a small coefficient satisfying its norm value equal to zero. Even if this forgery does not admit an adversary to change the message, this attack limits the applications of this version of NTRUSign, which does not use perturbation, in several applications.

Szydlo [181] introduced a new lattice reduction technique applicable to the class of hypercubic lattices which arise during transcript analysis of certain NTRUSign signature schemes. After a few thousand signatures, key recovery amounts to discovering a hidden unitary matrix from its Gram matrix [181]. This case of the Gram matrix factorization problem is equivalent to finding the shortest vectors in the hypercubic lattice defined by a quadratic form. This work on reduction of hypercubic lattices shows that the transcript attacks are relevant to the security of the NTRUSign schemes. The practical security threat to the schemes, however, was not clear given the required large number of oracle calls. Furthermore, the attack does not seem to be applicable to NTRUSign with perturbation.

Nguyen and Regev [143] presented the first successful key-recovery attack on NTRUSign-251 without perturbation [80, 81]. Experimentally, this attack requires about 400 signatures to recover the secret key of this non-perturbed version of NTRUSign. The main idea of the attack is based on the following learning problem: given many random points uniformly distributed over an unknown n-dimensional parallelepiped, recover the parallelepiped or an approximation thereof. Nguyen and Regev transformed this problem into a multivariate optimization problem which they solved by a gradient descent. Again, this attack does not seem to be directly applicable to NTRUSign with perturbation where, in this case, the attacker has to solve an extension of the hidden parallelepiped problem in which the parallelepiped is replaced by the Minkowski sum of two hidden parallelepipeds where the lattice spanned by one of the parallelepipeds is public but the other one is not.

Fault analysis is an example of SCAs in which the attacker is assumed to be able to induce faults in the cryptographic device and observe the faulty output. Then, by careful inspection of the faulty output, the attacker recovers the secret information, such as secret inner state or secret key. In fault analysis attacks, some kind of physical influence such as variations in the supply voltage, external clock, or temperature beyond the nominal operating range of the device, results in a corruption of the internal memory or the computation process. Other techniques for fault injections include subjecting the device to white light, laser beams, X-rays, or ion beams [16]. The examination of the output under such faults often reveals some information about the cipher key or the secret inner state. The first fault analysis attack targeted the RSA cryptosystem [29] and subsequently, fault analysis attacks were expanded to various cryptosystems (e.g., [23,79,154]) and were extended to other digital signature schemes (e.g., [23,24,66]). Furthermore, fault analysis attacks became a more realistic serious threat after cheap and low-tech methods of applying faults were presented [11].

Other variants of fault analysis attacks against the RSA cryptosystem were also investigated. The perturbation of public elements was considered as a real threat when Seifert presented an attack on the RSA signature checking mechanism [133, 163]. Then, Brier *et al.* [31]

extended this work to full recovery of the private signing exponent for various RSA implementations. Berzati *et al.* [21, 22] address the issue of modifying the modulus during the exponentiation. Fault analysis attacks were also extended to symmetric systems such as DES [25] and later to AES [54]. Fault attacks against stream ciphers were introduced by Hoch *et al.* [79].

Different assumptions are usually made regarding both timing and location of the injected faults [27]. The number of required faults in fault attacks varies depending on the assumed model. In particular, some strong fault models assume that the adversary has precise full control on both timing and location of injected faults which implies that the attacker has full control on the location of faulted bits as well as the attacked operation.

In this work, we present a fault analysis attack on the original NTRUEncrypt algorithm proposed in [84] and on the NTRUSign signature scheme. The fault model that we utilize in this work is slightly more relaxed. In particular, we assume that the attacker is able to fault a small number of coefficients in a specific polynomial during the decryption/ signing process but cannot control the exact location of the injected transient faults.

4.2 Fault analysis of NTRUEncrypt cryptosystem

4.2.1 Description of the NTRUEncrypt encryption algorithm

The NTRU encryption algorithm is a lattice-based public key cryptosystems that is parameterized by three integers: (N, p, q), where N is prime, gcd(p, q) = 1 and $p << q$. Let R, R_p, and R_q be the polynomial rings

$$R = \frac{\mathbb{Z}[x]}{x^N - 1}, R_p = \frac{(\mathbb{Z}/p\mathbb{Z})[x]}{x^N - 1}, R_q = \frac{(\mathbb{Z}/q\mathbb{Z})[x]}{x^N - 1}.$$

The product of two polynomials $a(x), b(x) \in R$ is given by

$$\begin{aligned}
&c(x) = a(x) \star b(x), \\
&\text{where,} \\
&c_k = \textstyle\sum_{i=0}^{k} a_i b_{k-i} + \sum_{i=k+1}^{N-1} a_i b_{N+k-i} = \sum_{j=k-i \ (\text{mod } N)} a_i b_j.
\end{aligned} \tag{4.1}$$

For any positive integers d_1 and d_2, let $\tau(d_1, d_2)$ denote the set of ternary polynomials given by

$$\left\{ a(x) \in R : \begin{array}{l} \text{a(x) has } d_1 \text{ coefficients equal to 1,} \\ \text{a(x) has } d_2 \text{ coefficients equal to -1,} \\ \text{all other coefficients equal to 0} \end{array} \right\}$$

In what follows, we briefly describe the key generation, encryption and decryption operations in the NTRU cryptosystem [87].

Key Generation

- Choose a private $f(x) \in \tau(d_f, d_f - 1)$ that is invertible in R_q and R_p.
- Choose a private $g(x) \in \tau(d_g, d_g)$.
- Compute $F_q(x) = f^{-1}(x)$ in R_q and $F_p(x) = f^{-1}(x)$ in R_p.
- Compute $h(x) = F_q(x) \star g(x)$ in R_q.

The polynomial $h(x)$ is the user's public key. The corresponding private key is the pair $(f(x), F_p(x))$. The following steps denote the encryption process for plaintext $m(x) \in R_p$.

Encryption

- Choose a random ephemeral key $r(x) \in \tau(d_r, d_r)$.
- Compute the ciphertext $e(x) = pr(x) \star h(x) + m(x)$ mod q.

Decryption

- Compute $a(x) = f(x) \star e(x) \bmod q$.
- Compute $b(x)$ = Centerlift $(a(x))$ such that its coefficients lie in the interval $(-q/2, q/2]$.
- Compute $m = F_p(x) \star b(x) \bmod p$.

Table 4.1: The parameter sets for NTRU in [85]

	N	p	q	d_f	d_g	d_r
Moderate Security	167	3	128	61	20	18
High Security	263	3	128	50	24	16
Highest Security	503	3	256	216	72	55

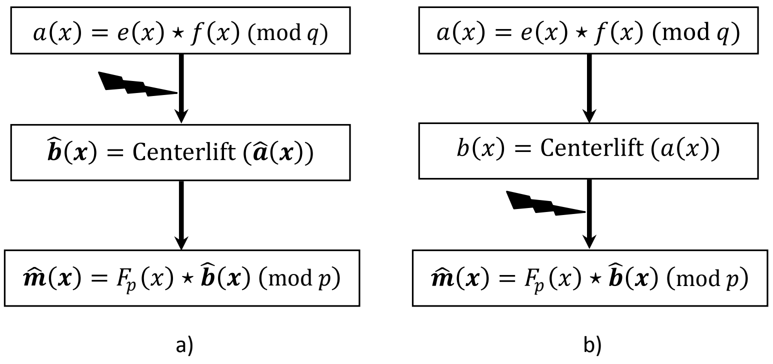
Table 4.1 shows some suggested choices for (N, p, q) for different security levels of the original NTRUEncryption algorithm [85].

By choosing $f(x) = 1 + pf_1(x)$, where $f_1(x) \in R$ in the key generation step, the polynomial multiplication in the last decryption step is eliminated since we will have $F_p(x) = 1 \bmod p$. In this case, Hoffstein and Silverman [88, 89] described a method for speeding up the encryption and decryption processes through the use of products of low Hamming weight polynomials. They use three low Hamming weight polynomials $r_1(x)$, $r_2(x)$, and $r_3(x)$ such that $r(x) = r_1(x) \star r_2(x) + r_3(x)$ in encryption and $f_1(x)$, $f_2(x)$, and $f_3(x)$ such that $f(x) = f_1(x) \star f_2(x) + f_3(x)$ in decryption. For example, at $(N, p, q) = (1171, 3, 2048)$ the number of non-zero coefficients for each $r_i(x)$ and $f_i(x)$ is 5. The convolution multiplication for encryption $t(x) = r(x) \star h(x) \bmod q$ can be calculated as follows:

$$\begin{aligned} t_1(x) &= r_2(x) \star h(x), \\ t_2(x) &= r_3(x) \star h(x), \\ t_3(x) &= r_1(x) \star t_1(x), \\ t(x) &= t_2(x) + t_3(x) \bmod q. \end{aligned} \tag{4.2}$$

4.2.2 Proposed attack

Similar to the case of attacking any public key cryptosystem, our proposed fault analysis attack targets the decryption process which, as explained in the previous section, contains three main operations. We assume that the attacker is able to fault a small number of coefficients of the polynomial input to (or output from) the second step of the decryption process but cannot control the exact location of the injected fault. In particular, as depicted in Figure 4.1, we assume that the attacker is able to fault small number of coefficients of the polynomial input to (or output from) the centerlift operation. Thus, the output of the faulty decryption process can be expressed as:



$a(x) = e(x) \star f(x) \pmod q$
$\hat{b}(x)$ = Centerlift $(\hat{a}(x))$
$\hat{m}(x) = F_p(x) \star \hat{b}(x) \pmod p$
a)
$a(x) = e(x) \star f(x) \pmod q$
$b(x)$ = Centerlift $(a(x))$
$\hat{m}(x) = F_p(x) \star \hat{b}(x) \pmod p$
b)

Figure 4.1: The decryption process after inducing faults: a) before the centerlift operation or b) after the centerlift operation

$$\widehat{m}(x) = \quad F_p \star (b(x) + \epsilon(x)) \bmod p$$

where $\epsilon(x) = \sum_{j=1}^{t} \epsilon_{i_j} x^{i_j} \in R_q, 0 \leq i_j < N$, denotes the resulting error polynomial with t non zero coefficients in the locations corresponding to the injected faults. Thus, the attacker can calculate

$$\Delta m(x) = \widehat{m}(x) - m(x) = \epsilon_p(x) \star F_p(x) \bmod p, \tag{4.3}$$

where $\epsilon_p(x) = \epsilon(x) \bmod p$.

Then, the attacker can obtain candidates for the secret key as

$$f(x) = (\Delta m(x))^{-1} \star \epsilon_p(x) \bmod p \tag{4.4}$$

by exhaustively trying all possible values for $\epsilon_p(x)$ with the pre-specified small number of non zero coefficients. The right secret key can be determined by performing the encryption process on the ciphetext and comparing the result with the original plaintext.

4.2.3 Complexity and success probability of the proposed attack

For typical values of p and q, a non zero coefficient in $\epsilon(x) \in R_q$ will be mapped to different values in $\mathbb{Z}_p$ with almost equal probabilities ($\approx \frac{1}{p}$) after reduction modulo p (e.g., for $p = 3$, a non zero coefficient $\in \mathbb{Z}_q^*$ will be mapped to $(0, 1, 2)$ with probabilities $\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$, respectively, for $q = 64$, $q = 256$ and $\frac{42}{127}, \frac{43}{127}, \frac{42}{127}$ for $q = 128$). To simplify our analysis, in what follows we assume that a non zero coefficient in $\epsilon(x) \in R_q$ will be mapped to different values in $\mathbb{Z}_p$ with probability $\frac{1}{p}$.

The above attack succeeds if, and only if, $\Delta m(x) \in R_p$ is invertible. From Equation (4.3), this requires that $\epsilon_p(x) = \epsilon(x) \bmod p$ to be invertible (note that F_p is guaranteed to be invertible by the key generation process.) Let $n \geq 1$ be the smallest integer such that $p^n \equiv 1 \bmod N$. Then, from [169], the probability that $\epsilon_p(x) \in R_P$ is invertible is given by

$$\left(1 - \frac{1}{p}\right)\left(1 - \frac{1}{p^n}\right)^{(N-1)/n}$$

which is approximately equal to $(1 - \frac{1}{p})$ for the typical choice of the parameters (e.g., for $p = 3$, we have $n = 83, 131, 251$ for $N = 167, 263, 503$, respectively.)

If the fault injection procedure ensures that the number of injected faults is less than or equal to t, then our attack requires $(p^t - 1)\binom{N}{t} = O((pN)^t)$ calculations of multiplicative inverse in R_p which can be done efficiently using the extended Euclidean algorithm.

4.2.4 Proposed countermeasures

To secure cryptographic devices against fault attacks, proper countermeasures have to be applied in order to detect any transient or permanent faults and prevent the attacker from accessing the faulty output by immediately disabling the device output or resting all the output bits to 0s. Several techniques of fault detection have been investigated [117]. These techniques include error detection codes and redundancy-based techniques. In what follows, we investigate different approaches to deploy the above fault detection techniques in hardware implementations of NTRUEncrypt and study the trade-off between the fault coverage and the hardware area and the throughput overheads.

Spatial and temporal duplication

Applying spatial duplication to the decryption process is quite straightforward. Spatial duplication requires redundant hardware to allow independent calculations so that faults injected into one hardware unit do not affect (in the same way) the other unit(s).

A different approach for applying duplication relies on having a separate hardware module for executing the encryption process. After completing the decryption, the encryption unit is applied to the resulting plaintext, and only if the result of the encryption is equal to the original ciphertext, then the system considered fault-free. In this section we explore the trade-off between different spatial and temporal duplication options applicable to NTRUEncrypt.

- Decryption-Decryption

  Figure 4.2 shows the case where error detection is achieved by duplicating the decryption operation. Note that throughout our work, we exclude the possibility of higher order faults where the attacker is able to fault the error detection circuity, e.g., by forcing the multiplexer in Figure 4.2 to always output m irrespective of the value of its *error* control signal.

  While full spatial duplication has practically no impact on the throughput and it can detect both permanent and transient faults, the associated area overhead is considerable. Instead, a large saving in the area can be achieved by utilizing temporal redundancy. A naïve implementation of this approach would practically double the decryption time. In our implementation, however, the convolution operation $f \star e$ required by the second decryption operation is performed at the same time with the operation $F_p \star b$. Thus the decryption time grows from $\approx 2N$ to $\approx 3N$ (instead of $\approx 4N$).

- Decryption-Rotation-Decryption

  Applying temporal redundancy only will not detect permanent faults. The following Lemma shows how this limitation can be alleviated by applying the redundant decryption operation on a related ciphertext.

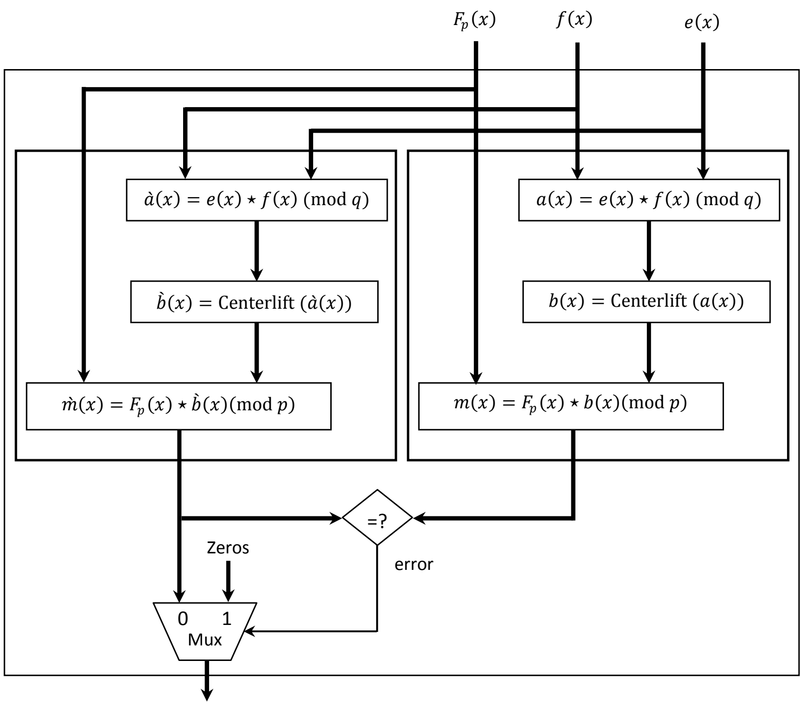

$F_p(x)$
$f(x)$
$e(x)$
$\grave{a}(x) = e(x) \star f(x)$ (mod q)
$\grave{b}(x) =$ Centerlift ($\grave{a}(x)$)
$\grave{m}(x) = F_p(x) \star \grave{b}(x)$(mod p)
$a(x) = e(x) \star f(x)$ (mod q)
$b(x) =$ Centerlift ($a(x)$)
$m(x) = F_p(x) \star b(x)$(mod p)
=?
Zeros
error
0 1
Mux

Figure 4.2: Detecting errors using the duplication method (decryption followed by decryption). When temporal redundancy is used, $\grave{a}(x)$ is calculated at the same time with $m(x)$ using the same hardware used to calculate $a(x)$.

Lemma 4.1 *Let* $e^{(s)}(x) \in R$ *denote an s-cyclically shifted version of* $e(x) = \sum_{i=0}^{N-1} e_i x^i$, $0 < s \le N-1$, *i.e.,* $e^{(s)}(x) = (e(x) >>> s)$ *(In other words, the coefficients of* $e^{(s)}(x)$

are obtained by rotating the coefficients of $e(x)$ *by* s *positions.) Then the plaintext* $\grave{m}$ *corresponding to the decryption of* $e^{(s)}(x)$ *is equal to* $(m(x) >>> s)$ *where* $m(x)$ *is the plaintext corresponding to the decryption of* $e(x)$*.*

Proof: Since, the coefficients of $e^{(s)}(x)$ are obtained by rotating the coefficients of $e(x)$ by s positions, we have $e^{(s)}(x) = \sum_{i=0}^{N-1} e_i x^{i+s \pmod N} = x^s \star e(x)$.

Let $\grave{a}$ and $\grave{b}$ denote the intermediate computation results during the decryption of $e^{(s)}(x)$. Then we have

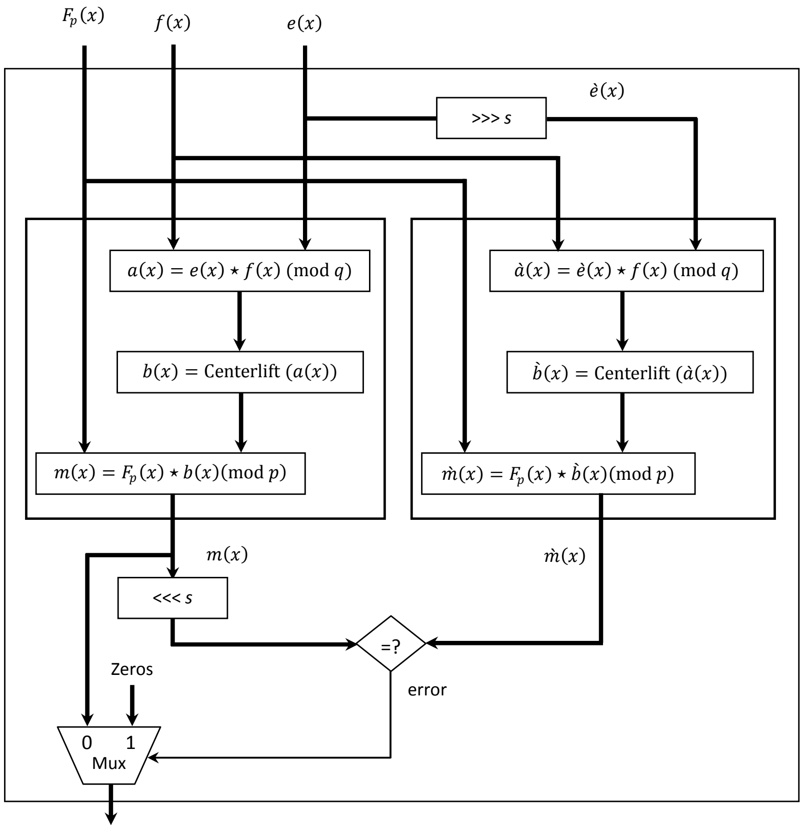
$$\begin{aligned}
\grave{a} &= e^{(s)}(x) \star f(x) \bmod q \\
&= (x^s \star e(x)) \star f(x) \bmod q \\
&= x^s \star (e(x) \star f(x)) \bmod q \\
&= x^s \star a(x) \Rightarrow \\
\grave{b} &= \text{ Centerlift } (\grave{a}) \\
&= x^s \star \text{ Centerlift } (a(x)) \\
&= x^s \star b(x) \Rightarrow \\
\grave{m} &= \grave{b} \star F_p(x) \bmod p \\
&= (x^s \star b(x)) \star F_p(x) \bmod p \\
&= x^s \star (b(x) \star F_p(x)) \bmod p \\
&= x^s \star m(x).
\end{aligned}$$

■

Thus, in order to detect permanent faults when temporal redundancy is utilized, the redundant computation can be performed using a rotated version of the ciphertext and the two plaintexts are compared as shown in Figure 4.3. Again, resource sharing and pipelining are used between the two decryption processes in the same way as when no rotation was employed.

- Decryption-Encryption

A naïve implementation for this approach would be to perform all the encryption steps



$F_p(x)$
$f(x)$
$e(x)$
$\grave{e}(x)$
$>>> s$
$a(x) = e(x) \star f(x) \pmod q$
$b(x) = \text{Centerlift}\,(a(x))$
$m(x) = F_p(x) \star b(x) (\text{mod } p)$
$\grave{a}(x) = \grave{e}(x) \star f(x) \pmod q$
$\grave{b}(x) = \text{Centerlift}\,(\grave{a}(x))$
$\grave{m}(x) = F_p(x) \star \grave{b}(x) (\text{mod } p)$
$m(x)$
$\grave{m}(x)$
$<<< s$
=?
Zeros
error
0
1
Mux

Figure 4.3: Detecting errors by using the duplication method (decryption of a ciphertext and its rotated version)

after the decryption is finished which dramatically reduces the systems throughput.

As depicted in Figure 4.4, in order to speed up this approach, all the encryption operations including the convolution step required by the encryption process can be performed before the plaintext is produced because only the last addition operation in the encryption process requires the availability of the plaintext. In our implementation, both convolution operations for the encryption $r \star h$ and decryption $F_p \star b$ are performed simultaneously in N clock cycles. Similar to the above two approaches, this scheme recovers 100% of the induced transient and permanent errors.

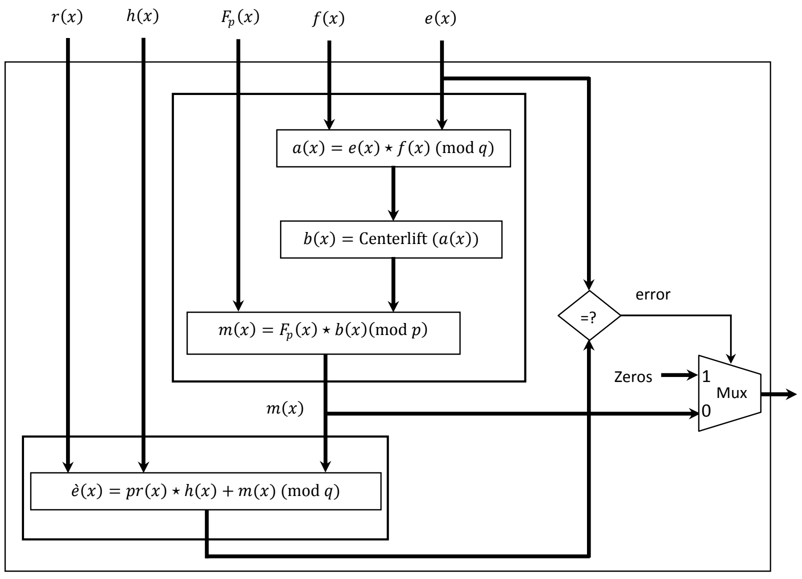

$r(x)$
$h(x)$
$F_p(x)$
$f(x)$
$e(x)$
$a(x) = e(x) \star f(x) \pmod q$
$b(x) = \text{Centerlift}(a(x))$
$m(x) = F_p(x) \star b(x) \pmod p$
$m(x)$
$\grave{e}(x) = pr(x) \star h(x) + m(x) \pmod q$
=?
error
Zeros
1
0
Mux

Figure 4.4: Detecting errors by using the duplication method (decryption followed by encryption)

Error-detecting codes

The basic idea of code-based fault detection schemes is that a checksum code for the output of a given module can be predicted from the input to this module and consequently it can be compared to the actual checksum calculated from the output. The disadvantage of this technique is that corrupted bits may affect the code in such a way that errors may not be detected.

Unlike the case of symmetric key ciphers, where traditional parity based error detection codes (over $GF(2)$) are somewhat straightforward to derive for the intermediate computation steps of the cipher, this does not seem to be the case for many public key systems including NTRUEncrypt. On the other hand, the algebraic properties of the polynomials convolutional product allow us develop simple check sum fault detection techniques which detect both transient and permanent faults. The following Lemmas will be utilized in the development of our EDC based techniques.

Lemma 4.2 *Let* $e(x) = \sum_{i=0}^{N-1} e_i x^i$*,* $m(x) = \sum_{i=0}^{N-1} m_i x^i$*, and* $a(x) = \sum_{i=0}^{N-1} a_i x^i, b(x) = \sum_{i=0}^{N-1} b_i x^i$ *denote the ciphertext, plaintext and intermediate computation results as defined in* *section 4.2.1. Then we have*

$$(\sum_{i=0}^{N-1} e_i) \textit{ mod } q = (\sum_{i=0}^{N-1} a_i) \textit{ mod } q = (\sum_{i=0}^{N-1} b_i) \textit{ mod } q$$

Proof: In matrix form, the convolution operation $a(x) = e(x) \star f(x) \bmod q$ can be expressed as

$$\begin{pmatrix} a_0 \\ a_1 \\ \cdot \\ \cdot \\ \cdot \\ a_{N-1} \end{pmatrix} = \begin{pmatrix} e_0 & e_{N-1} & \cdots & e_1 \\ e_1 & e_0 & \cdots & e_{N-2} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ e_{N-1} & e_{N-2} & \cdots & e_0 \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \cdot \\ \cdot \\ \cdot \\ f_{N-1} \end{pmatrix}$$

$$= \begin{pmatrix} (f_0 \times e_0 + f_1 \times e_{N-1} + \cdots + f_{N-1} \times e_1) \bmod q \\ (f_0 \times e_1 + f_1 \times e_0 + \cdots + f_{N-1} \times e_{N-2}) \bmod q \\ \cdot \\ \cdot \\ \cdot \\ (f_0 \times e_{N-1} + f_1 \times e_{N-2} + \cdots + f_{N-1} \times e_0) \bmod q \end{pmatrix}.$$

Thus we have

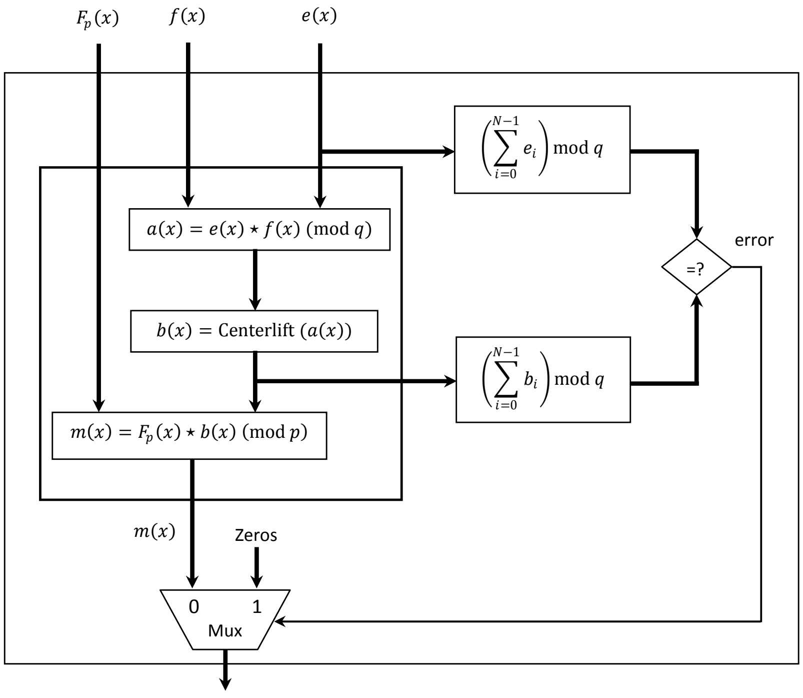
$$(\sum_{k=0}^{N-1} a_k) \bmod q = (\sum_{k=0}^{N-1} e_k \times \sum_{j=0}^{N-1} f_j) \bmod q.$$

From the key generation process, we have $f(x) \in \tau(d_f, d_f - 1)$. Thus $\sum_{j=0}^{N-1} f_j = 1$. Consequently, we have

$$(\sum_{k=0}^{N-1} a_k) \bmod q = (\sum_{k=0}^{N-1} e_k) \bmod q.$$

The second part of the Lemma follows by noting that from the definition of the Centerlift operation, we have $a_i = b_i \bmod q$, $i = 0, \cdots, N-1$. ■

Figure 4.5 shows how the checksum equation of Lemma 4.2 can be applied to implement error detection for errors inserted before or after the calculation of the Centerlift operation (corresponding to Figure 4.1-a and Figure 4.1-b, respectively).



$F_p(x)$
$f(x)$
$e(x)$
$\left(\sum_{i=0}^{N-1} e_i\right) \bmod q$
$a(x) = e(x) \star f(x) \pmod q$
error
=?
$b(x) = \text{Centerlift}\,(a(x))$
$\left(\sum_{i=0}^{N-1} b_i\right) \bmod q$
$m(x) = F_p(x) \star b(x) \pmod p$
$m(x)$
Zeros
0
1
Mux

Figure 4.5: Detecting errors by using checksum EDC from Lemma 4.2

Let $\widehat{b}(x) = b(x) + \epsilon(x)$ denote the faulty polynomial corresponding to $b(x)$, where $\epsilon(x) = \sum_{i_j=1}^{t} \epsilon_{i_j} x^{i_j} \in R_q, 0 \leq i_j < N$, denotes the resulting error polynomial with t non zero coefficients in the locations corresponding to the injected faults. From Lemma 4.2, it is clear that this checksum error detection method cannot detect errors if $\sum_{i_j=1}^{t} \epsilon_{i_j} = 0$. Assuming that the coefficients of the injected errors are uniformity distributed over $\mathbb{Z}_q$ and by noting that, in this case, the function $\sum_{i_j=1}^{t} \epsilon_{i_j}$ is a balanced function, i.e., it assumes all possible values in $\{0, 1, \cdots q-1\}$ with equal probability $= \frac{1}{q}$. Then, by ignoring the possibility of simultaneously faulting the error detection circuit itself, the error detection probability for this scheme is given by $\frac{q-1}{q} = 1 - \frac{1}{q}$.

In what follows, we show that a mod p checksum formula that holds between the input and output of the third decryption step. Using the same notation as in Lemma 4.2, in what follows and using Lemma 4.3, 4.4, we derive a checksum formula (mod p) that holds between the input and output of the last step in the decryption process.

Lemma 4.3 *Let $F_p = \sum_{i=0}^{N-1} F_p^i x^i$, where F_p^i is the coefficient of x^i in the polynomial F_p. Then we have*

$$(\sum_{i=0}^{N-1} F_p^i) \; mod \, p = 1.$$

Proof: By definition, $f(x) \star F_p$ mod p $= 1$. Thus we have

$$\begin{pmatrix} 1 \\ 0 \\ . \\ . \\ . \\ 0 \end{pmatrix} = \begin{pmatrix} f_0 & f_{N-1} & \cdots & f_1 \\ f_1 & f_0 & \cdots & f_{N-2} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ f_{N-1} & f_{N-2} & \cdots & f_0 \end{pmatrix} \begin{pmatrix} F_p^0 \\ F_p^1 \\ . \\ . \\ . \\ F_p^{N-1} \end{pmatrix}$$

$$= \begin{pmatrix} (F_p^0 f_0 + F_p^1 f_{N-1} + \cdots + F_p^{N-1} f_1) \text{ mod } p \\ (F_p^0 f_1 + F_p^1 f_0 + \cdots + F_p^{N-1} f_{N-2}) \text{ mod } p \\ . \\ . \\ . \\ (F_p^0 f_{N-1} + F_p^1 f_{N-2} + \cdots + F_p^{N-1} f_0) \text{ mod } p \end{pmatrix}.$$

Again, by noting that $\sum_{k=0}^{N-1} f_k = 1$, we have

$$(\sum_{k=0}^{N-1} F_p^k \times \sum_{j=0}^{N-1} f_j) \text{ mod } p = 1 \Rightarrow (\sum_{k=0}^{N-1} F_p^k) \text{ mod } p = 1.$$

■

Lemma 4.4

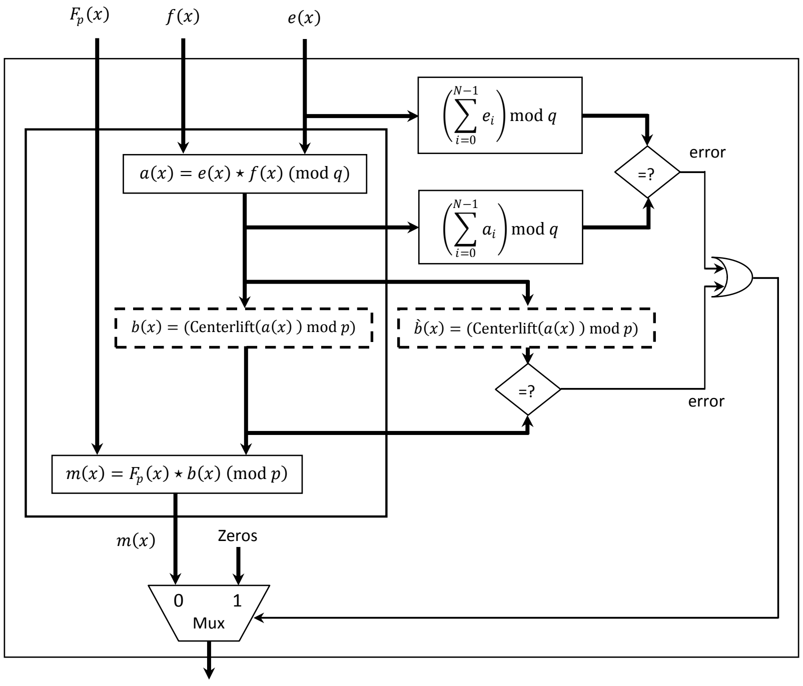
$$(\sum_{i=0}^{N-1} b_i)\ mod\, p = (\sum_{i=0}^{N-1} m_i)\ mod\, p.$$

Proof: Similar to the proof of Lemma 4.2, we express the convolution operation $m(x) = b(x) \star F_p(x) \bmod p$ in matrix form. Then, the proof follows by utilizing the result of Lemma 4.3 to simplify the resulting summation. ■

In our implementation, we did not utilize the above mod p checksum since it does not improve the overall error detection capability of our implementation. In particular, for the implementation shown in Figure 4.5, in order to be able to detect errors that might be injected in $b(x)$ during the calculations of m in the third step of the decryption process, this step is performed on the same set of registers that hold the result of the Centerlift operation mod q. Consequently, any injected errors in the coefficients of $b(x)$ during the calculation of $m(x)$ can be detected, with probability $1 - \frac{1}{q}$, by the checksum in the figure. In the next section we show how we can improve the area requirement of this approach without impacting the error detection capability.

Combining spatial redundancy and error detection codes

While visualizing the decryption process as composed of three separate steps (see section 4.2.1) allows us to better understand the mathematics behind it, for hardware implementations, we do not have to separate these three steps. In particular, since the coefficients of b are eventually reduced $\bmod p$ during the calculation of m in the third step of the decryption, one can directly evaluate the Centerlift operation and reduce the coefficient of $b(x) \bmod p$ in one step. In our FPGA prototype, modular reduction $\bmod p$ is implemented using the algorithm for modular reduction mod Mersenne primes (see Algorithm 3.1 in [189]). As depicted in Figure 4.6, spatial duplication is applied to detect errors in the mod p operations by duplicating the calculation of Centerlift$(a(x)) \bmod p$ and storing the results in two different registers, b and $\grave{b}$. In this case, by comparing the two registers, any error occurring during the calculation of any of them can be detected with 100% probability. On the other hand, if the attacker induces a fault in the polynomial $a(x)$, this duplication cannot detect this error because the registers b and $\grave{b}$ will be identical. However, in this case, the applied checksum mod q detects this error with probability $1 - \frac{1}{q}$.



$F_p(x)$
$f(x)$
$e(x)$
$\left(\sum_{i=0}^{N-1} e_i\right) \bmod q$
$a(x) = e(x) \star f(x) \pmod q$
$\left(\sum_{i=0}^{N-1} a_i\right) \bmod q$
=?
error
$b(x)$ = (Centerlift($a(x)$)) mod p)
$\grave{b}(x)$ = (Centerlift($a(x)$)) mod p)
=?
error
$m(x) = F_p(x) \star b(x) \pmod p$
$m(x)$
Zeros
0
1
Mux

Figure 4.6: Detecting errors using checksum EDC and spatial redundancy

4.2.5 FPGA implementation results

We used the Xilinx ISE 9.1i framework to prototype our protected NTRUEncrypt hardware with parameters $(N, p, q, df, dg, dr) = (167, 3, 128, 61, 20, 18)$. The synthesis was performed with XST application and the simulation was performed using Modelsim. The target FPGA is xcv1000e from Xilinx Virtex-E family.

At the beginning of the decryption operation, the private key $f(x) \in R_p$ is loaded into the chip through N parallel I/O PINS in $T_{ld(f)} = \lceil log_2(p) \rceil$ clock cycles. Similarly, the private key $F_p(x)$ is loaded in $T_{ld(F_p)} = \lceil log_2(p) \rceil$ clock cycles. The ciphertext $e(x)$ is then loaded in $log_2(q)$ clock cycles. When the encryption process in required by the redundancy scheme, h and r are loaded in $T_{ld(h)} = log_2(q)$ and $T_{ld(r)} = \lceil log_2(p) \rceil$ clock cycles, respectively. The most time consuming steps are the convolution operations which require $T_{conv(\cdot,\cdot)} = N$ clock cycles, each.

The Centerlift operation, $T_{cl(.)}$, consumes one clock cycle. Furthermore, $T_{out(m)} = \lceil log_2(p) \rceil$ clock cycles are be required to output the plaintext $m(x)$ using N parallel I/O PINS. Let $T_{sum(a)}$ and $T_{sum(b)}$ denotes the number of clock cycles for accumulating the coefficients of the register a mod q and b mod q, respectively. Figure 4.7 shows a simplified time-line for the operations required by the above approaches. The figure also illustrates the operations that are performed in parallel and those that utilize resource sharing. In the figure, the time between the vertical dashed lines represents the number of clock cycles required to perform one decryption operation for the input block assuming that the private key, and when applicable the public key, are already loaded into the chip.

The IEEE standard 1363.1-2008 specifies a message encoding scheme (from binary to ternary) for NTRUEncrypt. In particular, each three bits of the binary presentation of the message is converted to two ternary coefficients as follows

$$\begin{array}{ll} \{0,0,0\} \rightarrow \{0,0\}, & \{0,0,1\} \rightarrow \{0,1\}, \\ \{0,1,0\} \rightarrow \{0,-1\}, & \{0,1,1\} \rightarrow \{1,0\}, \\ \{1,0,0\} \rightarrow \{1,1\}, & \{1,0,1\} \rightarrow \{1,-1\}, \\ \{1,1,0\} \rightarrow \{-1,0\}, & \{1,1,1\} \rightarrow \{-1,1\}. \end{array}$$

Given this encoding scheme, each block with N coefficients can be used to encode a message of length $\frac{3}{2} \times N$ bits. Thus

$$\text{Throughput} \approx \frac{\text{Clock frequency} \times 1.5N}{\text{Clock cycles required to decrypt one block}}.$$

For all designs, the clock frequencies calculated by the synthesis tool were $\gtrsim 55.4$ MHz. Table 4.2 shows the FPGA resources, throughput and error detection capability of all the above proposed architectures when running at 55 MHz. It should be noted that the above performance figures refer to the raw NTRU encryption process. In practice, to encrypt a message securely,

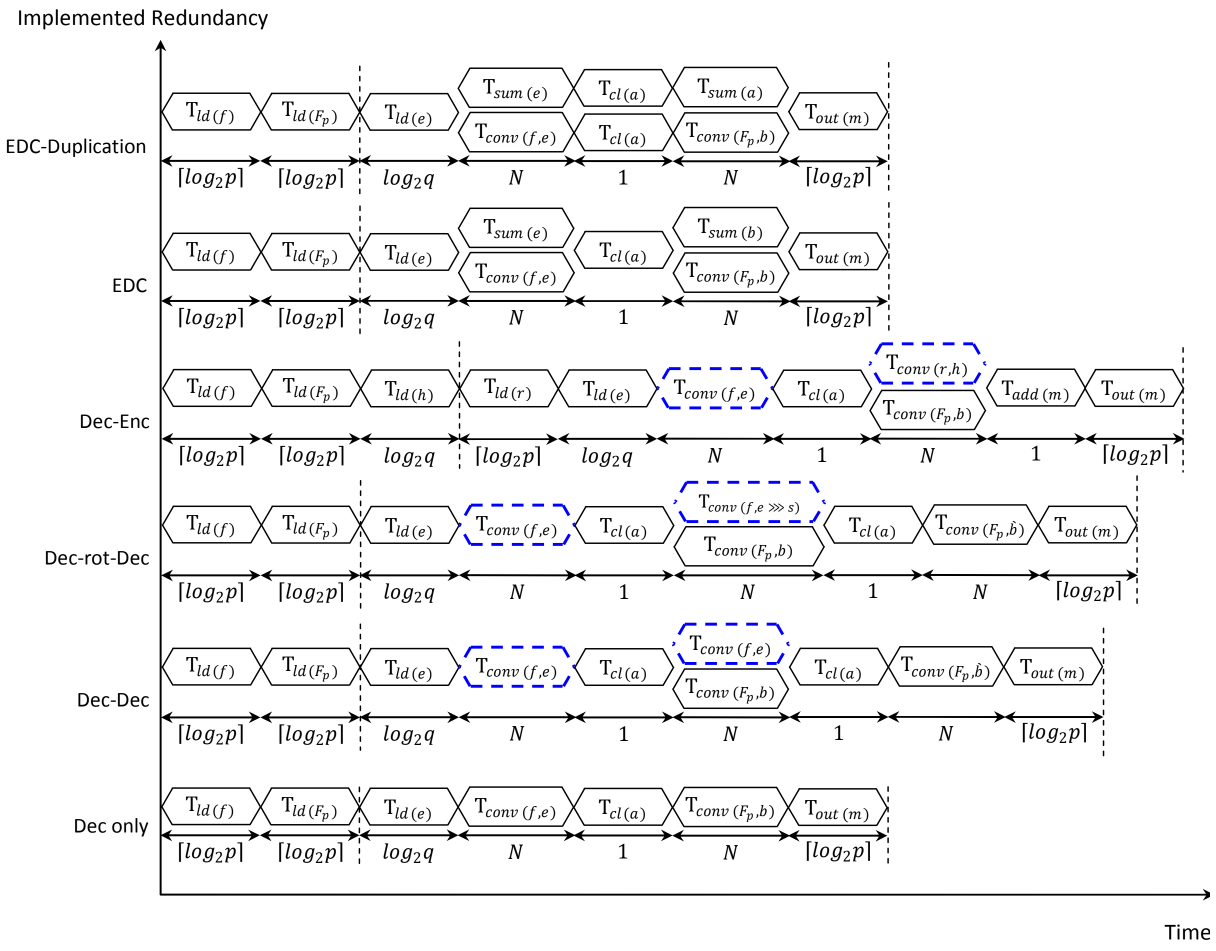


Figure 4.7: Time-line (not to scale) of operations performed by the proposed architectures. Operations performed using resource sharing are shown in dotted lines

one cannot simply convert the message to trinary and apply the raw NTRU encryption process. The message needs to be pre-processed before encryption and post-processed after encryption to protect against active attackers. The necessary processing is specified in IEEE Std 1363.1-2008, and discussed in [90].

Table 4.2: FPGA implementation results for the raw NTRUEncrypt decryption with parameters (N,p,q,df,dg,dr) = (167,3,128,61,20,18)

	Dec-only	Dec-Dec Figure 4.2	Dec-rot-Dec Figure 4.3	Dec-Enc Figure 4.4	EDC-only Figure 4.5	EDC-duplication Figure 4.6
# of Slices	3423	3636	4890	7853	4647	4260
# of Slice FFs	3919	4218	4234	5390	3579	4236
# of 4-input LUTs	5938	6094	8457	14399	8527	7651
# of IOBs	337	337	337	337	337	337
#Slices overhead (%)	-	6.22%	42.86%	129.42%	35.76%	24.45%
#Clk cycles per block	$2N + 10$	$3N + 11$	$3N + 11$	$2N + 13$	$2N + 10$	$2N + 10$
Throughput (K Blocks/sec)	160	107	107	159	160	160
Throughput (Mbps)	40.05	26.91	26.91	39.70	40.05	40.05
Throughput degradation (%)	-	32.81%	32.81%	0.86%	-	-
Type of detected faults	-	transient	transient, permanent	transient, permanent	transient, permanent	transient, permanent
Error detection probability	-	100%	100%	100%	99.22%	99.22%

4.3 Fault analysis of NTRUSign digital signature scheme

One main advantage of the NTRU family of cryptosystems (i.e., both NTRUEncrypt and NTRUSign) is that NTRU is smaller and faster than other public key cryptosystems. According to the performance benchmarks published on the NTRU company website (http://www.ntru.com/security-lab/), NTRU is about 5 to 200 times faster than RSA and ECC with similar security parameters. A recent comparison between hardware implementations of different digital signature schemes [53] shows that the power consumption of NTRUSign is significantly less than any other digital signature scheme published so far. These small footprints make NTRUsign ideal for handheld mobile devices, sensors, RFIDs, smartcards, and other resource constrained devices with limited computing resources. On the other hand, the typical deployment environment for these resource constrained devices makes it particularly susceptible to various forms of SCAs including fault analysis which has become a serious threat after cheap and low-tech methods of applying faults were presented [174]. It should also be noted that the limited resources and the extreme lightweight requirements make the implementation of various countermeasures against these attacks a more challenging task compared to other environments.

4.3.1 Description of NTRUSign

In this section, we briefly review some of the basic definitions and notations as defined in section 4.2.1. The key underlying structure of NTRUSign is the polynomial ring R, where N is a prime integer. The signatures and the messages use a polynomial ring R_q where the coefficients are taken mod q.

For a real number r, let $\lfloor r \rceil$ denote the closest integer to r. From the NTRUSign specifications [42], $\lfloor r \rceil$ is evaluated as $\lfloor r \rceil = \lfloor r + 0.5 \rfloor$ where $\lfloor \cdot \rfloor$ denotes the floor function of the enclosed argument. Similarly, if a is a polynomial with real coefficients, $\lfloor a \rceil$ denotes that this operation is applied to each coefficient in the polynomial a.

Let $a(x) = \sum_{i=0}^{N-1} a_i x^i$ be a polynomial in R. The centered norm of $a(x)$ is defined as the non-negative real number satisfying

$\|a(x)\|^2 = \sum_{i=0}^{N-1}(a_i - \mu_a)^2 = \sum_{i=0}^{N-1} a_i^2 - \frac{1}{N}(\sum_{i=0}^{N-1} a_i)^2$,

where $\mu_a = (1/N)\sum_{i=0}^{N-1} a_i$ is the average of the coefficients of $a(x)$. The centered norm in a direct sum module R^n is defined as the non-negative real number satisfying

$\|(a_0, a_1)\|^2 = \|a_0\|^2 + \|a_1\|^2$.

The original variant of the NTRUSign algorithm was proposed in [80]. A perturbation technique was later introduced in order to enhance the security of this scheme, particularly, against transcript attacks. Algorithm 2 shows this enhanced version of NTRUSign [42, 81, 87]. The original scheme can be seen as a special case of Algorithm 2 with $\mathcal{B} = 0$.

4.3.2 Proposed attack

To simplify our discussion, we first consider the NTRUSign with the "standard" NTRU lattice [81]. Steps 1-4 of the signing procedure in Algorithm 2 serve only to add perturbation δ to the original message m. Step 5 performs the actual signature operation, using the original NTRUSign scheme, on $m' = m + \delta$. Since δ is designed to be small enough, a valid signature on m' will be a valid signature on m. Our attack utilizes the fact that an attacker only needs to recover f_0, and consequently calculate g_0, in order to forge a signature that would pass the verification step in Algorithm 2. In other words, signatures generated by the setting $\mathcal{B} = 0$ in the signing algorithm, i.e., with no perturbation, will pass the verification algorithm even if the verifier assumes that it was generated with $\mathcal{B} > 0$. In fact, it will pass the verification step even if different sets of F and G are utilized as long as they satisfy the condition identified in step 2.ii in the key generation procedure of Algorithm 2.

The fault model in which we analyze NTRUSign is the one in which the attacker is assumed to be able to inject a transient fault in a small number of coefficients of the polynomial A or B in the signing algorithm (see step 6 in Figure 4.8) but cannot control the exact location of injected faults, i.e., cannot specify which polynomial coefficients are to be corrupted by the induced faults.

Algorithm 2 NTRUSign with perturbation [80–83]

```
Key Generation
1: INPUT: Integers($N$), $q$, $\mathcal{N}$, $\mathcal{B} \geq 0$, ($df$, $dg$ for binary polynomials), ($d$ for trinary polynomials), and the NTRU lattice type = "standard" or "transpose". In the above set of parameters, $N$ denotes the dimension of the polynomial ring used, i.e., polynomials are up to degree $N-1$. $\mathcal{N}$ denotes the norm-bound.
2: Generate $\mathcal{B}$ private lattice bases and one public lattice basis: Set $i$=$\mathcal{B}$. While $i \geq 0$ do:
    i. Randomly choose binary polynomials $f$, $g \in R$ with ($df$, $dg$) ones. For the parameter sets defined in [82, 83], choose trinary polynomials $f$, $g$ with $(d+1)$ +1s and $d$ -1s. $f$ needs to be invertible in $R_q$ in case of using NTRU lattice type = "standard" and $g$ needs to be invertible in $R_q$ in case of using NTRU lattice type = "transpose".
    ii. Find small polynomials $F, G \in R$ such that $f \star G - F \star g = q$.
    iii. Set $f_i = f$. If NTRU lattice type = "standard", set $f'_i = F_i$ and $h_i = f_i^{-1} \star g_i \bmod q$. If NTRU lattice type = "transpose", set $f'_i = g_i$ and $h_i = f_i^{-1} \star F_i \bmod q$. Set $i$=$i$-1.
3: PUBLIC OUTPUT: The input parameters and the public key $h = h_0$.
4: PRIVATE OUTPUT: The set of polynomial $\{f_i, f'_i, h_i\}$ for $i$=0..$\mathcal{B}$.
Signing
1: INPUT: A digital document $D \in \mathcal{D}$ (a digital document space) and the private key set $\{f_i, f'_i, h_i\}$ for $i = 0 \cdots \mathcal{B}$.
2: Set $r$=0.
3: Set $s$=0, $i$=0. Encode $r$ as a bit string. Set $m_0$=$H(D \parallel r)$, where "$\|$" denotes the concatenation. Set $m$=$m_0$.
4: Perturb the point using the private lattices: While $i \geq 1$:
    i. Calculate $A = \lfloor \frac{-f'_i \star m}{q} \rceil$, $B = \lfloor \frac{f_i \star m}{q} \rceil$, $s_i = A \star f_i + B \star f'_i \bmod q$.
    ii. Set $m = s_i \star h_i - h_{i-1} \bmod q$.
    iii. Set $s = s + s_i$. Set $i = i - 1$.
5: Sign the perturbed point using the lattice public key:
   Set $A = \lfloor \frac{-f'_0 \star m}{q} \rceil$, $B = \lfloor \frac{f_0 \star m}{q} \rceil$, $s_0 = A \star f_0 + B \star f'_0 \bmod q$, $s = s + s_0$.
6: Check the signature:
    i. Set $b = \|s, s \star h - m_0 \bmod q\|$.
    ii. If $b \geq \mathcal{N}$. Set $r = r + 1$ and go to step 3.
7: OUTPUT: The triplet ($D$, $r$, $s$).
Verifying
1: INPUT: A signed document($D$, $r$, $s$) and the public key $h$
2: Encode $r$ as a bit string. Set $m$=$H(D \parallel r)$.
3: Set $b = \|s, s \star h - m_0 \bmod q\|$.
4: OUTPUT: "valid" if $b < \mathcal{N}$, "invalid" otherwise.
```

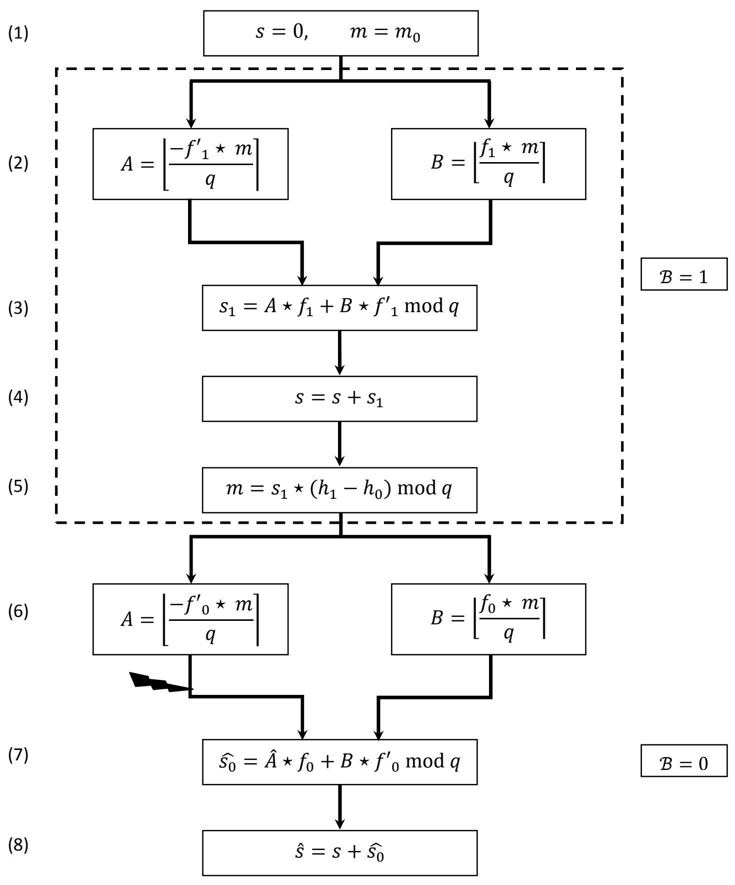

(1)
$s = 0, \quad m = m_0$
(2)
$A = \left\lfloor \frac{-f'_1 \star m}{q} \right\rceil$
$B = \left\lfloor \frac{f_1 \star m}{q} \right\rceil$
(3)
$s_1 = A \star f_1 + B \star f'_1 \bmod q$
$\mathcal{B} = 1$
(4)
$s = s + s_1$
(5)
$m = s_1 \star (h_1 - h_0) \bmod q$
(6)
$A = \left\lfloor \frac{-f'_0 \star m}{q} \right\rceil$
$B = \left\lfloor \frac{f_0 \star m}{q} \right\rceil$
(7)
$\widehat{s_0} = \hat{A} \star f_0 + B \star f'_0 \bmod q$
$\mathcal{B} = 0$
(8)
$\hat{s} = s + \widehat{s_0}$

Figure 4.8: Fault injection in the NTRUSign algorithm

If the fault injection process guarantees that number of faulted polynomial coefficients is upper bounded by t, then the resulting faulty signature is given by

$$\widehat{s} = s_1 + \widehat{s_0},$$

where

$$\widehat{s_0} = \widehat{A} \star f_0 + B \star f'_0 \bmod q,$$

$\widehat{A} = A + \epsilon \bmod q$, and $\epsilon = \sum_{j=1}^{t} \epsilon_{i_j} x^{i_j} \in R_q, 0 \leq i_j < N$, denotes the resulting error polynomial with t non zero coefficients in the locations corresponding to the injected faults. Thus, the attacker can calculate the difference between the faulty signature and the correct one as follows:

$$\begin{aligned}\Delta s = \widehat{s} - s &= (s_1 + \widehat{s_0}) - (s_1 + s_0) \\ &= \widehat{s_0} - s_0 = \epsilon \star f_0 \bmod q.\end{aligned}$$

The attacker can obtain the secret key f_0 as

$$f_0 \quad = \epsilon^{-1} \star \Delta s \bmod q$$

by exhaustively trying all possible $(q^t - 1)\binom{N}{t} = \mathcal{O}((qN)^t)$ values for $\epsilon(x)$ with the pre-specified small number of non zero coefficients. Calculating the multiplicative inverse of ϵ in R_q can be done efficiently using the Newton iteration method [170]. Then, the attacker calculates

$$g_0 \quad = f_0 \star h \bmod q,$$

The attacker can verify the correctness of the obtained solution by running the verification algorithm on signatures produced by these recovered keys.

The success of the above attack requires that ϵ is invertible in R_q. For $q = p^l$ is a power of a prime p and N is a prime number, let $n \geq 1$ be the smallest integer such that $p^n \equiv 1 \bmod N$. The probability that $\epsilon_q(x) \in R_q$ is invertible is given by [169]

$$\left(1 - \frac{1}{p}\right)\left(1 - \frac{1}{p^n}\right)^{(N-1)/n}$$

which is approximately equal to $(1 - \frac{1}{p})$ for the typical choice of the parameters of the NTRUSign algorithm. For example, for $N = 251$ and $q = 128$ [81], we have p=2 and n=50. Consequently, the probability that an element in R_q is invertible $\approx \frac{1}{2}$.

Our analysis above ignores the fact that the signature procedure might be implemented in such a way that the attacker may not have access to the faulty signature if it does not pass the norm-bound checking (step 6 in the signing procedure in Algorithm 2). While the NTRUsign designers have previously suggested that, depending on the application, this step might be skipped for NTRUSign with no perturbation in order to improve the efficiency of the signing process (see section 6.4 in [80]), this is not the case for NTRUSign with $\mathcal{B} > 0$.

Table 4.3 and Table 4.4 show the percentage of cases in which the faulted signatures pass this check for 10 random messages and 1000 fault injections for each message for the parameter set (N, q, df, dg, $\mathcal{B}$, "type", $\mathcal{N}$) =(251, 128, 73, 71, $\mathcal{B}$, standard, 310) [81] for $\mathcal{B} = 0$ and $\mathcal{B} = 1$, respectively. In the above set of parameters, N denotes the dimension of the polynomial ring used, i.e., polynomials are up to degree $N - 1$, $\mathcal{N}$ denotes the norm-bound, df, dg denote the Hamming weight of the binary polynomials f and g used in the key generation step of the algorithm, $\mathcal{B} = 0$ corresponds to the system with no perturbation and $\mathcal{B} = 1$ corresponds to the system with perturbation (see Algorithm 2 for further clarification of these parameters.) The experimental results were obtained by simulating the process of fault injection in our MAPLE software implementation of NTRUSign.

Table 4.3: Experimental results for NTRUSign with (N, q, df, dg, $\mathcal{B}$, "type", $\mathcal{N}$) =(251, 128, 73, 71, 0, standard, 310)

t	ϵ is non-invertible	ϵ is invertible		probability of successful fault injection
		fail	pass	
1	4901	3153	1946	0.1946
2	4917	4563	520	0.0520
3	5093	4789	118	0.0118

Table 4.4: Experimental results for NTRUSign with (N, q, df, dg, $\mathcal{B}$, "type", $\mathcal{N}$) =(251, 128, 73, 71, 1, standard, 310)

t	ϵ is non-invertible	ϵ is invertible		probability of successful fault injection
		fail	pass	
1	4948	4051	1001	0.1001
2	4990	4857	153	0.0153
3	5016	4962	22	0.0022

From Table 4.3, if the number of faulted coefficients in A is 1, then about $1/0.195 \approx 5$ fault injections are required before the attacker is able to access the faculty signature which

Table 4.5: Experimental results for NTRUSign with (N, q, df, dg, $\mathcal{B}$, "type", $\mathcal{N}$) =(251, 128, 73, 71, 0, transpose, 310)

t	ϵ is non-invertible	ϵ is invertible		probability of successful fault injection
		fail	pass	
1	4944	4437	619	0.0619
2	5064	4893	43	0.0043
3	4089	5008	3	0.0003

Table 4.6: Experimental results for NTRUSign with (N, q, df, dg, $\mathcal{B}$, "type", $\mathcal{N}$) =(251, 128, 73, 71, 1, transpose, 310)

t	ϵ is non-invertible	ϵ is invertible		probability of successful fault injection
		fail	pass	
1	4948	4775	277	0.0277
2	5026	4965	9	0.0009
3	5049	5051	0	0.0000

corresponds to an invertible error polynomial $\epsilon \in R_q$. It is also clear that the required number of fault injections increases with t. If, for practical reasons, inducing such a large number of transient fault injections is not possible, then a second order fault attack [111] can be utilized in order to reduce the number of required fault injections. In this case, the attacker induces a second fault in step 6 in the signing procedure of Algorithm 2 to skip this norm-bound check.

Similar analysis follows for type="transpose", i.e., when the signing process is performed using the NTRU transpose lattice. However, in this case, g_0 cannot be recovered from the knowledge of the public key and f_0. Thus the attacker has to repeat the attack by injecting another fault in the computation of the polynomial B (step 6 in Figure 4.8). Tables 4.5 and Table 4.6 show the corresponding experimental results for NTRUSign with the transpose lattice.

The above set of experiments corresponds to the security parameters defined in the Efficient Embedded Security Standards (EESS) standard [42] with a claimed security level equivalent to RSA-1024 which is roughly equivalent in strength to 80-bit symmetric keys [42]. In order to test the effect of changing the algorithm claimed theoretical security level on the per-

Table 4.7: Experimental results for NTRUSign with the set of parameters in [82, 83]

parameters						probability of successful fault injection		
k	N	d	q	β	$\mathcal{N}$	$t = 1$	$t = 2$	$t = 3$
80	157	29	256	0.38407	150.02	0.0303	0.0012	0.0000
112	197	28	256	0.51492	206.91	0.0241	0.0011	0.0001
128	223	32	256	0.65515	277.52	0.0266	0.0012	0.0000
160	263	45	512	0.31583	276.53	0.0188	0.0007	0.0000
192	313	50	512	0.40600	384.41	0.0221	0.0008	0.0000
256	349	75	512	0.18543	368.62	0.0350	0.0031	0.0000

formance of our attack when the attacker cannot bypass the signature verification step, we performed another set of experiments on NTRUSign with the new set of security parameters defined in [82, 83]. For this version of NTRUSign, the centered norm computation in step 3 of the verification procedure in Algorithm 2 is defined as $||s, s \star h - m_0||^2 = ||s||^2 + \beta^2 ||s \star h - m_0||^2$ where, as shown in Table 4.7, different values of β are defined for each set of the security parameters (see also section 2.3 in [82]). It should be noted that these parameters are only applicable to the NTRUsign with the transpose NTRU lattice. Furthermore, in this version of NTRUSign, the polynomials f and g have trinary coefficients $\in \{+1, -1, 0\}$ with $(d+1)$ +1s and (d) -1s while in the EESS version of NTRUSign, f and g are binary polynomials with Hamming weight df and dg, respectively.

The first column in Table 4.7 refers to the claimed security level, k, in bits. One interesting observation, that follows from the experimental results in Table 4.7, is that increasing the security parameter, which basically reflects the resistance against the best known lattice based attacks, does not necessarily improve the resistance against our attack. It should also be noted that if the attacker is able to skip the norm-bound checking step, then the probability of successful fault insertion $\approx 1 - \frac{1}{p}$, does not vary with the choice of the security parameters since $p = 2$ for all of them. On the other hand, as depicted in the table, in the case where the attacker cannot skip the norm-bound checking step, the precision of the fault injection process, i.e., the ability of the attacker to bound the number of faulted polynomial coefficients, t, has a large influence on the success probability of the attack and also keeps the computation steps required for the attack, $O((qN)^t)$, in the practical range.

4.3.3 Proposed countermeasures

To secure cryptographic devices against fault analysis, proper countermeasures have to be applied. Generally, these countermeasures detect temporal or permanent faults which happen in the cryptosystem, and immediately, disable the device output or rest all the output bits to 0s. As a result, the attacker is prevented from observing the output of the faulty cryptographic computations and hence the vulnerability of the cryptosystem to these attacks can be alleviated. Several approaches of fault detection techniques have been investigated. These techniques include error detecting codes and redundancy-based techniques [117].

Many ideas have been proposed to efficiently secure CRT based RSA signature computations against fault attacks. In [165], Shamir used a redundant way to compute the arguments used in the CRT computation of the RSA signature and checked their correctness before RSA combination. Shamir's countermeasure has two drawbacks. It requires the secret exponent which is not needed for the CRT computation of the signature and its error detection technique is based on decisional tests that should be avoided since they can be bypassed by higher order fault attacks which were introduced by Kim and Quisquater in [111] where they were able to practically break the first-order countermeasures for the RSA cryptosystem. The main idea is based on inducing the first fault during one of the exponentiation and then inducing a second fault to skip the error checking routine in the RSA-CRT scheme. Yen *et al.* [194] noted that fault injection on status register flags can bypass conditional checks in countermeasures. Hence, they introduced the concept of infective computation which aims at infecting the resulting signature, i.e., rendering it unusable for the attacker in the case where a fault is injected in one of the two exponentiations. Several subsequent countermeasures employed this infective computation approach [27, 28, 66, 96].

Unlike the case of symmetric key ciphers where parity based error detection codes are somewhat straightforward to derive for the intermediate computation steps of the cipher, this does not seem to be the case for NTRUsign because of the nonlinearity of the operations involved, especially the rounding operation, $\lfloor \cdot \rceil$, in step 2 and step 6 of Figure 4.8. On the other hand, the algebraic properties of the polynomials convolutional product defined by Equation (4.1), allow us develop a redundancy-based fault detection technique which detects both transient and permanent faults. We have also developed another, independent, technique to defend against second order fault analysis where the attacker is able to introduce another fault to skip the fault detection check.

Recomputing with cyclically shifted messages

Let $a^{(r)}(x) \in R$ denote an r-cyclically shifted version of $a(x) = \sum_i a_i x^i$, $0 < r \leq N-1$. The coefficients of $a^{(r)}(x)$ are obtained by rotating the coefficients of $a(x)$ by r positions. Thus we have $a^{(r)}(x) = \sum_{i=0}^{N-1} a_{i+r \pmod N} x^i$. Let $c(x) = a(x) \star b(x)$. Then we have $a^{(r)}(x) \star b(x) = c^{(r)}(x)$ where $c_i^{(r)} = c_{i+r \pmod N}$.

Consequently, the coefficients of $\lfloor \frac{a^{(r)}(x) \star b(x)}{q} \rceil$ correspond to r-cyclically shifted version of the coefficients of $\lfloor \frac{a(x) \star b(x)}{q} \rceil$. This fact can be utilized to introduce either spatial redundancy or temporal redundancy where the redundant computation is performed using a rotated version of m and the two signatures are compared as shown in Figure 4.9. If the result of the two operations do not match, up to the chosen rotation r, then the output of the device is disabled in order to prevent the attacker from accessing any information from the faulty signature. Note that this technique has an advantage over straightforward application of temporal redundancy, i.e., evaluating the signature on the same message for two times and comparing the result, since the latter approach does not protect against permanent faults. To speed up computation in the case of utilizing the temporal redundancy option, the added redundancy can be utilized at the level of smaller building blocks, i.e., during the computation of A and B in step 6 of Figure 4.8. However, applying this approach at the algorithm level has the advantage of being able to detect all faults, not necessarily the ones resulting from injecting errors at this particular step.

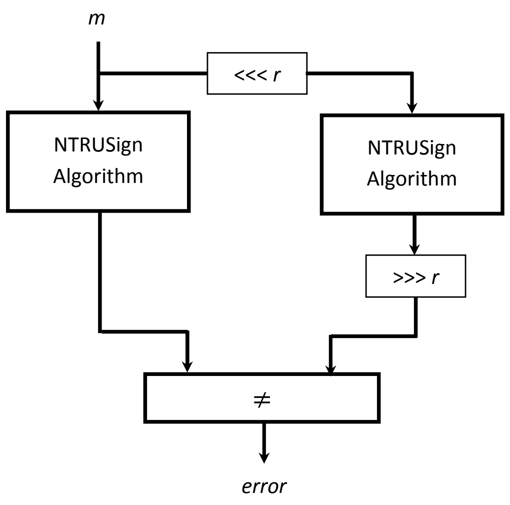


Figure 4.9: Fault detection by recomputing with cyclically shifted messages

Defending against second order fault analysis attacks

In this section, we introduce another countermeasure which avoids decision test (i.e., the error checking that is performed by comparing the signature with the one resulting from the redundant module such as the one shown in Figure 4.9) and consequently protects against sophisticated attackers who are able to induce higher order faults that skip fault detection operations. Our approach is inspired by the fault infective computation technique proposed by Yen *et al.* [194] to defend against fault attacks on the RSA digital signature scheme. The following two Lemmas will be used to prove the correctness of our approach

Lemma 4.5 *Let* $w = \frac{z}{q} - \lfloor \frac{z}{q} \rceil$*, where* z *and* q *are two integers. If an error occurs during the computation of* $\lfloor \frac{z}{q} \rceil$*, then we have* $\lfloor w \rceil \neq 0$*.*

Proof: Let w and w' denote the results of correct and incorrect computations, respectively. From to the definition of the round function $\lfloor \cdot \rceil$, $w \in [-\frac{1}{2}, \frac{1}{2})$. Since erroneous $\lfloor \frac{z}{q} \rceil$ differs from correct $\lfloor \frac{z}{q} \rceil$ by an integer, the difference between w and w' is given by $|w - w'| \geq 1$. Thus, $w' \notin [-\frac{1}{2}, \frac{1}{2})$ and consequently $\lfloor w' \rceil \neq 0$, which proves the Lemma. ■

Lemma 4.6 *If z_1, z_2 are two non-negative integers, then*

$$\lceil \frac{z_1 + z_2}{(z_1 + z_2 + 1)} \rceil = \begin{cases} 0 & \textit{if } z_1 = z_2 = 0 \\ 1 & \textit{otherwise}. \end{cases}$$

Proof: Let $z = z_1 + z_2$. It is clear that $\frac{z}{z+1} = 0$ for $z = 0$ and $\frac{1}{2} \leq \frac{z}{z+1} < 1$ for $z \geq 1$. It follows that $\lceil \frac{z}{z+1} \rceil = 0$ if $z = 0$ and $\lceil \frac{z}{z+1} \rceil = 1$ otherwise. The Lemma follows by noting that $z = 0$ if and only if $z_1 = z_2 = 0$. ∎

To defend against higher order fault analysis of NTRUSign, step 5 in the signing procedure of Algorithm 2 is replaced by the following steps

1. Calculate $A = \lfloor \frac{-f_0' \star m}{q} \rceil$, $B = \lfloor \frac{f_0 \star m}{q} \rceil$,
2. Calculate $AA = \lfloor \frac{-f_0' \star m}{q} - A \rceil$, $BB = \lfloor \frac{f_0 \star m}{q} - B \rceil$,
3. Set $s_0 = A \star f_0 + B \star f_0' \bmod q$,
4. Set $z_1 = \sum_{i=0}^{N-1} |AA_i|$, $z_2 = \sum_{i=0}^{N-1} |BB_i|$, where $|\cdot|$ denotes the absolute value of the enclosed coefficient.
5. Set $s = q^{\lceil \frac{z_1+z_2}{z_1+z_2+1} \rceil}(s + s_0) \bmod q$, where $\lceil \cdot \rceil$ denotes the ceiling operation.

From Lemma 4.5, the coefficients of the polynomials AA and BB will all be equal to zeros if and only if no errors occur during the computation of this step. Consequently, and by utilizing Lemma 4.6, in case of no errors, step 5 above evaluates

$$s = q^{\lceil \frac{z_1+z_2}{z_1+z_2+1} \rceil}(s + s_0) \bmod q = q^0 \times (s + s_0) \bmod q = (s + s_0) \bmod q$$

which corresponds to the correct output of Algorithm 2. On the other hand, if an error occurs, step (5) above results in

$$s = q^{\lceil \frac{z_1+z_2}{z_1+z_2+1} \rceil}(s + s_0) \bmod q = q \times (s + s_0) \bmod q = 0,$$

which prevents the attacker from recovering any useful information about the secret key. Note that we raise q to the power of $\lceil \frac{z_1+z_2}{z_1+z_2+1} \rceil \in \{0, 1\}$ instead of raising it to the power of $z_1 + z_2$ in order to avoid the unnecessarily computational complexity of performing the exponentiation operation with a large exponent.

4.4 Conclusion

One difficulty with analyzing the NTRU encryption algorithm is its large number of variants. Different choices of parameters can dramatically change the security of NTRU and attacks against a specific instantiation may not succeed against other instantiations.

In this chapter, we presented a fault analysis attack against the original variant of the NTRU encryption algorithm. Our attack does not work against more recent variants where $f = 1 + pF$ for ternary or binary polynomial F since the bijection between F_p and f does not exist in these variants. Extending our attack to constructions with padding is a challenging research problem. Another interesting research direction is to consider how to exploit the wraps in the centering algorithm by this kind of fault attacks. We also presented different techniques for strengthening the resistance of NTRUEncrypt hardware implementations against fault attacks. We provided a comparison between these different techniques in terms of their error detection capabilities as well as area and throughput overheads. We also provided FPGA implementation results for these approaches.

As shown in Table 4.2, EDC based approaches can provide error detection up to $1 - \frac{1}{q} = 99.22\%$ for the implemented system parameters. While this error detection capability might be suitable for protecting implementations against non malicious faults, in security sensitive applications where transient faults can be maliciously injected, we believe that this level of protection is not enough. Fault attacks are getting better and the number of required faults are getting smaller. Thus, from Table 4.2, in these situation the system designer can choose between the decryption-rotate-decryption approach (with throughput reduction by a factor of

$\approx \frac{3N}{2N} = 1.5$ and a relatively small area overhead) and the decryption-encryption option with a relatively large area overhead but practically very little impact on throughput. Developing the corresponding countermeasures for other variants of NTRUEncrypt with $f = 1 + pF_p$, or constructions with padding, is an interesting research directions.

One interesting observation that follows from our attack on NTRUSign is that the perturbation process, which was introduced to improve the security of NTRUSign towards previous attacks, does not improve the resistance of NTRUSign against this kind of fault attacks if the norm-bound checking step can be skipped by the attacker. Another observation is that using NTRUSign with a larger-security parameter, while improving the resistance against lattice based attacks, does not necessarily improve the resistance against fault analysis attacks especially in the cases where the attacker is able to precisely corrupt a small number of coefficients and is able to skip the norm-bound checking step. Thus, when NTRUSign is deployed in environments that are susceptible to this class of fault analysis attacks, implementing fault analysis countermeasures, such as the two countermeasures developed in this work, is necessary to protect the security of the system even if the largest-security parameters are utilized.

Chapter 5

Application of Scan-based SCAs on NTRUEncrypt Cryptosystems

5.1 Introduction

Scan-based Design-For-Test (DFT) [34] is a popular technique for validating the functionality of integrated circuits during fabrication and providing on-chip debugging capabilities in the field. When using this approach, all flip-flops in the circuit under test are tied to one or more scan chain that enable scanning out their states using the Joint Test Action Group (JTAG) boundary scan interface [93]. A JTAG interface is a special serial interface with the following pins: TDI (Test Data In), TDO (Test Data Out), TCK (Test Clock), TMS (Test Mode Select), and optional TRST (Test Reset). TMS selects between normal mode and test mode. TRST is the reset signal for the test controller. During testing, test vectors can be scanned in via the TDI pin, and flip-flops states can be scanned out via the TDO pin. As shown in Figure 5.1, a scan flip-flop is a flip-flop with a Multiplexer (MUX) at the input. In normal mode, it works like a normal flip-flop. In test mode, as shown in Figure 5.2, all scanned flip-flops are disconnected from the combinational circuit and connected to each other in a scan chain where their contents can be scanned in and out.

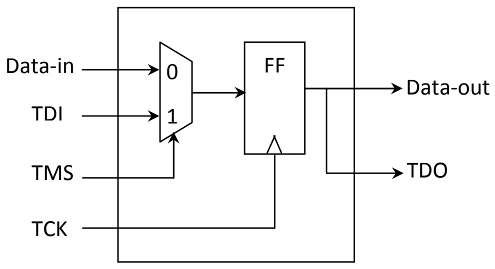


Figure 5.1: An illustration for a scan flip-flop (SFF)

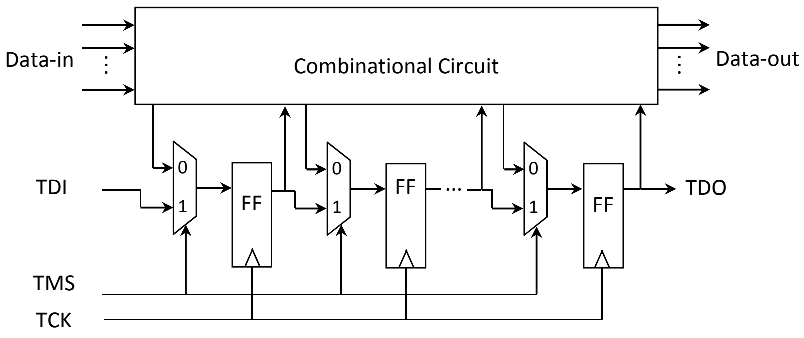


Figure 5.2: An illustration for a scan chain

While scan-based DFT improves the quality of testing, it also introduces a powerful side channel attacks against hardware implementations of cryptographic devices that utilize this technique. Despite the fact that the internal structure of the scan chain is usually not known to attackers, exploiting the information obtained from analyzing the scanned data allows cryptanalysts to ascertain this structure and retrieve the secret key from the cryptographic hardware devices that implement various cryptographic algorithms such as DES [192], AES [140, 193], RSA [139], ECC [141], and stream ciphers [1, 120].

As mentioned in the previous chapters, the NTRUEncrypt algorithm [84, 87] is a parameterized family of lattice-based public key cryptosystems. In the past few years, the security of NTRUEncrypt against mathematical attacks had been analyzed by many researchers (e.g., [95, 142]). Different classes of SCAs against NTRUEncrypt and their countermeasures were also considered [13, 105, 119, 171].

In this chapter, we present a scan-based SCA against a hardware implementations of

NTRUEncrypt. In general, all scan-based SCAs can be viewed as a kind of differential crypt-analysis where attackers take advantage of the scan chains to observe the bit changes between pairs of chosen plaintexts/ciphertexts and consequently, identify the secret keys. More precisely, scan-based attacks can be classified into two classes [167]: Constant Based Attacks (CBAs) and Fixed-Hamming-Distance-based Attacks (FHDAs). CBAs take advantages of the fact that in the encryption/decryption process, the contents of some special registers are independent of the input. By using several different inputs and scanning out the contents at different times of the cryptographic operation, the internal registers of the cryptographic device can be easily identified. Then, by setting these registers to specific states through scan operations, the complexity of the secret key recovery can be reduced. In FHDAs, several pairs of relevant plaintexts/ciphertexts are applied and then, for each pair, the number of different bits in the output are counted to recover the secret key.

Our attack can be seen as a type of FHDAs where we focus on determining the scan chain structure of the polynomial multiplication circuit in the decryption algorithm and then utilize the Hamming weight information of some particular registers, which can be obtained via the JTAG interface, to efficiently retrieve all the coefficients of the secret key polynomial. Another contribution of this chapter is showing that existing NTRUEncrypt hardware architectures that target low area implementations are easier to break using scan-based attacks compared to architectures that target high speed implementations because of the different methods used to perform the convolution multiplication in the decryption process in both approaches.

5.2 Hardware implementation options for NTRUEncrypt

Throughout the rest of the chapter, we focus on the NTRUEncrypt algorithm with the widely used parameters: $p = 3$ and q in the form of 2^n [84]. We assume that the attacker has access to the high level timing diagram of the target hardware implementation and that the secret key is stored in secure memory that cannot be accessed through the scan chain. We also assume that the attacker has direct access to the scan chains via the JTAG port or by breaking open the package and directly probing the buried JTAG ports [34].

From the description of the encryption and decryption operations in section 4.2.1, it is clear that the most time/area consuming step in both operations is the convolution multiplication required to compute $r(x) \star h(x)$ and $f(x) \star e(x)$ during the encryption and decryption operations, respectively.

While there are several hardware implementations for NTRUEncrypt (e.g., see [12, 13, 15, 110, 119, 130]), the majority of these implementations focus on optimizing either the time or the area required by theses convolution operations. In what follows, we focus on the decryption process because it is the only relevant part for our cryptanalysis since the encryption module does not contain any secret information.

For low area and low power implementations, which are viable for resource constrained applications such as RFIDs and sensor nodes, the convolution multiplication, $a = f \star e \bmod q$, is usually performed in approximately N^2 clock cycles [12, 110] as shown in Figure 5.3. In this case, the computation for the coefficients of the polynomial a are done sequentially where the coefficient a_i, of size $\log_2(q)$, is obtained after $(i+1) \times N$ clock cycles, $0 \leq i < N$. For example, a_0 is calculated by accumulating (mod q) the results of serially multiplying $e_0 \times f_0, e_{N-1} \times f_1, e_{N-2} \times f_2, \cdots, e_1 f_{N-1}$ into a register initialized to zero where each multiply-and-add step is performed in one clock cycle.

For this kind of implementations, scan-based SCAs can be used to recover the secret information in a straightforward way as follows: first, the attacker determines the locations of the flip flops of the ciphertext register e in the scan chain output. This is done by analyzing the bit differences between the scan chain outputs corresponding to the all zeroes ciphertext and the $N \times log_2(q)$ scan chain outputs corresponding to the ciphertext blocks with a (bit) Hamming weight equal to one. This step is performed right after the ciphertext loading operation.

In the next step, the attacker loads a ciphertext e in which all coefficients are set to 1. Since the value of a_0 is computed sequentially, the intermediate values of a_0 after the t^{th} clock cycle of the convolution multiplication step are given by

$$\begin{aligned} a_0^{(t)} &= \textstyle\sum_{j=0}^{t} f_j \times e_{(N-j) \bmod N} \bmod q \\ &= \textstyle\sum_{j=0}^{t} f_j \bmod q, 0 \le t < N. \end{aligned}$$

Consequently, the attacker can recover the value of the secret key by scanning out the intermediate values of a_0. Note that the attacker is able to determine the locations of the $log_2(q)$ bits corresponding to a_0. This is because the location of the bits corresponding to e are already determined in the previous step and the only bits of the scan chain output that change in this step would correspond to the a_0 bits. This holds because the bits corresponding to $a_i, 0 < i < N$ remain zeroes throughout the computation of a_0.

	e_0	e_1	e_2	...	e_{N-1}
$\star$	f_0	f_1	f_2	...	f_{N-1}
+	$e_0 \times f_0$	$e_1 \times f_0$	$e_2 \times f_0$	...	$e_{N-1} \times f_0$
+	$e_{N-1} \times f_1$	$e_0 \times f_1$	$e_2 \times f_1$	...	$e_{N-2} \times f_1$
+	$e_{N-2} \times f_2$	$e_{N-1} \times f_2$	$e_0 \times f_2$	...	$e_{N-3} \times f_2$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
+	$e_1 \times f_{N-1}$	$e_2 \times f_{N-1}$	$e_3 \times f_{N-1}$	...	$e_0 \times f_{N-1}$
	a_0	a_1	a_2	...	a_{N-1}

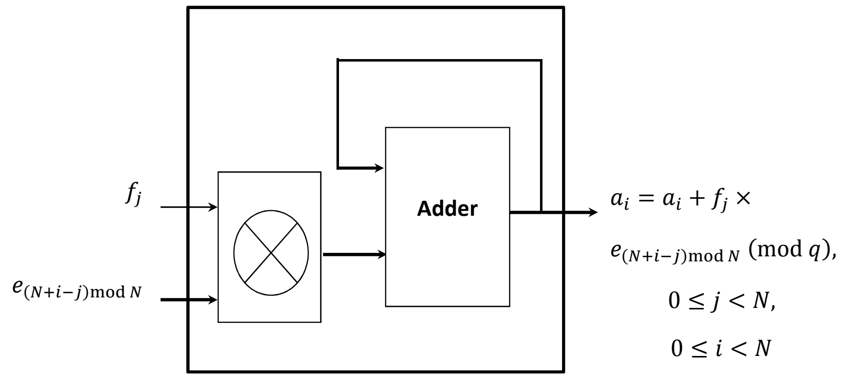


Figure 5.3: A typical low area implementation of the convolution multiplication $(f(x) \star e(x))$ in N^2 clock cycles. For each $0 \le i < N$, j varies from 0 to $N-1$.

In hardware implementations of NTRUEncrypt that targets higher speed applications (e.g., see [13, 119, 130]), instead of storing all the coefficients of f, the locations of the non-zero $(2 \times d_f - 1)$ coefficients are stored. This allows the convolution multiplication to be performed, as shown in Algorithm 3 [15], in $2 \times d_f - 1$ steps instead of N^2 steps (see Figure 5.4). Since the number of non-zero coefficients, $2 \times d_f - 1$, is typically much smaller than N, this implementation leads to a faster decryption.

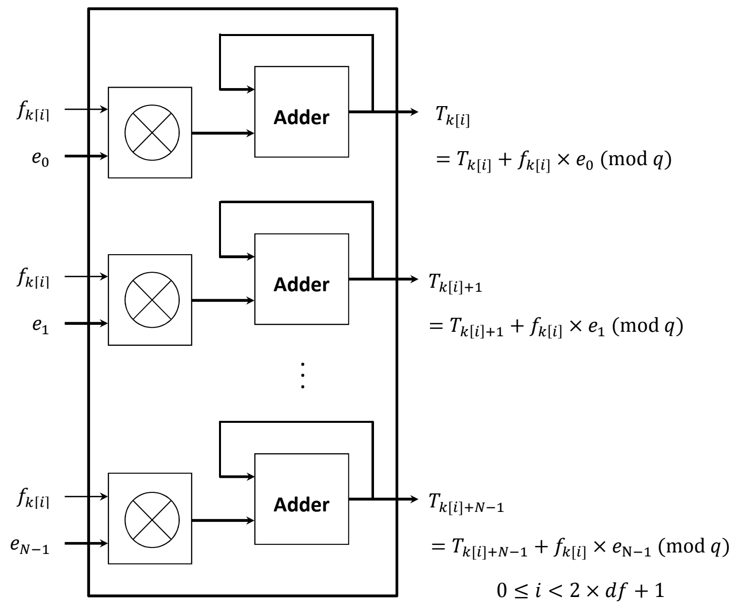

$f_{k[i]}$
e_0
Adder
$T_{k[i]}$
$= T_{k[i]} + f_{k[i]} \times e_0 \pmod q$
$f_{k[i]}$
e_1
Adder
$T_{k[i]+1}$
$= T_{k[i]+1} + f_{k[i]} \times e_1 \pmod q$
$f_{k[i]}$
e_{N-1}
Adder
$T_{k[i]+N-1}$
$= T_{k[i]+N-1} + f_{k[i]} \times e_{N-1} \pmod q$
$0 \le i < 2 \times df + 1$

Figure 5.4: The convolution multiplication between the polynomials $f(x)$ and $e(x)$ in $2 \times d_f - 1$ clock cycles

Determining the corresponding locations of the register e in the scan chain output can be performed using the same approach above, i.e., by loading different ciphertexts with a Hamming weight equal to one and analyzing the output differences in the scan chain output corresponding to these ciphertexts and the one corresponding to the all zero ciphertext. On the other hand, determining the coefficients of the secret key polynomial, f, requires some deeper analysis which will be developed though the rest of this chapter.

Algorithm 3 Fast Convolution Algorithm [15]

```
1: INPUT: An array $k$ of $d_f$+1 locations for +1 and $d_f$ locations for -1 representing $f$; $e$; $N$: the size of $f$ and $e$
2: OUTPUT: $T = f \star e \bmod q$.
3: for $0 \leq l < 2N$ do
4:     $T_l \longleftarrow 0$
5: end for
6: for $1 \leq t \leq d_f + 1$ do
7:     for $0 \leq l < N$ do
8:         $T_{l+k[t]} \longleftarrow T_{l+k[t]} + e_l$
9:     end for
10: end for
11: for $d_f + 2 \leq t \leq 2d_f + 1$ do
12:     for $0 \leq l < N$ do
13:         $T_{l+k[t]} \longleftarrow T_{l+k[t]} - e_l$
14:     end for
15: end for
16: for $0 \leq l < N$ do
17:     $T_l \longleftarrow (T_l + T_{l+N}) \bmod q$
18: end for
```

5.3 The proposed scan-based attack

As mentioned above, when the convolution multiplication is implemented as shown in Figure 5.4, a straightforward application of scan-based attacks to recover the full scan chain structure will not work because of the large number of flip-flops connected in the scan chain. Instead, in our attack, we focus on determining the relevant flip-flops in the scan chain structure of the polynomial multiplication circuit. The main idea of our attack is to distinguish the register T into two parts: T_R and T_L (the relevance of both parts will be explained below.) Then by single stepping through the $2d_f + 1$ clock cycles of the convolution multiplication, and by recording the Hamming weight of T_R and T_L in each clock, the attacker can construct a system of linear equations, with the potential positions of the non-zero elements of the secret key as unknowns. This set of equations can be solved to obtain a set of possible keys. Then, the correct key can be determined by verifying the correctness of the decryption operation for a known plaintext.

5.3.1 Summary of the attack

Conceptually, the steps of the attack can be summarized as follows:

1. Reset the chip and run it in the normal decryption mode to load an all zero ciphertext into the register e. Resetting the circuit allows the attacker to rest all flip-flops (including those that belong to the control circuit) into the same initial state before each attack step. This is necessary in order to allow the attacker to calculate the differences in the Hamming weight of the target registers before and after each attack step.

2. Switch to test mode and scan out the bit stream pattern.

3. Repeat steps 1 and 2 using all the $N \times log_2(q)$ ciphertexts with a (bit) Hamming weight equal to one and compare the output differences in the scan chain output corresponding to these ciphertexts with the one corresponding to the all zeroes ciphertext. At the end of this step, the attacker is able to determine the locations corresponding to the ciphertext register, e, in the scan chain.

4. Load the chip with a ciphertext of all 1s in normal mode and clock the system one time to evaluate $T = f \times e$ as shown in Algorithm 3.

5. Switch to test mode and scan out the bit stream pattern. In this case, the register T can be distinguished into two parts: T_R which contains all 1s and T_L which contains 0s. Note that while the attacker can identify the group of bits that belong to each of these two registers, the attacker cannot determine the exact location of these bits within the registers.

6. Clock the system in normal mode

7. Switch to test mode and scan out the bit stream to calculate the Hamming weight of the registers T_R and T_L.

8. Repeat steps 6-7 for $(2 \times d_f)$ times and record the Hamming weights of the registers T_R and T_L.

9. Use the Hamming weights obtained above as an input to Algorithms 4 and 5 to form a set of linear equations which can be solved to obtain the set of possible keys.

10. Determine the correct key by verifying the correctness of the decryption operation (using any arbitrary known plaintext-ciphertext pair obtained using the public key encryption process) for each of the keys obtained in step 9 above.

5.3.2 Recovering the secret key

For $A = [a_0, a_1, ..., a_{N-1}]$ and $B = [b_0, b_1, ..., b_{N-1}]$, $a_i, b_i \in \mathbb{Z}_q$, $0 \leq i < N$, the following notation will be used throughout the rest of the chapter.

- $A_{i..j}$ denotes the vector $[a_i, a_{i+1}, ..., a_j]$ of length $j - i + 1$, $0 \leq i < j$.

- $\overset{-}{A}_{i..j}$ denotes the vector $[(a_i + (q-1)) \bmod q, (a_{i+1} + (q-1)) \bmod q, ..., (a_j + (q-1)) \bmod q]$=$[(a_i - 1) \bmod q, (a_{i+1} - 1) \bmod q, ..., (a_j - 1) \bmod q]$, $0 \leq i < j$.

- $\overset{+}{A}_{i..j}$ denotes the vector $[(a_i - (q-1)) \bmod q, (a_{i+1} - (q-1)) \bmod q, ..., (a_j - (q-1)) \bmod q] = [(a_i + 1) \bmod q, (a_{i+1} + 1) \bmod q, ..., (a_j + 1) \bmod q]$, $0 \leq i < j$.

  To illustrate the above notation, consider the following example where $A = [5, 0, 4, 2, 7]$ and $q = 0xF$ (in hexadecimal). Then $\overset{-}{A}_{1..3} = [0 + (q-1) \bmod q, 4 + (q-1) \bmod q, 2 + (q-1) \bmod q] = [(0-1) \bmod q, (4-1) \bmod q)], (2-1) \bmod q = [0xE, 3, 1]$. Similarly, $\overset{+}{A}_{0..2} = [5 - (q-1) \bmod q, 0 - (q-1) \bmod q, 4 - (q-1) \bmod q] = [5+1, 0+1, 4+1] = [6, 1, 5]$.

- $A^{(t)}$ denotes the value of the register A at time t.

- HW$(\cdot)$ denotes the Hamming weight of the enclosed argument.

- $A|B$ denotes the vector obtained from the concatenation of A and B.

Recovering the locations of the +1 elements in the secret key

As mentioned above, after determining the corresponding locations of the flip-flops corresponding to e in the scan chain, the attacker divides the flip-flops corresponding to the register T into two parts: T_L and T_R which contain, after the first step of convolution multiplication, all zeroes and all ones, respectively. Algorithm 4 is then used to determine the locations of the +1 elements in f relative to the location of the first +1 element. More precisely, Algorithm 4 outputs an array whose t^{th} element is equal to $(k[t] - k[0])$, $1 \leq t < d_f + 1$. As shown in the algorithm, A and B, are used to simulate the intermediate values of T_L and T_R, respectively, during the computation of $T = f \star e \bmod q$. By examining the Hamming weight information of T_L and T_R obtained from the scan chain output bit stream observed via the JTAG port, one can derive information about $k[t] - k[0]$ by going through all valid guesses values for $k[t]$ (see the variable j in Algorithm 4) and choosing the value for which the Hamming weight of A and B match the corresponding one for T_L and T_R, respectively, at the corresponding time.

The steps of Algorithm 4 can be explained as follows. At $t = 1$, a ciphertext of all ones is circularly shifted by $k[0]$ elements and loaded into T (note that all ones at the bit level corresponds to $e_i = q - 1$.) This can be simulated by initializing $A_i = 0$ and $B_i = q - 1$, $0 \leq i < N$ (lines 3-6). At $t = 2$, the ciphetext polynomial is circularly shifted by $k[1]$ elements and added to T. Thus $(k[1] - k[0])$ elements of A will change from 0 to $q - 1$ and $(N - (k[1] - k[0]))$ elements of B will change from $q - 1$ to $= ((q - 1) + (q - 1)) \bmod q = q - 2)$. The notation $\bar{A}_{i..j}$ and $\bar{B}_{i..j}$ (lines 14-15) are used to reflect these updates in A and B. Similarly, depending on the value of $k[2] - k[0]$, at $t = 3$, some elements of A change from $q - 1$ to $q - 2$ and from 0 to $q - 1$ while some elements of B change from $q - 2$ to $q - 3$. This process continues in a similar way until $t = d_f + 1$. As shown in lines 8-17, we update $A_{0..N}$ and $B_{0..N}$ to $A_{0..N-j-1}|\bar{A}_{N-j..N-1}$, and $\bar{B}_{0..N-j-1}|B_{N-j..N-1}$, respectively, $1 \leq j < N$ to simulate the above process. For each j, we calculate HW$(A_{0..N-j-1}|\bar{A}_{N-j..N-1})$ – HW(A). This step is repeated after incrementing j until the difference between these Hamming weights matches the value of HW$({T_L}^{(t)})$ – HW$({T_L}^{(t-1)})$, $2 \leq t \leq d_f + 1$. Then we set $k[t] - k[0] = j$, $1 \leq t \leq d_f + 1$. Note that while observing the changes in the Hamming weight of A is enough to allow Algorithm 4 to calculate the elements of S_1, we still update B since it is needed by Algorithm 5 which is used to determine all possible valid locations of the -1s in f.

Algorithm 4 Recovery of the locations of the +1s in the private key polynomial f

```
 1: INPUT: HW($T_L{}^{(t)}$), $1 \leq t \leq d_f + 1$.
 2: OUTPUT: An array $S_1$ where $S_1[t] = k[t] - k[0]$ and $k[t]$ denotes the location of the $t^{th}$ +1 elements
    in $f$, $1 \leq t < d_f + 1$.
 3: for $0 \leq i < N$ do
 4:     $A_i \longleftarrow 0$
 5:     $B_i \longleftarrow q - 1$
 6: end for
 7: $j \longleftarrow 0$
 8: for $2 \leq t \leq d_f + 1$ do
 9:     diff $\longleftarrow 0$
10:     while (diff $\neq$ HW($T_L{}^{(t)}$) $-$ HW($T_L{}^{(t-1)}$)) do
11:         $j \longleftarrow j + 1$
12:         diff $\longleftarrow$ HW($A_{0..N-j-1}|\overline{A}_{N-j..N-1}$) $-$ HW($A$)
13:     end while
14:     $A \longleftarrow A_{0..N-j-1}|\overline{A}_{N-j..N-1}$
15:     $B \longleftarrow \overline{B}_{0..N-j-1}|B_{N-j..N-1}$
16:     $S_1[t-1] \longleftarrow j$
17: end for
18: return $S_1, A, B$
```

Recovering the locations of the -1 elements in the secret key

Algorithm 5 receives A, B, the set of +1 locations, S_1, evaluated by Algorithm 4 and the Hamming weights HW($T_L{}^{(t)}$) and HW($T_R{}^{(t)}$) obtained by analyzing the scan out bit stream, $d_f + 2 \leq t \leq 2 \times d_f + 1$. The algorithm operates in a way similar to Algorithm 4 except that, each element in S_2 represents a list of possible valid locations for the -1 elements as opposed to a single element for the case of S_1. Also, the content of register e is subtracted from register T, instead of addition in Algorithm 4 (see lines 11-15 in Algorithm 3.)

Lines 7-12 are used to initialize j to the starting values for our guesses for the location of the t -1 element which correspond to $k[t]$, $d_f + 2 \leq t \leq 2d_f + 1$. These steps can be explained by noting that S_2 represents the list of valid locations for the -1 elements and by the fact that

$k[t] > k[t-1]$ for $d_f + 2 \leq t \leq 2d_f + 1$ (in other words, the location of the t^{th} -1 element has to be greater than the location of the $(t-1)^{th}$ -1 element in f). At $t = d_f + 2$, j starts from 0 since at this stage, the attacker cannot yet determine the exact value of $k[0]$ (Also, the $j = k[0]$ step is skipped since two keys cannot be assigned to the same location.) More precisely, according to the attacker's knowledge at this step, $0 \leq k[0] < N - S_1[d_f]$. In lines 13-36, the updates of A and B can follow two different paths depending on whether $k[d_f + 2]$ is less than or greater than $k[0]$ (i.e., whether the first non-zero element in f is +1 or -1). Lines 26-31 are used to determine values of j that represent valid guesses for the location of the -1s to be appended to the list S_2. This is performed by comparing the Hamming weight of the simulated registers A and B with the Hamming weight of T_L and T_R which can be calculated by observing the scan out data. In particular, in lines 29-30, we add j to the list $S_2[t - d_f - 1]$, $d_f + 2 \leq t \leq 2d_f + 1$, as a possible solution and update the values of A and B and append it to list $L[t - d_f - 1]$.

Analyzing Algorithm 4 and Algorithm 5 shows their run time complexity to be $\mathcal{O}(d_f \times N)$ and $\mathcal{O}(d_f \times N^3)$, respectively.

Example 5.1 *Consider a toy version of the NTRUEncrypt cryptosystem with parameters (N, p, q, d_f) = (7,3,16,2) and with a private key*

$$f = [0, -1, -1, 1, 0, 1, 1].$$

Thus the locations of the non-zero coefficients of f are $k = [3, 5, 6, 1, 2]$ where the first $d_f + 1$ values denote the locations of +1s in the key and the last d_f values denote the locations of -1s.

As explained above, the T register is initialized with all 0s before starting the convolution computation of $f \star e$ mod q. At $t = 1$, T is loaded with a copy of the ciphertext after being circularly shifted by $k[0] = 3$ coefficients according to Algorithm 3. Recall that the attacker does not know $k[0]$. By scanning out the bit stream pattern via the JTAG port, the attacker can distinguish T into T_L and T_R. Assume that the attacker observed the Hamming weights of of T_L

Algorithm 5 Recovery of the -1s locations in the private key polynomial f

```
1: INPUT: $A$, $B$ and $S_1[d_f]$ (from Algorithm 4), HW($T_L{}^{(t)}$) and HW($T_R{}^{(t)}$), $d_f + 2 \leq t \leq 2 \times d_f + 1$
2: OUTPUT: An array $S_2$ of lists where $S_2[t - d_f]$ is a list containing all estimated possible values for
   $k[t]$, $d_f + 1 \leq t < 2 \times d_f + 1$.
3: **for** $0 \leq k[0] < (N - S_1[d_f])$ **do**
4:     **for** $d_f + 2 \leq t \leq 2 \times d_f + 1$ **do**
5:         $i = 1$
6:         **repeat**
7:             **if** $(t = d_f + 2)$ **then**
8:                 $j \longleftarrow 0$
9:             **else**
10:                $j \longleftarrow S_2[t - d_f - 2][i] + 1$
11:                $A|B \longleftarrow L[t - d_f - 2][i]$
12:            **end if**
13:            **while** $(j < N)$ **do**
14:                **if** $(j \neq k[0])$ **then**
15:                    **if** $(j < k[0])$ **then**
16:                        Temp$_A \longleftarrow \overset{+}{A}_{0..k[0]-j-1}|A_{k[0]-j..N-1}$
17:                        Temp$_B \longleftarrow B_{0..k[0]-j-1}|\overset{+}{B}_{k[0]-j..N-1}$
18:                        diff$_1 \longleftarrow$ HW(Temp$_A$) $-$ HW($A$)
19:                        diff$_2 \longleftarrow$ HW(Temp$_B$) $-$ HW($B$)
20:                    **else**
21:                        Temp$_A \longleftarrow A_{0..N-j+k[0]-1}|\overset{+}{A}_{N-j+k[0]..N-1}$
22:                        Temp$_B \longleftarrow \overset{+}{B}_{0..N-j+k[0]-1}|B_{N-j+k[0]..N-1}$
23:                        diff$_1 \longleftarrow$ HW(Temp$_A$) $-$ HW($A$)
24:                        diff$_2 \longleftarrow$ HW(Temp$_B$) $-$ HW($B$)
25:                    **end if**
26:                    **if** (diff$_1 =$ HW($T_L{}^{(t)}$) $-$ HW($T_L{}^{(t-1)}$) and diff$_2 =$ HW($T_R{}^{(t)}$) $-$ HW($T_R{}^{(t-1)}$)) **then**
27:                        $A \longleftarrow$ Temp$_A$
28:                        $B \longleftarrow$ Temp$_B$
29:                        append $A|B$ to $L[t - d_f - 1]$
30:                        append the location $j$ to $S_2[t - d_f - 1]$
31:                    **end if**
32:                    $j \longleftarrow j + 1$
33:                **else**
34:                    $j \longleftarrow j + 1$
35:                **end if**
36:            **end while**
37:            $i = i + 1$
38:        **until** $i >$ number of elements in the list $S_2[t - d_f - 1]$
39:    **end for**
40: **end for**
41: **return** $S_2$
```

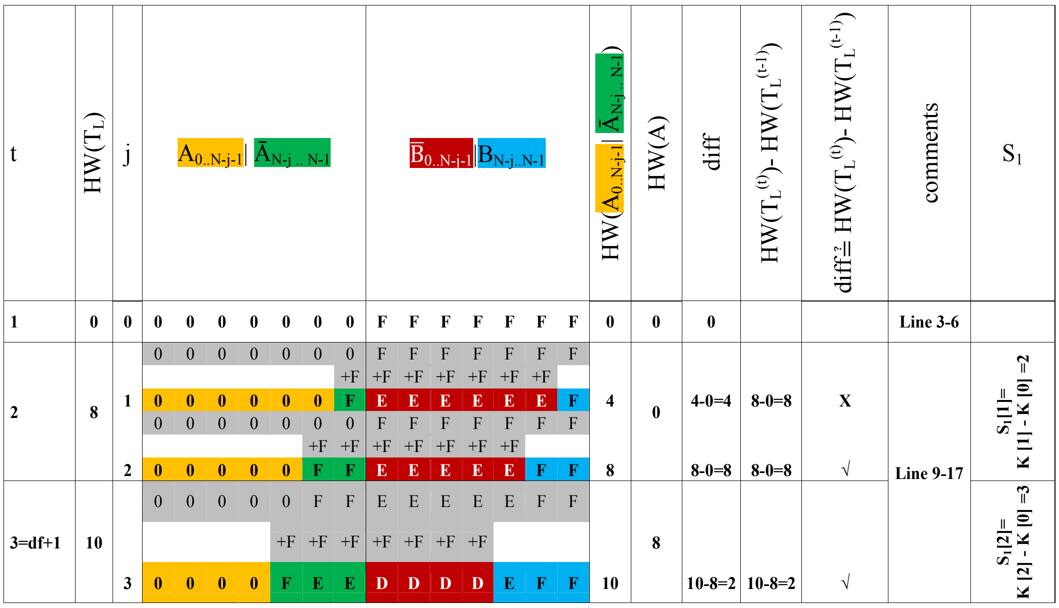
and T_R as shown in Table 5.1 (Obviously, at $t = 1$, these Hamming weights are always going to be 0 and $(N \times log_2(q))$, respectively). At $t = 2$, the register T_L is changed by two coefficients which implies that $k[1] - k[0] = 2$. While the attacker cannot associate the bits corresponding to T in the scan chain output with the individual coefficients in T_L and T_R, the attacker can still calculate the value of $k[1]-k[0]$ by using Algorithm 4 which simulates the content of $T = T_L|T_R$ (using A and B) for different possible values of $k([1] - k[0])$ (the variable j in lines 7-17) until the change in the Hamming weight of A, i.e., $HW(A_{0..N-j-1}|\bar{A}_{N-j..N-1}) - HW(A)$ in the simulation, matches HW($T_L{}^{(2)}$) – HW($T_L{}^{(1)}$) obtained from the scan out bit stream pattern. Following the same strategy, the attacker recovers the distances between $k[t]$ and $k[0]$ for $2 \leq t < d_f + 1$. In this example, the attacker recovers the set $S_1 = \{k[1]-k[0] = 2, k[2]-k[0] = 3\}$ which defines the distances between the locations of the +1s in the key.

Table 5.1: The Hamming weight of T_L and T_R as obtained from JTAG scan chain output in example 5.1

t	$HW(T_L^{(t)})$	$HW(T_R^{(t)})$
1	0	28
2	8	23
3=d_f+1	10	23
4	12	16
5=2×d_f+1	12	16

Figure 5.5 shows the input, output and intermediate computational results of Algorithm 4. To recover the locations of -1s in f, the attacker continues scanning out the bit stream pattern and, using Algorithm 5, calculates HW($T_L{}^{(t)}$) and HW($T_R{}^{(t)}$). Then the attacker calculates HW($T_L{}^{(t)}$) – HW($T_L{}^{(t-1)}$) and HW($T_R{}^{(t)}$) – HW($T_R{}^{(t-1)}$) for $d_f + 2 \leq t \leq 2 \times d_f + 1$. In this case and according to the obtained Hamming weights, different possibilities for these locations, at each t, can be recovered. The attacker appends all these possible locations of the -1s in a set of lists, S_2. In particular, for this example, the attacker evaluates $S_2 = \{k[3] = [1], k[4] = [2]\}$. Then, the values in S_1 represent the distances between the locations of +1s in the key in the form ($k[t] - k[0]$, $1 \leq t < d_f + 1$). The values of each element of S_2 represent a list of possible

locations of the -1s in the key. Enumerating all possible value for $k[0]$, $0 \leq k[0] < N - S_1[d_f]$ (in this example, $0 \leq k[0] < 4$), the attacker is able to uniquely determine the correct key locations $\{k[0] = 3, k[1] = 5, k[2] = 6, k[3] = 1, k[4] = 2\}$. Figures 5.6, 5.7 and 5.8 show the corresponding input, output and intermediate computational results of Algorithm 5.



t	$HW(T_L)$	j	$A_{0..N-j-1} \| \bar{A}_{N-j..N-1}$							$\bar{B}_{0..N-j-1} \| B_{N-j..N-1}$							$HW(A_{0..N-j-1} \| \bar{A}_{N-j..N-1})$	$HW(A)$	diff	$HW(T_L^{(t)}) - HW(T_L^{(t-1)})$	$diff \stackrel{?}{=} HW(T_L^{(t)}) - HW(T_L^{(t-1)})$	comments	S_1
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**0**	**0**	**0**			**Line 3-6**	
			0	0	0	0	0	0	0	F	F	F	F	F	F	F							
									+F	+F	+F	+F	+F	+F	+F								
2	**8**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**F**	**E**	**E**	**E**	**E**	**E**	**E**	**F**	**4**	**0**	**4-0=4**	**8-0=8**	**X**		**$S_1[1]$= K [1] - K [0] =2**
			0	0	0	0	0	0	0	F	F	F	F	F	F	F							
								+F	+F	+F	+F	+F	+F	+F									
		2	**0**	**0**	**0**	**0**	**0**	**F**	**F**	**E**	**E**	**E**	**E**	**E**	**F**	**F**	**8**		**8-0=8**	**8-0=8**	√	**Line 9-17**	
			0	0	0	0	0	F	F	E	E	E	E	E	F	F							
3=df+1	**10**						+F	+F	+F	+F	+F	+F	+F					**8**					
		3	**0**	**0**	**0**	**0**	**F**	**E**	**E**	**D**	**D**	**D**	**D**	**E**	**F**	**F**	**10**		**10-8=2**	**10-8=2**	√		**$S_1[2]$= K [2] - K [0] =3**

Figure 5.5: The computation steps in Algorithm 4 for Example 5.1

Because of the small value of N in the above example, the solution returned by Algorithms 4 and 5 is unique. The following example illustrates the more general case (the calculations details are omitted because of space limitations).

Example 5.2 *Consider a toy version of the NTRUEncrypt cryptosystem with parameters (N, p, q, d_f) = (11,3,32,3) and with a private key*

$$f = [0, -1, -1, 1, 0, -1, 1, 0, 0, 1, 1].$$

Assume that, based on the scan chain output bit stream observed via the JTAG port, the attacker distinguished the register T into two parts T_L and T_R and recorded the Hamming weights of both registers as shown in Table 5.2.

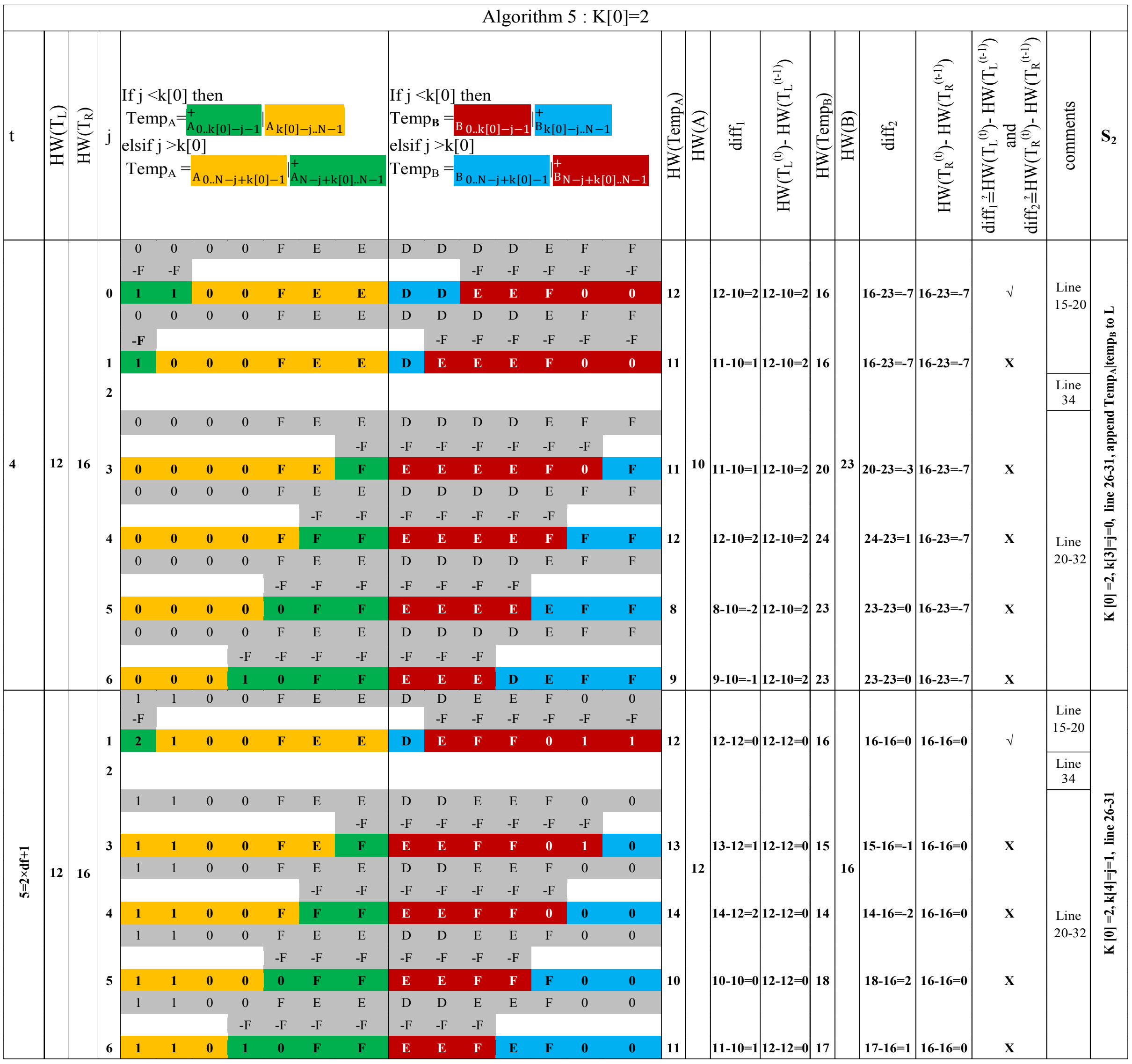
Algorithm 5: K[0]=0

t	$HW(T_L)$	$HW(T_R)$	j	If j <k[0] then $Temp_A = A_{0..k[0]-j-1} \overset{+}{\|} A_{k[0]-j..N-1}$ elsif j >k[0] $Temp_A = A_{0..N-j+k[0]-1} \overset{+}{\|} A_{N-j+k[0]..N-1}$							If j <k[0] then $Temp_B = B_{0..k[0]-j-1} \overset{+}{\|} B_{k[0]-j..N-1}$ elsif j >k[0] $Temp_B = B_{0..N-j+k[0]-1} \overset{+}{\|} B_{N-j+k[0]..N-1}$							$HW(Temp_A)$	$HW(A)$	$diff_1$	$HW(T_L^{(1)})- HW(T_L^{(t-1)})$	$HW(Temp_B)$	$HW(B)$	$diff_2$	$HW(T_R^{(1)})- HW(T_R^{(t-1)})$	$diff_1 \overset{?}{=} HW(T_L^{(1)})- HW(T_L^{(t-1)})$ and $diff_2 \overset{?}{=} HW(T_R^{(1)})- HW(T_R^{(t-1)})$	comments	S_2
			0																								Line 34	
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
										-F	-F	-F	-F	-F	-F	-F												
			1	0	0	0	0	F	E	F	E	E	E	E	F	0	F	11		11-10=1	12-10=2	20		20-23=-3	16-23=-7	X		
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
									-F	-F	-F	-F	-F	-F	-F													
			2	0	0	0	0	F	F	F	E	E	E	E	F	F	F	12		12-10=2	12-10=2	24		24-23=1	16-23=-7	X		
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
								-F	-F	-F	-F	-F	-F	-F														
1	12	16	3	0	0	0	0	0	F	F	E	E	E	E	E	F	F	8	10	8-10=-2	12-10=2	23	23	23-23=0	16-23=-7	X	Line 20-32	K [0] ≠0, line 26-31
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
							-F	-F	-F	-F	-F	-F	-F															
			4	0	0	0	1	0	F	F	E	E	E	D	E	F	F	9		9-10=-1	12-10=2	23		23-23=0	16-23=-7	X		
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
						-F	-F	-F	-F	-F	-F	-F																
			5	0	0	1	1	0	F	F	E	E	D	D	E	F	F	10		10-10=0	12-10=2	23		23-23=0	16-23=-7	X		
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
					-F	-F	-F	-F	-F	-F	-F																	
			6	0	1	1	1	0	F	F	E	D	D	D	E	F	F	11		11-10=1	12-10=2	23		23-23=0	16-23=-7	X		

Algorithm 5: K[0]=1

t	$HW(T_L)$	$HW(T_R)$	j	If j <k[0] then $Temp_A = A_{0..k[0]-j-1} \overset{+}{\|} A_{k[0]-j..N-1}$ elsif j >k[0] $Temp_A = A_{0..N-j+k[0]-1} \overset{+}{\|} A_{N-j+k[0]..N-1}$							If j <k[0] then $Temp_B = B_{0..k[0]-j-1} \overset{+}{\|} B_{k[0]-j..N-1}$ elsif j >k[0] $Temp_B = B_{0..N-j+k[0]-1} \overset{+}{\|} B_{N-j+k[0]..N-1}$							$HW(Temp_A)$	$HW(A)$	$diff_1$	$HW(T_L^{(t)})- HW(T_L^{(t-1)})$	$HW(Temp_B)$	$HW(B)$	$diff_2$	$HW(T_R^{(t)})- HW(T_R^{(t-1)})$	$diff_1 \overset{?}{=} HW(T_L^{(t)})- HW(T_L^{(t-1)})$ and $diff_2 \overset{?}{=} HW(T_R^{(t)})- HW(T_R^{(t-1)})$	comments	S_2
				0	0	0	0	F	E	E	D	D	D	D	E	F	F										Line 15-20	
				-F								-F	-F	-F	-F	-F	-F											
			0	1	0	0	0	F	E	E	D	E	E	E	F	0	0	11		11-10=1	12-10=2	16		16-23=-7	16-23=-7	X		
			1																								Line 34	
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
										-F	-F	-F	-F	-F	-F	-F												
			2	0	0	0	0	F	E	F	E	E	E	E	F	0	F	11		11-10=1	12-10=2	20		20-23=-3	16-23=-7	X		
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
									-F	-F	-F	-F	-F	-F	-F													
1	12	16	3	0	0	0	0	F	F	F	E	E	E	E	F	F	F	12	10	12-10=2	12-10=2	24	23	24-23=1	16-23=-7	X	Line 20-32	K [0] ≠1, line 26-31
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
								-F	-F	-F	-F	-F	-F	-F														
			1	0	0	0	0	0	F	F	E	E	E	E	E	F	F	8		8-10=-2	12-10=2	23		23-23=0	16-23=-7	X		
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
							-F	-F	-F	-F	-F	-F	-F															
			5	0	0	0	1	0	F	F	E	E	D	D	E	F	F	9		9-10=-1	12-10=2	23		23-23=0	16-23=-7	X		
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
						-F	-F	-F	-F	-F	-F	-F																
			6	0	0	1	1	0	F	F	E	E	D	D	E	F	F	10		10-10=0	12-10=2	23		23-23=0	16-23=-7	X		

Figure 5.6: The computation steps in Algorithm 5 for Example 5.1

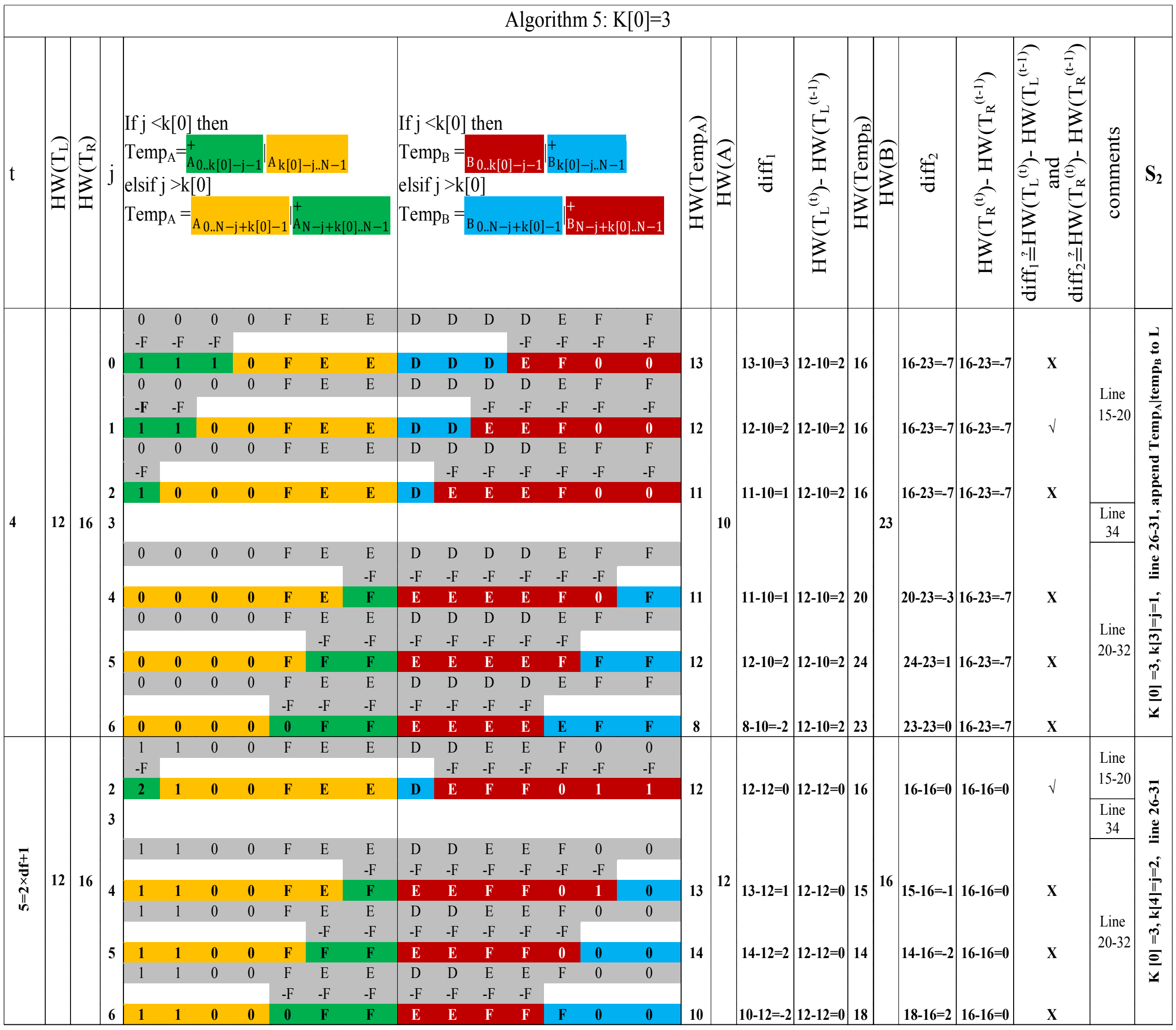


Algorithm 5 : K[0]=2

t	$HW(T_L)$	$HW(T_R)$	j	If j <k[0] then $Temp_A = {}^{+}A_{0..k[0]-j-1} \mid A_{k[0]-j..N-1}$ elsif j >k[0] $Temp_A = A_{0..N-j+k[0]-1} \mid {}^{+}A_{N-j+k[0]..N-1}$							If j <k[0] then $Temp_B = B_{0..k[0]-j-1} \mid {}^{+}B_{k[0]-j..N-1}$ elsif j >k[0] $Temp_B = B_{0..N-j+k[0]-1} \mid {}^{+}B_{N-j+k[0]..N-1}$							$HW(Temp_A)$	$HW(A)$	$diff_1$	$HW(T_L^{(t)})- HW(T_L^{(t-1)})$	$HW(Temp_B)$	$HW(B)$	$diff_2$	$HW(T_R^{(t)})- HW(T_R^{(t-1)})$	$diff_1 \overset{?}{=} HW(T_L^{(t)})- HW(T_L^{(t-1)})$ and $diff_2 \overset{?}{=} HW(T_R^{(t)})- HW(T_R^{(t-1)})$	comments	S_2
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											K [0] =2, k[3]=j=0, line 26-31, append $Temp_A$|$temp_B$ to L
				-F	-F								-F	-F	-F	-F	-F											
			0	1	1	0	0	F	E	E	D	D	E	E	F	0	0	12		12-10=2	12-10=2	16		16-23=-7	16-23=-7	√	Line 15-20	
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
				-F								-F	-F	-F	-F	-F	-F											
			1	1	0	0	0	F	E	E	D	E	E	E	F	0	0	11		11-10=1	12-10=2	16		16-23=-7	16-23=-7	X		
			2																								Line 34	
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
										-F	-F	-F	-F	-F	-F	-F												
4	12	16	3	0	0	0	0	F	E	F	E	E	E	E	F	0	F	11	10	11-10=1	12-10=2	20	23	20-23=-3	16-23=-7	X		
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
									-F	-F	-F	-F	-F	-F	-F													
			4	0	0	0	0	F	F	F	E	E	E	E	F	F	F	12		12-10=2	12-10=2	24		24-23=1	16-23=-7	X	Line 20-32	
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
								-F	-F	-F	-F	-F	-F	-F														
			5	0	0	0	0	0	F	F	E	E	E	E	E	F	F	8		8-10=-2	12-10=2	23		23-23=0	16-23=-7	X		
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
							-F	-F	-F	-F	-F	-F	-F															
			6	0	0	0	1	0	F	F	E	E	E	D	E	F	F	9		9-10=-1	12-10=2	23		23-23=0	16-23=-7	X		
				1	1	0	0	F	E	E	D	D	E	E	F	0	0											K [0] =2, k[4]=j=1, line 26-31
				-F								-F	-F	-F	-F	-F	-F											
			1	2	1	0	0	F	E	E	D	E	F	F	0	1	1	12		12-12=0	12-12=0	16		16-16=0	16-16=0	√	Line 15-20	
			2																								Line 34	
				1	1	0	0	F	E	E	D	D	E	E	F	0	0											
										-F	-F	-F	-F	-F	-F	-F												
			3	1	1	0	0	F	E	F	E	E	F	F	0	1	0	13		13-12=1	12-12=0	15		15-16=-1	16-16=0	X		
5=2×df+1	12	16		1	1	0	0	F	E	E	D	D	E	E	F	0	0		12				16					
									-F	-F	-F	-F	-F	-F	-F													
			4	1	1	0	0	F	F	F	E	E	F	F	0	0	0	14		14-12=2	12-12=0	14		14-16=-2	16-16=0	X	Line 20-32	
				1	1	0	0	F	E	E	D	D	E	E	F	0	0											
								-F	-F	-F	-F	-F	-F	-F														
			5	1	1	0	0	0	F	F	E	E	F	F	F	0	0	10		10-10=0	12-12=0	18		18-16=2	16-16=0	X		
				1	1	0	0	F	E	E	D	D	E	E	F	0	0											
							-F	-F	-F	-F	-F	-F	-F															
			6	1	1	0	1	0	F	F	E	E	F	E	F	0	0	11		11-10=1	12-12=0	17		17-16=1	16-16=0	X		

Figure 5.7: The computation steps in Algorithm 5 for Example 5.1: Continued from Figure 5.6

Running Algorithm 4, the attacker recovers the set $S_1 = \{k[1] - k[0] = 3, k[2] - k[0] = 6, k[3] - k[0] = 7\}$ *which defines the distances between the locations of the +1s in the key. Using Algorithm 5, the attacker evaluates* $S_2 = \{k[4] = 1, k[5] = [2, 4], k[6] = [4, 5, 6]\}$. *The values in* S_1 *represents the distances between the locations of +1s in the key in the form* ($k[t] - k[0]$, $1 \leq t < d_f + 1$) *and the values of each element in* S_2 *represents a list of possible locations of the -1s in the key. Enumerating all possible value for* $k[0]$, $0 \leq k[0] < N - S_1[d_f]$ *(in this example,* $0 \leq k[0] < 4$*) leads to* $4 \times 8 = 24$ *possible keys. The attacker can use the decryption*



Algorithm 5: K[0]=3

t	$HW(T_L)$	$HW(T_R)$	j	If j <k[0] then $Temp_A$= $\overset{+}{A}_{0..k[0]-j-1}$ | $A_{k[0]-j..N-1}$ elsif j >k[0] $Temp_A$ = $A_{0..N-j+k[0]-1}$ | $\overset{+}{A}_{N-j+k[0]..N-1}$							If j <k[0] then $Temp_B$ = $B_{0..k[0]-j-1}$ | $\overset{+}{B}_{k[0]-j..N-1}$ elsif j >k[0] $Temp_B$ = $B_{0..N-j+k[0]-1}$ | $\overset{+}{B}_{N-j+k[0]..N-1}$							$HW(Temp_A)$	$HW(A)$	$diff_1$	$HW(T_L^{(t)})- HW(T_L^{(t-1)})$	$HW(Temp_B)$	$HW(B)$	$diff_2$	$HW(T_R^{(t)})- HW(T_R^{(t-1)})$	$diff_1 \overset{?}{=} HW(T_L^{(t)})- HW(T_L^{(t-1)})$ and $diff_2 \overset{?}{=} HW(T_R^{(t)})- HW(T_R^{(t-1)})$	comments	S_2
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
				-F	-F	-F								-F	-F	-F	-F											
			0	**1**	**1**	**1**	**0**	**F**	**E**	**E**	**D**	**D**	**D**	**E**	**F**	**0**	**0**	**13**		**13-10=3**	**12-10=2**	**16**		**16-23=-7**	**16-23=-7**	**X**		
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
				-F	-F								-F	-F	-F	-F	-F											
			1	**1**	**1**	**0**	**0**	**F**	**E**	**E**	**D**	**D**	**E**	**E**	**F**	**0**	**0**	**12**		**12-10=2**	**12-10=2**	**16**		**16-23=-7**	**16-23=-7**	√	Line 15-20	
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
				-F								-F	-F	-F	-F	-F	-F											
			2	**1**	**0**	**0**	**0**	**F**	**E**	**E**	**D**	**E**	**E**	**E**	**F**	**0**	**0**	**11**		**11-10=1**	**12-10=2**	**16**		**16-23=-7**	**16-23=-7**	**X**		
4	**12**	**16**	3																**10**				**23**				Line 34	**K [0] =3, k[3]=j=1, line 26-31, append $Temp_A$|$temp_B$ to L**
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
										-F	-F	-F	-F	-F	-F	-F												
			4	**0**	**0**	**0**	**0**	**F**	**E**	**F**	**E**	**E**	**E**	**E**	**F**	**0**	**F**	**11**		**11-10=1**	**12-10=2**	**20**		**20-23=-3**	**16-23=-7**	**X**		
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
									-F	-F	-F	-F	-F	-F	-F													
			5	**0**	**0**	**0**	**0**	**F**	**F**	**F**	**E**	**E**	**E**	**E**	**F**	**F**	**F**	**12**		**12-10=2**	**12-10=2**	**24**		**24-23=1**	**16-23=-7**	**X**	Line 20-32	
				0	0	0	0	F	E	E	D	D	D	D	E	F	F											
								-F	-F	-F	-F	-F	-F	-F														
			6	**0**	**0**	**0**	**0**	**0**	**F**	**F**	**E**	**E**	**E**	**E**	**E**	**F**	**F**	**8**		**8-10=-2**	**12-10=2**	**23**		**23-23=0**	**16-23=-7**	**X**		
				1	1	0	0	F	E	E	D	D	E	E	F	0	0											
				-F								-F	-F	-F	-F	-F	-F											
			2	**2**	**1**	**0**	**0**	**F**	**E**	**E**	**D**	**E**	**F**	**F**	**0**	**1**	**1**	**12**		**12-12=0**	**12-12=0**	**16**		**16-16=0**	**16-16=0**	√	Line 15-20	
			3																								Line 34	
				1	1	0	0	F	E	E	D	D	E	E	F	0	0											
										-F	-F	-F	-F	-F	-F	-F												
5=2×df+1	**12**	**16**	4	**1**	**1**	**0**	**0**	**F**	**E**	**F**	**E**	**E**	**F**	**F**	**0**	**1**	**0**	**13**	**12**	**13-12=1**	**12-12=0**	**15**	**16**	**15-16=-1**	**16-16=0**	**X**		**K [0] =3, k[4]=j=2, line 26-31**
				1	1	0	0	F	E	E	D	D	E	E	F	0	0											
									-F	-F	-F	-F	-F	-F	-F													
			5	**1**	**1**	**0**	**0**	**F**	**F**	**F**	**E**	**E**	**F**	**F**	**0**	**0**	**0**	**14**		**14-12=2**	**12-12=0**	**14**		**14-16=-2**	**16-16=0**	**X**	Line 20-32	
				1	1	0	0	F	E	E	D	D	E	E	F	0	0											
								-F	-F	-F	-F	-F	-F	-F														
			6	**1**	**1**	**0**	**0**	**0**	**F**	**F**	**E**	**E**	**F**	**F**	**F**	**0**	**0**	**10**		**10-12=-2**	**12-12=0**	**18**		**18-16=2**	**16-16=0**	**X**		

Figure 5.8: The computation steps in Algorithm 5 for Example 5.1: Continued from Figure 5.7

process to uniquely determine the correct key locations $\{k[0] = 3, k[1] = 6, k[2] = 9, k[3] = 10, k[4] = 1, k[5] = 2, k[6] = 5\}$.

5.4 Experimental Results

In order to verify the correctness of the proposed attack, we implemented the NTRU-Encrypt decryption system with the convolution circuit depicted in Figure 5.4 using Synopsys design compiler and inserted a scan chain using Synopsys test compiler. Using this implemen-

Table 5.2: The Hamming weight of T_L and T_R as obtained from JTAG scan chain output in example 5.2

t	$HW(T_L^{(t)})$	$HW(T_R^{(t)})$
1	0	55
2	15	47
3	27	47
4=$d_f + 1$	29	43
5=$d_f + 2$	31	33
6	31	23
7=$2 \times d_f + 1$	31	24

tation, we confirmed our ability to determine the scan chain structure and the Hamming weight of T_R and T_L. The Hamming weights obtained from the ModelSim simulation were then used as input to Algorithm 4 and Algorithm 5 which were implemented using Python script. Table 5.3 shows our simulation results for Algorithm 4 and Algorithm 5 with 100 randomly selected keys for NTRUEncrypt with parameters $(N, p, q, d_f, d_g, d_r) = (167, 3, 128, 61, 20, 18)$, $(263, 3, 128, 50, 24, 16)$, and $(503, 3, 256, 216, 72, 55)$ which correspond to the moderate, high, and highest security parameters in [85]. As shown in the table, the average size of the list of keys, returned by Algorithms 4 and 5, is given by $\approx 2^{18}, 2^{24}$ and 2^{64} for these three set of parameters while the exhaustive search key security is given by $\frac{1}{d_g!}\sqrt{\frac{N!}{(N-2d_g)!}} \approx 2^{83}$, 2^{111} and 2^{285}, respectively [85]. It should be noted that these relatively large values for the average were dominated by some few cases where the size of the resulting key list were too large compared to the other cases (also note the relatively small values of the median in Table 5.3). Figure 5.9 shows the histogram distribution for the results of our simulations obtained from 100 runs with randomly generated keys for each N. From our data, and as can be deducted from Figure 5.9, for $N = 503$, in 28% of the cases, the size of the key list was less than 2^{40}. As mentioned above, the unique correct key can be determined by going through this list and verifying the correctness of the decryption operation for a known plaintext. This off-line step does not require physical access to the cryptographic device. It also does not require large memory space since the size of the sets S_1 and S_2 (outputs from Algorithm 4 and 5) is limited to $O(d_f N)$. Also, since there

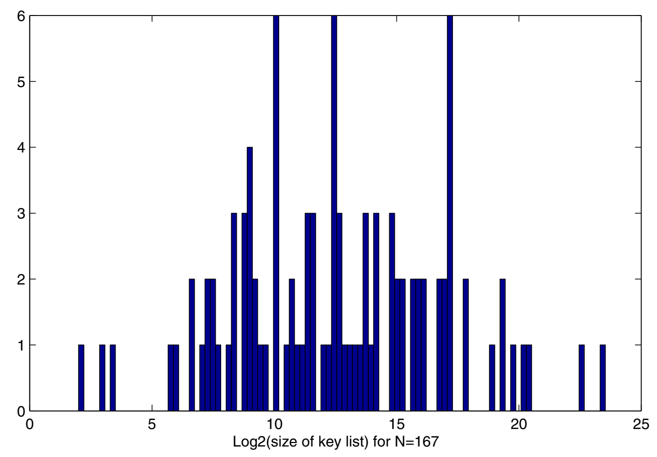
Table 5.3: Average and median size of the list of suggested keys

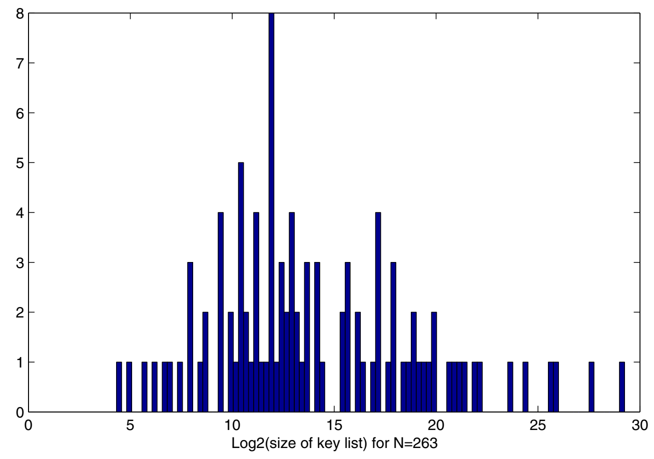
	$N = 167$	$N = 263$	$N = 503$
Average $\approx$	$2^{17.98}$	$2^{23.28}$	$2^{63.52}$
Median $\approx$	$2^{8.61}$	$2^{8.97}$	$2^{30.98}$

is no dependency between the different search paths, this exhaustive search step can be easily parallelized. It should be noted that the overall complexity of the attack is dominated by the complexity of Algorithm 5 and the complexity of going through the list of keys calculated by Algorithms 4 and 5 since the number of steps required by the scan-in and scan-out operations is negligible compared to these two steps.

5.5 Conclusion

In this chapter, we proposed a scan-based SCA to recover the NTRUEncrypt secret key. By analyzing the Hamming weight of the scan chain output at carefully chosen clock cycles, the attacker is able to efficiently recover the locations of the +1s and -1s of the secret key polynomial. The presented attack clearly shows the need to utilize secure scan chains for hardware implementations of NTRUEncrypt with the scan-based DFT feature.





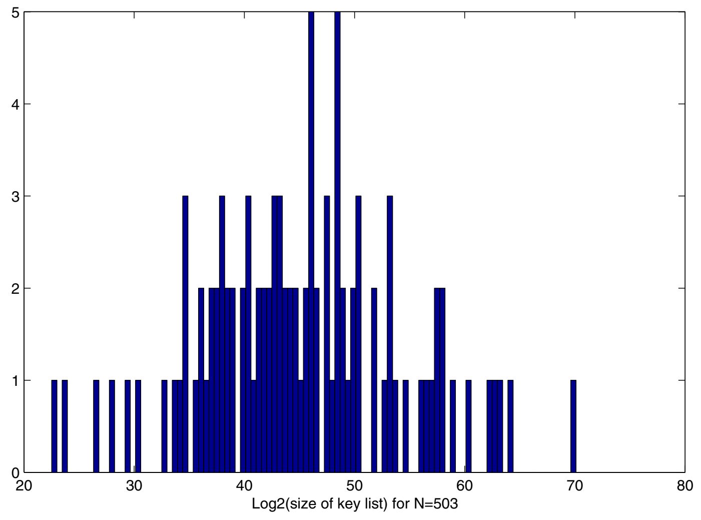


Figure 5.9: Histogram distribution for the size of the list of suggested keys

Chapter 6

Cryptanalysis of Key Exchange Schemes Based on Matrix Algebra

6.1 Introduction

Public-key cryptography [124] provides key exchange mechanisms in which secret keys can be exchanged between users over insecure communication channels. These key exchange mechanisms are usually based on number theory problems such as the discrete logarithm problem (DLP) [57], integer factorization [158] and elliptic curve DLP [26]. These systems require a large number of arithmetic operations, which makes them hard to implement in most resource constrained applications. To overcome this problem, key exchange protocols based on efficient matrix algebra have been proposed (e.g., see [191]). Odoni *et al.* [147] introduced the discrete logarithm problem for matrices over $\mathbb{F}_q$ and proposed a Diffie-Hellman key exchange protocol based on matrices. Menezes and Wu [125] reduced the discrete logarithm problem for matrices to some discrete logarithm problems over small extensions of $\mathbb{F}_q$. In what follows, we present a cryptanalysis of three different key exchange schemes based on matrix algebra.

6.2 Cryptanalysis of Álvarez *et al.* key exchange scheme

Recently, Álvarez *et al.* [5] proposed a key exchange scheme utilizing the non-abelian group of block upper triangular matrices (see also [5, 7]). Álvarez *et al.* claimed that one of the main advantages of this scheme is the absence of big prime numbers, which yields faster arithmetic operations and avoids the need for primality testing. Moreover, they also claimed that the proposed scheme is very efficient since it employs fast exponentiation algorithms for this type of matrices. In particular, by analyzing the order of the non-abelian group generated by these matrices as a function of the security parameters (r, s, p), as well as the implementation efficiency of these schemes, Álvarez *et al.* concluded that their system with security parameters $(r = 2, s = 89, p = 2903)$ has better performance than the Diffie-Hellman scheme with a similar level of security (key size of approximately 1024 bits).

In [187], Vasco *et al.* showed that breaking the Álvarez scheme can be reduced to solving a small set of discrete logarithm problems in an extension of the base field. Consequently, Vasco *et al.* concluded that the Álvarez scheme does not offer any computational advantage over the original Diffe-Hellman key exchange scheme. While the presented results in [187] challenges the efficiency claims made by Álvarez *et al.* [5] by showing that working with the proposed non-abelian group of block upper triangular matrices does not offer a computational advantage over working in the base field, these results do not present a practical attack on the Álvarez scheme for the recommended size of the security parameters (see table 3 in [5]).

In this section, we show that breaking this scheme is equivalent to solving a set of $3(r+s)^2$ consistent linear equations with $2(r+s)^2$ unknowns in $\mathbb{Z}_p$, which renders this system insecure for the suggested practical choices of security parameters. The rest of this context is organized as follows. In section 6.2.1, we briefly describe some details of the Álvarez *et al.* key exchange scheme. The proposed attack is described in section 6.2.2.

6.2.1 Description of the Álvarez *et al.* key exchange scheme

In this section, we briefly review the relevant definitions and details of the Álvarez *et al.* key exchange scheme. For further details, the reader is referred to [5].

Let $\mathrm{Mat}_{r\times s}(\mathbb{Z}_p)$ denote the set of matrices of size $r \times s$ with elements in $\mathbb{Z}_p$ where p is a prime number. Let $Gl_r(\mathbb{Z}_p)$ denote the general linear group of invertible matrices of sizes $r \times r$, also with elements in $\mathbb{Z}_p$.

Let $\Theta = \left\{ \begin{bmatrix} A & X \\ \mathbf{0} & B \end{bmatrix} : A \in Gl_r(\mathbb{Z}_p), B \in Gl_s(\mathbb{Z}_p), X \in \mathrm{Mat}_{r\times s}(\mathbb{Z}_p) \right\}$.

If $M \in \Theta$ and $h \geq 0$ then $M^h = \begin{bmatrix} A^h & X^{(h)} \\ \mathbf{0} & B^h \end{bmatrix}$ where

$$X^{(h)} = \left\{ \begin{array}{ll} \mathbf{0} & \text{if } h = 0 \\ \sum_{i=1}^{h} A^{h-i} X B^{i-1} & \text{if } h \geq 1 \end{array} \right\}.$$

Let $M_1 = \begin{bmatrix} A_1 & X_1 \\ \mathbf{0} & B_1 \end{bmatrix}$ and $M_2 = \begin{bmatrix} A_2 & X_2 \\ \mathbf{0} & B_2 \end{bmatrix}$ be two elements of the set Θ with order m_1 and m_2, respectively.

For x, $y \in \mathbb{N}$, we define

$$\begin{aligned} A_{xy} &= A_1^x A_2^y \\ B_{xy} &= B_1^x B_2^y \\ C_{xy} &= A_1^x X_2^{(y)} + X_1^{(x)} B_2^y \end{aligned}$$

The Álvarez *et al.* key exchange scheme can be summarized as follows [5]

1. Alice and Bob agree on a prime p and two matrices M_1, $M_2 \in \Theta$ with large orders m_1 and m_2, respectively.

2. Alice generates two random private keys[1] l, $m \in \mathbb{N}$ such that $1 \leq l \leq m_1 - 1$, $1 \leq m \leq m_2 - 1$, and computes A_{lm}, B_{lm}, C_{lm} constructing

$$C = \begin{bmatrix} A_{lm} & C_{lm} \\ \mathbf{0} & B_{lm} \end{bmatrix}$$

3. Alice sends C to Bob.

4. Bob generates two random private keys v, $w \in \mathbb{N}$ such that $1 \leq v \leq m_1 - 1$, $1 \leq w \leq m_2 - 1$, and computes A_{vw}, B_{vw}, C_{vw} constructing

$$D = \begin{bmatrix} A_{vw} & C_{vw} \\ \mathbf{0} & B_{vw} \end{bmatrix}$$

5. Bob sends D to Alice.

6. The public keys of Alice and Bob are respectively the matrices C and D.

7. Alice computes $K_a = A_1^l A_{vw} X_2^{(m)} + A_1^l C_{vw} B_2^m + X_1^{(l)} B_{vw} B_2^m$. It should be noted that K_a is the upper right $r \times s$ matrix in

$$M_a = M_1^l D M_2^m = \begin{bmatrix} A_a & K_a \\ \mathbf{0} & B_a \end{bmatrix}. \tag{6.1}$$

8. Bob computes $K_b = A_1^v A_{lm} X_2^{(w)} + A_1^v C_{lm} B_2^w + X_1^{(v)} B_{lm} B_2^w$. Similarly, we have

$$M_b = M_1^v C M_2^w = \begin{bmatrix} A_b & K_b \\ \mathbf{0} & B_b \end{bmatrix}. \tag{6.2}$$

[1] In [5], the symbols r, s were mistakenly used to simultaneously refer to both the security parameters and the secret exponents chosen by Alice, in step 2 of the key exchange algorithm. In this context, to avoid any possible confusion, we use r, s to refer to the system parameters and l, m to refer to the secret exponents chosen by Alice.

Finally, Alice and Bob share the key $K = K_a = K_b$.

6.2.2 The proposed attack

The above construction for M_1 and M_2 is used to guarantee a large order of the non-abelian group generated by these matrices and to attain a fast exponentiation algorithm for this type of matrices. On the other hand, our attack does not depend on the particular method by which the matrices M_1 and M_2 are constructed. From the analysis provided in [5], we have

$$\begin{aligned} C &= M_1^l M_2^m, \\ D &= M_1^v M_2^w. \end{aligned}$$

Thus, despite the apparent complexity of the above key exchange scheme, when analyzing its security, one can simply view it as follows

1. Alice and Bob agree on a prime p and two matrices M_1, $M_2 \in \Theta$.
2. Alice sends $C = M_1^l M_2^m$ to Bob.
3. Bob sends $D = M_1^v M_2^w$ to Alice.
4. Both Alice and Bob calculate $M_1^{l+v} M_2^{w+m}$ and extract the secret key using Equations (6.1), (6.2).

In what follows, we show that, given the public matrices C and D, the attacker can easily recover the secret key.

Lemma 6.1 *Let W_1 and W_2 be two invertible matrices of dimension $(r+s)\times(r+s)$ that satisfy*

$$W_1 M_1 = M_1 W_1 \tag{6.3}$$

$$W_2 M_2 = M_2 W_2 \tag{6.4}$$

$$D = W_1 W_2 \tag{6.5}$$

Then we have

$$M_1^{l+v} M_2^{w+m} = W_1 C W_2.$$

Proof: Using mathematical induction, it is easy to show that $W_1 M_1 = M_1 W_1$ and $W_2 M_1 = M_2 W_2$ implies that $W_1 M_1^l = M_1^l W_1$ and $W_2 M_2^m = M_2^m W_2$, respectively. The rest of the proof follows by noting that

$$\begin{aligned} W_1 C W_2 &= W_1 M_1^l M_2^m W_2 \\ &= M_1^l W_1 W_2 M_2^m \\ &= M_1^l D M_2^m. \end{aligned}$$

■

The above Lemma shows that while the attacker may not be able to recover the secrets chosen by Alice and Bob, i.e., l, v, w, m, or the associated matrices $M_1^l, M_1^v, M_2^w, M_2^m$, the attacker can still recover the overall secret key agreed upon between Alice and Bob if she is able to find any W_1 and W_2 that satisfy the above set of equations. This seemingly nonlinear system of equations can be easily linearized as follows

From Equation (6.3), we have

$$\begin{aligned} W_1 M_1 = M_1 W_1 &\iff W_1 M_1 W_1^{-1} = M_1 \\ &\iff M_1 W_1^{-1} = W_1^{-1} M_1 \end{aligned}$$

The attacker can easily solve a linear system of equations for W_1^{-1} and W_2 by replacing Equation (6.3) by $M_1 W_1^{-1} = W_1^{-1} M_1$ and Equation (6.5) by $W_1^{-1} D = W_2$. In other words, the attacker solves the system of equations given by

$$\begin{aligned} W_1^{-1}M_1 &= M_1W_1^{-1} \\ W_2M_2 &= M_2W_2 \\ W_1^{-1}D &= W_2, \end{aligned} \tag{6.6}$$

which corresponds to solving a set of $3(r+s)^2$ linear equations with $2(r+s)^2$ unknowns, corresponding to the elements of W_1^{-1} and W_2 over $\mathbb{Z}_p$. The following Lemma shows that the attacker is always able to find a valid solution for Equation (6.6).

Lemma 6.2 *The linear system of equations defined in Equation (6.6) is consistent.*

Proof: The proof follows directly by noting that $W_1 = M_1^v$ and $W_2 = M_2^w$ is a valid solution for this system of equations. ■

Remark 6.1 *A closer look at the Álvarez scheme reveals that it resembles the completely wrong and insecure implementation of the Diffe-Hellman key exchange in which Alice and Bob agree on $g^{(x+y)} = g^x \times g^y$ instead of $g^{xy} = (g^x)^y = (g^y)^x$, and hence it should be completely abandoned. It is also interesting to note that the claimed efficiency of this system is also a direct consequence of this mistake; the system uses matrix multiplication (e.g., see step 4 of the algorithm description in section 3) instead of matrix exponentiation.*

The following toy example illustrates the idea of the attack.

Example 6.1 *Let* $p = 37$, $r = 2$, $s = 3$, $l = 11$, $m = 32$, $v = 17$, $w = 39$,

$$M_1 = \begin{bmatrix} 3 & 31 & 24 & 12 & 13 \\ 9 & 24 & 28 & 20 & 26 \\ 0 & 0 & 9 & 16 & 14 \\ 0 & 0 & 25 & 17 & 2 \\ 0 & 0 & 23 & 12 & 30 \end{bmatrix}, M_2 = \begin{bmatrix} 7 & 14 & 18 & 12 & 4 \\ 22 & 16 & 15 & 12 & 6 \\ 0 & 0 & 29 & 36 & 8 \\ 0 & 0 & 33 & 15 & 35 \\ 0 & 0 & 5 & 24 & 5 \end{bmatrix}$$

Alice calculates $C = M_1^l M_2^m = \begin{bmatrix} 31 & 14 & 31 & 19 & 31 \\ 35 & 10 & 10 & 32 & 21 \\ 0 & 0 & 36 & 8 & 30 \\ 0 & 0 & 9 & 18 & 10 \\ 0 & 0 & 27 & 5 & 11 \end{bmatrix}$ *and sends it to Bob.*

Bob calculates $D = M_1^v M_2^w = \begin{bmatrix} 7 & 25 & 32 & 23 & 21 \\ 16 & 28 & 18 & 15 & 32 \\ 0 & 0 & 33 & 12 & 17 \\ 0 & 0 & 16 & 25 & 20 \\ 0 & 0 & 33 & 18 & 14 \end{bmatrix}$ *and sends it to Alice.*

Thus we have

$$M_a = M_b = M_1^{l+v} M_2^{m+w} = \begin{bmatrix} 2 & 15 & 33 & 18 & 26 \\ 14 & 2 & 3 & 27 & 16 \\ 0 & 0 & 28 & 1 & 5 \\ 0 & 0 & 17 & 18 & 14 \\ 0 & 0 & 11 & 13 & 5 \end{bmatrix}$$

and the secret calculated by Alice and Bob is $\begin{bmatrix} 33 & 18 & 26 \\ 3 & 27 & 16 \end{bmatrix}$.

It is easy to verify that

$$W_2 = \begin{bmatrix} 5 & 14 & 24 & 21 & 19 \\ 22 & 14 & 32 & 29 & 12 \\ 0 & 0 & 4 & 20 & 21 \\ 0 & 0 & 26 & 8 & 0 \\ 0 & 0 & 11 & 10 & 10 \end{bmatrix} \textit{ and}$$

$$W_1^{-1} = \begin{bmatrix} 20 & 17 & 2 & 20 & 31 \\ 30 & 16 & 34 & 31 & 24 \\ 0 & 0 & 9 & 4 & 6 \\ 0 & 0 & 36 & 10 & 16 \\ 0 & 0 & 34 & 1 & 32 \end{bmatrix} \Longrightarrow W_1 = \begin{bmatrix} 19 & 33 & 31 & 31 & 13 \\ 6 & 33 & 18 & 21 & 18 \\ 0 & 0 & 22 & 16 & 11 \\ 0 & 0 & 30 & 9 & 13 \\ 0 & 0 & 15 & 7 & 18 \end{bmatrix}$$

is one valid solution to the systems of equations given by (6.6), from which the attacker calculates

$$W_1 C W_2 = \begin{bmatrix} 2 & 15 & 33 & 18 & 26 \\ 14 & 2 & 3 & 27 & 16 \\ 0 & 0 & 28 & 1 & 5 \\ 0 & 0 & 17 & 18 & 14 \\ 0 & 0 & 11 & 13 & 5 \end{bmatrix} = M_a = M_b.$$

It is obvious that the secret key is given by the upper right $r \times s = 2 \times 3$ *matrix of* $W_1 C W_2$.

6.3 Cryptanalysis of a key exchange protocol based on the endomorphisms ring $End(\mathbb{Z}_p \times \mathbb{Z}_{p^2})$

Climent *et al.* [40] identified the elements of the endomorphisms ring $End(\mathbb{Z}_p\times\mathbb{Z}_{p^2})$ [17] with elements in a new set, denoted by E_p, of matrices of size 2×2, whose elements in the first row belong to $\mathbb{Z}_p$ and the elements in the second row belong to $\mathbb{Z}_{p^2}$. The following results were established in [40]:

The set

$$E_p = \left\{ \begin{bmatrix} a & b \\ pc & d \end{bmatrix} | a, b, c \in \mathbb{Z}_p \text{ and } d \in \mathbb{Z}_{p^2} \right\}$$

is a noncommutative unitary ring where addition is defined by

$$\begin{bmatrix} a_1 & b_1 \\ pc_1 & d_1 \end{bmatrix} + \begin{bmatrix} a_2 & b_2 \\ pc_2 & d_2 \end{bmatrix} = \begin{bmatrix} (a_1 + a_2) \bmod p & (b_1 + b_2) \bmod p \\ p(c_1 + c_2) \bmod p^2 & (d_1 + d_2) \bmod p^2 \end{bmatrix},$$

and multiplication is defined by

$$\begin{bmatrix} a_1 & b_1 \\ pc_1 & d_1 \end{bmatrix} \cdot \begin{bmatrix} a_2 & b_2 \\ pc_2 & d_2 \end{bmatrix} = \begin{bmatrix} (a_1a_2) \bmod p & (a_1b_2 + b_1d_2) \bmod p \\ p(c_1a_2 + d_1c_2) \bmod p^2 & (pc_1b_2 + d_1d_2) \bmod p^2 \end{bmatrix}.$$

The additive and multiplicative identities of E_p are given by

$$O = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \text{ and } I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \text{ respectively.}$$

Let $M = \begin{bmatrix} a & b \\ pc & pu + v \end{bmatrix} \in E_p$ with $a, b, c, u, v \in \mathbb{Z}_p$. Then M is invertible if and only if

$a \neq 0$ and $v \neq 0$, and in this case we have $M^{-1} =$

$$\begin{bmatrix} a^{-1} & (-a^{-1}bv^{-1}) \bmod p \\ p[(-a^{-1}cv^{-1}) \bmod p] & p[(ca^{-1}b(v^{-1})^2 - u(v^{-1})^2 - \lfloor \frac{vv^{-1}}{p} \rfloor v^{-1}) \bmod p] + v^{-1} \end{bmatrix}.$$

Climent *et al.* [40] proved that the ring $End(\mathbb{Z}_p \times \mathbb{Z}_{p^2})$ is isomorphic to the ring E_p. Furthermore, they proved that the fraction of invertible elements in E_p is given by

$$\left(\frac{p-1}{p}\right)^2 \approx 1 \text{ for large } p. \tag{6.7}$$

Thus, for large values of p, almost all elements in E_p are invertible. During the last decade, several cryptographic primitives using algebraic systems rather than traditional finite cyclic groups or finite fields have been proposed (e.g., see [138, 183]). In this context, and by trying to take advantage of matrix arithmetic, Climent *et al.* proposed a key exchange protocol using polynomial functions over E_p defined by polynomials in $\mathbb{Z}[X]$. In this section, we show that this protocol is not secure. In particular, we show that this protocol can be broken by solving a set of 10 consistent homogeneous linear equations in 8 unknowns over $\mathbb{Z}_{p^2}$. The rest of this context is organized as follows. In section 6.3.1, we briefly describe some details of the Climent *et al.* key exchange scheme. The proposed attack is described in section 6.3.2.

6.3.1 Description of Climent *et al.* key exchange scheme

For completeness, we briefly review the relevant details of the Climent *et al.* key exchange scheme. For further details, the reader is referred to [40].

Let $f(X) = a_0 + a_1X + a_2X^2 + ... + a_nX^n \in \mathbb{Z}[X]$. For an element $M \in E_p$, the element

$$f(M) = a_0I + a_1M + a_2M^2 + ... + a_nM^n \in E_p$$

where I is the multiplicative identity of E_p. The key exchange protocol proposed in [40] can be summarized as follows

1. Alice and Bob agree on the public parameters $r, s \in \mathbb{N}$ and $M, N \in E_p$ for a large prime p.
2. Alice and Bob choose their private keys $f(X)$ and $g(X) \in \mathbb{Z}[X]$, respectively.
3. Alice computes her public key $P_A = f(M)^r N f(M)^s$ and sends it to Bob.
4. Bob computes his public key $P_B = g(M)^r N g(M)^s$ and sends it to Alice.
5. Alice and Bob compute $S_A = f(M)^r P_B f(M)^s$ and $S_B = g(M)^r P_A g(M)^s$ respectively.
6. Finally, Alice and Bob share the secret key $S_A = S_B$.

6.3.2 The proposed attack

The main idea of the attack is based on the following Lemma.

Lemma 6.3 *Let*

$$W_1 = \begin{bmatrix} a_1 & b_1 \\ pc_1 & d_1 \end{bmatrix} \textit{ and } W_2 = \begin{bmatrix} a_2 & b_2 \\ pc_2 & d_2 \end{bmatrix}$$

be two matrices in E_p such that

$$W_1 M = M W_1 \tag{6.8}$$

$$W_2 M = M W_2 \tag{6.9}$$

$$P_B W_2 = W_1 N. \tag{6.10}$$

Then we have

$$S_A = S_B = W_1 P_A W_2^{-1}.$$

Proof:

Note that $W_i, i = 1, 2$, commutes with M implies that W_i commutes with $f(M)$ and consequently W_i commutes with $f(M)^h$ for any $h \in \mathbb{N}$. Also W_i commutes with M implies that W_i^{-1} commutes with M (This follows by noting that $W_iM = MW_i \Rightarrow W_iMW_i^{-1} = M \Rightarrow MW_i^{-1} = W_i^{-1}M$). Thus we have

$$\begin{aligned} W_1P_AW_2^{-1} &= W_1f(M)^rNf(M)^sW_2^{-1} \\ &= f(M)^rW_1NW_2^{-1}f(M)^s \\ &= f(M)^rP_Bf(M)^s \\ &= S_A. \end{aligned}$$

■

It is easy to verify that $W_1 = g(M)^r$ and $W_2 = g(M)^{-s}$ is a valid solution to the system of equations in Lemma 6.3. Thus, this linear system of equations is consistent and consequently the attacker is guaranteed to find at least one solution. In what follows we show how the attacker can solve this system of equations. Let

$$M = \begin{bmatrix} m_1 & m_2 \\ pm_3 & m_4 \end{bmatrix} \in E_p.$$

Because of the structure of the elements in E_p, it is easy to verify that the equation resulting from equating the top left element on both sides of the resulting matrices products in Equation (6.8) does not add any constraints to the system of equations and hence it can be eliminated (in other words, $(a_1m_1 + pb_1m_3) \equiv (a_1m_1 + pm_2c_1) \bmod p$ is always satisfied for all choices of a_1 and b_1). Consequently, Equation (6.8) leads to the following three equations:

$$\begin{aligned} a_1m_2 + b_1m_4 - b_1m_1 - d_1m_2 &\equiv 0 \bmod p \\ p(c_1m_1 + d_1m_3 - a_1m_3 - c_1m_4) &\equiv 0 \bmod p^2 \\ p(c_1m_2 - b_1m_3) &\equiv 0 \bmod p^2 \end{aligned} \tag{6.11}$$

with unknowns $a_1, b_1, c_1 \in \mathbb{Z}_p$ and $d_1 \in \mathbb{Z}_{p^2}$.

Similar argument applies to Equation (6.9) (note, however that Equation (6.10) leads to 4 equations). The solution for the above system of equations can be obtained by solving it over $\mathbb{Z}_{p^2}$ and then reducing the obtained solution for a_i, b_i and c_i modulo p, $i = 1, 2$ (recall that, for any multivariate polynomial f, $f(x_1, \cdots, x_n) \equiv 0 \bmod p^2 \Rightarrow f(x_1, \cdots, x_n) \equiv 0 \bmod p$.)

Thus the solution for the system of $3 + 3 + 4 = 10$ equations corresponding to Lemma 6.3 can be obtained by solving all equations over $\mathbb{Z}_{p^2}$ and then reducing the obtained solution for a_i, b_i and c_i modulo p, $i = 1, 2$.

Based on our experimental results, this system of equations is always under-determined and many solutions exist for W_1 and W_2. Choosing any solution such that W_2 is invertible leads to the right key. Note that for large p, which is the case of interest for this cryptosystem, our experimental results confirm this condition and practically holds for almost all valid solutions (also see Equation (6.7)). In what follows, we illustrate our attack using the same toy example that was provided in [40] to explain the steps of the protocol.

Example 6.2 *Assume that Alice and Bob agree on $p = 11$, $r = 3$, $s = 5$,*

$$M = \begin{bmatrix} 5 & 8 \\ 44 & 102 \end{bmatrix} \text{ and } N = \begin{bmatrix} 10 & 3 \\ 77 & 37 \end{bmatrix}.$$

Then, Alice chooses her secret key as $f(X) = 3 + 3X + 9X^2 + 5X^3 \in \mathbb{Z}[X]$ and Bob chooses his secret key as $g(X) = 9 + 6X + 5X^2 \in \mathbb{Z}[X]$. Thus we have

$$f(M) = 3 + 3M + 9M^2 + 5M^3 = \begin{bmatrix} 10 & 8 \\ 44 & 19 \end{bmatrix},$$

$$g(M) = 9 + 6M + 5M^2 = \begin{bmatrix} 10 & 5 \\ 88 & 72 \end{bmatrix}.$$

Alice computes her public key, P_A, as

$$P_A = f(M)^3 N f(M)^5 = \begin{bmatrix} 10 & 5 \\ 110 & 119 \end{bmatrix}$$

and sends it to Bob. Bob computes his public key, P_B, as

$$P_B = g(M)^3 N g(M)^5 = \begin{bmatrix} 10 & 10 \\ 11 & 16 \end{bmatrix}$$

and sends it to Alice. Alice computes her secret key $S_A = f(M)^3 P_B f(M)^5$ and Bob computes his secret key $S_B = g(M)^3 P_A g(M)^5$ to obtain

$$S_A = S_B = \begin{bmatrix} 10 & 7 \\ 22 & 113 \end{bmatrix}.$$

As explained above, the solution for the system of equations in Lemma 6.3 can be obtained by solving

$$\begin{bmatrix} 8 & 9 & 0 & 3 & 0 & 0 & 0 & 0 \\ 77 & 0 & 22 & 44 & 0 & 0 & 0 & 0 \\ 0 & 77 & 88 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 2 & 0 & 8 \\ 0 & 0 & 0 & 0 & 44 & 0 & 99 & 77 \\ 0 & 0 & 0 & 0 & 0 & 44 & 33 & 0 \\ 10 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 110 & 77 & 110 & 0 & 66 & 0 \\ 0 & 0 & 33 & 37 & 0 & 110 & 0 & 105 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \end{bmatrix} \equiv \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \textit{mod}\, 121$$

and then reducing the obtained solution for a_i, b_i and c_i, $i = 1, 2$, modulo p. Solving this system of linear equations, we obtain

$$\begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \end{bmatrix} \equiv \begin{bmatrix} 41z_1 + z_2 & \textit{mod}\, 11 \\ 3z_1 + 65z_2 & \textit{mod}\, 11 \\ 7z_1 + 5z_2 + 11z_3 & \textit{mod}\, 11 \\ 43z_1 + 4z_2 & \textit{mod}\, 121 \\ 74z_1 + 111z_2 & \textit{mod}\, 11 \\ 99z_1 + 88z_2 & \textit{mod}\, 11 \\ 11z_4 & \textit{mod}\, 11 \\ 8z_1 + 12z_2 & \textit{mod}\, 121 \end{bmatrix}$$

where z_1, z_2, z_3, z_4 can assume any arbitrary values in $\mathbb{Z}_{121}$. The attacker chooses any random values for z_1, z_2, z_3, z_4 such that W_2 is invertible (which happens with probability ≈ 1 for large values of p). In this example, suppose that the attacker randomly chooses $[z_1, z_2, z_3, z_4]^T = [1, 1, 10, 7]^T$, then we have $[a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2]^T = [9, 2, 1, 47, 9, 0, 0, 20]^T$ and conse-

quently we have

$$W_1 = \begin{bmatrix} 9 & 2 \\ 11 & 47 \end{bmatrix} \textit{ and } W_2 = \begin{bmatrix} 9 & 0 \\ 0 & 20 \end{bmatrix} \Rightarrow W_2^{-1} = \begin{bmatrix} 5 & 0 \\ 0 & 115 \end{bmatrix}.$$

Finally, the attacker recovers the secret key by calculating

$$W_1 P_A W_2^{-1} = \begin{bmatrix} 9 & 2 \\ 11 & 47 \end{bmatrix} \begin{bmatrix} 10 & 5 \\ 110 & 119 \end{bmatrix} \begin{bmatrix} 5 & 0 \\ 0 & 115 \end{bmatrix} = \begin{bmatrix} 10 & 7 \\ 22 & 113 \end{bmatrix} = S_A = S_B.$$

6.4 Cryptanalysis of a $GL(r, \mathbb{Z}_n)$-based Public Key System

In [175], Salvin presented a key exchange protocol using matrices in the general linear group, $GL(r, \mathbb{Z}_n)$, where n is the product of two distinct large primes. The system is fully specified in the United States patent number 7346162 issued in 2008. In the patent claims, Salvin argued that the best way to break this system is to factor n. Furthermore, for efficiency reasons, it was suggested to choose $r = 2$. In this section, we show that this cryptosystem can be trivially broken by solving a consistent set of r^2 homogenous linear equations in $2r$ unknowns over $\mathbb{Z}_n$. The rest of this context is organized as follows. In section 6.4.1, we briefly describe some details of the Salvin's key exchange scheme. The proposed attack is described in section 6.4.2.

6.4.1 Description of Salvin's key exchange scheme

In this section, we briefly review the relevant details of Salvin's key exchange protocol. Let $GL(r, \mathbb{Z}_n)$ denote the General Linear group of invertible square matrices of rank r and elements in $\mathbb{Z}$ mod n. To establish a shared secret key K between a sender and a receiver, the scheme proceeds as follows [175]

1. The receiver chooses two distinct large primes, p and q, and calculates $n = p \times q$.

2. The receiver generates two random matrices, A and $C \in GL(r, \mathbb{Z}_n)$, such that $AC \neq CA$, i.e., A and C do not commute.

3. The receiver calculates a matrix $B \in GL(r, \mathbb{Z}_n)$ such that $B = CAC$.

4. The receiver generates a matrix G which commutes with C, i.e., $GC = CG$.

5. The receiver publishes A, B, G, n, r as the public key and keeps p, q and the matrix C as the secret key (note that only C is used in the decryption process).

6. The sender uses the public matrix G and generates a secret matrix $D \in GL(r, \mathbb{Z}_n)$ such that $GD = DG$.

7. The sender generates the secret key $K \in GL(r, \mathbb{Z}_n)$ using the matrices B, D such that $K = DBD$.

8. The sender generates the matrix $E \in GL(r, \mathbb{Z}_n)$ using the public matrix A such that $E = DAD$ and sends it to the receiver.

9. The receiver retrieves the secret key $K \in GL(r, \mathbb{Z}_n)$ by calculating $K = CEC$. Finally, the sender and receiver share the key $K = DBD = CEC$.

In step 4 of the above protocol, the matrix G is generated by one of the following two methods

$$G = C^k \tag{6.12}$$

where k must be even and preferably small integer for faster key generation or

$$G = \sum_{i=0}^{r-1} a_i C^i, \tag{6.13}$$

where a_i are integers randomly generated in $\mathbb{Z}_n$ with at least one $a_i \neq 0$ for $i > 0$ and $C^0 = I$. Using the fact that powers of a matrix commute, the matrix D is also constructed by one of the two methods above to ensure that D commutes with C, i.e.,

$$D = G^{k'} \text{ or } D = \sum_{i=0}^{r-1} b_i G^i \tag{6.14}$$

where k' and b_i, $i = 0, \cdots, r-1$, are generated in the same way as k and a_i. For justification of the above choice of parameters (e.g., why k has to be even), the reader is referred to the security analysis in [175] where it is shown that recovering C for such choice of parameters is equivalent to factoring n.

6.4.2 The proposed attack

Most of the security argument in [175] is based on the fact that recovering the secret matrices C and D from the public parameters is hard. In here, we show that the cryptanalyst can recover the overall secret key, K, agreed upon between the sender and receiver without recovering any of these two matrices. The main idea of the attack is based on the following Lemma.

Lemma 6.4 *Let E, K and D be three matrices constructed as in the cryptosystem above, i.e., $D = \sum_{i=0}^{r-1} b_i G^i$, $b_i \in \mathbb{Z}_n$, $E = DAD$, and $K = DBD$ for arbitrary matrices A, B and G in $GL(r, \mathbb{Z}_n)$. Let W_1, W_2 be matrices in $GL(r, \mathbb{Z}_n)$ that satisfy*

$$\begin{aligned} W_1 G &= G W_1, \\ W_2 G &= G W_2, \end{aligned} \tag{6.15}$$

and

$$W_1 A = B W_2. \tag{6.16}$$

Then we have

$$K = W_1 E W_2^{-1}.$$

Proof: The proof follows by noting that

$$\begin{aligned} W_1 A &= B W_2 \quad \Rightarrow \\ D W_1 A D &= D B W_2 D \end{aligned}$$

From Equation (6.15), and given the construction of D, we have $W_i D = D W_i$, $i = 1, 2$. Consequently

$$\begin{aligned} D W_1 A D &= D B W_2 D \quad \Rightarrow \\ W_1 D A D &= D B D W_2 \quad \Rightarrow \\ W_1 E &= K W_2 \quad \Rightarrow \\ W_1 E W_2^{-1} &= K. \end{aligned}$$

■

It is straightforward to verify that $W_1 = C$ and $W_2 = C^{-1}$ is a valid non-trivial solution for the homogeneous system of equations defined by Lemma 6.4. Thus despite the fact that the number of unknowns in this homogenous system of linear equations, $2r^2$, is less than the number of equations, $3r^2$, this system of equations is consistent and the attacker is guaranteed to find a non-trivial (i.e., non-zero) solution for W_1 and W_2.

The complexity of the system of equations in Lemma 6.4 (i.e., number of unknown and number of equations) can be reduced by choosing W_1 and W_2 in the form

$$W_1 = \sum_{i=0}^{r-1} u_i G^i \text{ and } W_2 = \sum_{i=0}^{r-1} v_i G^i \tag{6.17}$$

where $u_i, v_i \in \mathbb{Z}_n$. This form guarantees that the condition in Equation (6.15) is always satisfied and consequently reduces the system of equations in Lemma 6.4 to

$$\sum_{i=0}^{r-1} u_i G^i A - v_i B G^i = \mathbf{0} \tag{6.18}$$

with $2r$ unknowns and r^2 equations. In what follows we prove that this reduced system of equations is also consistent.

Lemma 6.5 *For any matrix $H \in GL(r, \mathbb{Z}_n)$, and an integer power m, H^m can be expressed as*

$$H^m = \sum_{i=0}^{r-1} h_i H^i, h_i \in \mathbb{Z}_n.$$

Proof: The proof follows by recursively applying Cayley Hamilton theorem [14], which states that every square matrix over any commutative ring satisfies its own characteristic equation, and by noting that the characteristic of H has degree r. ■

Since G is constructed according to Equation (6.12) or Equation (6.13), Lemma 6.5 implies that the construction in Equation (6.17) is equivalent to

$$W_1 = \sum_{i=0}^{r-1} u'_i C^i, W_2 = \sum_{i=0}^{r-1} v'_i C^i, u'_i, v'_i \in \mathbb{Z}_n.$$

By noting that Lemma 6.5 also implies that C^{-1} can be expressed in the same form as W_2, the system in Equation (6.18) is consistent and has at least one non-trivial solution ($W_1 = C$ and $W_2 = C^{-1}$). In fact, our experimental results show that the rank of this system of equations is usually less than $2r$, i.e., it is under-determined and many solutions exist for W_1 and W_2. Choosing any solution such that W_2 is invertible leads to the right key. For a large n, which is the case of interest for this cryptosystem, this condition practically holds for almost all valid solutions of u_i, v_i, $i = 0, \cdots, r-1$ excluding the trivial all zeroes solution. Using random p and q of size 512 bits, we were able to recover the key in all the 1000 experiments that we conducted for each $r = 2, 3 \cdots, 10$ and we did not encounter any case corresponding to a non-invertible W_2. The following toy example illustrates the idea of our attack.

Example 6.3 *Let $r = 2$, $p = 13$, $q = 17$, and $n = p \times q = 221$. The receiver generates two random matrices A and $C \in GL(2, Z_{221})$*

$$A = \begin{bmatrix} 16 & 173 \\ 207 & 140 \end{bmatrix}, C = \begin{bmatrix} 112 & 142 \\ 177 & 122 \end{bmatrix}$$

and verifies that $AC \neq CA$. Then the receiver calculates

$$B = CAC = \begin{bmatrix} 220 & 176 \\ 132 & 196 \end{bmatrix}$$

and G using Equation (6.13) with randomly selected $a_0 = 102$ and $a_1 = 135$ to obtain

$$G = \sum_{i=0}^{r-1} a_i C^i = \begin{bmatrix} 265 & 108 \\ 7 & 173 \end{bmatrix}$$

which ensures that $GC = CG$. The receiver publishes n, A, B, and G as the public key and keeps the other parameters secret. The sender generates a matrix D such that $DG = GD$ using Equation (6.14) with $b_0 = 30$ and $b_1 = 2$ to obtain

$$D = \sum_{i=0}^{r-1} b_i G^i = \begin{bmatrix} 118 & 216 \\ 14 & 155 \end{bmatrix}.$$

The sender calculates

$$E = DAD = \begin{bmatrix} 66 & 58 \\ 11 & 38 \end{bmatrix}$$

and sends it to the receiver who recovers the shared secret key

$$K = \begin{bmatrix} 30 & 14 \\ 107 & 178 \end{bmatrix}.$$

Given the public parameters and E, the attack proceeds as follows. First, the attacker *generates two matrices W_1 and W_2 in $\mathbb{Z}_n$ such that*

$$W_1 = \sum_{i=0}^{r-1} u_i G^i = \begin{bmatrix} u_1 + 44\,u_2 & 108\,u_2 \\ 7\,u_2 & u_1 + 173\,u_2 \end{bmatrix}$$

$$W_2 = \sum_{i=0}^{r-1} v_i G^i = \begin{bmatrix} v_1 + 44\,v_2 & 108\,v_2 \\ 7\,v_2 & v_1 + 173\,v_2 \end{bmatrix}$$

Thus the system of linear equations in (6.16) can be expressed as

$$W_1 A - B W_2 = 0 \Rightarrow$$

$$\begin{bmatrix} 16 & 76 & 1 & 138 \\ 173 & 190 & 45 & 158 \\ 207 & 121 & 89 & 113 \\ 140 & 16 & 25 & 14 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

By solving this system of linear equations, the attacker obtains $u_1 = 34z$, $u_2 = 199z$, $v_1 = 106z$, and $v_2 = z$ where z can be set to any arbitrary value in $\mathbb{Z}_n$. To avoid the trivial solution, we avoid choosing $z = 0$. Then we have

$$W_1 = z \begin{bmatrix} 171 & 55 \\ 67 & 206 \end{bmatrix}$$

$$W_2 = z \begin{bmatrix} 150 & 108 \\ 7 & 58 \end{bmatrix} \Rightarrow W_2^{-1} = \frac{1}{z} \begin{bmatrix} 32 & 9 \\ 19 & 98 \end{bmatrix},$$

where $\frac{1}{z}$ denotes the multiplicative inverse of z mod n. Consequently, the attacker calculates the *secret key as*

$$K = W_1 E W_2^{-1} = \begin{bmatrix} 30 & 14 \\ 107 & 178 \end{bmatrix}.$$

6.5 Conclusion

In the first part of this chapter, we showed that Álvarez *et al.* key exchange scheme is insecure for all suggested practical choices of the security parameters (r, s, p). Our attack is also directly applicable to the new system recently proposed by Álvarez *et al.* in [6]. As mentioned above, our attack does not depend on the particular method by which the involved matrices are generated, and hence the idea of linearization used in this chapter can be applied to a wider class of similar key exchange schemes.

In the second part, we showed that the key exchange protocol proposed by Climent *et al.* is not secure. In fact, Climent's scheme can be seen as a partial generalization of Stickel's key agreement scheme [178] which was broken by Shpilrain in [168] (see also [134, 135, 177]). In particular, Shpilrain [168] deployed the same linearization approach used in our attack and suggested to use non-invertible matrices to foil such linear algebra attacks and to repair Stickel's scheme but his proposal was also broken [134]. The fact that there are so few non-invertible elements in E_p is a weakness of the scheme since it makes the attacker's job easier. It should also be noted that Stickel's scheme is only an instance of the group Diffie-Hellman scheme [37] which generalizes the original Ko-Lee *et al.* braid group based protocol [114]. Later on, several braid groups were suggested as platform groups. Linear algebra attacks on these braid-based schemes using the same techniques were also deployed (e.g., see [39, 92, 98, 118]).

Finally, we showed that key exchange scheme proposed in the US patent number 7346162 is completely insecure. In particular, while the attacker is not able to recover the secret parameters chosen by the sender and receiver, the attacker can easily fully recover the secret key agreed upon between the sender and receiver by solving r^2 linear equations in $2r$ unknowns over $\mathbb{Z}_n$. For the suggested choice of the parameter $r = 2$, this corresponds to solving 4 linear equations in 4 unknowns. It is interesting to note that the effort required by the attacker to break this system, for any r, is less than the effort required by the sender during the key exchange process. To our knowledge, the majority of published matrix-based cryptosystems fall into the same trap and base the security argument on the inability of the cryptanalyst to recover some secret parameters. However, as shown in our work, this argument does not ensure the security of these schemes because attackers might be able to recover the overall secret key without recovering the individual components (e.g., let $Z = XY$. Then being able to find Z does not mean that we are able to recover X and/or Y. Conversely, proving that one cannot find X or Y does not imply that one cannot find Z given the availability of other information about Z).

Chapter 7

Conclusions and Future Research Directions

7.1 Summary and Conclusions

This section briefly summarizes the accomplished work and the major contributions of our thesis. In chapters 1, 2, the essential background, mathematical assumptions and motivation for this work were presented.

In chapter 3, we modeled the problem of key recovery of the AES-128 key schedules from its corresponding decayed memory images as a Boolean SAT problem and solved it using CryptoMiniSat. Our experimental results confirm the versatility of our proposed approach which allows us to efficiently recover the AES-128 key schedules for large decay factors. The presented method can be extended in a straightforward way to AES-192, AES-256 and other ciphers with key schedules that can be presented as a set of Boolean equations and, hence, lend themselves naturally to SAT solvers.

In chapter 4, we presented a fault analysis attack against the original variant of NTRUEncrypt. Our attack does not work against more recent variants of NTRUEncrypt where $f = 1+pF$ for ternary or binary polynomial F since the bijection between F_p and f does not exist in these variants. We also presented different techniques for strengthening the resistance of NTRUEncrypt hardware implementations against fault attacks. We provided a comparison between these different techniques in terms of their error detection capabilities as well as area and throughput overheads. We also provided FPGA implementation results for these approaches. One interesting observation that follows from our attack on NTRUSign is that the perturbation process, which was introduced to improve the security of NTRUSign against previous attacks, does not improve the resistance of NTRUSign against this kind of fault attacks if the norm-bound checking step can be skipped by the attacker. Another observation is that using NTRUSign with a larger-security parameter, while improving the resistance against lattice based attacks, does not necessarily improve the resistance against fault analysis attacks especially in the cases where the attacker is able to precisely corrupt small number of coefficients and is able to skip the norm-bound checking step.

In chapter 5, we presented a scan-based SCA on NTRUEncrypt hardware implementations that employ scan-based DFT techniques. By analyzing the Hamming weight of the scan chain output at carefully chosen clock cycles, the attacker is able to efficiently recover the locations of the +1s and -1s of the secret key polynomial. The presented attack clearly shows the need to utilize secure scan chains [74] for hardware implementations of NTRUEncrypt with the scan-based DFT feature.

Finally, in chapter 6, we showed that the three key exchange schemes proposed by Álvarez *et al.* [8], Climent *et al.* [40], and Keith Salvin [175] are completely insecure. The presented results show the difficulty of designing efficient key agreement schemes based on matrices.

7.2 Future works

In what follows we list some topics of interest for future extension of our research.

- Recent advances in the field of leakage-resilient cryptography (e.g., see [2, 9, 36, 127]) is concerned with the design of cryptographic primitives resistant to arbitrary SCAs. The main assumption used in all these works is that, while an attacker can repeatedly and adaptively learn information about the secret key, the overall amount of such information is bounded by some parameter. Several new cryptographic primitives are being designed using such theoretical assumption but no concrete engineering methodology to guarantee, or even measure, this information leakage bound has been considered. The development of methods and techniques that achieve bounded leakage grantee is certainly a very challenging and interesting research direction.

- Post-quantum cryptography refers to research on cryptographic primitives, usually public-key cryptosystems, that are not breakable using quantum computers. One possible (long term) theoretical research project is to investigate the existence of quantum algorithms that may help speed up the cryptanalysis of ciphers currently included in the list of post-quantum algorithms (e.g., lattice based cryptosystems including NTRU, code-based cryptosystems [123, 144] and systems based on multivariate-quadratic-equations [91]). Another research direction is exploring the application of different types of SCAs to these algorithms.

- Examples of current side channels include timing information, power consumption, electromagnetic leaks, acoustic leaks, and fault analysis. It is not hard to argue that other side channel might be explored in future research but coming up with a new concrete example for such channels is the real challenge.

- The security of key exchange schemes that are based on braid groups [113, 114] is still not very well understood. Exploring different cryptanalytic techniques against this class of group-based cryptography is an interesting research direction.

Appendix A

An FPGA Implementation of the NTRUEncrypt Cryptosystem

In this appendix, we present an architecture that offers different area-speed trade-off for NTRUEncrypt cryptosystem and analyze its performance. We utilize the statistical properties of the distance between the non-zero elements in the polynomials involved in the encryption and decryption operations. An FPGA prototype for the proposed design is also presented.

A.1 Hardware implementation of the NTRUEncrypt cryptosystem

In what follows, we assume that at the beginning of the encryption operation, the public key $h(x) \in R_q$ is loaded into the chip through N parallel I/O PINS in $T_{Ld(h)} = log_2(q)$ clock cycles. Similarly, for each plaintext $m(x) \in R_p$, an ephemeral key $r(x) \in \tau(d, d)$ will be loaded in $T_{ld(r)} = \lceil log_2(3) \rceil$ clock cycles and $m(x)$ will be loaded in $T_{Ld(m)} = \lceil log_2(p) \rceil$ clock cycles. Furthermore, $T_{op(e)} = log_2(q)$ clock cycles will be required to output the ciphertext $e(x) \in R_q$ using N parallel I/O PINS.

From the description of the encryption and decryption operations in section 4.2.1, it is clear that the most time consuming part in both operations is the convolution product, T_{conv}, required to compute $r(x) \star h(x)$ and $f(x) \star e(x)$ during the encryption and decryption operations, respectively.

In general, the convolution product $a(x) \star b(x)$ requires N^2 multiplications and additions where each of the c_k terms, $k = 0 \cdots N-1$, can be evaluated in one clock cycle using a shift, multiply and add operation. However, because of the ternary nature of both $f(x) \in \tau(d+1, d)$ and $r(x) \in \tau(d, d)$, these convolution can be evaluated without any multiplications. Furthermore, the time required to load h can be ignored when encrypting a large number of blocks. This is because h is loaded only once into the chip and remains unchanged throughout the encryption process. Hence, and by ignoring the few extra clock cycles required by the control circuity, a straightforward implementation of the encryption algorithm would require approximately

$$\begin{array}{ccccccc} T_{Ld(r)} & + & T_{Ld(m)} & + & T_{conv} & + & T_{op(e)} = \\ \lceil log_2(p) \rceil & + & \lceil log_2(p) \rceil & + & N & + & log_2(q) \end{array}$$

clock cycles to encrypt each plaintext block.

On the other hand, d is usually much smaller than N. For example, for NTRU with parameters $(N, p, q) = (251, 3, 128)$, we have $d = 36$, i.e., the number of non-zero coefficients in $r(x)$ is 72 [145, 153]. Thus if the locations of these non-zero elements are loaded into the chip, together with one bit indicating the value of each coefficient (i.e., 1 or -1), then T_{conv} can be significantly reduced from N to $2d$ operations. To simplify the control circuity, we assume that the j^{th}-bits of the locations of the non-zero elements in r, $j = 1, \cdots \lceil log_2(N) \rceil$ are loaded in one clock cycle. Thus we have $T_{ld(r)} = \lceil log_2(N) \rceil + 1$ and hence the number of clock cycles required to implement the encryption operation will be reduced to

$$\begin{array}{ccccccc} T_{Ld(r)} & + & T_{Ld(m)} & + & T_{conv} & + & T_{op(e)} = \\ (\lceil log_2(N) \rceil + 1) & + & \lceil log_2(p) \rceil & + & (2d) & + & log_2(q). \end{array}$$

This reduction in T_{conv}, however, requires the availability of a hardware circuity that can be used to perform an arbitrary number of shifts, up to $N - 2d$, for the $h(x)$ coefficients in one clock cycle, e.g., a Barrel shifter. However, implementing $\lceil log_2(q) \rceil$ N-bit Barrel shifters would require a prohibitively large hardware.

A considerable reduction in this hardware complexity can be achieved if we only require the shifter to be able to shift the coefficients in $h(x)$ by a relatively small number of locations, $s << N$, in each clock cycles.

Figure A.1 shows a typical implementation of such a circuit that can circularly shift its N bit inputs by up to s bits in one clock cycle.

Let d_i, $1 \leq i \leq 2d$, denote the location of the i^{th} non-zero element in $r(x)$. For convenience of notation, we set $d_0 = 0$. Then the number of clock cycles required to evaluate the convolution operation during the encryption operation is given by

$$T_{conv(s)} = \sum_{i=1}^{2d} \lceil \frac{d_i - d_{i-1}}{s} \rceil$$

Since these non-zero coefficients are distributed uniformly across the N terms of the corresponding polynomials, then, on average, we have

$$d_i - d_{i-1} \approx \frac{N}{2d} << N.$$

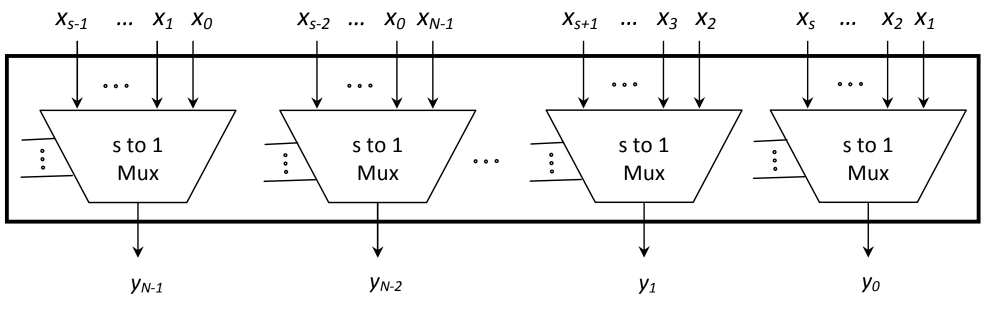


Figure A.1: Example of (N, s)-shifter

Throughout the rest of the appendix we focus on the $(251, 3, 128)$-version of NTRU.

Figure A.2 shows how the average value of $T_{conv(s)}$ decreases with s, for $1000,000$ randomly generated polynomials $r(x) \in \tau(36,36)$. Figure A.3 shows the number of slices required to implement this (251,s)-shifter. From both figures, it is clear that choosing $s > 8$ offers a small marginal increase in throughput at the expense of a relatively large area overhead.

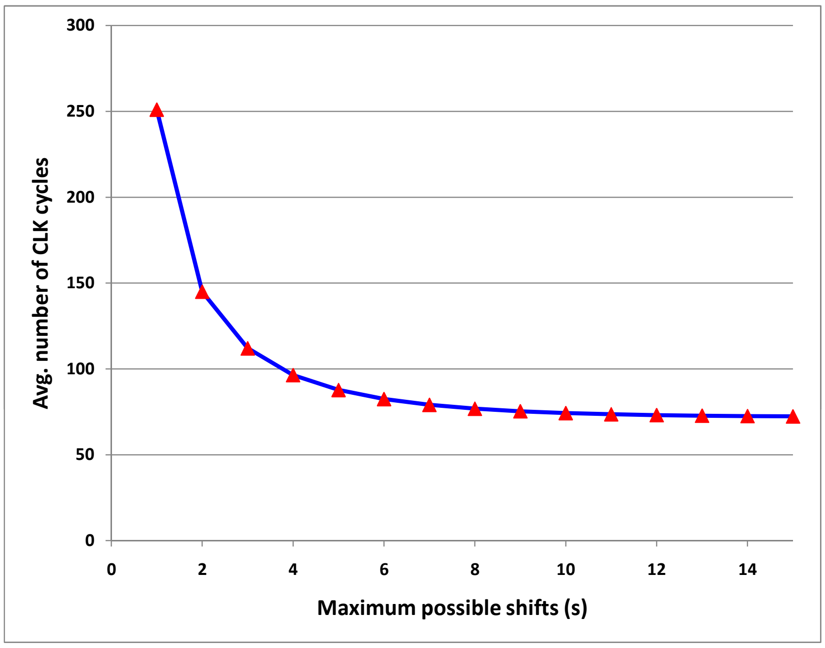


Figure A.2: Average value of $T_{conv(s)}$

Figures A.4 and A.5 show the histogram of $T_{conv(s)}$ for $1000,000$ randomly generated $r(x) \in \tau(36,36)$ for $s = 4,8$ respectively. The average, minimum and maximum values for $T_{conv(s=4)}$ are 96.4, 85 and 106, respectively. The corresponding values for $T_{conv(s=8)}$ are 76.85, 72, and 85.

Similar argument applies in optimizing the decryption algorithm except that calculating mod p in the final decryption step is non trivial. In this work, we investigated two methods to calculate the mod p operation. The first method is using the Mersenne primes algorithm [189]. In this method, $a(x)$ can be split into sections each of length $log_2(p+1)$ bits which can be added to produce $a(x)$ mod p (see Algorithm 6). The second method is to use Look-Up Tables (LUTs). The LUT approach is relatively efficient in case of small p. Note that choosing q in the form of 2^n eliminates the need for such circuits during the encryption operation.

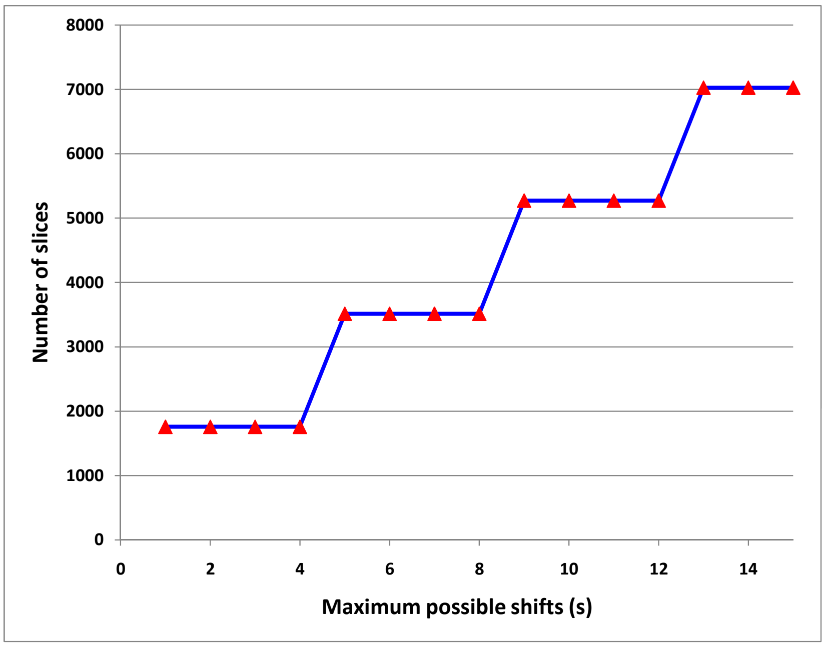


Figure A.3: The number of slices for $(251, s)$-shifter

Algorithm 6 Modular reduction using Mersenne primes algorithm

```
INPUT: an integer $a$, a Mersenne prime $p$
OUTPUT: $b = a \bmod p$
$b = a$
while $b > p$ do
    Split $b$ into sections $c_i \mid c_{i-1} \mid \cdots \mid c_1 \mid c_0$ each of length $log_2(p+1)$ bits.
    $b = c_i + c_{i-1} + \cdots + c_1 + c_0$.
end while
```

The targeted hardware platform for our implementation is the xcv1600e-8-fg860 FPGA using Xilinx ISE 9.1i as the Synthesis, Translation, Mapping, Place and Route File Generation tool. The control unit for our encryption/decryption engine is implemented using a Finite State Machine Data-path (FSMD) model. Simulations were conducted using the ISE model and implemented using VHDL. This ensures the performance was inaccurate representation of the actual FPGA.

Table A.1 shows the implementation results for the encryption-decryption engine using the naïve approach for polynomial multiplication. Table A.2 shows the corresponding results when using $(251, s)$ shifters with $s = 4, 8$. For $s = 4$, the average encryption throughput

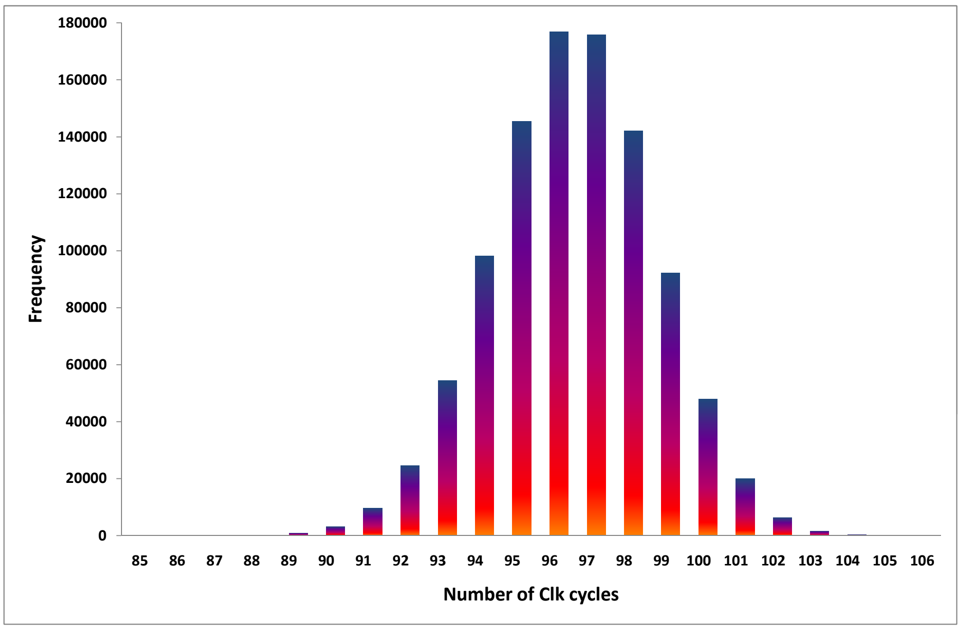


Figure A.4: Histogram of $T_{conv(s=4)}$

increased from 51.22 to 134 Mbps, i.e., by about 161.62%, when using the Mersenne primes algorithm and from 51.22 to 129 Mbps, i.e., by 151.85% when using the LUT method. Similarly, for decryption, the average throughput increased from 51.81 to 143.99 Mbps, i.e., by 177.92% when using the Mersenne primes algorithm and from 51.81 to 138.6 Mbps, i.e., by 167.52%, when using the LUT method. On the other hand, the area cost has increased from 8973 to 13028 slices, i.e., by 45.19% when using the Mersenne primes algorithm and from 8648 to 12472, i.e., by about 44.22% when using the LUT method.

Similarly, for $s = 8$, the average encryption throughput increased by about 216.83% and the average decryption throughput increased by about 241.75%. This occurs at the expense of about 60%-66% increase in the number of slices.

It should be noted that the implementation presented here is susceptible to timing analysis. It is interesting to explore how to modify it in a way that eliminates this weakness without scarifying the achieved speedup gain.

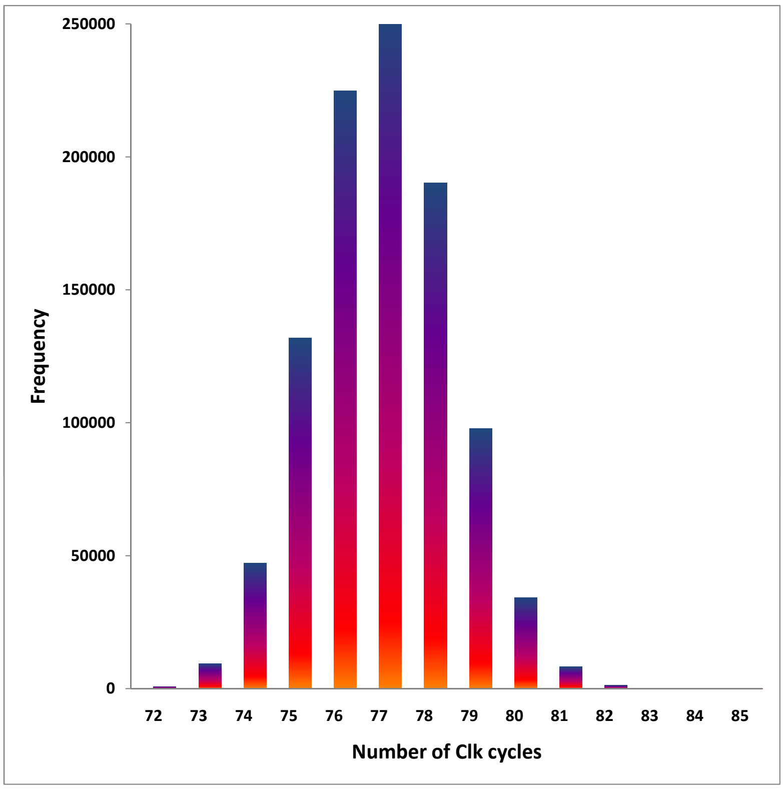


Figure A.5: Histogram of $T_{conv(s=8)}$

Table A.1: Implementation results using the naïve polynomial multiplication algorithm

	Mersenne algorithm	LUTs
Device components report		
# of Slices	8973	8648
# of Slice FFs	4922	4922
# of 4-input LUTs	16273	15913
# of Roms (128x2-bits)	-	251
# of IOBs	506	506
Timing report		
Clk Freq.(MHz)	54.08	54.08
Throughput(Mbps)	51.22 Enc., 51.81 Dec.	

Table A.2: Implementation results using the proposed approach for $s = 4$ and $s = 8$

	$s = 4$		$s = 8$	
	Mersenne algorithm	LUTs	Mersenne algorithm	LUTs
Device components report				
# of Slices	13028	12472	14406	14352
# of Slice FFs	5424	4838	5469	5160
# of 4-input LUTs	24360	21654	27216	27292
# of Roms (128x2-bits)	-	251	-	251
# of IOBs	579	579	579	579
Timing report				
Clk Freq.(MHz)	61.61	59.31	61.61	62.33
Avg. Throughput (Mbps) Enc.	134.00	129.00	161.34	163.22
Avg. Throughput (Mbps) Dec.	143.99	138.6	176.03	178.09

Appendix B

Enhanced Implementation of the NTRUEncrypt Algorithm Using Graphics Cards

B.1 Introduction

A Graphical Processing Unit (GPU) is a dedicated hardware for rendering graphics on devices such as personal computers, workstations, game consoles and embedded systems. State of the Art technologies made it possible to combine multiple GPUs in a single machine and hence achieving an affordable level of parallel processing. One of the main differences between GPUs and CPUs is that, in general, a CPU is optimized for executing high performance sequential code and hence the majority of its transistors are dedicated for flow control and branch prediction. On the other hand, a GPU has a more parallel nature and the majority of its transistors are dedicated for computation. This is because of the characteristics of its target applications. During the past few years, the power of GPUs have been increasing at a higher rate than that of CPUs. Consequently, general purpose computing on GPUs, i.e., the use of GPUs to accelerate the computations of different algorithms, has become a trend in parallel computing and played an important role in developing faster implementations for many algorithms in different areas. Because of their inherent computational complexity, encryption techniques are good candidates to benefit from the affordable high level of parallelism available on current GPUs. This is because many cryptographic algorithms show good characteristics for data parallel processing such as high computation intensity and independent work loads. Furthermore, security is becoming increasingly important and there is a great demand to secure data in all phases of its life cycle from communication to active or archived storage. This trend requires increased processing power which can be met by a combination of standard CPUs and GPUs acting as cryptographic accelerators. This is possible because GPUs are now ubiquitous and, unless involved in intensive graphics processing, their computational power are under-utilized most of the time.

Several researchers have explored the potential to use this available GPU power in a role similar to existing hardware cryptographic accelerators. In symmetric key cryptography, several works (e.g., [43, 73, 122, 151, 159]) have reported different GPU implementations for the Advanced Encryption Standard (AES). A very fast GPU implementation for the Korean ARIA block cipher was achieved by Yung *et al.* [121] where they reached an encryption throughput of 4.8 Gbps. In asymmetric key cryptography, R. Szerwinski *et al.* [179, 180] exploited the power of GPUs to speed up the computations of expensive operations such as modular exponentiation for RSA and DSA and point multiplication for ECC. They obtained 813 modular exponentiation per second for RSA and DSA-based systems with 1024 bit integers and 1412 point multiplications per second for ECC over P-224 NIST elliptic curves. A fast GPU implementation for one of the NIST SHA-3 hash function candidates, Blue Midnight Wish (BMW), was presented and analyzed in [149]. A GPU implementation for NTRU was also given in [76] where, using a modern 1.2 GHz GTX280 GPU, a throughput of up to 200,000 encryptions per second was reached at a security level of 256 bits. This gives a theoretical data throughput of 47.8 MB/s (see also [15, 102] for other software and hardware implementations of NTRUEncrypt).

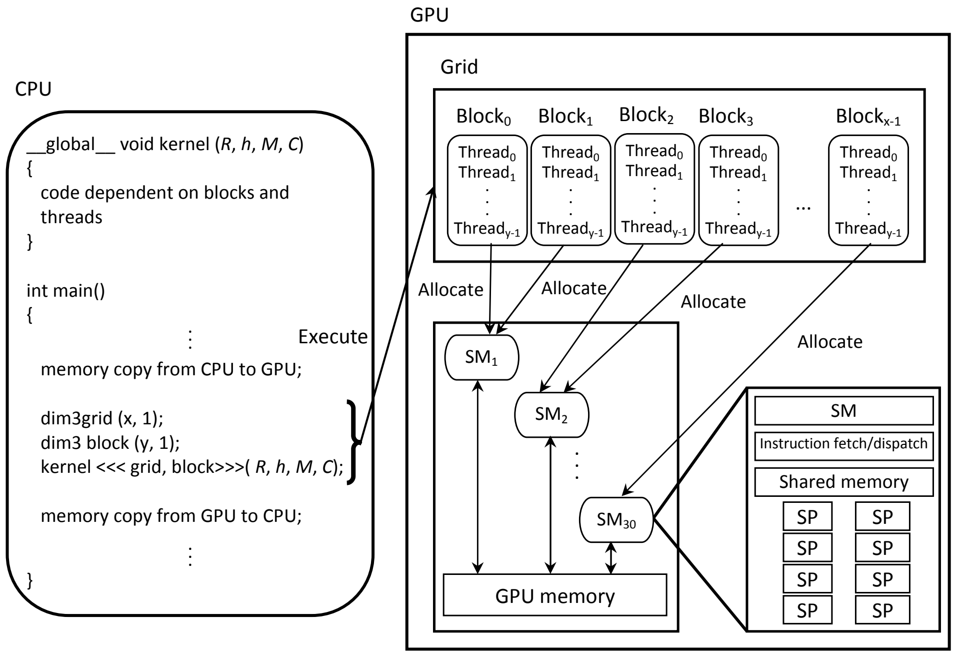
In this appendix, we investigate different implementation options for the NTRU encryption on the NVIDIA GTX275 GPU. Using the Compute Unified Device Architecture (CUDA) framework, our implementation achieves more than 351,000 parallel encryption operations per second for NTRU with 256 bits security level (parameter set $(N, q, p) = (1171, 2048, 3)$) which corresponds to an encryption throughput of about 77.21 MB/s.

B.2 The CUDA framework

Early attempts to use GPUs in cryptography were not very encouraging due to their previously poor suitability to the problem space, especially, the lack of integer processing support. Furthermore, GPU programming was challenging for those not familiar with graphics. Nowadays, high level language libraries that support parallel operations on GPUs have been developed by the main manufacturers of GPUs and are widely available [33, 41, 137, 164]. In what follows, we briefly summarize the thread organization and memory model of the Compute Unified Device Architecture (CUDA) [146] framework developed by NVIDIA.

B.2.1 Thread organization and memory model

The CUDA framework defines normal C like functions called kernels. Typical candidates for a kernel are functions that are executed many times but on multiple independent data. An execution configuration is used to specify the number of times the function should be executed as threads on the GPU and how the threads are organized. To manage the large number of threads executed, CUDA uses a thread hierarchy to identify and organize the threads. Unique coordinates are used to distinguish threads that execute the same function. As shown in Figure B.1, for every launch, the threads are lined up in a grid which is divided into a two level hierarchy of thread blocks and threads identified by coordinates called *blockIdx* and *threadIdx* which are assigned to them by the CUDA runtime system. These coordinates are accessible in the kernel to identify the different threads. The threads within a thread block can be organized in a one, two or three-dimension. In one kernel launch, a dimension in the grid cannot exceed 65535 and



```
__global__ void kernel (R, h, M, C)
{
  code dependent on blocks and
  threads
}

int main()
{
                    ⋮
  memory copy from CPU to GPU;

  dim3grid (x, 1);
  dim3 block (y, 1);
  kernel <<< grid, block>>>( R, h, M, C);

  memory copy from GPU to CPU;
                    ⋮
}
```

Figure B.1: The CUDA programming model

the size of a thread block is limited to 512. When the grid of thread blocks are launched, each thread block is assigned to an arbitrary Streaming Multiprocessor (SM). Every thread within the thread block is computed on the same SM. To schedule the threads, the SMs use a technique that NVIDIA calls Single-Instruction Multiple-Thread (SIMT). It is similar to Single Instruction Multiple Data (SIMD) but SIMT specifies the execution and branching behavior of each thread. The SMs are allocated thread blocks, and the SIMT unit splits the threads into groups of 32 threads called warps. Each warp consists of threads with consecutive increasing thread IDs with the first warp of a thread block containing thread ID 0-31. For every clock cycle, the SM chooses a warp that is ready to execute for scheduling, and hands each Streaming Processor (SP) a thread.

While Kernels run as threads on the GPU, the memory copying and the execution configuration is done by the CPU (the host). This separates the code into code executed on the CPU and code executed on the GPU. While both the host and the GPU manage their own memory space, data can be copied between them. Figure B.2 shows the memory model in CUDA [146]. Global Memory (GM) is the most frequently used memory space since it is the only space that

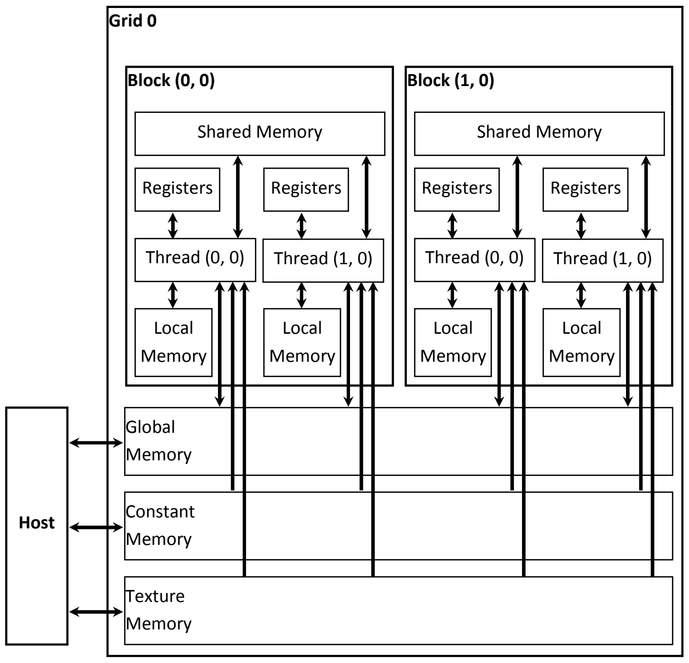


Figure B.2: The CUDA memory model

can be both read and written by both the CPU and the GPU. Constant and texture memory is read-only on the GPU, and is setup from the host for read-only data. The constant and texture memory is located in the same physical memory (DRAM) as global memory, but uses the texture unit in combination with a cache available to each SM to increase speed up reads from the memory spaces. Each SM has an 8 kB working set of constant and texture memory that is cached, which makes it useful for algorithms using data patterns that are read-only. Global memory, texture and constant memory can be accessed from any thread on the grid. Shared memory (SH) is the only memory space that is available on the actual GPU core and it is local to each SM, i.e., it can be accessed only within a thread block. Since the shared memory is very quick, it is optimal for sharing data across a thread block, or for performing computation before writing back to global memory. Thus to avoid the latency of global memory, a good way to compute data is to load it from global memory into shared memory, perform the computations in shared memory, synchronize each thread block and then write the result back to the global memory so that the host can read it when ready.

B.3 Implementation options for the NTRUEncrypt

The target platform for our implementation is the NVIDIA GTX275 GPU installed on the PCI Express 2.0 bus of a PC running AMD Athlon 64 X2 5000+ dual-core processor at 2.6 GHz and 2 GB of RAM. CUDA enables us to use the GTX275 GPU card, which runs at 1.404 GHz, as a parallel machine which contains 30 SMs, each contains 8 SPs with a total of 240 processing cores. The total memory of the card is 896 MB and each SM has a 16 KB shared memory.

To utilize the advantages of the GPU, several plaintext messages $m_i(x), i = 1, \cdots n$, are encrypted in parallel. The GPU receives the plaintext messages as a matrix M where the i^{th} row in M corresponds to the plaintext message $m_i(x)$. Similarly, the corresponding random ephemeral keys are transferred to the GPU as a matrix R where the i^{th} row represents the random ephemeral key $r_i(x)$ used to encrypt the message $m_i(x)$. Finally, The GPU receives the public key $h(x)$ as a vector and compute the the ciphertext as a matrix C where the i^{th} row in C corresponds to the ciphertext $c_i(x)$. Depending on to the method used to present the coefficients of $r_i(x)$, $h_i(x)$, $m_i(x)$ and $c_i(x)$, and the use of shared memory, we have the following implementation choices:

- *Representation of the polynomial* $r(x)$: the polynomial $r_i(x)$ can be represented in the naïve form as a polynomial with N coefficients or in the product form as shown in Equation (4.2).

- *Representation of the polynomial coefficients:* Traditionally, each coefficient of the polynomials $r_i(x)$ and $c_i(x)$ is stored in one integer variable (normal coefficients representation). In order to reduce the memory copying time between the CPU and the GPU, we employed a technique referred to as bit-packing [131] where each coefficient in $r_i(x)$ is represented in 2-bits. Therefore, each $\lfloor \frac{32}{2} \rfloor = 16$ coefficients can be represented in one integer variable. Similarly, since $q = 2048 = 2^{11}$, $\lfloor \frac{32}{11} \rfloor = 2$ coefficient in $c_i(x)$ can be stored in one integer. This helps reduce the time required to copy the data from/to the

GPU at the expense of the added computation time required to pack/un-pack the polynomial coefficients before using them in the actual computations.

- *Storage options:* Data can be stored in global memory or in the shared memory. The advantage of using shared memory over global memory is the low latency in calculation. However, the size of the GPU shared memory is limited (16 KB for each SM of the GPU).

We assume that the key setup and the ephemeral keys generation are done off-line by the host (i.e., the CPU). To encrypt n messages, each of size N elements, the convolution multiplication between $r_i(x)$ and $h_i(x)$ is done first and then the plaintext messages $m_i(x), i = 1 \cdots n$ are added (mod q) to the convolution output.

Figure B.3 illustrates the convolution operation using the Normal Polynomial form (NP), the Normal Coefficients representation (NC) and the global memory as a storage. In Figure B.3, the $N \times n$ matrix R corresponds to the n random ephemeral keys, each of size N elements. The i^{th} row of the $N \times N$ matrix H corresponds to a cyclically shifted version of the public key $h(x)$ by i elements. It should be noted that the matrix H shown in this Figure is a virtual matrix used for illustration purpose only, i.e., it is not physically stored on the GPU or the CPU memory. In other words, only $h(x)$ is physically stored and the different rows of H are created by the way that different threads use to access $h(x)$ using different values for $blockIdx.x$ and $blockIdx.y$ as shown in Algorithm 7. Each thread reads one row of the matrix R and one column of the matrix H and then computes the ciphertext by adding the result of the convolution to the corresponding element of the plaintext message $m_i(x)$. One kernel is used to calculate the encryption operation with $32 \times 16 = 512$ threads (to utilize the full capabilities of the SMs) which corresponds to $\lceil N/32 \rceil \times \lceil n/16 \rceil$ thread blocks.

As shown in Figure B.4, when the Bit-Packing approach (BP) is used to present the coefficients of R and C, the size of the R matrix is reduced from $n \times N$ to $n \times \lceil N/16 \rceil$. Similarly, the size of the corresponding ciphertext matrix is reduced from $n \times N$ to $n \times \lceil N/2 \rceil$. Consequently, R can be transferred from the CPU to the GPU in a shorter time. Also copying C from the GPU to the CPU will require less time.

Algorithm 7 Pseudo code for naïve implementation of NTRUEncrypt on GPU

```
 1: INPUT: Plaintext matrix $M_{n \times N}$, random ephemeral key matrix $R_{n \times N}$, the public key $h_{1 \times N}$.
 2: OUTPUT: The ciphertext matrix $C_{n \times N}$
 3: $tx \longleftarrow blockIdx.x * blockDim.x + threadIdx.x$
 4: $ty \longleftarrow blockIdx.y * blockDim.y + threadIdx.y$
 5: sum = 0
 6: for $i = 1$ to $N$ do
 7:     if $(R[ty * N + i] = 1)$ then
 8:         $sum \longleftarrow sum + h[(tx + N - i) \bmod N]$
 9:     else if $(R[ty * N + i] = -1)$ then
10:         $sum \longleftarrow sum - h[(tx + N - i) \bmod N]$
11:     end if
12: end for
13: $C[ty * N + tx] \longleftarrow (sum + M[ty * N + tx]) \bmod q$
```

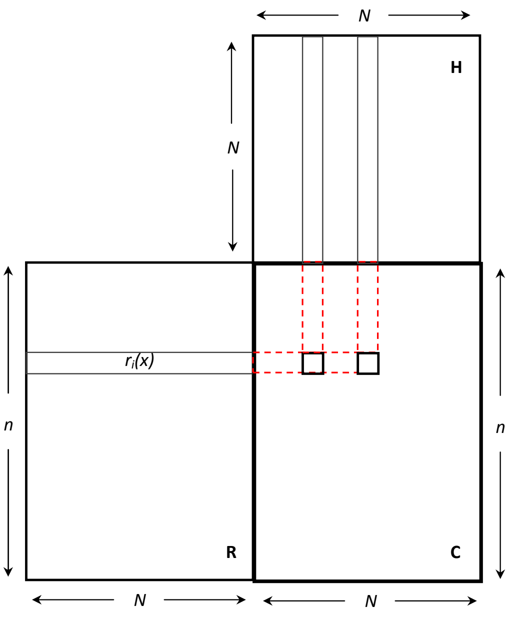


Figure B.3: The convolution operation using NP, NC, and GM

The computation efficiency on the GPU can be further improved when a thread block can load a block of data into the on-chip shared memory, process it there, and then write the final results back to external memory. Figure B.5 shows the case where the shared memory is used as a storage when $r_i(x)$ is presented in the naïve polynomial form and the normal representation is used to encode the polynomial coefficients. Each thread block is responsible for computing one block of the convolution matrix and each thread within the block is responsible for computing one element of the block. Two kernels are used to calculate the encryption operation. The first

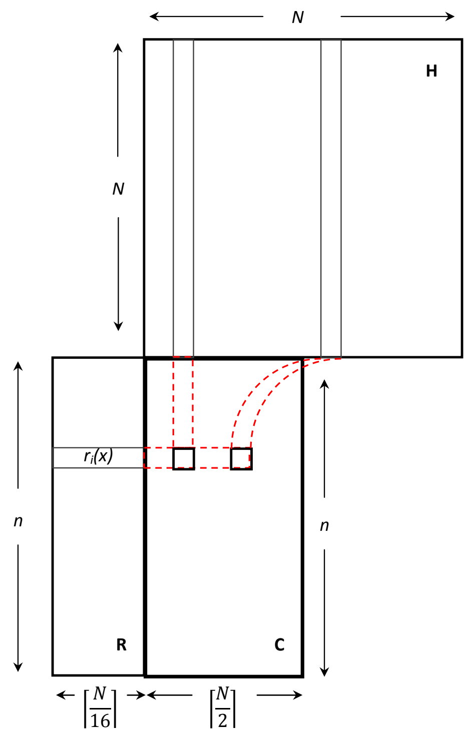


Figure B.4: The convolution operation using NP, BP, and GM

one calculates the matrix H from the vector $h(x)$ and then loads it into the shared memory. The second kernel completes the encryption operation via the shared memory. It also performs the necessary padding to ensure that the length of all blocks in R and H is divisible by 16. The CUDA $\_\_syncthreads()$ function is used to ensure that the data is copied to the shared memory before performing any calculations on it. In the first kernel, $32 \times 16 = 512$ threads and $\lceil N/32 \rceil \times \lceil n/16 \rceil$ blocks are used. The second kernel uses $16 \times 16 = 256$ threads and $\lceil N/16 \rceil \times \lceil n/16 \rceil$ blocks.

Figure B.6 shows the corresponding scenario where the bit-packing method is used to represent the polynomial coefficients and the shared memory is used to store the matrices R and H. Again, one kernel calculates the matrix H from the vector $h(x)$ and then loads it into the shared memory. The second kernel completes the encryption operation via the shared memory with $16 \times 16 = 256$ threads and $\lceil \lceil N/2 \rceil / 16 \rceil \times \lceil n/16 \rceil$ blocks. The dimension of the matrix R is reduced to $\lceil N/16 \rceil \times n$ and divided into blocks of size 16×16 KB. Also, the matrix H is

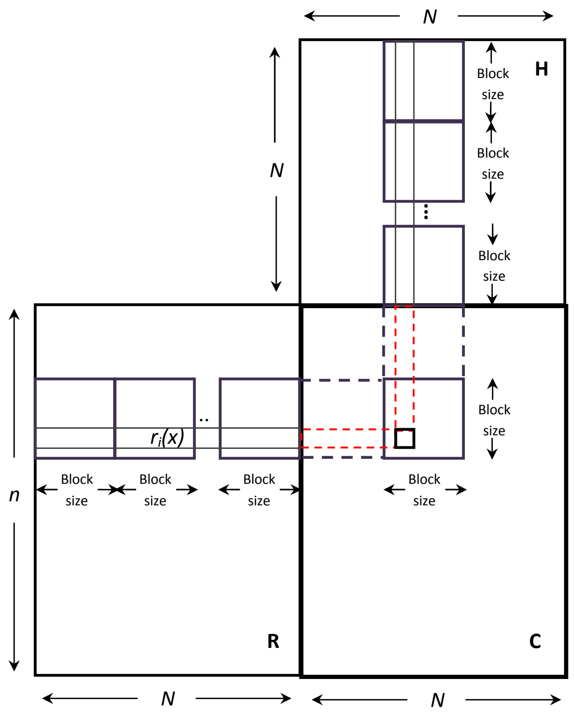


Figure B.5: The convolution operation using NP, NC, and SH

divided into blocks of size 16×16 KB. The blocks in R and the corresponding blocks in H are loaded into the shared memory. After performing the computation, the results are written into the global memory.

Similar analysis applies when the Product Form (PF) is used except that the convolution operation is performed three times per each encryption operation as shown in Equation (4.2).

As mentioned before, the key setup and the ephemeral keys generation are assumed to be generated off-line by the host. Thus, the total encryption time is equal to the time required to transfer the data from/to GPU and the time required to perform the necessary computational operations inside the GPU. The use of shared memory requires a padding of $h(x)$, $m_i(x)$, and $r_i(x)$ in order to obtain a polynomial size dividable by the block size in the shared memory. Since the decryption performs merely the same as encryption, we only illustrate the results for encryption.

According to the discussion in the previous section, there are $2^3 = 8$ combinations of implementation options for the NTRUEncrypt on the GPU. Figure B.7 shows how the total

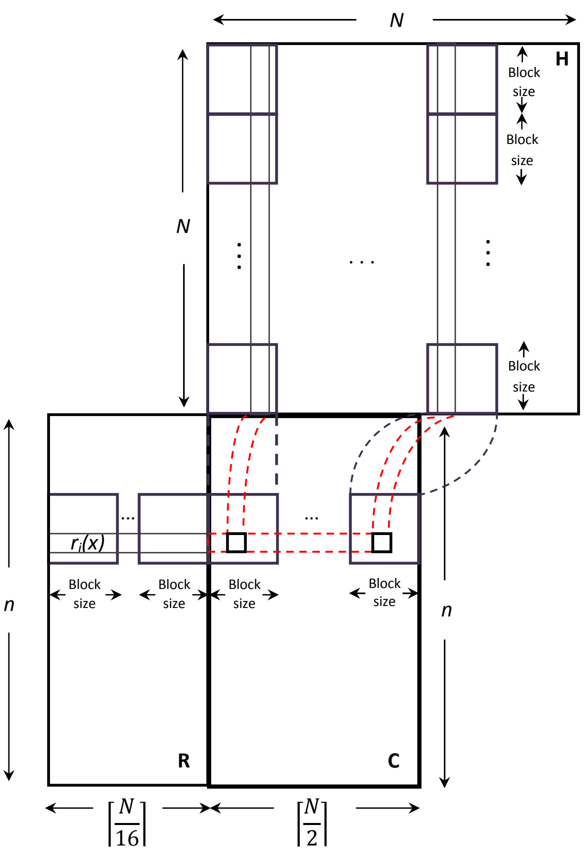


Figure B.6: The convolution operation using NP, BP, and SH

encryption time required to encrypt n messages in parallel varies with n (i.e., as n messages are loaded in one time from the host to the GPU and then encrypted in parallel on the GPU). Figure B.8 shows the corresponding encryption throughput. Due to the limited memory of the GPU, the maximum number of possible parallel operations is limited and varies depending on the memory requirements associated with the eight different implementation choices. It is clear that the use of product form, bit packing and shared memory allows us to achieve the best throughput where, for example at $n = 32768$, the achieved number of parallel encryption operations per second is 351,635. Since each message block has 1171 tuples and each tuple can assume the value of $+1, -1, 0$, i.e., each tuple has $2 \times \frac{3}{4}$ bits of information, then encrypting 351,635 message blocks per second corresponds to an encryption throughput of $351,635 \times 1171 \times 1.5 \times \frac{1}{8} \approx 77.21$ MB/sec.

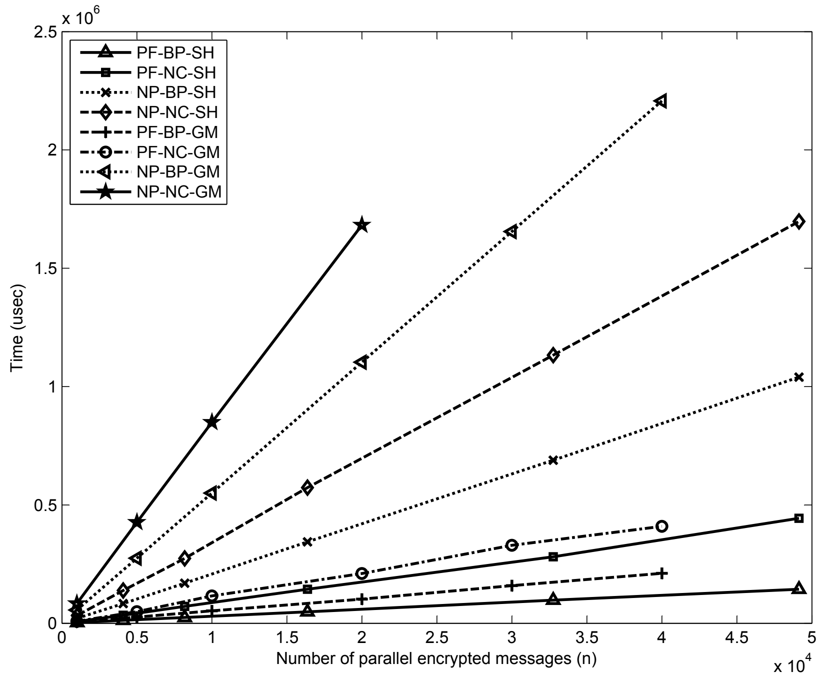


Figure B.7: Total time required to encrypt n messages in parallel

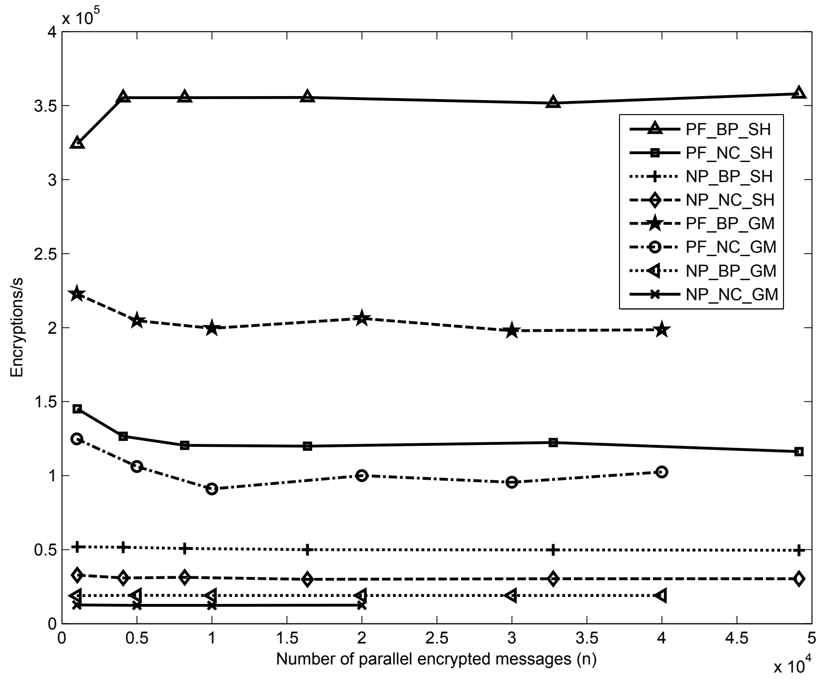


Figure B.8: The number of parallel encrypted messages per second.

Bibliography

[1] M. Agrawal, S. Karmakar, D. Saha, and D. Mukhopadhyay. Scan based side channel attacks on stream ciphers and their counter-measures. In *Proceedings of the International Conference on Cryptology in India (INDOCRYPT'08)*, volume 5365 of *Lecture Notes in Computer Science*, pages 226–238, Kharagpur, India, 2008. Springer.

[2] A. Akavia, S. Goldwasser, and V. Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *Proceedings of Theory of Cryptography Conference (TCC'09)*, volume 5444 of *Lecture Notes in Computer Science*, pages 474–495, San Francisco, CA, USA, 2009. Springer.

[3] M.-L. Akkar, R. Bevan, P. Dischamp, and D. Moyart. Power analysis, what is now possible... In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'00)*, volume 1976 of *Lecture Notes in Computer Science*, pages 489–502, Kyoto, Japan, 2000. Springer.

[4] M.-L. Akkar and C. Giraud. An implementation of DES and AES, secure against some attacks. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES'01)*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318, Paris, France, 2001. Springer.

[5] R. Álvarez, F. Ferrández, J.-F. Vicent, and A. Zamora. Applying quick exponentiation for block upper triangular matrices. *Applied Mathematics and Computation*, 183(2): 729–737, 2006.

[6] R. Álvarez, F. Martínez, J.-F. Vicent, and A. Zamora. Cryptographic applications of 3x3 block upper triangular matrices. In *Proceedings of the International Conference on Hybrid Artificial Intelligent Systems (HAIS'12)*, volume 7209 of *Lecture Notes in Computer Science*, pages 97–104, Salamanca, Spain, 2012. Springer.

[7] R. Álvarez, L. Tortosa, J. Vicent, and A. Zamora. A non-abelian group based on block upper triangular matrices with cryptographic applications. In *Proceedings of the International Symposium on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (AAECC'09)*, volume 5527 of *Lecture Notes in Computer Science*, pages 117–126, Tarragona, Catalonia, Spain, 2009. Springer.

[8] R. Álvarez, L. Tortosa, J.-F. Vicent, and A. Zamora. Analysis and design of a secure key exchange scheme. *Information Sciences*, 179(12):2014–2021, 2009.

[9] J. Alwen, Y. Dodis, and D. Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *Proceedings of the International Cryptology Conference (CRYPTO'09)*, volume 5677 of *Lecture Notes in Computer Science*, pages 36–54, Santa Barbara, CA, USA, 2009. Springer.

[10] R. J. Anderson, M. Bond, J. Clulow, and S. P. Skorobogatov. Cryptographic processors–a survey. *Proceedings of the IEEE*, 94(2):357–369, 2006.

[11] R. J. Anderson and M. G. Kuhn. Low cost attacks on tamper resistant devices. In *Proceedings of the International Workshop on Security Protocols*, volume 1361 of *Lecture Notes in Computer Science*, pages 125–136, Paris, France, 1997. Springer.

[12] A. C. Atici, L. Batina, J. Fan, I. Verbauwhede, and S. B. Örs. Low-cost implementations of NTRU for pervasive security. In *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors (ASAP'08)*, pages 79–84, Leuven, Belgium, 2008. IEEE Computer Society.

[13] A. C. Atici, L. Batina, B. Gierlichs, and I. Verbauwhede. Power analysis on NTRU implementation for RFIDs: First results. In *Proceedings of the Workshop on RFID Security (RFIDSec'08)*, pages 128–139, Budapest, Hungary, 2008.

[14] M. F. Atiyah and I. G. MacDonald. *Introduction to commutative algebra.* Westview Press, New York, NY, USA, 1969.

[15] D. V. Bailey, D. Coffin, A. J. Elbirt, J. H. Silverman, and A. D. Woodbury. NTRU in constrained devices. In *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems (CHES'01)*, volume 2162 of *Lecture Notes in Computer Science*, pages 262–272, Paris, France, 2001. Springer.

[16] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerers apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, 2006.

[17] G. M. Bergman. Some examples in PI ring theory. *Israel Journal of Mathematics*, 18(3): 257–277, 1974.

[18] D. L. Berre and A. Parrain. The SAT4J library, release 2.2. *Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

[19] D. L. Berre and O. Roussel. The international SAT competitions. http://www.satcompetition.org/, 2009.

[20] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri. Error analysis and detection procedures for a hardware implementation of the advanced encryption standard. *IEEE Transactions on Computers*, 52(4):492–505, 2003.

[21] A. Berzati, C. Canovas, J.-G. Dumas, and L. Goubin. Fault attacks on RSA public keys: Left-to-right implementations are also vulnerable. In *Proceedings of the Cryptographers' Track at the RSA Conference (CT-RSA'09)*, volume 5473 of *Lecture Notes in Computer Science*, pages 414–428, San Francisco, CA, USA, 2009. Springer.

[22] A. Berzati, C. Canovas, and L. Goubin. Perturbating RSA public keys: An improved attack. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES'08)*, volume 5154 of *Lecture Notes in Computer Science*, pages 380–395, Washington, DC, USA, 2008. Springer.

[23] I. Biehl, B. Meyer, and V. Müller. Differential fault attacks on elliptic curve cryptosystems. In *Proceedings of the International Cryptology Conference (CRYPTO'00)*, volume 1880 of *Lecture Notes in Computer Science*, pages 131–146, Santa Barbara, CA, USA, 2000. Springer.

[24] J. Biernat and M. Nikodem. Fault cryptanalysis of ElGamal signature scheme. In *Proceedings of the International Conference on Computer Aided Systems Theory (EUROCAST'05)*, volume 3643 of *Lecture Notes in Computer Science*, pages 327–336, Las Palmas de Gran Canaria, Spain, 2005. Springer.

[25] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In *Proceedings of the International Cryptology Conference (CRYPTO'97)*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525, Santa Barbara, CA, USA, 1997. Springer.

[26] I. F. Blake, G. Seroussi, and N. P. Smart. *Elliptic curves in cryptography*. London Mathematical Society Lecture Notes in Computer Science. Cambridge University Press, Cambridge, UK, 1999.

[27] J. Blömer and M. Otto. Wagner's attack on a secure CRT-RSA algorithm reconsidered. In *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC'06)*, volume 4236 of *Lecture Notes in Computer Science*, pages 13–23, Yokohama, Japan, 2006. Springer.

[28] J. Blömer, M. Otto, and J.-P. Seifert. A new CRT-RSA algorithm secure against Bellcore

attacks. In *Proceedings of the Conference on Computer and Communications Security (CCS'03)*, pages 311–320, Washington, DC, USA, 2003. ACM.

[29] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT'97)*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51, Konstanz, Germany, 1997. Springer.

[30] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of eliminating errors in cryptographic computations. *Cryptology*, 14(2):101–119, 2001.

[31] E. Brier, B. Chevallier-Mames, M. Ciet, and C. Clavier. Why one should also secure RSA public key elements. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES'06)*, volume 4249 of *Lecture Notes in Computer Science*, pages 324–338, Yokohama, Japan, 2006. Springer.

[32] B. Buchberger. Gröbner bases: An algorithmic method in polynomial ideal theory. In N. K. Bose and J. P. Guiver, editors, *Multidimensional systems theory: Progress, directions and open problems in multidimensional systems*, pages 184–232. Reidel Publishing Company, 1985.

[33] I. Buck, T. Foley, D. R. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. Brook for GPUs: Stream computing on graphics hardware. *ACM Transactions on Graphics*, 23(3):777–786, 2004.

[34] M. L. Bushnell and V. D. Agrawal. *Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits*. Kluwer Academic, Boston, MA, USA, 2000.

[35] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai. Exposure-resilient functions and all-or-nothing transforms. In *Proceedings of the International Conference on the*

Theory and Application of Cryptographic Techniques (EUROCRYPT'00), volume 1807 of *Lecture Notes in Computer Science*, pages 453–469, Bruges, Belgium, 2000. Springer.

[36] D. Cash, Y. Z. Ding, Y. Dodis, W. Lee, R. J. Lipton, and S. Walfish. Intrusion-resilient key exchange in the bounded retrieval model. In *Proceedings of Theory of Cryptography Conference (TCC'07)*, volume 4392 of *Lecture Notes in Computer Science*, pages 479–498, Amsterdam, Netherlands, 2007. Springer.

[37] J. C. Cha, K. H. Ko, S. Lee, J. W. Han, and J. H. Cheon. An efficient implementation of braid groups. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'01)*, volume 2248 of *Lecture Notes in Computer Science*, pages 144–156, Gold Coast, Australia, 2001. Springer.

[38] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *Proceedings of the International Cryptology Conference (CRYPTO'99)*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412, Santa Barbara, CA, USA, 1999. Springer.

[39] J. H. Cheon and B. Jun. A polynomial time algorithm for the braid Diffie-Hellman conjugacy problem. In *Proceedings of the International Cryptology Conference (CRYPTO'03)*, volume 2729 of *Lecture Notes in Computer Science*, pages 212–225, Santa Barbara, CA, USA, 2003. Springer.

[40] J.-J. Climent, P. R. Navarro, and L. Tortosa. On the arithmetic of the endomorphisms ring $End(\mathbb{Z}_p \times \mathbb{Z}_{p^2})$. *Applicable Algebra in Engineering, Communication and Computing*, 22 (2):91–108, 2011.

[41] Close to Metal. Technology unleashes the power of stream computing. http://www.amd.com/us/press-releases/Pages/Press_Release_114147.aspx, 2006.

[42] Consortium for Efficient Embedded Security. Efficient embedded security stan-

dard (EESS) #1: Implementation aspects of NTRUEncrypt and NTRUSign. http://grouper.ieee.org/groups/1363/lattPK/submissions/EESS1v2.pdf, 2003.

[43] D. L. Cook, J. Ioannidis, A. D. Keromytis, and J. Luck. CryptoGraphics: Secret key cryptography using graphics cards. In *Proceedings of the Cryptographers' Track at the RSA Conference (CT-RSA'05)*, volume 3376 of *Lecture Notes in Computer Science*, pages 334–350, San Francisco, CA, USA, 2005. Springer.

[44] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Symposium on Theory of Computing (STOC'71)*, pages 151–158, Shaker Heights, Ohio, USA, 1971. ACM.

[45] N. Courtois and G. V. Bard. Algebraic cryptanalysis of the data encryption standard. In *Proceedings of the International Conference on Cryptography and Coding*, volume 4887 of *Lecture Notes in Computer Science*, pages 152–169, Cirencester, UK, 2007. Springer.

[46] N. Courtois, G. V. Bard, and D. Wagner. Algebraic and slide attacks on KeeLoq. In *Proceedings of the International Workshop on Fast Software Encryption (FSE'08)*, volume 5086 of *Lecture Notes in Computer Science*, pages 97–115, Lausanne, Switzerland, 2008. Springer.

[47] N. Courtois, S. O'Neil, and J.-J. Quisquater. Practical algebraic attacks on the Hitag2 stream cipher. In *Proceedings of the International Conference on Information Security (ISC'09)*, volume 5735 of *Lecture Notes in Computer Science*, pages 167–176, Pisa, Italy, 2009. Springer.

[48] N. T. Courtois, K. Nohl, and S. O'Neil. Algebraic attacks on the Crypto-1 stream cipher in MiFare classic and Oyster cards. *International Association for Cryptologic Research (IACR)*, 166, 2008.

[49] J. Daemen and V. Rijmen. *The design of Rijndael: AES–The advanced encryption standard*. Springer, New York, NY, USA, 2002.

[50] M. Davis and H. Putnam. A computing procedure for quantification theory. *ACM*, 7(3): 201–215, 1960.

[51] D. De, A. Kumarasubramanian, and R. Venkatesan. Inversion attacks on secure hash functions using SAT solvers. In *Proceedings of the International Conference on Theory and applications of satisfiability testing (SAT'07)*, volume 4501 of *Lecture Notes in Computer Science*, pages 377–382, Lisbon, Portugal, 2007. Springer.

[52] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems. A practical implementation of the timing attack. In *Proceedings of the International Conference on Smart Card Research and Applications (CARDIS'98)*, volume 1820 of *Lecture Notes in Computer Science*, pages 167–182, Louvain-la-Neuve, Belgium, 1998. Springer.

[53] B. Driessen, A. Poschmann, and C. Paar. Comparison of innovative signature algorithms for WSNs. In *Proceedings of the Conference on Wireless Network Security (WISEC'08)*, pages 30–35, Alexandria, VA, USA, 2008. ACM.

[54] P. Dusart, G. Letourneux, and O. Vivolo. Differential fault analysis on AES. In *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS'03)*, volume 2846 of *Lecture Notes in Computer Science*, pages 293–306, Kunming, China, 2003. Springer.

[55] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518, Santa Margherita Ligure, Italy, 2003. Springer.

[56] T. Eibach, E. Pilz, and G. Völkel. Attacking Bivium using SAT solvers. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*

(SAT'08), volume 4996 of *Lecture Notes in Computer Science*, pages 63–76, Guangzhou, China, 2008. Springer.

[57] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[58] J. Erickson, J. Ding, and C. Christensen. Algebraic cryptanalysis of SMS4: Gröbner basis attack and SAT attack compared. In *Proceedings of the International Conference on Information, Security and Cryptology (ICISC'09)*, volume 5984 of *Lecture Notes in Computer Science*, pages 73–86, Seoul, Korea, 2009. Springer.

[59] Federal Information Processing Standards Publication (FIPS). Announcing the advanced encryption standard (AES). *National Institute of Standards and Technology (NIST)*, 197, 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[60] Federal Information Processing Standards Publication (FIPS). Security requirements for cryptographic modules. *National Institute of Standards and Technology (NIST)*, 140(2), 2001. http://http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf.

[61] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES'01)*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261, Paris, France, 2001. Springer.

[62] C. H. Gebotys and R. J. Gebotys. Secure elliptic curve implementations: An analysis of resistance to power-attacks in a DSP processor. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES'02)*, volume 2523 of *Lecture Notes in Computer Science*, pages 114–128, Redwood Shores, CA, USA, 2002. Springer.

[63] L. Genelle, E. Prouff, and M. Quisquater. Thwarting higher-order side channel analysis with additive and multiplicative maskings. In *Proceedings of the International Workshop*

on Cryptographic Hardware and Embedded Systems (CHES'11), volume 6917 of *Lecture Notes in Computer Science*, pages 240–255, Nara, Japan, 2011. Springer.

[64] C. Gentry, J. Jonsson, J. Stern, and M. Szydlo. Cryptanalysis of the NTRU signature scheme (NSS) from Eurocrypt 2001. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'01)*, volume 2248 of *Lecture Notes in Computer Science*, pages 1–20, Gold Coast, Australia, 2001. Springer.

[65] C. Gentry and M. Szydlo. Cryptanalysis of the revised NTRU signature scheme. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'02)*, volume 2332 of *Lecture Notes in Computer Science*, pages 299–320, Amsterdam, Netherlands, 2002. Springer.

[66] C. Giraud, E. W. Knudsen, and M. Tunstall. Improved fault analysis of signature schemes. In *Proceedings of the International Conference on Smart Card Research and Advanced Application (CARDIS'10)*, volume 6035 of *Lecture Notes in Computer Science*, pages 164–181, Passau, Germany, 2010. Springer.

[67] E. I. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT-solver. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE'02)*, pages 142–149, Paris, France, 2002. IEEE Computer Society.

[68] O. Goldreich, S. Goldwasser, and S. Halevi. Public-key cryptosystems from lattice reduction problems. In *Proceedings of the International Cryptology Conference (CRYPTO'97)*, volume 1294 of *Lecture Notes in Computer Science*, pages 112–131, Santa Barbara, CA, USA, 1997. Springer.

[69] J. D. Golic and C. Tymen. Multiplicative masking and power analysis of AES. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Sys-*

tems (CHES'02), volume 2523 of *Lecture Notes in Computer Science*, pages 198–212, Redwood Shores, CA, USA, 2002. Springer.

[70] L. Goubin. A sound method for switching between Boolean and arithmetic masking. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES'01)*, volume 2162 of *Lecture Notes in Computer Science*, pages 3–15, Paris, France, 2001. Springer.

[71] L. Goubin and A. Martinelli. Protecting AES with Shamir's secret sharing scheme. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES'11)*, volume 6917 of *Lecture Notes in Computer Science*, pages 79–94, Nara, Japan, 2011. Springer.

[72] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *Proceedings of the USENIX Security Symposium*, pages 45–60, San Jose, CA, USA, 2008. USENIX Association.

[73] O. Harrison and J. Waldron. AES encryption implementation and analysis on commodity graphics processing units. In *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems (CHES'07)*, volume 4727 of *Lecture Notes in Computer Science*, pages 209–226, Vienna, Austria, 2007. Springer.

[74] D. Hély, F. Bancel, M.-L. Flottes, and B. Rouzeyre. Secure scan techniques: A comparison. In *Proceedings of the International On-Line Testing Symposium (IOLTS'06)*, pages 119–124, Como, Italy, 2006. IEEE.

[75] N. Heninger and H. Shacham. Reconstructing RSA private keys from random key bits. In *Proceedings of the International Cryptology Conference (CRYPTO'09)*, volume 5677 of *Lecture Notes in Computer Science*, pages 1–17, Santa Barbara, CA, USA, 2009. Springer.

[76] J. Hermans, F. Vercauteren, and B. Preneel. Speed records for NTRU. In *Proceedings of the Cryptographers' Track at the RSA Conference (CT-RSA'10)*, volume 5985 of *Lecture Notes in Computer Science*, pages 73–88, San Francisco, CA, USA, 2010. Springer.

[77] M. Heule and H. van Maaren. March_dl: Adding adaptive heuristics and a new branching strategy. *Satisfiability, Boolean Modeling and Computation*, 2(1-4):47–59, 2006.

[78] A. Hevia and M. A. Kiwi. Strength of two data encryption standard implementations under timing attacks. *ACM Transactions on Information and System Security*, 2(4):416–437, 1999.

[79] J. J. Hoch and A. Shamir. Fault analysis of stream ciphers. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES'04)*, volume 3156 of *Lecture Notes in Computer Science*, pages 240–253, Cambridge, MA, USA, 2004. Springer.

[80] J. Hoffstein, N. Howgrave-Graham, J. Pipher, J. H. Silverman, and W. Whyte. NTRUSign: Digital signatures using the NTRU lattice. Technical report, Security Innovation Company, April 2002.

[81] J. Hoffstein, N. Howgrave-Graham, J. Pipher, J. H. Silverman, and W. Whyte. NTRUSign: Digital signatures using the NTRU lattice. In *Proceedings of the Cryptographers' Track at the RSA Conference (CT-RSA'03)*, volume 2612 of *Lecture Notes in Computer Science*, pages 122–140, San Francisco, CA, USA, 2003. Springer.

[82] J. Hoffstein, N. Howgrave-Graham, J. Pipher, J. H. Silverman, and W. Whyte. Performance improvements and a baseline parameter generation algorithm for NTRUSign. *International Association for Cryptologic Research (IACR)*, 274, 2005.

[83] J. Hoffstein, N. Howgrave-Grahama, J. Pipher, and W. Whyte. Practical lattice-based cryptography: NTRUEncrypt and NTRUSign. In P. Q. Nguyen and B. Vallée, editors, *The LLL algorithm survey and applications*, pages 349–390. Springer, 2010.

[84] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In *Proceedings of the Third International Symposium on Algorithmic Number Theory (ANTS-III'98)*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288, Portland, Oregon, USA, 1998. Springer.

[85] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A public key cryptosystem. Avilable at http://grouper.ieee.org/groups/1363/lattPK/ submissions/ntru.pdf, 1999.

[86] J. Hoffstein, J. Pipher, and J. H. Silverman. NSS: An NTRU lattice-based signature scheme. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT'01)*, volume 2045 of *Lecture Notes in Computer Science*, pages 211–228, Innsbruck, Austria, 2001. Springer.

[87] J. Hoffstein, J. Pipher, and J. H. Silverman. *An introduction to mathematical cryptography*. Undergraduate Texts in Mathematics. Springer, New York, NY, USA, 2008.

[88] J. Hoffstein and J. Silverman. Optimizations for NTRU. In *Proceedings of the Conference of Public-Key Cryptography and Computational Number Theory*, pages 77–88, Warsaw, Poland, 2000. De Gruyter.

[89] J. Hoffstein and J. H. Silverman. Random small hamming weight products with applications to cryptography. *Discrete Applied Mathematics*, 130(1):37–49, 2003.

[90] N. Howgrave-Graham, J. H. Silverman, A. Singer, and W. Whyte. NAEP: Provable security in the presence of decryption failures. *International Association for Cryptologic Research (IACR)*, 172, 2003.

[91] Y.-J. Huang, F.-H. Liu, and B.-Y. Yang. Public-key cryptography from new multivariate quadratic assumptions. *International Association for Cryptologic Research (IACR)*, 273, 2012.

[92] J. Hughes. A linear algebraic attack on the AAFG1 braid group cryptosystem. In *Proceedings of the Australian Conference on Information Security and Privacy (ACISP'02)*, volume 2384 of *Lecture Notes in Computer Science*, pages 176–189, Melbourne, Australia, 2002. Springer.

[93] IEEE Std 1149.7-2009. IEEE standard for reduced-pin and enhanced-functionality test access port and boundary-scan architecture, 2010.

[94] Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. In *Proceedings of the International Conference on Advances in Cryptology (CRYPTO'03)*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481, Santa Barbara, CA, USA, 2003. Springer.

[95] É. Jaulmes and A. Joux. A chosen-ciphertext attack against NTRU. In *Proceedings of the International Cryptology Conference (CRYPTO'00)*, volume 1880 of *Lecture Notes in Computer Science*, pages 20–35, Santa Barbara, CA, USA, 2000. Springer.

[96] M. Joye and M. Ciet. Practical fault countermeasures for Chinese remaindering based RSA. In *Proceedings of the International Workshop on Fault Diagnosis and Tolerance (FDTC'05)*, Lecture Notes in Computer Science, pages 124–131, Edinburgh, Scotland, UK, 2005. Springer.

[97] B. Kaliski. Considerations for new public-key algorithms. *Network Security*, 2000(9): 9–10, 2000.

[98] A. G. Kalka. Representation attacks on the braid Diffie-Hellman public key encryption. *Applicable Algebra in Engineering, Communication and Computing*, 17(3–4):257–266, 2006.

[99] A. Kamal and A. M. Youssef. An area optimized implementation of the advanced encryption standard. In *Proceedings of the International Conference on Microelectronics (ICM'08)*, pages 159–162, Sharjah, UAE, 2008. IEEE.

[100] A. Kamal and A. M. Youssef. An area-optimized implementation for AES with hybrid countermeasures against power analysis. In *Proceedings of the International Symposium on Signals, Circuits and Systems (ISSCS'09)*, pages 1–4, Iasi, Romania, 2009. IEEE.

[101] A. Kamal and A. M. Youssef. An FPGA implementation of AES with fault analysis countermeasures. In *Proceedings of the International Conference on Microelectronics (ICM'09)*, pages 217–220, Marrakech, Morocco, 2009. IEEE.

[102] A. Kamal and A. M. Youssef. An FPGA implementation of the NTRUEncrypt cryptosystem. In *Proceedings of the International Conference on Microelectronics (ICM'09)*, pages 209–212, Marrakech, Morocco, 2009. IEEE.

[103] A. Kamal and A. M. Youssef. Applications of SAT solvers to AES key recovery from decayed key schedule images. In *Proceedings of the International Conference on Emerging Security Information, Systems and Technologies (SECURWARE'10)*, pages 216–220, Venice/Mestre, Italy, 2010. IEEE Computer Society.

[104] A. Kamal and A. M. Youssef. Enhanced implementation of the NTRUEncrypt algorithm using graphics cards. In *Proceedings of the International Conference on Parallel, Distributed and Grid Computing (PDGC'10)*, pages 168–174, Solan, India, 2010. IEEE.

[105] A. Kamal and A. M. Youssef. Fault analysis of the NTRUEncrypt cryptosystem. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E94-A(4):1156–1158, 2011.

[106] A. Kamal and A. M. Youssef. Cryptanalysis of a $GL(r, \mathbb{Z}_n)$-based public key system. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E95-A(4):829–831, 2012.

[107] A. Kamal and A. M. Youssef. Cryptanalysis of a key exchange protocol based on the endomorphisms ring $End(\mathbb{Z}_p \times \mathbb{Z}_{p^2})$. *Applicable Algebra in Engineering, Communication and Computing*, 2012. Published Online.

[108] A. Kamal and A. M. Youssef. Fault analysis of the NTRUSign digital signature scheme. *Cryptography and Communications, Discrete Structures, Boolean Functions and Sequences*, 4(2):131–144, 2012.

[109] A. Kamal and A. M. Youssef. A scan-based side channel attack on the NTRUEncrypt cryptosystem. In *Proceedings of the International Workshop on Modern Cryptography and Security Engineering (MoCrySEN'12)*, Prague, Czech Republic, 2012. IEEE. Accepted.

[110] J.-P. Kaps. *Cryptography for ultra-low power devices*. PhD thesis, Worcester Polytechnic Institute, MA, USA, 2006.

[111] C. H. Kim and J.-J. Quisquater. Fault attacks for CRT based RSA: New attacks, new results, and new countermeasures. In *Proceedings of the International Workshop on Information Security Theory and Practices (WISTP'07)*, volume 4462 of *Lecture Notes in Computer Science*, pages 215–228, Heraklion, Crete, Greece, 2007. Springer.

[112] H. Kim, S. Hong, and J. Lim. A fast and provably secure higher-order masking of AES s-box. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES'11)*, volume 6917 of *Lecture Notes in Computer Science*, pages 95–107, Nara, Japan, 2011. Springer.

[113] K. H. Ko, J. W. Lee, and T. Thomas. Towards generating secure keys for braid cryptography. *International Association for Cryptologic Research (IACR)*, 149, 2007.

[114] K. H. Ko, S.-J. Lee, J. H. Cheon, J. W. Han, J.-S. Kang, and C. Park. New public-key cryptosystem using braid groups. In *Proceedings of the International Cryptology Conference (CRYPTO'00)*, volume 1880 of *Lecture Notes in Computer Science*, pages 166–183, Santa Barbara, CA, USA, 2000. Springer.

[115] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems.

[116] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Proceedings of the International Cryptology Conference (CRYPTO'99)*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397, Santa Barbara, CA, USA, 1999. Springer.

[117] I. Koren and C. M. Krishna. *Fault-tolerant systems*. Morgan Kaufmann, San Francisco, CA, USA, 2007.

[118] E. Lee and J. H. Park. Cryptanalysis of the public-key encryption based on braid groups. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'03)*, volume 2656 of *Lecture Notes in Computer Science*, pages 477–490, Warsaw, Poland, 2003. Springer.

[119] M.-K. Lee, J. E. Song, D. Choi, and D.-G. Han. Countermeasures against power analysis attacks for the NTRU public key cryptosystem. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E93-A(1):153–163, 2010.

[120] Y. Liu, K. Wu, and R. Karri. Scan-based attacks on linear feedback shift register based stream ciphers. *ACM Transactions on Design Automation of Electronic Systems*, 16(2): 1–15, 2011.

[121] T. Malkin, F.-X. Standaert, and M. Yung. A comparative cost/security analysis of fault attack countermeasures.

[122] S. A. Manavski. CUDA compatible GPU as an efficient hardware accelerator for AES cryptography. In *Proceedings of the International Conference on Signal Processing and Communications (ICSPC'07)*, pages 65–68, Dubai, UAE, 2007. IEEE.

[123] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. Technical report, Jet Propulsion Laboratory, February 1978.

[124] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of applied cryptography*. Discrete Mathematics and its Applications. CRC Press, Boca Raton, FL, USA, 1996.

[125] A. Menezes and Y.-H. Wu. The discrete logarithm problem in $GL(n, q)$. *Ars Combinatoria*, 47:23–32, 1997.

[126] T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Examining smart-card security under the threat of power analysis attacks. *IEEE Transactions on Computers*, 51(5):541–552, 2002.

[127] S. Micali and L. Reyzin. Physically observable cryptography (extended abstract). In *Proceedings of Theory of Cryptography Conference (TCC'04)*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296, Cambridge, MA, USA, 2004. Springer.

[128] S. Min, G. Yamamoto, and K. Kim. Weak property of malleability in NTRUSign. In *Proceedings of the Australasian Conference on Information Security and Privacy (ACISP'04)*, volume 3108 of *Lecture Notes in Computer Science*, pages 379–390, Sydney, Australia, 2004. Springer.

[129] I. Mironov and L. Zhang. Applications of SAT solvers to cryptanalysis of hash functions. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT'06)*, volume 4121 of *Lecture Notes in Computer Science*, pages 102–115, Seattle, WA, USA, 2006. Springer.

[130] M. Monteverde. NTRU software implementation for constrained devices. Master's thesis, Katholike Universiteit, Leuven, Belgium, 2006.

[131] M. Monteverde. NTRU software implementation for constrained devices. Master's thesis, Department of Electrical Engineering–ESAT, Katholieke Universiteit Leuven, Leuven, Belgium, 2008.

[132] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the Design Automation Conference (DAC'01)*, pages 530–535, Las Vegas, NV, USA, 2001. ACM.

[133] J. A. Muir. Seifert's RSA fault attack: Simplified analysis and generalizations. In *Proceedings of the International Conference on Information and Communications Security (ICICS'06)*, volume 4307 of *Lecture Notes in Computer Science*, pages 420–434, Raleigh, NC, USA, 2006. Springer.

[134] C. Mullan. Cryptanalysing variants of Stickel's key agreement protocol. *Mathematical Cryptology*, 4(4):365–373, 2011.

[135] C. Mullan. *Some results in group-based cryptography*. PhD thesis, Department of Mathematics Royal Holloway, University of London, Egham, Surrey, UK, 2011.

[136] T. Müller, F. C. Freiling, and A. Dewald. TRESOR runs encryption securely outside RAM. In *Proceedings of the USENIX Security Symposium*, San Francisco, CA, USA, 2011. USENIX Association.

[137] A. Munshi. OpenCL parallel computing on the GPU and CPU. http://s08.idav.ucdavis.edu/munshi-opencl.pdf, 2008.

[138] A. Myasnikov, V. Shpilrain, and A. Ushakov. *Non-commutative cryptography and complexity of group-theoretic problems*. Mathematical Surveys and Monographs. American Mathematical Society, Providence, RI, USA, 2011.

[139] R. Nara, K. Satoh, M. Yanagisawa, T. Ohtsuki, and N. Togawa. Scan-based side-channel attack against RSA cryptosystems using scan signatures. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E93-A(12):2481–2489, 2010.

[140] R. Nara, N. Togawa, M. Yanagisawa, and T. Ohtsuki. A scan-based attack based on discriminators for AES cryptosystems. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E92-A(12):3229–3237, 2009.

[141] R. Nara, N. Togawa, M. Yanagisawa, and T. Ohtsuki. Scan-based attack against elliptic curve cryptosystems. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'10)*, pages 407–412, Taipei, Taiwan, 2010. IEEE.

[142] P. Q. Nguyen and D. Pointcheval. Analysis and improvements of NTRU encryption paddings. In *Proceedings of the International Cryptology Conference (CRYPTO'02)*, volume 2442 of *Lecture Notes in Computer Science*, pages 210–225, Santa Barbara, CA, USA, 2002. Springer.

[143] P. Q. Nguyen and O. Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'06)*, volume 4004 of *Lecture Notes in Computer Science*, pages 271–288, St. Petersburg, Russia, 2006. Springer.

[144] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15(2):159–166, 1986.

[145] NTRU Cryptosystems. The NTRU public key cryptosystem–a tutorial. http://www.securityinnovation.com/security-lab/crypto/155.html, 1998.

[146] NVIDIA Developer Zone. Introducing CUDA. http://developer.nvidia.com/category/zone/cuda-zone, 2010.

[147] R. Odoni, vijay varadharajan, and P. W. Sanders. Public key distribution in matrix rings. *IEE Electronics Letters*, 20(9):386–387, 1984.

[148] L. Ordu and S. B. Ors. Power analysis resistant hardware implementations of AES. In *Proceedings of the International Conference on Electronics, Circuits and Systems (ICECS'07)*, pages 1408–1411, Marrakech, Morocco, 2007. IEEE.

[149] G. L. Osa. Fast implementation of two hash algorithms on Nvidia CUDA GPU. Mas-

ter's thesis, Department of Telematics, Norwegian University of Science and Technology, Trondheim, Norway, 2009.

[150] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen. A side-channel analysis resistant description of the AES s-box. In *Proceedings of the International Workshop on Fast Software Encryption (FSE'05)*, volume 3557 of *Lecture Notes in Computer Science*, pages 413–423, Paris, France, 2005. Springer.

[151] A. Ottesen. Efficient parallelisation techniques for applications running on GPUs using the CUDA framework. Master's thesis, Department of Informatics, University of Oslo, Oslo, Norway, 2009.

[152] D. Page. Theoretical use of cache memory as a cryptanalytic sidechannel. Technical report, Computer Science Department, University of Bristol, June 2002.

[153] J. Pipher. Lectures on the NTRU encryption algorithm and digital signature scheme. http://www.math.brown.edu/ jpipher/grenoble.pdf, 2002.

[154] G. Piret and J.-J. Quisquater. A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES'03)*, volume 2779 of *Lecture Notes in Computer Science*, pages 77–88, Cologne, Germany, 2003. Springer.

[155] E. Prouff and T. Roche. Higher-order glitches free implementation of the AES using secure multi-party computation protocols. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES'11)*, volume 6917 of *Lecture Notes in Computer Science*, pages 63–78, Nara, Japan, 2011. Springer.

[156] J.-J. Quisquater and D. Samyde. ElectroMagnetic analysis (EMA): Measures and counter-measures for smart cards. In *Proceedings of the International Conference on Research in Smart Cards (E-smart'01)*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210, Cannes, France, 2001. Springer.

[157] M. Rivain, E. Dottax, and E. Prouff. Block ciphers implementations provably secure against second order side channel analysis. In *Proceedings of the International Workshop on Fast Software Encryption (FSE'08)*, volume 5086 of *Lecture Notes in Computer Science*, pages 127–143, Lausanne, Switzerland, 2008. Springer.

[158] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[159] U. Rosenberg. Using graphic processing unit in block cipher calculations. Master's thesis, Institute of Computer Science, University of Tartu, Tartu, Estonia, 2007.

[160] W. Schindler. A timing attack against RSA with the Chinese remainder theorem. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES'00)*, volume 1965 of *Lecture Notes in Computer Science*, pages 109–124, Worcester, MA, USA, 2000. Springer.

[161] J.-M. Schmidt and C. Herbst. A practical fault attack on square and multiply. In *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC'08)*, pages 53–58, Washington, DC, USA, 2008. IEEE Computer Society.

[162] K. Schramm and C. Paar. Higher order masking of the AES. In *Proceedings of the Cryptographers' Track at the RSA Conference (CT-RSA'06)*, volume 3860 of *Lecture Notes in Computer Science*, pages 208–225, San Jose, CA, USA, 2006. Springer.

[163] J.-P. Seifert. On authenticated computing and RSA-based authentication. In *Proceedings of the Conference on Computer and Communications Security (CCS'05)*, Lecture Notes in Computer Science, pages 122–127, Alexandria, VA, USA, 2005. ACM.

[164] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: A many-core x86 architecture for visual computing. *ACM Transactions on Graphics*, 27 (3):18:1–18:15, 2008.

[165] A. Shamir. *Method and apparatus for protecting public key schemes from timing and fault attacks*. US Patent No. 5991415, 23 November 1999.

[166] A. Shamir and E. Tromer. Acoustic cryptanalysis: On nosy people and noisy machines. In *Proceedings of the EUROCRYPT ramp session*, Interlaken, Switzerland, 2004.

[167] Y. Shi, N. Togawa, M. Yanagisawa, and T. Ohtsuki. Robust secure scan design against scan-based differential cryptanalysis. *IEEE Transactions on Very Large Scale Integration Systems*, 20(1):176–181, 2012.

[168] V. Shpilrain. Cryptanalysis of Stickel's key exchange scheme. In *Proceedings of the International Computer Science Symposium in Russia (CSR08)*, volume 5010 of *Lecture Notes in Computer Science*, pages 283–288, Moscow, Russia, 2008. Springer.

[169] J. H. Silverman. Invertibility in truncated polynomial rings. Technical report, Security Innovation Company, October 1998.

[170] J. H. Silverman. Almost inverses and fast NTRU key creation. Technical report, Security Innovation Company, March 1999.

[171] J. H. Silverman and W. Whyte. Timing attacks on NTRUEncrypt via variation in the number of hash calls. In *Proceedings of the Cryptographers' Track at the RSA Conference (CT–RSA'07)*, volume 4377 of *Lecture Notes in Computer Science*, pages 208–224, San Francisco, CA, USA, 2007. Springer.

[172] R. Singel. Declassified NSA document reveals the secret history of TEMPES. http://www.wired.com/threatlevel/2008/04/nsa-releases-se/, 2008.

[173] S. Skorobogatov. Low-temperature data remanence in static RAM. Technical report, University of Cambridge Computer Laboratory, June 2002.

[174] S. P. Skorobogatov and R. J. Anderson. Optical fault induction attacks. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems*

(CHES'02), volume 2523 of *Lecture Notes in Computer Science*, pages 2–12, Redwood Shores, CA, USA, 2002. Springer.

[175] K. Slavin. *Public key cryptography using matrices*. US Patent No. 7346162, 18 March 2008.

[176] M. Soos, K. Nohl, and C. Castelluccia. Extending SAT solvers to cryptographic problems. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257, Swansea, UK, 2009. Springer.

[177] M. Sramka. On the security of Stickel's key exchange scheme.

[178] E. Stickel. A new method for exchanging secret keys. In *Proceedings of the International Conference on Information Technology and Applications (ICITA05)*, pages 426–430, Sydney, Australia, 2005. IEEE Computer Society.

[179] R. Szerwinski. Efficient cryptography on graphics hardware. Master's thesis, Department of Electrical Engineering and Information Sciences, Ruhr-University of Bochum, Bochum, Germany, 2008.

[180] R. Szerwinski and T. Güneysu. Exploiting the power of GPUs for asymmetric cryptography. In *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems (CHES'08)*, volume 5154 of *Lecture Notes in Computer Science*, pages 79–99, Washington, DC, USA, 2008. Springer.

[181] M. Szydlo. Hypercubic lattice reduction and analysis of GGH and NTRU signatures. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'03)*, volume 2656 of *Lecture Notes in Computer Science*, pages 433–448, Warsaw, Poland, 2003. Springer.

[182] E. Trichina, D. D. Seta, and L. Germani. Simplified adaptive multiplicative masking for AES. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES'02)*, volume 2523 of *Lecture Notes in Computer Science*, pages 187–197, Redwood Shores, CA, USA, 2002. Springer.

[183] B. Tsaban. Combinatorial group theory and cryptography bulletin (CGC bulletin). http://u.cs.biu.ac.il/ tsaban/CGC/cgc.html.

[184] A. Tsow. An improved recovery algorithm for decayed AES key schedule images. In *Proceedings of the International Workshop on Selected Areas in Cryptography (SAC'09)*, volume 5867 of *Lecture Notes in Computer Science*, pages 215–230, Calgary, Alberta, Canada, 2009. Springer.

[185] Y. Tsunoo, E. Tsujihara, K. Minematsu, and H. Miyauchi. Cryptanalysis of DES implemented on computers with cache.

[186] Y. Tsunoo, E. Tsujihara, K. Minematsu, and H. Miyauchi. Cryptanalysis of block ciphers implemented on computers with cache. In *Proceedings of the International Symposium on Information Theory and Its Applications (ISITA'02)*, pages 803–806, Xian, China, 2002. IEEE Information Theory Society.

[187] M. I. G. Vasco, A. L. P. del Pozo, and P. T. Duarte. Cryptanalysis of a key exchange scheme based on block matrices. *International Association for Cryptologic Research (IACR)*, 553, 2009.

[188] W. Wang and T. Stransky. Stateless key distribution for secure intra and inter-group multicast in mobile wireless network.

[189] K. Wilhelm. Aspects of hardware methodologies for the NTRU public key cryptosystem. Master's thesis, Kate Gleason College of Engineering, Rochester Institute of Technology, Rochester, NY, USA, 2008.

[190] P. Wright and P. Greengrass. *Spycatcher: The candid autobiography of a senior intelligence officer*. Viking Adult Press, New York, NY, USA, 1987.

[191] C.-K. Wu and E. Dawson. Generalized inverses in public key cryptosystem design. *IEE Computers and Digital Techniques*, 145(5):321–326, 1998.

[192] B. Yang, K. Wu, and R. Karri. Scan based side channel attack on dedicated hardware implementations of data encryption standard. In *Proceedings of the International Test Conference (ITC'04)*, pages 339–344, Charlotte, NC, USA, 2004. IEEE.

[193] B. Yang, K. Wu, and R. Karri. Secure scan: A design-for-test architecture for crypto chips. In *Proceedings of the Design Automation Conference (DAC'05)*, pages 135–140, San Diego, CA, USA, 2005. IEEE.

[194] S.-M. Yen, S. Kim, S. Lim, and S.-J. Moon. RSA speedup with Chinese remainder theorem immune against hardware fault cryptanalysis. *IEEE Transaction on Computers*, 52(4):461–472, 2003.

[195] B. Zakeri, M. Salmasizadeh, A. Moradi, M. Tabandeh, and M. T. M. Shalmani. Compact and secure design of masked AES s-box. In *Proceedings of the International Conference on Information and Communications Security (ICICS'07)*, volume 4861 of *Lecture Notes in Computer Science*, pages 216–229, Zhengzhou, China, 2007. Springer.

[196] Y. Zhou and D. Feng. Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing. *International Association for Cryptologic Research (IACR)*, 388, 2005.