Klepto for post-quantum encryption

Kwant, R.J.

Award date:
2016

Bachelor Final Project

Klepto for Post-Quantum Encryption

Author:
Robin Kwant

Supervisor:
Prof. Dr. Tanja Lange

Eindhoven University of Technology
Department of Mathematics and Computer Science

August 22, 2016

Abstract

The quantum future, a future in which full scale quantum computers exist, seems apparent. In this future most cryptography currently used for online communications will be completely broken. This gives rise to the field of post-quantum cryptography, cryptography resistant to attacks of both classical and quantum computers. Young and Yung initiated the field of cryptography with backdoors, called kleptography. They showed for the most commonly used schemes that one should not blindly trust black box implementations in terms of security, as they might contain backdoors. This thesis will show that they can also exist in a post-quantum cryptosystem.

Contents

Chapter 1

Introduction

Online communication relies on cryptography for security and privacy such that parties can converse without eavesdroppers. If a quantum computer is ever built, those who have access to it will likely have the ability to eavesdrop on encrypted communications. In order to prevent this, many advancements in cryptography have been made in the past decades. Implementations of this new kind of cryptography can however contain backdoors and subliminal channels.

1.1 Motivation

Most of current online communications are encrypted in such a way that an adversary could break them using a quantum computer. Fast algorithms for quantum cryptanalysis have already been published by Shor in 1994 [14]. Full scale quantum computers do not exist yet as of today, but post-quantum cryptography does and already has applications. Cryptography resistant to quantum computers is used primarily to encrypt information that should remain secure for a longer period of time than just a few years. In this application, post-quantum cryptography is a better choice than cryptography that gets broken by quantum computers, because an adversary could record messages over the years and wait until a quantum computer is built to break them. Research has shown that black box implementations of commonly used cryptographic schemes can contain backdoors and subliminal channels. For general security it is important to investigate whether or not this is also feasible in post-quantum cryptography and to what extent.

1.2 Outline

This thesis first gives some mathematical background, a general introduction to cryptography and kleptography. Then the post-quantum encryption scheme NTRU is described after which modifications with a backdoor and a subliminal channel are described. In the end conclusions will be presented and recommendations for future research will be given.

Chapter 2

Background information

This section provides some background on the necessary mathematics and cryptography used in the rest of this thesis. Most of the mathematics is based on [4, Chapters 2,3,4,7] and [3], in which mathematical proofs can be found.

2.1 Modular Arithmetic

Definition 2.1.1 (Divisibility)**.** *Let $a, b \in \mathbb{Z}$. The integer b is called a multiple of a if and only if some integer n exists such that $b = a \cdot n$. The integer a is called a divisor of b if and only if some integer m exists such that $a \cdot m = b$. If a is a divisor of b, this is denoted as $a|b$.*

Definition 2.1.2 (Congruence)**.** *Let $a, b, n \in \mathbb{Z}$, a and b are congruent modulo n if and only if $n|(a - b)$. This is denoted as $a \equiv b \mod n$.*

Definition 2.1.3 (Congruence classes)**.** *Let $n \in \mathbb{Z}$. A congruence class modulo n an equivalence class modulo n. The set of all congruence classes modulo n is denoted by $\mathbb{Z}/n$ or $\mathbb{Z}_n$.*
Define n distinct equivalence classes as $n \cdot \mathbb{Z}, 1 + n \cdot \mathbb{Z}, ..., n - 1 + n \cdot \mathbb{Z}$.

2.2 Rings

Definition 2.2.1 (Ring)**.** *A ring is a set of elements R with the operations addition $(+)$ and multiplication $(\cdot)$ for which the following ring axioms hold.*

Addition $(+)$:
$\forall a, b, c \in R$

- *R is closed under addition, meaning $a + b \in R$.*

- *Addition is associative and commutative meaning* $(a+b)+c = a+(b+c)$ *(associativity) and* $a + b = b + a$ *(commutativity).*
- *There exists a neutral element* $0 \in R$ *such that* $a + 0 = a$.
- *Every element* a *has an inverse* $-a \in R$ *under addition such that* $a + (-a) = 0$.

Multiplication $(\cdot)$ *:*
$\forall a, b, c \in R$

- R *is closed under multiplication, meaning* $a \cdot b \in R$.
- *Multiplication is associative meaning* $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.
- *There exists a neutral element* $1 \in R$ *such that* $a \cdot 1 = 1 \cdot a = a$.
- *Multiplication is distributive with respect to addition meaning* $a \cdot (b + c) = a \cdot b + a \cdot c$ *and* $(a + b) \cdot c = a \cdot c + b \cdot c$.

Definition 2.2.2 (Field)**.** *A field* K *is a ring in which*

- *Multiplication is commutative meaning* $a \cdot b = b \cdot a$ *for* $a, b \in K$.
- *Every element* $a \in K \backslash \{0\}$ *has an inverse* $b \in K$ *under multiplication, such that* $a \cdot b \equiv 1$.

2.3 Polynomials

Definition 2.3.1 (Polynomial)**.** *A polynomial of degree* n *over a ring* R *in the indeterminate* X *is defined as:*
$\sum_{i=0}^{n} a_i \cdot X^i$ *with* $n \in \mathbb{N}$ *and* $a_i \in R$ *for all* $i \in \{0, \ldots, n\}$ *and* $a_n \neq 0$*. The elements* $a_0, ..., a_n$ *are called the coefficients of the polynomial. The degree of a polynomial is the exponent of the nonzero term with the largest exponent. The degree of the polynomial* 0 *is defined as* $-\infty$.

Definition 2.3.2 (Addition)**.** *The sum of polynomials* a *and* b *of degrees* n, m *with coefficients* a_j, b_i *respectively and* $m \geq n$ *is defined as:*

$$a + b = \sum_{i=0}^{m} (a_i + b_i) \cdot X^i,$$

where $\forall_{n<j\leq m}\ a_j = 0$.

Definition 2.3.3 (Multiplication)**.** *The product of two polynomials a and b of degrees n, m with coefficients a_i, b_i respectively and $m \geq n$ is defined as:*

$$a \cdot b = \sum_{i=0}^{n+m} (\sum_{j=0}^{i} a_j \cdot b_{i-j}) \cdot X^i,$$

where $a_j = 0$ for $j > n$ and $b_i = 0$ for $i > m$.

Lemma 2.3.1. *The set of polynomials $R[X]$ with multiplication and addition as defined above forms a ring.*

Definition 2.3.4 (Polynomial Ring)**.** *The ring $R[X]$ is called the polynomial ring over R.*

Definition 2.3.5 (Congruence)**.** *Let R be a ring and $f, g, p \in R[X]$ be polynomials. The polynomials f and g are congruent modulo p if and only if $p(X)|(f(X) - g(X))$. This is denoted by $f \equiv g \mod p$.*

Definition 2.3.6 (Congruence classes)**.** *Let R be a ring and f, g, p be polynomials in $R[X]$. The congruence class $f(X)$ modulo $p(X)$ denoted as $[f(X)]_{p(X)}$ is defined by:*

$$[f(x)]_{p(X)} = \{g(x) \in R[X] | g(X) \equiv f(X) \mod p(X).\}$$

The set of congruence classes of polynomials in $R[X]$ modulo $p(X)$ is denoted as $R[X]/p(X)$.

Definition 2.3.7 (Inverse)**.** *Let $f \in R[X]$. The polynomial f is called invertible in $R[X]$ if and only if a polynomial $f^{-1} \in R[X]$ exists such that $f \cdot f^{-1} = 1$. The polynomial f^{-1} is called the inverse of f.*

2.4 Cryptography

The term *cryptography* refers to the art of writing messages in such a way that they are readable by the intended recipients and at the same time unreadable by any third party. In cryptography third parties are often called adversaries. Cryptography is vital in modern day society as it makes secure online communications possible. Most of this security is ensured by using *encryption*. Encryption is the conversion of a so called *plaintext* which is regular readable text, into a *ciphertext* which is text that appears gibberish but can be reverted back to the plaintext by the intended party. A specific way of encrypting information is often called an *encryption scheme* or *cipher*.

2.4.1 Symmetric encryption

Symmetric encryption is encryption in which the same key k is used for both encryption and decryption. In the case that Alice sends a message to Bob, Alice encrypts the plaintext using some encryption function E_k. Bob receives the ciphertext and is able to recover the

plaintext using his decryption function D_k. If Bob wants to send a message back, the process is analogous and they can use the same key.
The advantage of symmetric encryption schemes is that they can be extremely secure and usually very fast in comparison to asymmetric schemes defined in section 2.4.2. The main disadvantages are that it is generally hard to share keys, and that if a key is compromised communications in both directions are compromised. Furthermore the number of keys necessary for multiple parties to communicate grows very large as there are more parties. The number of keys needed for n parties to communicate independently equals $\frac{n\cdot(n-1)}{2}$ (the number of edges in a complete graph with n vertices).

2.4.2 Asymmetric encryption

Asymmetric encryption is encryption in which a key k_e often called *public key* is used for encryption, and a different key k_d often called *private key* is used for decryption. Asymmetric encryption is therefore often referred to as *public key encryption*. When Alice sends a message to Bob, Alice encrypts the plaintext using some encryption function E_{k_e} with k_e being Bob's public key, which is available to everyone. Bob receives the ciphertext and is able to recover the plaintext using his decryption function D_{k_d}. The main advantage of asymmetric algorithms is that key distribution is easier than in symmetric encryption. Public keys do not have to be kept secret, they can just be published online, so it is very easy to obtain a key for encryption. Disadvantages are that they often rely on complex mathematical problems that are assumed to be hard which often affects performance. A very common encryption scheme is the *RSA* encryption scheme which relies on the integer factorization problem. For details on RSA see [13].

2.4.3 Hybrid encryption

As seen in Sections 2.4.1 and 2.4.2 symmetric and asymmetric encryption have their advantages and disadvantages. The choice on which one is better to use depends on the situation. In most situations however it is best to use a combination of both. A problem with symmetric encryption is key exchange This can be solved by agreeing upon keys in an asymmetric way and using those keys in a symmetric encryption scheme. One of those methods is the *Diffie-Hellman*(DH) key exchange. For details on Diffie-Hellman see [6]. Later in this thesis a combination of DH and the symmetric encryption *AES* will be used, where an instantation of DH is considered using elliptic curves (ECDH). Using elliptic curves leads to particularly compact ciphertexts, which is desirable for use in a backdoor situation because space is often limited. Background information on the use of elliptic curves can be found in the works of Miller [10] and Koblitz [9] and a description of AES can be found in [5].

2.4.4 Post-quantum encryption

The security of most asymmetric encryption algorithms relies on mathematical problems which are assumed computationally hard. Some of these problems however are solvable while making use of a still theoretical type of computer called a *quantum computer*. A quantum computer is a computer using quantum mechanical phenomena to perform computations. Algorithms that could break widely deployed cryptographic schemes like RSA and DH using a quantum computer are already described by Shor [14]. So the creation of full scale quantum computers would break a large part of the schemes used today. This gives rise to *post-quantum encryption*. In his chapter of [2, Chapter 1] Bernstein describes several types of encryption that are believed to be resistant to both a classical and quantum computer. These types are:

- *Hash-based cryptography*
- *Code-based cryptography*
- *Lattice-based cryptography*
- *Multivariate-quadratic-equations cryptography*
- *Secret key (symmetric) cryptography,*

of which according to Bernstein Lattice-based cryptography has attracted the most interest. One of those lattice based schemes is the *NTRU* encryption scheme, described in Section 4.

Chapter 3

Kleptography

The term *kleptography* refers to the art of stealing cryptographic information in a secure and unnoticeable way. Kleptography is a phenomenon that can occur in so called *black-box cryptosystems*, cryptosystems in which the exact inner workings are not known to the user. The user just has to trust what is inside according to the manufacturer. Black boxes are very common on personal computers. Enterprises and consumers install so called closed source software all the time. Closed source software is software of which the source code is not publicly available. This is common on proprietary products in order to protect intellectual property. It does however have the side effect that it is essentially a black box, because nobody can verify what actually happens inside.

3.1 SETUP

In 1996 Young and Yung published an article [17] in which they introduce and standardize kleptography in terms of the *SETUP* mechanism. SETUP is an abbreviation of Secretly Embedded Trapdoor with Universal Protection. For a detailed overview of SETUP implementations in commonly used cryptosystems, see [1]. The definition of a SETUP mechanism by Young and Yung [17] is given as follows:

Definition 3.1.1 (SETUP)**.** *Let C be a publicly known cryptosystem. A SETUP mechanism is an algorithmic modification made to C to get C' such that:*

- *The input of C' agrees with the public specifications of the input of C.*
- *C' computes using the attacker's public encryption function E (and possibly other functions as well), contained within C'.*
- *The attacker's private decryption function D is not contained within C' and is known only by the attacker.*
- *The output of C' agrees with the public specifications of the output of C. At the same time, it contains published bits which are easily derivable by the attacker but are otherwise hidden.*

- *Furthermore, the outputs of C and C' are polynomially indistinguishable to everyone except the attacker.*

3.2 Weak SETUP

The definition of a weak SETUP mechanism is a relaxation of a regular SETUP mechanism. A weak SETUP is the same as a regular SETUP with the exception that it does not require the polynomial indistinguishability between the output of C and C' [16]. This may seem very easily detected, but in practice this still works well because an end user does not know that the implementation contains a SETUP. Furthermore, an end user often does not know what the output of C should be.

3.3 Subliminal Channel

A subliminal channel is a secondary channel of communication hidden inside a communications channel that is presumed to be compromised. The concept of a subliminal channel was introduced as a solution to the *prisoners problem* by Simmons in 1984 [15]. In the prisoners problem two people Alice and Bob are incarcerated and wish to plan a breakout. Their only way of communicating is by passing over messages via Eve who is one of the guards. They are allowed to use encryption, but Eve will only pass along the messages if she is allowed to read the messages, so she needs acces to the keys and the decryption function. As Eve will report any breakout plan, Alice and Bob have to hide their communications about breaking out within their communication.
This subliminal channel seems to solve a very specific problem, yet as of 2016 this problem is and will be more frequently seen in practice. More and more countries propose laws which oblige citizens to give up their private keys if requested. If they want to continue having secure communications, this creates a situation directly analogous to the prisoners problem.

Chapter 4

NTRU

This chapter follows the documents [8] and [12] with a description of the NTRU encryption scheme. NTRU is an assymetric cryptosystem commonly used in a hybrid cryptosystem to share keys for a symmetric encryption algorithm. NTRU works with operations on elements of the ring $R = \mathbb{Z}[X]/(X^N - 1)$. An element can be represented as either a polynomial of degree at most $N - 1$ or a vector of length N containing the coefficients. The operation denoted as $\circledast$ is the cyclic convolution product, and is used for multiplication on R. Using the property $X^N \equiv 1 \mod (X^N - 1)$ it is defined as $F \circledast G = H$ with

$$H_k = \sum_{i=0}^{k} F_i \cdot G_{k-1} + \sum_{i=k+1}^{N-1} F_i \cdot G_{N+k-i} = \sum_{i+j \equiv k \mod N} F_i \cdot G_j. \tag{4.1}$$

4.1 Parameters

NTRU is specified by six public parameters, the integers (N, p, q, d_f, d_g, d_r), in which $\gcd(p, q) = 1$ and q is much larger than p. In practice p is usually chosen as 3 and q a power of 2. The integer N specifies the ring $R = \mathbb{Z}[X]/(X^N - 1)$. The parameters (d_f, d_g, d_r) specify the sets $(\mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_r, \mathcal{L}_m)$, which are sets of polynomials of degree at most $N - 1$ with a fixed number of nonzero coefficients.

4.2 Sample spaces

Definition 4.2.1 (Message space)**.** *The message space $\mathcal{L}_m$ is defined as*

$$\mathcal{L}_m = \{m \in R | m \text{ has coefficients in } [-\frac{p-1}{2}, \frac{p-1}{2}]\},$$

assuming p is odd.

Messages are assumed to be integers in a radix p representation, with every digit a coefficient of the polynomial. The rest of this section follows definitions from [8].

Definition 4.2.2 (The set $\mathcal{L}(d_1, d_2)$)**.** *The set of ternary polynomials* $\mathcal{L}(d_1, d_2)$ *is defined as:*

$$\mathcal{L}(d_1, d_2) = \left\{ F \in R \,\middle|\, F \text{ has exactly } \begin{array}{l} d_1 \text{ coefficients equal to } 1, \\ d_2 \text{ coefficients equal to } -1 \\ \text{the rest of the coefficients equal } 0 \end{array} \right\} \tag{4.2}$$

The key and randomness spaces $(\mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_r)$ are defined as :

$$\begin{aligned} \mathcal{L}_f &= \mathcal{L}(d_f, d_f - 1) \\ \mathcal{L}_g &= \mathcal{L}(d_g, d_g) \\ \mathcal{L}_r &= \mathcal{L}(d_r, d_r) \end{aligned}$$

$\mathcal{L}_f$ is not set as $\mathcal{L}(d_f, d_f)$ because a polynomial $f \in \mathcal{L}(d_f, d_f)$ would have $f(1) = 0$ which is not invertible, while f needs to be invertible for key creation explained in Section 4.3.

4.3 Key creation

To create a key, two random polynomials $f \in \mathcal{L}_f$ and $g \in \mathcal{L}_g$ are chosen such that inverses F_q and F_p of f exist in R modulo q and p respectively.
The public key

$$h \equiv F_q \circledast g \mod q, \tag{4.4}$$

is computed.
The private key is the pair (f, F_p), in which F_p is derivable from f and p but is precomputed for practical purposes. The modulo q reduction of the polynomial means a reduction of the coefficients to equivalent representatives in the interval $[-\frac{q}{2}, \frac{q}{2}]$.

4.4 Encryption

A message $m \in \mathcal{L}_m$ is chosen and a random $r \in \mathcal{L}_r$ is selected. Now ciphertext

$$c \equiv p \cdot r \circledast h + m \mod q, \tag{4.5}$$

is computed.

4.5 Decryption

To obtain message m the quantity $a \equiv f \circledast c \mod q$ is computed. Because

$$a \equiv f \circledast (p \cdot r \circledast h + m) \equiv f \circledast (p \cdot r \circledast F_q \circledast g + m) \equiv p \cdot r \circledast g + f \circledast m \mod q, \tag{4.6}$$

reducing modulo p yields $f \circledast m$ if the polynomials are sparse enough.
In general

$$m \equiv a \circledast F_p \mod p, \tag{4.7}$$

Section 4.6 explains possible exceptions to this equivalence.

4.6 Decryption failures

Definition 4.6.1 (Width)**.** *Let $l \in R$. The width of l is defined as $|l|_\infty = \max_{0 \le i \le N-1} l_i - \min_{0 \le i \le N-1} l_i$.*

When Equation (4.6) is not an exact equation over the integers m might be only partially recovered, due to the modular reductions in the decryption step. In Equation 4.6 first the term $a \equiv p \cdot r \circledast g + f \circledast m$ is reduced modulo q after which it is reduced modulo p. Since $\gcd(p, q) = 1$, this resulting a reduced modulo q and p is not well defined, as reducing a different representative of a modulo p could give a different result. In practice this problem is avoided by choosing the uniquely defined representative of a with coefficients in the interval $[-\frac{q}{2}, \frac{q}{2}]$. However, if the width of the term $S = p \cdot r \circledast g + f \circledast m$ exceeds q, some coefficients in the recovered polynomial will differ from the coefficients of m. This is called a decryption failure and has to be taken into account in parameter selection.

Chapter 5

NTRU backdoor

In this section an example of a modified NTRU encryption with a backdoor using a weak SETUP is described and analysed, after which countermeasures are given. The backdoor has the purpose of leaking information fromt the encrypting party to a third party. This information can be exclusively available to third party if it is public key encrypted to that party.

5.1 Description

This version differs from regular NTRU in the sense that a secondary encrypted message along with the regular message is included in the ciphertext. This secondary message is available to a third party. The public key encryption of this secondary message will be denoted as M. The key setup on the receiving end stays exactly the same.

5.1.1 Encryption

For simplicity a message M is assumed to be represented as a polynomial in R with coefficients modulo the integer parameter ρ which is less than or equal to q and coprime to p so, $M \in \mathbb{Z}_\rho[X]/(X^N - 1)$, $\rho < q$ and $\gcd(\rho, p) = 1$.

On the sending end a slight adaptation of Equation (4.5) is used. First ciphertext c is computed as in Equation (4.5). Now the new ciphertext c' including the secondary message, is computed as

$$c' = c + k \cdot p, \tag{5.1}$$

with k a polynomial in R with coefficients in $\mathbb{Z}_\rho$ such that $c' \equiv M \mod \rho$. This polynomial k can be obtained by solving the integer equation $M_i \equiv c_i + k_i \cdot p \mod \rho$ for every coefficient of k. Having the $\gcd(\rho, p) = 1$ by definition, ensures the existence of these solutions.

5.1.2 Decryption by the attacker

The attacker reduces $c' \mod \rho$ and recovers M, since $M \equiv c' \mod \rho$. The attacker now decrypts M with its private key and obtains the leaked information.

5.1.3 Decryption by the intended receiver

Decryption at the receiver end stays exactly the same. First the quantity $A = f \circledast c' \mod q$ is computed as in Equation (4.6). Because

$$\begin{aligned} A &\equiv f \circledast (p \cdot k + p \cdot r \circledast h + m) \\ &\equiv f \circledast (p \cdot k + p \cdot r \circledast F_q \circledast g + m) \\ &\equiv p \cdot k \circledast f + p \cdot r \circledast g + f \circledast m \mod q, \end{aligned} \tag{5.2}$$

reducing A modulo p still yields $f \circledast m$ if the coefficients are not too large (see Section 5.2.1). Thus $m \equiv A \circledast F_p \mod p$.

5.1.4 Encryption of M

To ensure that the leaked information is exclusively available to one party, it is important that there is encryption on it. To obtain this encryption an elliptic curve implementation of DH can be used in combination with AES. Using elliptic curves is preferred in this case because it is very space efficient. A downside is that it is not post-quantum, this is not a big problem because the backdoor itself strongly relies on it is secrecy. For M to be an AES encryption a static ECDH key needs to be placed in the backdoored implementation. A ciphertext M would consist of a session key generated in the backdoored implementation, appended with the AES encryption of the message which possibly includes some authentication tag. For this a 256 bit ECC key is recommended, M would then consist of 256 bits of session key, which is a point on the curve, appended with 128 bit message blocks encrypted with the ECDH key obtained from the static key and the session key. The authentication tag is usually chosen as one 128 bit block but smaller is possible. At least 64 bits is recommended for authentication.

5.2 Analysis

5.2.1 Decryption failures

As pointed out in Section 4.6, a decryption failure occurs when the polynomial

$$S = p \cdot r \circledast g + f \circledast m,$$

has a width larger than q. Adding the $k \cdot p$ term to the ciphertext in equation (5.1) makes decryption failures more likely because now a decryption failure occurs when the polynomial

$$T = p \cdot k \circledast f + p \cdot r \circledast g + f \circledast m,$$

has a width larger than q, as generally $|T|_\infty > |S|_\infty$. Because for a single coefficient of T it applies that

$$T_l = S_l + \sum_{i+j \equiv l \mod N} p \cdot k_i \cdot f_j,$$

the contribution of this extra convolution product $p \cdot k \circledast f$ to a single coefficient is at most

$$p \cdot (\lceil \frac{\rho - 1}{2} \rceil) \cdot (2 \cdot d_f - 1). \tag{5.3}$$

Let $\alpha = \min(d_g, d_r)$, the maximum width of S is given by

$$\max |S|_\infty = 2 \cdot p \cdot \alpha + (2 \cdot d_f - 1) \cdot \lfloor \frac{p}{2} \rfloor, \tag{5.4}$$

so the maximum width of T would be

$$\max |T|_\infty = 2 \cdot p \cdot \alpha + (2 \cdot d_f - 1) \cdot (\lfloor \frac{p}{2} \rfloor + p \cdot \lceil \frac{\rho - 1}{2} \rceil). \tag{5.5}$$

5.2.2 Parameter choices

Because of the possible decryption failures it is important to pick parameters that minimize this phenomenon while maintaining global security. It is recommended to keep ρ as small as possible. In the case where $p = 3$ the value $\rho = 2$ is quite suitable. Other options would be $\rho = 4$ and $\rho = 5$ as this would give space to leak more information, but as noted in section 5.2.1 decryption failures will be much more likely because the extra contribution of the term in equation 5.3 can become much larger. For most parameter sets $\rho = 2$ will most likely be the only option that works without increasing the probability of decryption failures too much.

5.2.3 Optimization

Decryption failures will be less likely if the vector $k \cdot p$ added in equation (5.1) is sparse. This is the case when k is sparse. A way to keep k sparse is to minimize the number of bits needed to store M and pad it with zeros. Depending on what information will be leaked it might even be possible to split up M over several messages. In that case M will only be partially leaked, but can be recovered if multiple messages containing all the parts are recorded.
Another optimization that works in the case where $\rho = 2$ is to append a one bit shorter message M' with an indicator bit i such that instead of M, $[i|M']$ is leaked. The polynomial k is now computed regularly. If this k contains more ones then it contains zeroes the term $\bar{k} \cdot p$ is added instead of $k \cdot p$, with $\bar{k}$ the bitwise complement of k. The attacker now recovers either $[0|M']$ or $[1|\bar{M}']$ and is able to recover M' by taking the complement if $i = 1$. Note that this indicator costs one bit in space, so M' has at most $N - 1$ bits where M would have N.

5.2.4 Practical implementation

In a sage implementation of NTRU with a backdoor added, experiments were run to look at the impact of the backdoor with respect to decryption failures. The sage code can be found in appendix A. In every experiment a pseudorandom ternary message m is generated along with a pseudorandom N bit binary message M. Subliminal message M is binary because the parameter $\rho = 2$ is set. No optimisations discussed in the previous section were applied in the implementation.The first set of experiments counts the number of decryption failures caused by the backdoor in 2 different ways using NTRU parameters from [7]. First a subset of experiments was conducted in which a new key is generated with every trial, in this case all trials are independent. Secondly a subset of experiments was conducted in which the same key is used more than once, these trials are not independent but they do represent a real world situation in which keys are generated once and then reused often. Doing multiple trials with the same keys also allows for more experiments as generating a new key is relatively expensive computationally.

Table 5.1: Decryption failure check results

Parameters						# keys	# trials per key	# failures
N	p	q	d_f	d_g	d_r			
613	3	2048	55	204	55	20000	1	0
						100	10000	0
887	3	2048	81	295	81	10000	1	0
						100	10000	0
1171	3	2048	106	390	106	5000	1	0
						100	10000	0

Since no decryption failures occured in these experiments, the increased probability of a decryption failure caused by the backdoor will probably go unnoticed in practice. With Equation 5.3 the maximum contribution to the width of T with the parameters used can be computed, with respect to q this difference is relatively small enough. Looking at Equations 5.4 and 5.5 the maximum width T is only slightly larger than the maximum width of S with the parameters used in the experiments. This could possibly explain the lack of decryption failures. Note that the maximum width was not expected to be obtained. These extreme widths are in general very rare, as f and r are chosen to be sparse. They are intentionally centred around 0 to let a lot of cancelations occur. This behaviour is not unique to the parameters used in the experiments, in most parameter sets used in practice $d_f = d_r$. When $\rho = 2$ is chosen, the most significant term in Equation 5.5 is generally the first one, so the contribution of the backdoor to the maximum width is generally small enough. In the implementation any message can be leaked as long as its encryption does not exceed N bits. The trialled version with parameter $N = 613$ will fit a 256 bit subliminal message encrypted as explained in Section 5.1.4 with 256 bits security and leave some space for

authentication. The versions with $N = 887$ and $N = 1171$ will even fit a 512 or 768 bit subliminal message respectively. For instance, in $N = 1171$ there could be a 256 bit key and a 128 bit authentication tag, which leaves $(1171 - 256 - 128 = 787)$ bits for a message. The 787 bits fit 6 blocks of 128 bit ciphertext and the remaining 19 bits could be used for the optimizations discussed in Section 5.2.3. To get an idea of how much the probability of decryption failures increases on average instead of just the worst case, a second set of experiments was run. In the second set of experiments, the width of the terms S and T were stored. A decryption failure occurs when $|S|_\infty > q$ or $|T|_\infty > q$, without and with the backdoor respectively. For these experiments the parameters from Table 5.1 were used. The results are presented in histograms where red corresponds to $|S|_\infty$ and green corresponds to $|T|_\infty$.

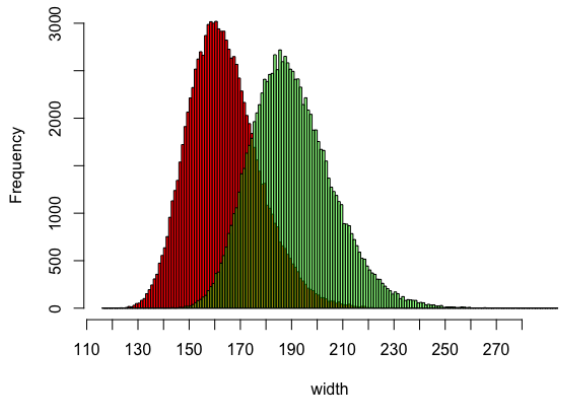

Figure 5.1: $(N, p, q, d_f, d_g, d_r) = (613, 3, 2048, 55, 204, 55)$,
10 keys, 10000 trials per key.

	$\|S\|_\infty$	$\|T\|_\infty$
μ	164.0471	190.7234
σ	13.81889	15.95371
min	116	139
max	251	294

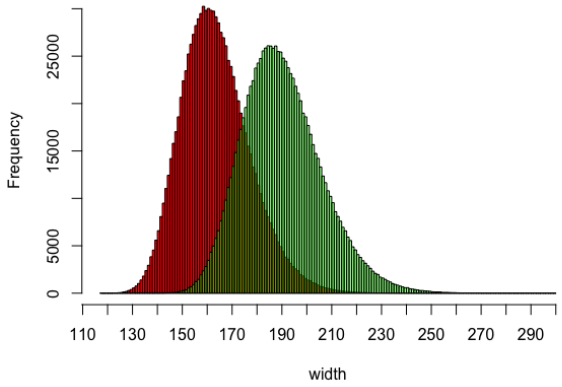

Figure 5.2: $(N, p, q, d_f, d_g, d_r) = (613, 3, 2048, 55, 204, 55)$,
100 keys, 10000 trials per key

	$\|S\|_\infty$	$\|T\|_\infty$
μ	163.9682	190.6541
σ	13.79992	15.95437
min	117	140
max	257	300

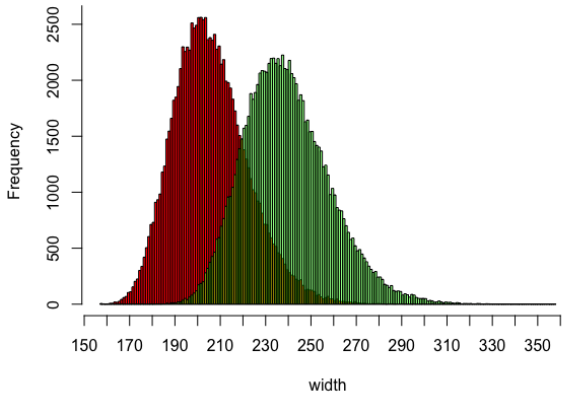

Figure 5.3: $(N, p, q, d_f, d_g, d_r) = (887, 3, 2048, 81, 295, 81)$,
10 keys, 10000 trials per key.

	$\|S\|_\infty$	$\|T\|_\infty$
μ	206.3269	239.786
σ	16.43092	18.94655
min	157	182
max	298	358

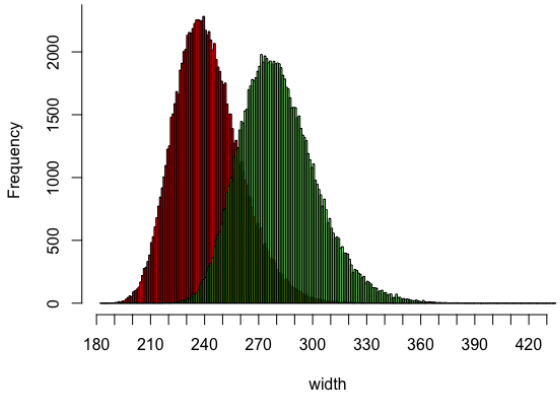

Figure 5.4: $(N, p, q, d_f, d_g, d_r) = (1171, 3, 2048, 106, 390, 106)$,
10 keys, 10000 trials per key

	$\|S\|_\infty$	$\|T\|_\infty$
μ	242.3299	281.6776
σ	18.47012	21.31923
min	182	218
max	365	435

These results confirm that on average the the probability of a decryption failure increases, but this increase is small enough to go unnoticed in a practical situation because large widths are rare. An interesting side effect is that the standard deviation also increases when the backdoor is added. The green spike is generally lower and less steep, which means that the $|T|_\infty$ values are less predictable than the $|S|_\infty$ values. This phenomena gives rise to some questions explained in Section 7.2.3.

5.2.5 Countermeasures

There are ways to find out that the ciphertext was tampered with. One of those being the recovery of the randomness. From equation (4.5) the equation

$$c - m = r \circledast h \mod q \tag{5.6}$$

can be obtained. Meaning r can be recovered by

$$r = (c - m) \circledast h^{-1} \mod q \tag{5.7}$$

if inverse h^{-1} exists in R modulo q. In the case of a ciphertext with an extra term added, doing the same computation will result in $r + k \cdot p \circledast h^{-1}$ instead of r, which with high probability will not be an element of $\mathcal{L}_r$. Since by specification $r \in \mathcal{L}_r$, the receiver can check whether $(c - m) \circledast h^{-1} \in \mathcal{L}_r$. If this is not the case and h is invertible in R modulo q, the ciphertext has been tampered with and a warning can be sent back to the sender. To make sure that this is possible, it is important that h is always invertible in R modulo q. h depends on the choice of f and g so a change has to be made to the selection of those. Public key h is defined as $h = F_q \circledast g \mod q$. By definition F_q is invertible so the only extra requirement is that g must also be invertible, this can be done by choosing $g \in \mathcal{L}_g$ in a similar manner as f. Since invertibility is required, $\mathcal{L}_g$ can no longer be defined as $\mathcal{L}_g = \mathcal{L}(d_g, d_g)$ and would need to be defined as $\mathcal{L}_g = \mathcal{L}(d_g, d_g - 1)$.

Chapter 6

NTRU subliminal channel

In this chapter a modification of NTRU with a SETUP is shown in which an extra possibly secret channel for information is added. This channel differs from the backdoor discussed in Chapter 5 as it is intended for the receiver of the message instead of a third party.

6.1 Description

In this adaptation, Bob sends a regular message m and a subliminal encrypted message M to receiver Alice. To include this M, a technique inspired by the countermeasures described in Section 5.2.5 is used. In addition to the regular setup, both Alice and Bob agree upon an injective map ϕ which maps M to an element of $\mathcal{L}_r$ and a pair of keys to encrypt an decrypt M.

6.2 Key setup

Alice chooses $f \in \mathcal{L}_f$ and computes F_q and F_p as in Section (4.3). Now $g \in \mathcal{L}(d_g, d_g - 1)$ is chosen such that inverse g^{-1} exists in R_q and public key h is computed normally as in Equation (4.4). Choosing $g \in \mathcal{L}(d_g, d_g - 1)$ is justifiable as a protection against a specific backdoor mentioned earlier. Alice publishes her public key h so that others including Bob, can send her messages.

6.3 Encryption

Bob uses the function ϕ to map M to an element $r \in \mathcal{L}_r$ and encrypts m by computing c using Equation (4.5). Bob now sends c to Alice.

6.4 Decryption

Alice receives c and recovers m using Equation (4.6). She now computes h^{-1} and uses this to recover $r \equiv (c - m) \circledast h^{-1} \mod q$. She now recovers M as the preimage of r using ϕ^{-1}. For efficiency it is possible to precompute h^{-1}.

6.5 Encoding messages

In this section an example for the injective map ϕ mentioned earlier is described. It is somewhat similar to Algorithm 2.2 in [2, Chapter 4]. Let M be an encrypted message represented as a unique number chosen in the discrete interval $[0, \binom{N}{d_r} \cdot \binom{N-d_r}{d_r} - 1]$. Then ϕ is an injective map $[0, \binom{N}{d_r} \cdot \binom{N-d_r}{d_r} - 1] \to \mathcal{L}_r$ that encodes an encrypted message M to an $r \in \mathcal{L}_r$. The inverse ϕ^{-1} gives preimage M from the image r.

The set $\mathcal{L}_r$ can be represented as a tree, with every level representing one coefficient. The root is defined as representing r_0, the level of the leaves r_n. Every leaf corresponds to a unique element of $\mathcal{L}_r$, and is defined by the unique path from the root to the leaf. Every node has at most 3 branches depending on whether it can still be completed, because left and right branches are limited. The leftmost branch corresponds to choosing a -1, the middle branch a 0 and the right branch a 1 on that level. Now the set $\mathcal{L}_r$ can be indexed by counting the leaves from left to right, where the leftmost leaf has index 0.
The tree itself does not have to be stored in memory, at every node the number of leaves can be computed by $\binom{n}{k} \cdot \binom{n-k}{l}$, with n being the number of levels from the node to a leaf, k the number of -1s and l the number of 1s that are not used yet at that node. The left middle and right subbranches of a node have $\binom{n-1}{k-1} \cdot \binom{n-k}{l}$,$\binom{n-1}{k} \cdot \binom{n-k-1}{l}$ and $\binom{n-1}{k} \cdot \binom{n-k-1}{l-1}$ leaves respectively.

To convert an index M into an $r \in \mathcal{L}_r$ the tree is traversed starting from the root, and a running index j is kept, so at the root $i = 0$ and $j = M$. At every level i the number of leaves in the left subbranch L_i is computed. If $j \leq L_i$, the left branch is taken and $r_i = -1$. If this is not the case, the number of leaves in the middle subbranch is computed and added to L_i to obtain L_i' which is the number of leaves in the left and middle subbranch combined. Now if $L_i < j \leq L_i'$ the middle branch is taken, $r_i = 0$ and $j := j - L_i$. If $j > L_i'$ the right branch is taken, $r_i = 1$ and $j := j - L_i'$. This process repeats until a leaf is reached.

The inverse ϕ^{-1} works in a similar matter, the tree is traversed starting from the root according to the path specified in r. A running index j is kept for which $j = 0$ at the root. Now at every level i the number of leaves that are "skipped" by not choosing the left or middle branch respectively, are added to j. So if $r_i = 0$, the middle branch is taken, L_i is computed and $j := j + L_i$. Else if $r_i = 1$, the right branch is taken, L_i' is computed and $j := j + L_i'$. This process repeats until all the bits of r are evaluated. Now $M := j$.

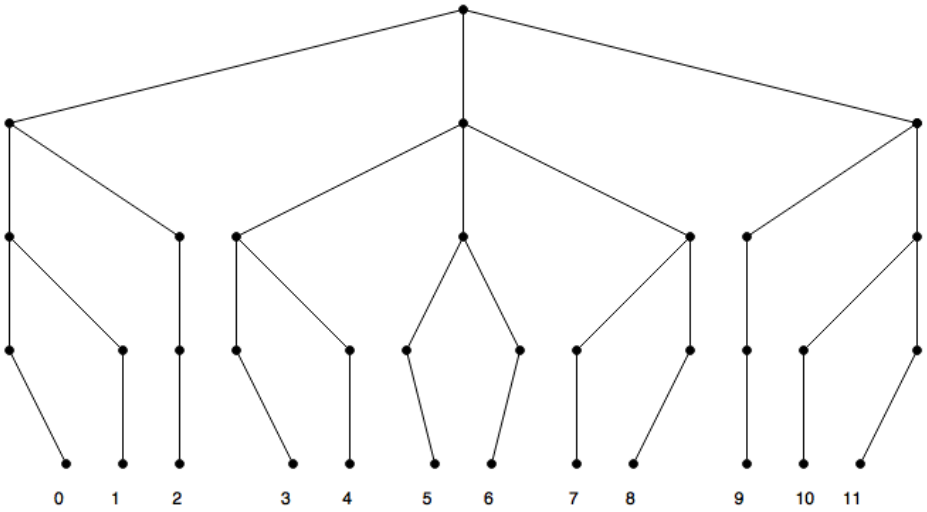

Figure 6.1: Example of a tree for $d_r = 1$ and $N = 4$

6.6 Why does it work?

As pointed out in Section 5.2.5 the randomness r can be recovered by the receiver if g is chosen to be invertible. This phenomenon is exploited by putting a message in r rather than choosing r randomly.

Chapter 7

Conclusions

7.1 Final Remarks

As shown in Chapters 5 and 6 it is feasible and practical to modify the NTRU cryptosystem in such a way that it contains a backdoor or subliminal channel. Countermeasures against the backdoor have been described in Section 5.2.5.

7.2 Future research

7.2.1 SETUP in NTRU

In Chapter 5 a modification of NTRU with a weak SETUP is described. It is probably possible to put a regular SETUP in there by choosing the vector r rather than randomly generating it. This would be done by influencing r in such a way that c in equation (4.5) has a desired outcome. However because r should be an element of $\mathcal{L}_r$ the set of possible outcomes for c is limited. Research needs to be done in order to find if this set can be generalized, to design an invertible function that converts M to an element of this set and find out how many bits can be leaked using this technique.

7.2.2 Minimization of decryption failures

In Section 5.2.3 some optimizations have been given in order to reduce the increased probability of decryption failures with the backdoor added. In Section 5.2.4 some experimental results are given. By doing more experiments and with more parameter sets, the increased probability of decryption failures might be estimated. From this parameters can be selected which allow for more information to be leaked without increasing the probability too much. Research can also be done to find the theoretical probability instead of an estimation. With this estimation parameters can be computed that preserve global security, but at the same time minimize the probability of decryption failures.

7.2.3 Statistical countermeasures

In Section 5.2.4 experimental results were given on the width of the polynomial T with respect to the width of S. These results showed that the width of T is less predictable. The standard deviation was larger for the values of T. This occurs because adding an extra message to the ciphertext means adding some randomness. This yields the question, could a receiver of messages distinguish the ones that were tampered with from the ones that were not and alert the sender? How many messages would it need to be able to do so? These are questions that might be worthwhile looking into.

Bibliography

[1] Milou Antheunisse. "Kleptography Cryptography with Backdoors". MA thesis. Eindhoven University of Technology, 2015. URL: http://repository.tue.nl/801620.

[2] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen. *Post-quantum cryptography*. Springer, 2009. ISBN: 978-3-540-88702-7.

[3] Lindsay N. Childs. "Congruence Classes Modulo a Polynomial". In: *A Concrete Introduction to Higher Algebra*. New York, NY: Springer New York, 1995, pp. 414–431. ISBN: 978-1-4419-8702-0.

[4] Hans Cuypers, Hans Sterk, and Arjeh M. Cohen. *Algebra-Interactive*. Springer-Verlag Berlin Heidelberg, 1999. ISBN: 978-3-540-65368-4.

[5] Joan Daemen and Vincent Rijmen. "Rijndael for AES". In: *AES Candidate Conference*. 2000, pp. 343–348.

[6] Whitfield Diffie and Martin E. Hellman. "New directions in cryptography". In: *IEEE Trans. Information Theory* 22.6 (1976), pp. 644–654.

[7] Philip S. Hirschhorn et al. "Choosing NTRUEncrypt Parameters in Light of Combined Lattice Reduction and MITM Approaches". In: *ACNS*. Vol. 5536. Lecture Notes in Computer Science. 2009, pp. 437–455.

[8] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. "NTRU: A Ring-Based Public Key Cryptosystem". In: *ANTS*. Vol. 1423. Lecture Notes in Computer Science. Springer, 1998, pp. 267–288.

[9] Neal Koblitz. "Elliptic curve cryptosystems". In: *Mathematics of computation* 48.177 (1987), pp. 203–209.

[10] Victor S. Miller. "Use of Elliptic Curves in Cryptography". In: *CRYPTO*. Vol. 218. Lecture Notes in Computer Science. Springer, 1985, pp. 417–426.

[11] Constantinos Patsakis. *NTRU sage*. 2015 (retrieved August 2016). URL: https://github.com/kpatsakis/NTRU_Sage/blob/master/ntru.sage.

[12] Jill Pipher. *Lectures on the NTRU encryption algorithm and digital signature scheme*. June 2002 (retrieved June 2016). URL: http://www.math.brown.edu/~jpipher/grenoble.pdf.

[13] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. In: *Commun. ACM* 21.2 (1978), pp. 120–126.

[14] Peter W. Shor. “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer”. In: *SIAM Review* 41.2 (1999), pp. 303–332.

[15] Gustavus J. Simmons. “The Prisoners’ Problem and the Subliminal Channel”. In: *CRYPTO*. Plenum Press, New York, 1983, pp. 51–67.

[16] Adam L. Young and Moti Yung. “Kleptography: Using Cryptography Against Cryptography”. In: *EUROCRYPT*. Vol. 1233. Lecture Notes in Computer Science. Springer, 1997, pp. 62–74.

[17] Adam L. Young and Moti Yung. “The Dark Side of ”Black-Box” Cryptography, or: Should We Trust Capstone?” In: *CRYPTO*. Vol. 1109. Lecture Notes in Computer Science. Springer, 1996, pp. 89–103.

Appendix A

Implementation code

This chapter includes the code used for the experiments discussed in section 5.2.4. The code is written in sage using the work of Patsakis [11] as a basis. There are 3 parameters to set: **Security_level**, **No_of_keygenerations** and **No_of_trials_per_keyset** which set the security level, either 128,192 or 256. The number of keys to be generated and the number of encryptions and decryptions with a key pair. The program generates an NTRU key-pair using parameters chosen with the security level, it encrypts a pseudorandom ternary message with and without the backdoor and decrypts both of them. In the backdoored encryption a subliminal pseudorandom N bit binary message is included. After this the result is verified to check wether or not a decryption failure has occurred.
The output is the total runtime and the number of failures.

```
from time import time

R.<x> = ZZ['x'];



class NTRUEncrypt(object):



        #generates a random number in [j,k]
        def random_between(self, j,k) :

                a=int(random()*(k-j+1))+j

                return a



        #generates a pseudorandom polynomial
        #with coefficients in {-1,0,1}
        def random_trinary(self) :

                x = self.random_between(0, self.n-1)

                y = self.random_between(0, self.n-1-x)

                s = [x] + [y]

                shuffle(s)

                poly = self.sample(self.n, s[0], s[1])

                return poly
```

```
#generates pseudorandom polynomial with binary coefficients
def random_binary(self) :

        x = self.random_between(0, self.n-1)

        poly = self.sample(self.n, x, 0)

        return poly



#generates a pseudorandom ternary polynomial
#with exactly o ones and mo minus ones
def sample(self,NN, o,mo):

        s=[1]*o+[-1]*mo+[0]*(NN-o-mo)

        shuffle(s)

        return R(s)



#reduces the coefficients of polynomial f modulo pp

#and "fixes" them so that they belong to [-pp/2,pp/2]

def modCoeffs(self,f,pp):

        clist=f.list()

        p2=pp/2

        for i in range(len(clist)):

                clist[i] = clist[i]%pp

                if clist[i]>p2:

                        clist[i]-=pp

        return R(clist)


#computes the polynomial k
def generate_k(self, c, ml):
        a = []
        b = []
        clist = []

        for i in range (0, self.n - 1 ):

                a.append(c[i] % 2)

                clist.append((a[i] + ml[i]) % 2)

                self.k = R(clist)

        return R(clist)



# adds subliminal message ml to ciphertext c
def add_backdoor(self, c, ml):

        k = self.generate_k(c, ml)

        c = (c + k * self.p)

        c=self.modCoeffs(c,self.q)

        return c



# encrypts message m using public key h
def encrypt(self,h,m):

        self.m=m

        s=self.sample(self.n, self.Dr, self.Dr-1)
```

```
        self.r = s
        c=s*h+m
        c=c%(x^self.n-1)
        c=self.modCoeffs(c,self.q)
        return c


#decrypts cipher-text c using private key Priv
def decrypt(self,c,Priv):
        f,fp=Priv
        a=f*c
        a=a%(x^self.n-1)
        a=self.modCoeffs(a,self.q)
        a=a*fp
        a=a%(x^self.n-1)
        a=self.modCoeffs(a,self.p)
        return a


#computes the inverse of poly in R_2
def __inv_poly_mod2__(self,poly):
        k=0;b=1;c=0*x;
        f=poly;g=x^self.n-1
        f=self.modCoeffs(f, 2)
        res=False
        while True:
                while f(0)==0 and not f.is_zero():
                        f=f.shift(-1)
                        c=c.shift(1)
                        c=self.modCoeffs(c, 2)
                        k+=1
                if f.is_one():
                        e=(-k)%self.n
                        retval= x^e*b
                        res=True
                        break
                elif f.degree()==-1 or f.is_zero():
                        break
                if f.degree()<g.degree():
                        f,g=g,f
                        b,c=c,b
                f=f+g
                b=b+c
                f=self.modCoeffs(f, 2)
                c=self.modCoeffs(c, 2)
        if res:
```

```
                retval=retval%(x^self.n-1)
                retval=self.modCoeffs(retval, 2)
                return True, retval
        else:
                return False,0


#computes the inverse of poly in R_3
def __inv_poly_mod3__(self,poly):
        k=0;b=1;c=0*x;
        f=poly;g=x^self.n-1
        res=False
        while True:
                while f(0)==0 and not f.is_zero():
                        f=f.shift(-1)
                        c=c.shift(1)
                        k+=1
                if f.is_one():
                        e=(-k)%self.n
                        retval= x^e*b
                        res=True
                        break
                elif (-f).is_one():
                        e=(-k)%self.n
                        retval= -x^e*b
                        res=True
                        break
                elif f.degree()==-1 or f.is_zero():
                        break
                if f.degree()<g.degree():
                        f,g=g,f
                        b,c=c,b
                if f(0)==g(0):
                        f=f-g
                        b=b-c
                else:
                        f=f+g
                        b=b+c
                f=self.modCoeffs(f, 3)
                c=self.modCoeffs(c, 3)
        if res:
                retval=retval%(x^self.n-1)
                retval=self.modCoeffs(retval, 3)
                return True, retval
        else:
```

```
                return False,0


    #computes the inverse of poly in R_{2^x}
    def __inv_poly_mod_prime_pow__(self,poly):

            res,b=self.__inv_poly_mod2__(poly)

            if res:

                    qr=2

                    while qr<self.q:

                            qr=qr^2

                            b=b*(2-poly*b)

                            b=b%(x^self.n-1)

                            b=self.modCoeffs(b, self.q)

                    return True,b

            else:

                    return False,0


    #generates private key
    def __gen_priv_key__(self):

            res=False

            while (res==False):

                    poly=self.sample(n,self.Df,self.Df-1)

                    ppInv=self.__inv_poly_mod3__(poly)[1]

                    res,pqInv=self.__inv_poly_mod_prime_pow__(poly)

            return poly,ppInv,pqInv


    #computes the width of a polynomial
    def width(self, poly) :

            cofs = poly.coefficients()

            maxi = max(cofs)

            mini = min(cofs)

            return (maxi-mini)


    #generates a keypair
    def gen_keys(self):

            f,fp,fq=self.__gen_priv_key__()

            g=self.sample(n,self.Dg,self.Dg) #-1)

            self.f = f

            self.g = g

            h=self.p*g*fq

            h=h%(x^self.n-1)

            h=self.modCoeffs(h,self.q)

            return h,(f,fp)


    #sets of NTRU parameters
```

```
        def __init__(self, SECLEVEL):

                self.p=3

                self.q=2048

                #variables for computing widths
                self.m=0
                self.r=0
                self.f=0
                self.g=0
                self.k=0

                if SECLEVEL==128:

                        self.n = 613
                        self.Df = 55
                        self.Dg = 204
                        self.Dr = 55

                elif SECLEVEL==192:

                        self.n = 887
                        self.Df = 81
                        self.Dg = 295
                        self.Dr = 81

                else:

                        self.n = 1171
                        self.Df = 106
                        self.Dg = 390
                        self.Dr = 106



#test code
Security_level = 128
No_of_keygenerations = 1
No_of_trials_per_keyset = 1

#output file
output = file('/Users/robin/Desktop/128.txt','w')

failures = 0
failureswith = 0

ts=time()

for i in range(0, No_of_keygenerations):

        ntru=NTRUEncrypt(Security_level)

        n=ntru.n

        h,Priv=ntru.gen_keys()


        for i in range(0, No_of_trials_per_keyset):

                m=ntru.random_trinary()

                ml = ntru.random_binary()

                cc=ntru.encrypt(h,m)

                ccb=ntru.add_backdoor(cc, ml)

                mm=ntru.decrypt(cc,Priv)

                S = ((ntru.p*ntru.r*ntru.g + ntru.f * ntru.m)%(x^n-1))

                output.write(str(ntru.width(S)))
                output.write("\t")

                mmb=ntru.decrypt(ccb,Priv)

                T = ((S + ntru.p*ntru.k*ntru.f) %(x^n-1))

                output.write(str(ntru.width(T)))
                output.write("\n")

                if mm!=m:

                        failures += 1
```

```
                if mmb!=m :
                        failureswith += 1
runtime=time()-ts
print "runtime", runtime
print "failures without backdoor:", failures
print "failures with backdoor:", failureswith
output.close()
```

Appendix B

Notation

Table B.1: Notation table

Variable	Notation
$\mathbb{N}$	Set of all the positive integers including 0
$\mathbb{Z}$	Set of all the integers
p, q, n, m, i, j	Integers
X	Indeterminate
f, g, h, r	Polynomials
K	Usually denotes a field
R	Usually denotes a ring
F_q	Inverse of polynomial f in R_q
m	Message
M	Encrypted information intended to leak out of the system.
c	Ciphertext
$R[X]$	Polynomial ring over R
C	Cryptosystem
D	Decryption function
E	Encryption function
S	The polynomial $p \cdot r \circledast g + f \circledast m$
T	The polynomial $p \cdot r \circledast g + f \circledast m + p \cdot k \circledast f$
ϕ	Encoding function
$\|g\|_\infty$	The width of polynomial g
$\lfloor a \rfloor$	The number a, rounded to the nearest integer smaller than or equal to a
$\lceil a \rceil$	The number a, rounded to the nearest integer larger than or equal to a
$\mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_r$	Subsets of R defined in section 4.2.
$\mathcal{L}_m$	Set of polynomials in R_p