# INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

# An Investigation of the Lattice Based Digital Signature Scheme and Its Application in E-Commerce

**By**

**Xinzhou Wei**

A Dissertation submitted to the Graduate Faculty in Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy, the City University of New York

2002

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

March 1, 2002
Date

Professor Michael Anshel
Chair of Examining Committee

3/6/02
Date

Professor Theodore Brown
Executive Officer

Professor Jerome Liang

Professor Victor Pan

Professor Jerry Waxman

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

# ABSTRACT

## An Investigation of the Lattice Based Digital Signature Scheme and Its Application in E-Commerce

XINZHOU WEI
B.S., XI'AN JIAOTONG UNIVERSITY
M.S. XI'AN JIAOTONG UNIVERSITY
M.Philosophy, THE CITY UNIVERSITY OF NEW YORK
Ph.D., THE CITY UNIVERSITY OF NEW YORK

Advisor: Professor Michael Anshel

In order to protect commercially sensitive information and communications on embedded and wireless consumer devices, emerging markets require security at different levels. The first-generation security technologies like RSA and ECC are not practical for wide scale consumer applications because they are usually too large and too slow for the weak computation ability devices. An efficient NTRU Public Key Cryptography System

(PKCS) has been developed to uniquely suit to this purpose. This dissertation presents an investigation of the NTRU PKCS, a lattice based polynomial ring encryption/decryption scheme. We provide a performance evaluation of the NTRU and those of other popular PKCS, such as Elliptic Curve Cryptosystem (ECC) in the key creation, encryption, and decryption at different security levels. We focus on developing a lattice based digital signature/authentication scheme, in which the "Secure Hashing Algorithm-1" (SHA-1) has been employed. We also present an efficient way to create a simple format function with collision detection. The simulation results for the Personal Digital Assistants (PDAs) are given. Finally, we present an application of the lattice based digital signature with the advanced Java Database Connectivity (JDBC) technology in E-commerce.

To my wife Lihong, and our daughter Katherine

# ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my mentor, professor Michael Anshel for introducing me to cryptography analysis and earnestly supervising my dissertation work. His insightful guidance, generous encouragement, and patience make this work possible. His instructions and suggestions are the best nutrition for my research life.

I would also like to express my great appreciation to the members of my dissertation committee, professor Ted Brown, professor Jerry Waxman, professor Jerome Liang, and professor Victor Pan for their advices and help during my research.

I wish to thank professor Robert R. Goldberg and professor Zhigang Xiang from Queens College/CUNY for their time and suggestions during my qualifying exam. I sincerely appreciate the assistance and support from professor Stanley Habib and Mr. Joseph Driscoll in the computer science Ph.D. program of the City University New York.

I am grateful for all the friendship and support I have received from the faculty, staff, and graduate students of the City University of New York.

I am fortunate to be blessed with the love and support from my wife, Lihong and our daughter, Katherine. Thanks for their support and encouragement during my study.

# TABLE OF CONTENTS

# LISTS OF TABLES

# LIST OF FIGURES

# Guide to the Reader

This dissertation presents an investigation of the NTRU PKCS, a lattice based polynomial ring encryption/decryption scheme. We provide a performance evaluation of the NTRU and those of other popular PKCS, such as Elliptic Curve Cryptosystem (ECC) in the key creation, encryption, and decryption at different security levels. We focus on developing a lattice based digital signature/authentication scheme, in which the "Secure Hashing Algorithm-1" (SHA-1) has been employed. We also present an efficient way to create a simple format function with collision detection. The simulation results for the Personal Digital Assistants (PDAs) are given. Finally, we present an application of the lattice based digital signature with the advanced Java Database Connectivity (JDBC) technology in E-commerce. The thesis outline is as follows.

In Chapter 2, we cover the mathematical principles. The definition and properties of polynomial ring are also presented in this Chapter. If the reader has known these backgrounds, he/she can skip Chapter 2.

NTRU, a lattice based polynomial ring encryption/decryption scheme, is described in Chapter 3. We investigate the performance NTRU Public Key Cryptography System, including the key generation, encryption and decryption.

The NTRU PKCS is a fast and efficient technique for public key authentication, digital signature and encryption, with low memory and processor requirement. The "hard problem" that NTRU PKCS is based on is related to the difficulty of finding particularly small vectors in certain lattices with high dimensions.

It has the following features:

- Operates 100 times faster than the current cryptosystems;
- Low memory footprint;
- Fast key generation, enabling disposable keys;
- More security options; and
- Scalable from embedded devices to mainframes.

For a communication process with PKCS, there are two entities –Alice as sender and Bob as receiver. We suppose that Bob wants to receive a message from Alice, and Bob needs to create a public/private key pair for the lattice based public key cryptosystem.

**Key Creation:** The key creation process starts from two polynomials $f$ and $\mathbf{g}$ randomly chosen in the ring of truncated polynomials $\mathbf{R}$ according to the following rule:

- $f$ has $d_f$ of its coefficients equal to 1, it has $d_f$-1 of its coefficients equal to -1, and it has all of the rest of its coefficients equal to 0.
- $\mathbf{g}$ has $d_g$ of its coefficients equal to 1, it has $d_g$ of its coefficients equal to -1, and it has all of the rest of its coefficients equal to 0.

Next step is to compute the inverse of $f$ modulo q, $\mathbf{F}_q$ and the inverse of $f$ modulo p, $\mathbf{F}_p$, with the property that

$$f^*F_q = 1 \text{ (modulo q)} \quad \text{and} \quad f^*F_p = 1 \text{ (modulo p).}$$

Now Bob computes the product

$$\mathbf{h} = pF_q^*\mathbf{g} \text{ (modulo q).}$$

Bob's private key is the pair of polynomials $f$ and $F_p$. Bob's public key is the polynomial **h**.

**Encryption:** Alice wants to send a message to Bob using Bob's public key **h**. She first puts her message in the form of a polynomial **m** whose coefficients are chosen modulo p, say between -p/2 and p/2. Next she chooses randomly a polynomial **r** according to the following rule:

- **r** has $d_r$ of its coefficients equal to 1, it has $d_r$ of its coefficients equal to -1. and it has all of the rest of its coefficients equal to 0.

She uses the message **m**, her randomly chosen polynomial **r**, and Bob's public key **h** to compute the polynomial

$$\mathbf{e} = \mathbf{r}^*\mathbf{h} + \mathbf{m} \text{ (modulo q).}$$

The polynomial **e** is the encrypted message which Alice sends to Bob.

**Decryption:** Now Bob has received Alice's encrypted message **e** and he wants to decrypt it. He begins by using his private polynomial $f$ to compute the polynomial

$$\mathbf{a} = f^*\mathbf{e} \text{ (modulo q).}$$

$$\mathbf{b} = \mathbf{a} \text{ (modulo p).}$$

$$\mathbf{c} = \mathbf{F_p} * \mathbf{b} \text{ (modulo p)}.$$

The polynomial **c** will be the original message **m**.

The parameters of NTRU PKCS are as follows:

- **N** the polynomials in the truncated polynomial ring having degree N-1.
- **q** the large modulus.
- **p** the small modulus.
- $\mathbf{d_f}$ the private f key has $d_f$ coefficients equal to 1 and $d_f$-1 coefficients equal to −1.
- $\mathbf{d_g}$ the private g key has $d_g$ coefficients equal to each of 1 and −1.
- $\mathbf{d_r}$ the random encryption polynomial has $d_r$ coefficients equal to each of 1 and −1.

By using C++ as a programming language, we evaluated its performance by comparing with the Elliptic Curve Cryptosystem (ECC) at different security options.

Table 3.6 shows a speed comparison between NTRU and the Elliptic Curve Cryptosystem (ECC) PKCS in the key generation. We run our program on a Pentium III (500 MHz) with Windows NT4.0 operation system. It is noticed that the speed of NTRU is 6.25, 7.2 and 4.6 times faster than that of ECC in a speed of key generation at three different security levels as shown in Table 3.6.

From Table 3.10, it is noted that the encryption speed of the NTRU is about 44 and 40 times faster than that of the ECC in both moderate and high security level respectively. Meanwhile, the speed of the NTRU is about 27 times faster than that of the ECC at the highest security level for encryption.

Table 3.14 lists a speed comparison between NTRU and ECC PKCS during the decryption process. It is noted that the speed of the NTRU PKCS is approximately one order faster than that of the ECC PKCS at both moderate and high security levels. While at the highest security level, the decryption speed of the NTRU PKCS is 6 times faster than that of the ECC PKCS.

In Chapter 4, we provide the lattice based digital signature scheme. By extending the work from [HLS99], we modified the PASS, a lattice based authentication scheme, as a digital signature scheme by applying the SHA-1 hash function and format function. Our format function creates four polynomials with collision detection in the signing and verifying process. Experimental results are provided in this chapter. The simulation results of the NTRU 107 and NTRU 167 for PDAs are given at the end of the Chapter 4.

The NTRU 107 and NTRU 167 are considered not very secure in reality. Higher secure level of digital signature scheme is demanded in the E-Commerce. If we pursue the higher-level end-to-end security of information exchange by using NTRU 503 or NTRU 768, it'll take much more time, especially on the wireless device side. This is due to its weak computation capability. Therefore, we apply a statistic hypothesis test to determine the required verification sample size to achieve a high probability for a verification success. By this way, we can greatly increase operating speed while decreasing memory requirements. The results are listed in Table 4.20.

The lattice based polynomial digital signature scheme is based on the hard mathematical problem of finding a binary polynomial $f(X)$ that takes on prescribed values $f(S)$ modulo q are at a given collection of a set:

$$S = \{ \alpha_1, \alpha_1, \alpha_2, \ldots \alpha_{N/2} \}$$

This is an equivalent problem to solve the closest vector problem in a certain lattice. As the signer publishes the set of values $f(\alpha_i)$ as his/her public key, the verifier can use the set S and signer's public key to verify the signature of the message.

Similar to the security estimation of the RSA and ECC digital signature scheme, the security level of the lattice based polynomial digital signature scheme was given by the estimated breaking time when best-known attack method is applied to it. The best-known attack method for the lattice based polynomial digital signature scheme is the improved LLL algorithm.

The foundation of the latticed based polynomial authentication and signature scheme is the ring **R**. It consists of all truncated polynomials of degree N-1 having coefficients modulo q, i.e.,

$$f(X) = a_0 + a_1X + a_2X^2 + a_3X^3 + \ldots + a_{N-2}X^{N-2} + a_{N-1}X^{N-1} \pmod q.$$

The definitions and properties of the polynomial ring have been described in Chapter 2. For more details, please see [KRO98], [LEV90], [ORE76], and [ROS93].

There are two important operations in this truncated polynomial: Addition and multiplication. Polynomials are added in the usual way. They are also multiplied more-or-less as usual, except that $X^N$ is replaced by 1, $X^{N+1}$ is replaced by X, $X^{N+2}$ is replaced by $X^2$, and so on. Parameters of the NTRU Digital Signature Scheme are described in the section 4.2.

The message digest can be used as input to the digital signature algorithm that generates or verifies the signature for the message. Signing the message digest rather than the message itself often improves the efficiency of the process because the message digest's size is usually much smaller than that of the message. In addition, the hash algorithm used by the verifier of a digital signature must be the same as that was used by the creator of the digital signature.

In Table 4.1, we list a comparison of security level which is based on time estimation that is approximately how long it would take a single machine to break a single key by using the best-known attack method for the RSA and the NTRU cryptography system.

Table 4.2 demonstrates the approximate equivalent security level of NTRU as compared with RSA and ECC.

Figure 4.1 and Figure 4.2 demonstrate the digital signature generation and verification process by applying the SHA-1 message digest algorithm, respectively.

The lattice based digital signature scheme depend on public key set S. Let a set of values, $S = \{\alpha_1, \alpha_2, \ldots \alpha_{N/2}\} \pmod q$, is one of the public keys of the signer. the signer will publish all of the values of set S to the public. For every value of set S, if $\alpha_i \pmod q$ is in set S, then $\alpha_i^{-1} \pmod q$ must be also in set S.

The method constInv (i, q) is designed for finding $i^{-1} \pmod q$ for integer i by using the Euclidean algorithm. We have described it in details in Chapter 2. Table 4.5 shows an example of set S when q is equivalent to 769.

The private key of the lattice based digital signature scheme includes two random generated polynomials, $f_1$ and $f_2$. From the parameters explanation, we know both $f_1$ and $f_2$ have $d_f$ of their coefficients equal to 1 and −1. All the rest of their coefficients are equal to 0. In our implementation, we use a vector as our data structure for both $f_1$ and $f_2$ because a vector is more flexible than a one-dimensional array. Table 4.6 and Table 4.7 give the $f_1$ and $f_2$ private key when q is equal to 769. Table 4.8 gives the value of the public key $f$ (S).

The signer randomly selects two polynomial $g_1$ and $g_2$, which have $d_g$ of their coefficients equal to each of 1 and -1, and all of their other coefficients equal to 0. Table 4.9 and Table 4.10 show the private key $g_1$ and $g_2$ when q equals to 769.

The signer also computes the values of $g_1$ and $g_2$ for each of the elements of the set S. That is, g (S) = $(g_1(S), g_2(S)) = \{g_1(\alpha_1), g_1(\alpha_2), ...., g_1(\alpha_{N/2}), g_2(\alpha_1), g_2(\alpha_2), .... g_2(\alpha_{N/2})\}$ (mod q). The polynomials $g_1$ and $g_2$ will be kept as the signer's secret key. Then the signer sends the set of values g(S) to the verifier. We call that the set g(S) is the signer's commitment. An example of g(S) is shown in Table 4.11.

In order to use PASS to create a digital signature, we assume the hash function SHA-1 has been applied and a 160-bit string output has been produced.

The signer constructs the four polynomials $C_{ij} = (C_{11}, C_{12}, C_{21}, C_{22})$ by computing $C_{ij} = F(H(g(S)|M))$. Here M is the message, H is the hash function and F is the formatting function, respectively. Table 4.13 lists 160 bit hash values and the corresponding $C_{11}$, $C_{12}$, $C_{21}$, $C_{22}$ polynomials.

The signer computes the response **h** to the polynomial **g** and the polynomial **c**, using his/her private key $f$, in the usual way:

$$\mathbf{h} = \mathbf{c}_{11} * f_1 * \mathbf{g}_1 + \mathbf{c}_{12} * f_1 * \mathbf{g}_2 + \mathbf{c}_{21} * f_2 * \mathbf{g}_1 + \mathbf{c}_{22} * f_2 * \mathbf{g}_2 .$$

The signer's signed message for the message **M** is the signature (**g**(S),**h**). Table 4.14 shows the signature value **h**.

To verify the signed message **M** and its signature, the Verifier computes the **C** polynomials from **g**(**S**) and the signed message **M** by using the publicly available hash function **H** (i.e., SHA-1 in our example) and formatting function **F**. He/she then uses the signer's public key $f$(**S**) to verify whether the signature value **h** was generated by the signer.

We describe the details of the verification process as follows.

1.) Verifier computes four polynomials $\mathbf{C}_{ij}$ from **g**(**S**) and **M** by using the publicly available hash function **H** (e.g. SHA-1) and the formatting function **F**. Then use the polynomial **C** to computer $\mathbf{C}_{ij}$(**S**). The Table 4.15 shows the experimental results. After verifier obtains the 160-bit hash values from the SHA-1 algorithm, he applies the formatting function **F** as described before. The four polynomials $\mathbf{C}_{ij}$ are created. Table 4.16 lists the values of $\mathbf{C}_{ij}$ during the verification process.

2.) The verifier then uses the signer's public key **h** and the set **S** to calculate the value **h**(**S**). Table 4.17 shows the experimental results of **h**(**S**) values during the verification process.

3.) The verifier then uses the signer's public key $f(S)$ and $g(S)$ to verify whether signature values **h** was generated by the signer, i.e., by someone with knowledge of the private key $f$. This verifies the signer's signature on the message **M**. There are two conditions for signed message to pass the verification process.

(A) $h_0^2 + h_1^2 + h_2^2 + \ldots + h_{N-1}^2 < (B_h * q)^2$.

(B) $h(s_i) = c_{11}(s_i)*f_1(s_i)*g_1(s_i) + c_{12}(s_i)*f_1(s_i)*g_2(s_i)$

$+ c_{21}(s_i)*f_2(s_i)*g_1(s_i) + c_{22}(s_i)*f_2(s_i)*g_2(s_i) \pmod q$.

4.) The signer's signature value **h** will pass test (A) if the polynomials $f$, **g** and **c** all have very small coefficients. Thus, the polynomial:

$$\mathbf{h} = \mathbf{c}_{11}*f_1*\mathbf{g}_1+\mathbf{c}_{12}*f_1*\mathbf{g}_2+\mathbf{c}_{21}*f_2*\mathbf{g}_1+\mathbf{c}_{22}*f_2*\mathbf{g}_2$$

will also have small coefficients. Therefore, with an appropriate choice of parameters (such as the norm bound $B_h$), **h** will certainly pass test (A). On the other hand, because **h** is equal to $\mathbf{c}_{11}*f_1*\mathbf{g}_1+\mathbf{c}_{12}*f_1*\mathbf{g}_2+\mathbf{c}_{21}*f_2*\mathbf{g}_1+\mathbf{c}_{22}*f_2*\mathbf{g}_2$, the equality

$$\mathbf{h}(s_i) = \mathbf{c}_{11}(s_i)*f_1(s_i)*\mathbf{g}_1(s_i) + \mathbf{c}_{12}(s_i)*f_1(s_i)*\mathbf{g}_2(s_i) +$$
$$\mathbf{c}_{21}(s_i)*f_2(s_i)*\mathbf{g}_1(s_i) + \mathbf{c}_{22}(s_i)*f_2(s_i)*\mathbf{g}_2(s_i) \pmod q$$

will be true for all values of a modulo q. In particular, it will be true for all $s_i$ in the set S, so the response will also pass test (B). Table 4.18 lists the $c*f*g$ values during the verification process.

NTRU technology enables secure wireless services such as mobile commerce or so called m-commerce, streaming media, interactive banking, and mobile enterprise applications. It will be the emerging standard for m-commerce and transactional protection. We investigate and simulated the whole digital signature generation/ verification process on a desktop computer, which has the same CPU speed as the current PDAs. The current (May 2001) palm-based products have 33-MHz processor with 8 MB RAM (Sony's Clie, Palm m505 etc.), and the latest class PDAs has 206-MHz processor with 32 MB memory (COMPAQ: iPAQ, please see http://www.compaq.com). Both of the mobile products have the ability to surf the web. The simulation results are presented in the section 4.9.

When the customer uses a wireless device like mobile phone and the Wireless Application Protocol (WAP), the privacy of the data is in fact not guaranteed in current version of WAP1.2.1. The WAP gateway constitutes a security hole since the data is transmitted in its original, un-encrypted form in side the gateway. In WAP, the gateway is the point between the two end points where the data may be compromised.

There are three major remedies to the breach of end-to-end security in WAP. One of them is to apply application level security on top of the WAP. The lattice based digital signature scheme is the best candidate to ensure the information integrity between the wireless device and the server.

As we will know from chapter 3, the NTRU 107 and NTRU 167 are not very secure for the sensitive information, such as on-line banking or e-trade on the Internet by the wireless device. If we pursue the higher-level end-to-end security of information

exchange by using NTRU 503 or NTRU 768, it'll take much more time. Because the wireless device has relatively weak computation capability compared with the server, the process will take much longer on the wireless device side. When applying higher-level security like the NTRU 503 or NTRU 769 for m-commerce, we need find a way to decrease the computation time on the wireless device side.

By analyzing the lattice based digital signature scheme in the verifying process, we have,

$$\mathbf{h}(s_i) = \mathbf{c}_{11}(s_i)*\boldsymbol{f}_1(s_i)*\mathbf{g}_1(s_i) + \mathbf{c}_{12}(s_i)*\boldsymbol{f}_1(s_i)*\mathbf{g}_2(s_i) + \mathbf{c}_{21}(s_i)*\boldsymbol{f}_2(s_i)*\mathbf{g}_1(s_i) + \mathbf{c}_{22}(s_i)*\boldsymbol{f}_2(s_i)*\mathbf{g}_2(s_i) \pmod q.$$

This is the most timing consuming process in the verification process because there are 8 polynomial multiplication operations for verifying every number in set S. We can apply a statistic hypothesis test to determine the required sample size (less than the size of S) to achieve a high success probability for a verification success. Table 4.20 demonstrates an improvement of the verification process speed with selected sample size ($\hat{S}$) on the wireless device side.

In Chapter 5, we present an application of the lattice based digital signature scheme on a "paperless" report/billing system. With more and more of business activities being replaced by an electronic or "paperless" system, the authentication and integrity become more important than ever. Such applications as banking record and payment billing systems, internet brokerage trades, business to business portals, postal applications and healthcare professionals in hospitals or clinical areas who are performing inquiries on

sensitive patient's medical record information. This work is based on the query results of a relational Database Management System (DBMS). In addition, the Java JDBC technology has been successfully integrated into the system. JDBC technology is an API that lets you access virtually any tabular data source from the Java programming language [HC99]. It provides cross-DBMS connectivity to a wide range of SQL databases, and now, with the new JDBC API, it also provides access to other tabular data sources such as spreadsheets. The following steps show main function of the JDBC API in our "paperless" report/billing system.

1. Open the template file and reads the data into a buffer.
2. Load the JDBC-ODBC driver.
3. Set up a connection to the named data-source.
4. Use the connection to create a statement.
5. Call executeQuery on the statement to retrieve a result set.
6. Iterates over all the tuples in result set, retrieving a particular attribute each time.
7. Close the connection.
8. Merge the template file with query results.
9. Create new query result file.
10. Sign the query result file.

The Figure 5.1 shows the structure of our "paperless" report/billing system. From the figure 5.1 we can see that the whole reporting system can be divided into five steps:

- Query from DBMS.
- Query results data merge with the template file.
- Signing the merged data.
- Send signed document through the Internet.
- Verification process by the receiver.

The system can use any relational DBMS in the current market. In our example, we use Microsoft Access 2000 as our database driver. The reason we use Access is that relational databases are very popular and have many advantages over other methods for the construction of data processing systems. Figure 5.2 shows the structure of the database ETS.mdb.

The template file is a standard file for the "paperless" report/billing system. In our TOFEL "paperless" reporting system, before we run the query, the template file looks like Figure 5.3. The template file, which includes some basic information about ETS, doesn't contain any information for a specific examinee before we merge it with query results. Figure 5.4 shows query results for Mr. John Doe, with a registration number equal to 7715703 for the TOFEL exam.

During the transmission, if the file is modified, the modified file can't pass the verification process as it is described in the example of section 5.5. Table 5.2 lists the signature of the file after the sender signs query results.

The NTRU PKCS has achieved considerable attentions in the cryptographic community. It also receives attacks from time to time since it's proposed. Following these attacks, the NTRU PKCS has been enhanced from both encoding and decoding methods. The parameters of NTRU have been carefully re-selected after its first version was cracked. To maintain the integrity of the NTRU PKCS, we described attacks on the NTRU PKCS in the Chapter 6.

The section 6.1 described the lattice reduction attack on the NTRU PKCS. The first attack that NTRU received came from Coppersmith and Shamir. It indicated the NTRU system was not reliably secure after they analyzed the first version of the NTRU using the lattice reduction method [CS97]. Because of this, the parameters of NTRU were cautiously chosen and the system was significantly improved. Table 6.1 shows the average time of recovering private key $f$ from $\{f(\alpha)\}$ by using the lattice reduction method. A sequence of primes q , N = q-1 and $d_f \approx [q/3]$ are used in the experiments [HLS99]. By using the regression line to extrapolate the breaking time for larger values of N, we estimate the breaking time for a large number N as shown in Table 6.2.

The section 6.2 introduced the Zero-forced Lattice Attack. Alexander May has given an improved method for searching small vectors that have a comparatively large number of coordinates equal to 0 [MAY99]. These ideas lead to the notion of zero-forced lattices, in which one guesses that r particular coordinates of the target are 0, and forces them to be zero, thereby reducing the dimension of the lattice. Alexander May pointed out in [MAY99] that there is a very efficient method in using the lattice reduction algorithm to find the target vectors from a smaller dimensional lattice in which the specified coordinates are forced to equal to zero.

The section 6.3 described a passive attack from signature's interception introduced by Ilya Mironov [MIR00]. This showed a passive attack for private key recovery is only from the signature interception.

Chapter 7 gives the conclusions and future work of this thesis.

# CHAPTER 1

# Introduction

Today's rapid escalation in intelligence devices presents a unique opportunity for an enterprise to extend applications and services beyond the desktop, but also brings big challenges to computer security developers. Cellular phones, Personal Digital Assistants (PDAs), vendor machines, casher-machines, parking meters, or ATMs that previously stood alone can now easily be connected to the Internet or an enterprise's intranet via wire or wireless devices. In order to protect commercially sensitive information and communications on these devices and servers, the security requirements of these servers increases, and so does the burden of cryptography on these commercial servers. How to provide fast, top-quality, fully supported security protocols to protect communications between all these wireless or embedded devices and commercial servers is critical for computer security developers.

The NTRU PKCS delivers significant advantages for Internet applications and electronic devices that use "real-time" technology, such as Internet telephony and video/audio streaming. Mobile phones, smart cards and personal digital assistants--

devices that use "embedded" chipsets also benefit from NTRU. Neither Internet streaming techniques nor mobile electronic devices have the bandwidth to use current, memory intensive encryption techniques. NTRU's encryption approach, on the other hand, makes it much easier to quickly bring excellent security to these systems, many of which couldn't support encryption and authentication at all prior to this.

NTRU PKCS is based on a unique and highly sophisticated mathematical system that simplifies and speeds the creation of encryption ``keys" necessary for all types of secure digital transactions. Because NTRU's mathematics significantly reduces the memory requirements needed by the current encryption systems, it can generate new "disposable" keys in real-time for each and every secure transaction. This not only increases security, but also means the NTRU approach can be applied to an increasingly popular Internet telephony, using video and audio streaming techniques. These opportunities, in turn, open entirely new encryption and authentication markets.

## 1.1 Background of the Public Key Cryptography System

The concept of the public key cryptography was invented by Whitfield Diffie and Martin Hellman [DF76B], as well as independently by Ralph Merkle [MER78]. A Public Key Cryptosystem is an algorithmic method for securely sending private information over an insecure channel where the communication parties have no common-shared secret (e.g., key). At the heart of a PKC is a two-party secure computation referred to as a

protocol. The major PKCs and their associated protocols in use today are based on finite abelian groups [ANS99].

The development of the Public Key Cryptography System is the "most significant" and perhaps the only true revolution in the entire history of cryptography [STA98].

### 1.1.1 Characteristics of the PKCS

Public-key algorithms rely on one key for encryption and a different, but related, key for decryption [STA98]. They have the following characteristics:

- It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.
- Either of the two related keys can be used for encryption, while the other is used for decryption.
- Each end system in a network generates a pair of keys to be used for encryption and decryption of messages that it will receive.
- Each system publishes its encryption key by placing it in a public register or file. This is the public key. The companion key is kept private.
- If the sender wishes to send a message to the receiver, he encrypts the message by using the receiver's public key.

- When the receiver obtains the message, the receiver decrypts it using the receiver's private key. No other recipient can decrypt the message because only the receiver knows his private key.

### 1.1.2 General Description of the PKCS Digital Signature Scheme

The digital signature is analogous to a handwritten signature. It must have the following properties [STA98]:

- It must be able to verify the author, date and time of the signature.
- It must be able to authenticate the contents at the time of the signature.
- The signature must be verifiable by third parties, to resolve disputes.

On June 30, 2000, former president Bill Clinton signed the "Digital Signature" bill. The bill, officially known as the "Electronic Signatures in Global and National Commerce Act" [http://www.ecommerce.gov/ecomnews], gives electronic signatures and documents the same force in law as those done with ink on paper. The act eliminates legal barriers to use electronic technology in forms and signed contracts, collect and store documents, send and receive notices and disclosures.

There are several possible protocols for signing in the public-key cryptography, depending on the services desired. Usually, there are three situations as follows [STA98].

1. In the simplest case, entity A simply signs a message M by sending:

$$S = D_A(M)$$

to entity B. Here, D represents Digital signature algorithms and S means signature.

2. If a hash function H is used, entity A computes:

$$S = D_A(H(M))$$

and sends both M and S to entity B. Entity B validates entity A's signature S by computing:

$$H(M) = E_A(S)$$

It is noted here that because $E_A$ is a trapdoor one-way function. It should not be possible for an intruder to find S' such that:

$$H(M) = E_A(S')$$

for a given message M. Thus entity A's signature cannot be forged. Also, if entity A attempts to repudiate the M sent to entity B, entity B may present M and S to a judge. The judge can access $E_A$ and hence can verify:

$$H(M) = E_A(S)$$

If we assume the trapdoor function $D_A$ is indeed secret, only entity A could have sent S. Thus a priori we have is non-repudiation.

3. For both authenticity and secrecy, entity A could send:

$$M' = E_B(M, D_A(H(M)))$$

to entity B; Entity B computes:

$$(M,D_A(H(M))) = D_B(M')$$

and then verifies that:

$$H(M) = E_A(D_A(H(M)))$$

The scheme is illustrated in Figure 1.1:

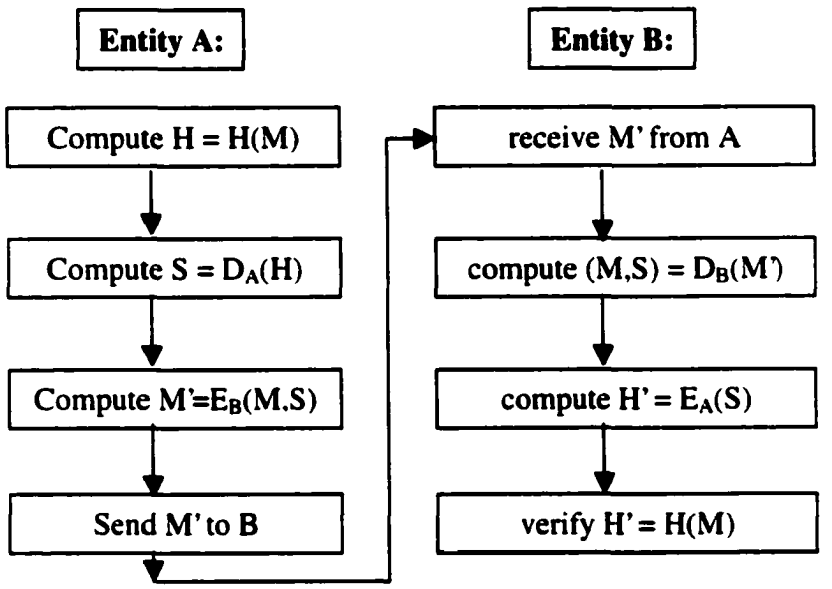



Figure 1.1 A digital signature scheme for Signing with the hash function.

For a non-repudiation case, entity B retains:

$$D_B(M') = (M,D_A(H(M))).$$

A judge can apply $E_A$ to $D_A(H(M))$ and compare to H(M). This again assumes common domains for the Es and Ds. The preceding schemes also satisfy another desirable property: they do not compromise security by exposing private components to a judge during the adjudication process. The use of a central authority is suggested for this

purpose. In this scheme, the receiver of a message sends a copy to the central authority. The latter can attest to the instantaneous validity of the sender's signature. The sender's private component has not been compromised at the time of sending.

### 1.1.3 The Lattice Based Public Key Cryptography System NTRU

The lattice based public key cryptography system NTRU is a new PKCS. It was invented by 3 scientists in the Brown University with the following features [HPS98]:

- High security at high speed
- Short, easily generated keys
- Low memory requirements
- Low processing power
- High flexibility & customization
- Disposable key options
- Authentication and digital signatures

The basic principles of the lattice based PKCS is a collection of mathematical ideas based on manipulating vectors of very small integers [HLS99] [HPS00A]. All of these integers are less than q which is a big prime in the lattice based PKCS. These allow lattice based PKCS to achieve high speeds with the use of minimal computing power. Comparable to the current public key cryptosystems at different security levels, the computations performed in the execution of the lattice based PKCS algorithms involve more simplified processes, such as the addition and multiplication of short integers, that are performed very quickly on even inexpensive 8-bit processors.

### 1.1.4 The "Hard Problem" of the Lattice Based Digital Signature Scheme

The "hard problem" in NTRU, the lattice based authentication and digital signature scheme, is that it is difficult to simultaneously control both the values and the coefficients of a polynomial over a finite field [HPS98]. This principle is the discrete analogue for finite fields of the "Heisenberg Uncertainty Principle." The problem can be solved using the lattice reduction methods. But if N is sufficiently large, the underlying lattice problem is too difficult to solve using current techniques. Briefly, the Prover publishes a set of values $f(\alpha_i)$ as his public key, and he proves his identity by demonstrating he possesses a binary polynomial of those values.

## 1.2 Contributions of the thesis

1. Evaluation of the NTRU system including key generation, encryption and decryption. A performance evaluation of the NTRU and ECC schemes is given at moderate, high and highest security levels, respectively. The simulation results of the NTRU on the PDAs are presented.

2. Apply the "Secure Hashing Algorithm-1" ("SHA-1") to the Polynomial Authentication Signature Scheme (PASS) [HLS99].

The "SHA-1" hashing algorithm is described in Section 4.4. The "SHA-1" is a very secure message digest algorithm. It produces a total of 160 bits as the output. We modified the PASS as a digital signature scheme by applying our format function using these 160-bit data to create the four polynomials.

3. Create a simple format function with collision detection.

A format function is presented in section 4.6 by extending the work in [HLS99]. After the "SHA-1" is applied, the digital signature scheme needs four polynomials: $C_{11}$, $C_{12}$, $C_{21}$, $C_{22}$ from the digested message for signing and verifying. We describe an efficient way to create these four polynomials with collision detection. These polynomials will be used in the digital signature scheme.

4. Provide a solution for current WAP security problem.

Because the wireless device has relatively weak computation capability compared with the server, the process will take much longer on the wireless device side when applying high-level security. By analyzing the lattice based digital signature scheme in the verifying process, we apply a statistic hypothesis test to determine the required sample size (less than the size of S) to achieve a high success probability for a verification success.

5. Apply the lattice based digital signature in an electronic report/billing system.

The application is presented in chapter 5, where the lattice based digital signature scheme is applied in a database query system. When a query is executed, the results are sent to buffer. The query results will merge with the template file in a buffer and the secure hashing algorithm "SHA-1" is applied to create message digest. Finally, the lattice based digital signature scheme is used and the signature is produced. The query result

data could be sent out with its digital signature via the Internet. By using the "disposable keys" function from NTRU PKCS, the application can sign the query results in real time.

# CHAPTER 2

# Polynomial Ring

In this chapter we briefly introduce the mathematical backgrounds and the properties of the polynomial ring. All these properties are the basis for the following chapters.

We frequently refer to four sets of numbers: integers, rational numbers, real numbers and complex numbers. We use special symbols for these sets, as follows:

$\mathbb{Z}$ represents the set of integers.

$\mathbb{Q}$ represents the set of rational numbers.

$\mathbb{R}$ represents the set of real numbers.

$\mathbb{C}$ represents the set of complex numbers.

## 2.1 Congruence in $\mathbb{Z}$ and Modular Arithmetic

**Definition:** Let a, b and n be integers, with $n > 0$. Then a is congruent to b modulo n, written

$a \equiv b \pmod{n}$, provided that n divides a-b, or $(a-b) \mid n$.

**Theorem 2.1** Let n be a positive integer and a, b, $c \in \mathbb{Z}$,

(1) $a \equiv a \pmod{n}$;

(2) if $a \equiv b \pmod{n}$, then $b \equiv a \pmod{n}$;

(3) if $a \equiv b \pmod{n}$ and $b \equiv c \pmod{n}$, then $a \equiv c \pmod{n}$;

(4) if $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then $a + c \equiv b + d \pmod{n}$ and $ac \equiv bd \pmod{n}$;

The proofs could be found on any number theory books, so we skip them here. For further details, please see [CAM98], CFV86], and [LEV90].

**Definition:** Let a and n be integers with $n > 0$. The congruence class of a modulo n, which denoted [a] as a set of all those integers that are congruent to a modulo n, that is,

$$[a] = \{ b \mid b \in \mathbb{Z} \text{ and } b \equiv a \pmod{n} \}.$$

Here $b \equiv a \pmod{n}$ means b-a is a multiple of n, or $b-a = n*k$ for some integer of k. Then we have:

$$[a] = \{ b \mid b \equiv a \pmod{n} \} = \{ b \mid b = a + k*n \text{ with } k \in \mathbb{Z} \} = \{ a + k*n \mid k \in \mathbb{Z} \}.$$

**Definition:** The set of all congruence classes modulo n is denoted $\mathbb{Z}_n$.

The finite set $\mathbb{Z}_n$ is closely related to the infinite set $\mathbb{Z}$. $\mathbb{Z}_n$ also has addition and multiplication operations.

**Definition:** Addition and multiplication in $\mathbb{Z}_n$ are defined as follows:

(1) $[a] + [c] = [a + c]$;

(2) $[a][c] = [ac]$;

The properties of $\mathbb{Z}_n$ are listed as follows:

1. If [a] $\in \mathbb{Z}_n$ and [b] $\in \mathbb{Z}_n$, then [a] + [b] $\in \mathbb{Z}_n$.
2. [a] + ([b] +[c]) = ([a]+[b]) + [c].
3. [a] + [b] = [b] + [a].
4. [a] + [0] = [a] = [0] + [a].
5. For each [a] in $\mathbb{Z}_n$, the equation [a] + X = [0] has a solution in $\mathbb{Z}_n$.
6. If [a] $\in \mathbb{Z}_n$ and [b] $\in \mathbb{Z}_n$, then [a][b] $\in \mathbb{Z}_n$.
7. If [a] $\in \mathbb{Z}_n$ and [b] $\in \mathbb{Z}_n$, then [a][b] $\in \mathbb{Z}_n$.
8. [a]([b] + [c]) = [a][b] + [a][c]; ([a]+[b])[c] = [a][c] + [b][c].
9. [a][b] = [b][a].
10. [a][1] = [a] = [1][a].

We also skip the proofs for these properties. The reader can find them on [KRO98], [LEV90], [ORE76] and [ROS93]. At the end of this section we give the most important theorem in the integer modular operation.

**Theorem 2.2(Fermat)** If a is an integer and $p$ is a prime, then $a^p \equiv a \pmod{p}$.

Let $p$ be a prime and P(n) be the proposition that $n^p \equiv n \pmod{p}$. Then P(0) and P(1) are hold. From the binomial expansion of $(n+1)^p$, we know that every coefficient except the first and the last is divisible by $p$. So we get $(n+1)^p \equiv n^p + 1 \pmod{p}$. P(n) implies $(n+1)^p \equiv n+1 \pmod{p}$, which means the proposition P(n+1) hold.

We will present the polynomial ring and its properties in the next section.

## 2.2 Ring and Its Properties

**Definition**: A ring R is a set with two binary operations: addition (denoted by a + b) and multiplication (denoted by ab), such that for all a, b, c in R:

1. $a + b = b + a$. **(Commutative law of addition)**
2. $(a + b) + c = a + (b + c)$. **(Associative law of addition)**
3. There exists $0 \in R$ such that $a + 0 = a$. **(Neutral element for addition)**
4. For each $a \in R$ there exists $-a \in R$ such that $a + (-a) = 0$. **(Additive inverse)**
5. $a(bc) = (ab)c$. **(Associative law of multiplication)**
6. $a(b+c) = ab + ac$ and $(b+c)a = ba + ca$. **(Distributive laws)**

A ring R also has additional algebraic properties. We list as follows:

(1) A ring R is commutative if $ab = ba$ for all a, b in R.

(2) A ring R has an identity if there is an element 1 in R such that

$$1a = a1 = a \quad \text{for all } a \text{ in } R.$$

(3) A ring R is an integral domain if it is commutative and if, for all a, b, in R, the equality $ab = 0$ implies $a = 0$ or $b = 0$.

(4) An element u in a ring R with an identity is a unit if there is an element $u^{-1}$ in R such that $uu^{-1} = u^{-1}u = 1$. We call $u^{-1}$ the multiplicative inverse of u.

So, a ring is an Abelian group under addition, also has an associative multiplication that is left and right distributive over addition.

A nonzero element a in a commutative ring R is called a **zero-divisor** if there is a nonzero element b in R such that ab = 0. A commutative ring with a unity is said to be an **integral domain** if it has no zero-divisor.

**Definition:** Let R be a commutative ring. The set of formal symbols $R[x] = \{ a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 \mid a_i \in R$, n is a nonnegative integer $\}$ is called the ring of polynomials over R in the indeterminate $x$.

Two elements

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

and

$$b_m x^m + b_{m-1} x^{m-1} + \ldots + b_1 x + b_0$$

of R[x] are considered equal if, and only if, $a_i = b_i$ for all nonnegative integers i. ( Define $a_i = 0$ when $i > n$ and $b_i = 0$ when $i > m$).

In this definition, the symbols x, $x^2$, . . . $x^n$ do not represent "unknown" elements or variables from the ring R. Rather, their purpose is to serve as convenient placeholders that separate the ring elements $a_n, a_{n-1}, \ldots, a_0$.

A polynomial over R can also be represented by a sequence

$$(a_0, a_1, \ldots, a_{n-1}, a_n)$$

in which each term $a_i \in R$.

**Definition**: Addition and Multiplication in R[x]

Let R be a commutative ring and let

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

$$g(x) = b_m x^m + ba_{m-1} x^{m-1} + \ldots + b_1 x + b_0$$

belong to R[x]. Then

$$f(x) + g(x) = (a_s + b_s) x^s + (a_{s-1} + b_{s-1}) x^{s-1} + \ldots + (a_1 + b_1) x + a_0 + b_0$$

where s is the maximum of m and n, $a_i = 0$ for $i > n$, and $b_i = 0$ for $i > m$. Also,

$$f(x)g(x) = c_{m+n} x^{m+n} + c_{m+n-1} x^{m+n-1} + \ldots + c_1 x + c_0,$$

where $c_k = a_k b_0 + a_{k-1} b_1 + \ldots + a_1 b_k$, for $k = 0, 1, \ldots, m + n$.

**Definition**: If polynomial ring $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$, where $a_n \neq 0$, we say $f(x)$ has degree n and we write $\deg(f) = n$. The term $a_n$ is called the leading coefficient of $f(x)$.

## 2.3 The Division Algorithm and Consequences

### 2.3.1 The Euclidean Algorithm

The division algorithm plays an important role in the study of the divisibility of the natural numbers. Its form is: given any two integers a, b, if $b \neq 0$, there exist integers q, r such that:

$$a = bq + r, \ 0 \leq r < |b|.$$

Finding the numbers q and r may be interpreted as a partial division of a by b, with quotient q and remainder r. Writing $a = a_0$, $b = a_1$, we can form a chain $a_0, a_1, a_2, \ldots$

where $a_{i+1}$ is the remainder in the partial division of $a_{i-1}$ by $a_i$. The process, called the Euclidean algorithm (after Euclid of Alexandria, ca. 300 BC), terminates when we reach a zero remainder, as we will, because the values $a_2$, $a_3$, ... form a strictly decreasing sequence of positive integers; we shall find that the last now-zero remainder is the highest common factor of a and b.

It turns out that such a Euclidean algorithm also exists for the integers in certain algebraic fields, and for polynomial rings over a field. Next we shall describe it in general terms before applying it in particular cases.

**2.3.2 Theorem: Division Algorithm in R[x]**

Let R be a ring and let $f(x) \in R[x]$ be a polynomial of degree n as following:

$$f(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$$

whose leading coefficient $a_n$ is a unit of R.

For any polynomial $g(x) \in R[x]$, there exist unique polynomials q(x) and r(x) such that

$$g(x) = f(x)q(x) + r(x) \qquad (1)$$

and either $r(x) = 0$ or $\deg(r(x)) < \deg(f(x))$.

We begin by showing the existence of q(x) and r(x). First, in the case $g(x) = 0$, we simply choose $q(x) = r(x) = 0$. Next we proceed by induction method on the degree of g(x). For a fixed polynomial $f(x)$, whose leading coefficient $a_n$ is a unit, if $\deg(g(x)) = k$, there exist polynomials q(x) and r(x) such that (1) holds. The crucial fact is that $a_n$, being a unit, has an inverse $a_n^{-1}$.

The induction hypothesis states the division algorithm may be carried out on polynomials g(x) of a degree less than k. Suppose deg(g(x)) = k, then we have:

$$g(x) = b_0 + b_1x + \ldots + b_kx^k.$$

Of course, if deg($f$(x)) > deg(g(x)), choose q(x) = 0 and r(x) = g(x) to obtain the result. This will always be the case if k = 0, which starts the induction, unless n = k = 0. In this case $f$ is a scalar polynomial that is a unit of R, and hence divides all polynomials in R[x]. Now, we handle the induction step:

Suppose deg(g(x)) = k ≥ deg($f$(x)). Find

$$f(x)(a_n)^{-1}\, b_kx^{k-n} = b_0 + \ldots + b_kx^k$$

so

$$h(x) = g(x) - f(x)(a_n)^{-1}\, b_kx^{k-n}$$

has a degree that is less than k. Hence, by induction, there exist polynomials q(x) and r(x) such that

$$h(x) = f(x)q(x) + r(x)$$

where either r(x) = 0 or deg(r(x)) < deg($f$(x)). Now equate the two expressions for h(x), solve for g(x), and collect terms:

$$g(x) = f(x)(a_n)^{-1}b_kx^{k-n} + f(x)q(x) + r(x)$$

$$= f(x)((a_n)^{-1}b_kx^{k-n} + q(x)) + r(x)$$

so the inductive step has been completed.

To show uniqueness, suppose that g(x) = $f$(x)q(x) + r(x) and g(x) = $f$(x)q*(x) + r*(x), where r(x) = 0 or deg(r(x)) < deg(g(x)) and r*(x) = 0 or deg( r*(x)) < deg(g(x)). Then, subtracting these two equations, we obtain:

$$0 = f(x)[q(x)- q^*(x)] + [r(x)- r^*(x)]$$

or $r^*(x) - r(x) = f(x)[q(x) - q^*(x)]$.

Thus, $r^*(x) - r(x)$ is 0, or the degree of $[r^*(x) - r(x)]$ is at least that of $f(x)$. Since the latter is clearly impossible, we have $r^*(x) = r(x)$ and $q(x) = q^*(x)$ as well.

The polynomials $q(x)$ and $r(x)$ in the division algorithm are called the quotient and remainder in the division of $g(x)$ by $f(x)$.

**2.3.3 Remainder Theorem:** if R is a commutative ring with identity, then for any $g(x) \in R[x]$ and $a \in R$

$$g(x) = (x-a)q(x) + g(a).$$

By using the division algorithm above with $f(x) = x - a$, a polynomial of degree 1. to obtain

$$g(x) = (x-a)q(x) + r$$

where the remainder either is 0 or has degree 0. In any event, r is an element of R. Use the substitution theorem to find

$$g(a) = (a - a)q(a) + r$$

and hence, $r = g(a)$.

## 2.4 Unique Factorization in Z[x]

We are all quite familiar with the factorization of an integer into prime numbers. When the integers are introduced, it is shown that every integer $m>1$, can either be factorized to $m = rs$ -- where the factors satisfy $r, s > 1$ -- or no such factorization is possible. In the latter case, m is called a prime number. It is further shown that every

positive integer has a complete factorization, i.e., it can be expressed as a product of primes, and any two complete factorizations differ only in the order of the prime factors. This result is known as the "Fundamental Theorem of Arithmetic." Its proof depends on the Euclidean algorithm, and we shall see that a corresponding result holds for any Euclidean domain.

Now we consider the problem of factoring polynomials. In order to discuss the factorization of polynomials, we must first introduce the polynomial analog of a prime integer.

**Definition:** Irreducible Polynomial and Reducible Polynomial

Let I be an integral domain. A polynomial $f(x)$ from I[x] that is neither the zero polynomial nor a unit in I[x] is said to be irreducible over I if, whenever $f(x)$ is expressed as a product $f(x) = g(x)h(x)$, with $g(x)$ and $h(x)$ from I[x], then $g(x)$ or $h(x)$ is a unit in I[x]. A nonzero, nonunit element of I[x] that is not irreducible over I is called reducible over I.

If the integral domain is a field F, the irreducible is to define a nonconstant $f(x) \in F[x]$ to be irreducible if $f(x)$ cannot be expressed as a product of two polynomials of lower degree.

**Definition:** Content of Polynomial and Primitive Polynomial

The content of a nonzero polynomial $a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$, where the a's are integers, is the greatest common divisor of the integers $a_n$, $a_{n-1}$, ..., $a_0$. A primitive polynomial is an element of Z[x] with content 1.

**Gauss's Lemma**: The product of two primitive polynomials is primitive.

Let $f(x)$ and $g(x)$ be primitive polynomials, and suppose that $f(x)g(x)$ is not primitive. Let p be a prime divisor of the content of $f(x)g(x)$, and let $\overline{f}(x)$, $\overline{g}(x)$ and $\overline{f(x)g(x)}$ be the polynomials obtained from $f(x)$, $g(x)$ and $f(x)g(x)$ by reducing the coefficients modulo p. Then, $\overline{f}(x)$ and $\overline{g}(x)$ belong to the integral domain $Z_p[x]$ and $\overline{f}(x)\overline{g}(x) = \overline{f(x)g(x)} = 0$, the zero element of $Z_p[x]$. Thus, $\overline{f}(x)$ or $\overline{g}(x) = 0$. This means that either p divides every coefficient of $f(x)$ or p divides every coefficient of $g(x)$. Hence, either $f(x)$ is not primitive or $g(x)$ is not primitive.

**Theorem**: Unique Factorization in Z[x]

Every polynomial in Z[x] that is not the zero polynomial or a unit in Z[x] can be written in the form:

$$b_1b_2\ldots b_sp_1(x)p_2(x)\ldots p_m(x) = c_1c_2\ldots c_tq_1(x)q_2(x)\ldots q_n(x),$$

where the b's and c's are irreducible polynomials of degree 0, and the p(x)'s and q(x)'s are irreducible polynomials of positive degree, then s=t, m=n, and, after renumbering the c's and q(x)'s, we have $b_i = \pm c_i$ for i=1, ... , s; and $p_i(x) = \pm q_i(x)$ for i=1, ... , m.

Let $f(x)$ be a nonzero, nonunit polynomial from Z[x]. If deg $(f(x)) = 0$, then $f(x)$ is constant and the result follows from the Fundamental Theorem of Arithmetic. If deg $(f(x)) > 0$, let b denote the content of $f(x)$, and let $b_1b_2\ldots b_s$ be the factorization of b as a product of primes. Then $f(x) = b_1b_2\ldots b_sf_1(x)$, where $f_1(x)$ belongs to Z[x], is primitive and deg $(f_1(x))$ = deg $(f(x))$. Thus, to show the existence portion of the theorem, it suffices

to show that a primitive polynomial $f(x)$ of positive degree can be written as a product of irreducible polynomials of positive degree. We proceed by induction on deg ($f(x)$). If deg ($f(x)$) = 1, then $f(x)$ is already irreducible, and we are done. Now, suppose every primitive polynomial of a degree less than deg($f(x)$) can be written as a product of irreducibles of positive degree. If $f(x)$ is irreducible, there is nothing to prove. Otherwise, $f(x) = g(x)h(x)$, where both $g(x)$ and $h(x)$ are primitive and have a degree less than that of $f(x)$. Thus, by induction, both $g(x)$ and $h(x)$ can be written as a product of irreducibles of positive degree. Clearly, then $f(x)$ is also such a product.

To show the uniqueness portion of the theorem, suppose that

$$f(x) = b_1b_2\ldots b_sp_1(x)p_2(x)\ldots p_m(x) = c_1c_2\ldots c_tq_1(x)q_2(x)\ldots q_n(x),$$

where the b's and c's are irreducible polynomials of degree 0, and the p(x)'s and q(x)'s are irreducible polynomials of degree 0, and the p(x)'s and q(x)'s are irreducible polynomials of positive degree. Let $b = b_1b_2\ldots b_s$ and $c = c_1c_2\ldots c_t$. Since the p(x)'s and q(x)'s are primitive, it follows from Guass's Lemma that $p_1(x)p_2(x)\ldots p_m(x)$ and $q_1(x)q_2(x)\ldots q_n(x)$ are primitive. Hence, both b and c must equal plus or minus the content of $f(x)$ and, therefore, are equal in absolute value. It then follows from the Fundamental Theorem of Arithmetic that s = t and, after renumbering, $b_i = \pm c_i$ for i = 1, 2, . . . s. Thus, by canceling the constant terms in the two factorizations for $f(x)$, we have:

$$p_1(x)p_2(x)\ldots p_m(x) = \pm q_1(x)q_2(x)\ldots q_n(x).$$

Now, viewing the p(x)'s and q(x)'s as elements of Q[x] and noting that $p_1(x)$ divides $q_1(x)q_2(x)\ldots q_n(x)$, it follows from induction that $p_i(x)|q_i(x)$ for some i. By renumbering, we may assume i =1. Then since $q_1(x)$ is irreducible, we have $q_1(x) = (r/s)p_1(x)$, where r, s $\in$ Z. However, because both $q_1(x)$ and $p_1(x)$ are primitive, we must have r/s = $\pm$1. So,

$q_1(x) = \pm p_1(x)$. Also, after canceling, we have $p_2(x)...p_m(x) = \pm q_2(x)...q_n(x)$. Now, we may repeat the above argument with $p_2(x)$ in place of $p_1(x)$. If $m < n$, after m such steps we would have 1 on the left and a nonconstant polynomial on the right. Clearly, this is impossible. On the other hand, if $m > n$, after n steps we would have $\pm 1$ on the right and a nonconstant polynomial on the left – another impossibility. So, $m = n$ and $p_i(x) = \pm q_i(x)$ after suitable renumbering of the q(x)'s.

# CHAPTER 3

# The Lattice Based Encryption/Decryption Scheme

## 3.1 Overview of the Lattice Based NTRU PKCS

Finding and applying a fast encryption/decryption algorithm for the embedded or wireless devices with low memory and weak microprocessor such as PDAs, cellular phones, gas pumps, and casher machines is a big problem for cryptanalyst. A lattice based Public Key Crypto System, which is named NTRU, was first presented in 1996 by three mathematicians, Jeffrey Hoffstein, Jill Pipher and Joseph H. Silverman from Brown University. It is a good solution for this problem.

The NTRU PKCS is a fast and efficient collection of the techniques for public key authentication, digital signature and encryption, with low memory and processor requirement. The "hard problem" that NTRU PKCS is based on is related to the difficulty of finding particularly small vectors in certain lattices with high dimensions.

It has the following features as claimed by its inventors:

- Operates 100 times faster than the current cryptosystems;
- Low memory footprint;
- Fast key generation, enabling disposable keys;
- More security options; and
- Scalable from embedded devices to mainframes.

### 3.2 The NTRU PKCS Parameters

The Lattice Based Public Key Cryptosystem is based on basic operations of the ring **R** that consists of all truncated polynomials of degree N-1, having integer coefficients as the follows:

$$f(x) = a_0 + a_1X + a_2X^2 + a_3X^3 + \ldots + a_{N-2}X^{N-2} + a_{N-1}X^{N-1}.$$

There are two operations in the lattice based PKCS: addition and multiplication. For more details of the descriptions, please see [ORE76], [SIL97], [COH93] and [KOB94].

The addition operations of polynomials are like the ordinary mathematic addition operation. The multiplications are done in almost the same way as usual, except that $X^N$ is replaced by 1, $X^{N+1}$ is replaced by X, $X^{N+2}$ is replaced by $X^2$ and so on.

In the NTRU Public Key Cryptosystem, the number N specifies the degree of the truncated polynomials. Other parameters are described as follows:

| | |
|---|---|
| **N** | the polynomials in the truncated polynomial ring having degree N-1 |
| **q** | the large modulus |
| **p** | the small modulus |
| $\mathbf{d_f}$ | the private ***f*** key has $d_f$ coefficients equal to 1 and $d_f$-1 coefficients equal to –1 |
| $\mathbf{d_g}$ | the private **g** key has $d_g$ coefficients equal to each of 1 and -1 |
| $\mathbf{d_r}$ | the random encryption polynomial has $d_r$ coefficients equal to each of 1 and –1 |

Table 3.1 lists some typical values for the NTRU PKCS parameters at various security levels. In order to ensure security, it is crucial that p and q have no common factors.

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

Table 3.1 Parameters of the lattice based PKCS

We will describe every step of the NTRU PKCS by using an example of N is equivalent to 167 in the following sections.

### 3.3 Key Generation of the Lattice Based NTRU PKCS

In our cryptosystem, there are two entities –Alice as sender and Bob as receiver. Now suppose Bob wants to receive a message from Alice, and Bob needs to create a public/private key pair for the lattice based public key cryptosystem. He first randomly chooses two polynomials, *f* and g, in the ring of the truncated polynomials R, according to the following rules:


Figure 3.1 Communication through the public channel.

- *f* has $d_f$ of its coefficients equal to 1; it also has $d_f$-1 of its coefficients equal to -1, and all of the rest of its coefficients equal to 0. Table 3.2 lists the *f* polynomial values when N is equivalent to 167.

| The private key *f* polynomial (N = 167) |
|---|
| 0 0 -1 -1 0 1 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 -<br>1 1 -1 0 0 0 0 -1 0 0 0 0 0 1 -1 0 1 0 1 0 1 0 0 0 0 0 1 -<br>1 -1 0 -1 1 0 0 0 -1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0<br>0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 -1 0 0 -1 1 0 -1 0 0 1 |

Table 3.2 The *f* polynomial values.

- g has $d_g$ of its coefficients equal to 1; it also has $d_g$ of its coefficients equal to -1, and all of the rest of its coefficients equal to 0. Table 3.3 shows the g polynomial values with N equivalent to 167.

| The private key g polynomial (N = 167) |
|---|
| 0 0 0 -1 0 1 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 -1<br>1 -1 0 0 0 0 -1 0 0 0 0 0 1 -1 0 1 0 1 0 1 0 0 0 0 1 -1 -<br>1 0 -1 1 0 0 0 -1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0<br>0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 -1 0 0 -1 0 0 1 |

Table 3.3 The g polynomial values.

In the NTRU, because the values of $f$ and g are the two important components needed to generate the public and private key, Bob must keep them private. Anyone who knows the value of either one could decrypt messages sent to Bob.

Next, Bob should compute the inverse values of $f$ modulo q and the inverse values of $f$ modulo p. Thus, he computes polynomials $F_q$ and $F_p$ with the property that

$$f*F_q = 1 \text{ (modulo q)} \quad \text{and} \quad f*F_p = 1 \text{ (modulo p)}.$$

If these inverse values do not exist, Bob needs to go back and choose another $f$ until he can get both $F_q$ and $F_p$. Table 3.4 list the inversions of $f$ polynomials, $F_q$ and $F_p$.

| $F_p$ (N = 167, p = 3) |
|---|
| 1 1 0 1 0 2 2 0 0 1 1 2 1 2 0 0 2 0 0 2 2 0 1 2 0 1 0 0 0 2 1 2 2 0 1 1 2 1 0 1 1 0 0 2 1 2 0 2 0 0 1 0 2 0 2 1 0 2 1 0 0 1 0 1 1 1 0 0 1 0 2 1 2 2 1 2 0 1 1 2 1 2 1 0 1 0 1 1 0 0 2 0 1 1 0 0 1 0 1 2 0 2 0 2 0 1 1 |
| $F_q$ (N = 167, q = 64) |
| 5 4 18 49 52 35 44 60 38 0 27 62 34 46 56 52 13 47 16 27 20 6 5 63 56 47 51 51 53 36 35 1 63 6 4 28 11 29 38 8 60 32 39 18 53 53 60 53 20 24 26 16 20 22 2 57 9 38 52 57 12 40 30 50 12 28 36 11 6 31 12 53 50 37 38 44 40 43 32 20 25 3 22 10 24 9 45 43 35 18 5 50 30 49 26 28 59 63 34 45 16 59 49 9 41 44 48 |

Table 3.4 The inversion of $f$ polynomial.

Now Bob computes the product:

$$\mathbf{h} = p\mathbf{F}_q * \mathbf{g} \pmod{q}.$$

| The public key h polynomial (N = 167, p = 3, q = 64) |
|---|
| 34 18 16 15 33 40 37 17 24 50 20 12 28 20 19 9 41 42 27 20 40 44 14 53 63 46 12 4 26 43 35 35 59 41 57 62 24 1 23 17 11 13 0 46 62 62 28 20 55 44 16 30 10 27 36 53 26 48 6 0 26 20 47 7 9 36 54 49 46 55 51 34 47 63 13 16 41 7 44 34 15 34 58 30 13 59 20 39 26 36 36 52 31 29 20 58 17 11 31 19 53 36 5 27 18 8 34 |

Table 3.5 The public key h polynomial values.

Bob's public key is the polynomial h, and his private key is the pair of polynomials $f$ and $F_p$. The public key h is listed in Table 3.5.

Table 3.6 shows a speed comparison between NTRU and the Elliptic Curve Cryptosystem (ECC) PKCS in the key generation. We run our program on a Pentium III (500 MHz) with Windows NT4.0 operation system. It is noticed that the speed of NTRU is 6.25, 7.2 and 4.6 times faster than that of ECC in a speed of key generation at three different security levels as shown in Table 3.6.

| Key creation Comparison between NTRU and ECC | | | |
|---|---|---|---|
| **PKCS Name** | **Key Creation (ms)** | **NTRU vs. ECC** | **Security Level** |
| NTRU 167 | 4.0 | 6.25 | Moderate |
| ECC 112 | 25 | | |
| NTRU 263 | 9.0 | 7.2 | High |
| ECC 168 | 65 | | |
| NTRU 503 | 25 | 4.6 | Highest |
| ECC 196 | 115 | | |

Table 3.6 A speed comparison between NTRU and ECC.

### 3.4 The Lattice Based PKCS Encryption Scheme

Now, suppose Alice wants to send a message to Bob. Alice knows Bob's public key h, of course. She first puts her message in the form of a polynomial m whose coefficients are chosen modulo p, say between -p/2 and p/2. In Table 3.7, we list the values of our randomly chosen message m in a polynomial format.

| Original message m |
|---|
| -1 0 0 1 1 1 -1 1 1 1 1 0 -1 1 -1 0 1 1 1 1 1 -1 -1 -1 -1 0 0 0 1 -1 1 1 0 0 -1 0 -1 1 0 1 0 1 0 1 1 0 -1 0 1 1 0 0 1 0 -1 0 -1 0 1 1 0 -1 0 -1 -1 0 1 0 -1 -1 0 -1 1 0 -1 -1 0 0 -1 1 1 -1 1 0 1 1 -1 -1 -1 -1 0 -1 0 0 0 -1 1 -1 -1 -1 0 -1 1 1 0 0 0 1 |

Table 3.7 The original message values.

Next, she randomly chooses a polynomial r according to the following rule:

- **r** has $d_r$ of its coefficients equal to 1; it has $d_r$ of its coefficients equal to -1, and all of the rest of its coefficients are equal to 0. The private key r polynomial values with N equivalent to 167 are shown in Table 3.8.

| **The private key r polynomial (N = 167)** |
|---|
| 0 0 0 -1 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 1 0 1 0 0 0 0 0 0 -1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0 1 |

Table 3.8 The r polynomial values.

She uses the message **m**, her randomly chosen polynomial **r**, and Bob's public key **h** to compute the polynomial:

$$\mathbf{e} = \mathbf{r}*\mathbf{h} + \mathbf{m} \text{ (modulo q)}.$$

| **The encrypted message e (N = 167)** |
|---|
| 49 11 58 3 14 59 18 6 62 3 48 32 18 20 17 58 55 10 4 12 21 21 16 33 10 3 13 62 2 47 13 59 36 36 61 24 9 41 30 44 4 14 14 56 47 7 30 33 62 16 5 54 50 26 16 2 26 27 16 13 3 19 16 34 37 60 46 31 59 1 9 25 9 52 62 26 55 49 56 12 3 45 5 11 63 44 54 6 6 6 29 50 6 16 38 38 22 47 14 25 16 31 23 32 57 26 46 |

Table 3.9 The encrypted message values.

The polynomial **e** is the encrypted message that Alice sends to Bob. The encrypted message **e** is listed in the Table 3.9. A speed comparison for encryption process between ECC and NTRU is listed in Table 3.10. The experiments operate on a Pentium III 500 MHZ with the Windows NT4.0 operating system.

| **A speed comparison between NTRU and ECC in the encryption scheme:** | | | |
|---|---|---|---|
| **PKCS Name** | **Encryption (ms)** | **NTRU vs. ECC** | **Security Level** |
| NTRU 167 | 1.36 | 44.11 | Moderate |
| ECC 112 | 60 | | |
| NTRU 263 | 3.50 | 40.00 | High |
| ECC168 | 140 | | |
| NTRU 503 | 9.33 | 27.33 | Highest |
| ECC 196 | 255 | | |

Table 3.10 A comparison of encryption rate between ECC and NTRU.

From Table 3.10, it is noted that the encryption speed of the NTRU is about 44 and 40 times faster than that of the ECC in both moderate and high security level respectively. Meanwhile, speed of the NTRU is about 27 times faster than that of the ECC in the highest security level for encryption.

### 3.5 The Lattice Based PKCS Decryption Scheme

During the decryption process, if Bob receives Alice's encrypted message **e** and wants to decrypt it, he begins to use one of his private key polynomial $f$ to compute the polynomial **a** as follows:

$$\mathbf{a} = f * \mathbf{e} \text{ (modulo q)}.$$

| **The decrypted message a = $f$*e mod q (N = 167)** |
|---|
| -11 2 2 1 10 -3 -11 -7 -5 6 8 -6 14 -8 -5 -1 5 0 15 -7 -1 -8 0 13 3 8 -7 -2 -8 -6 -1 6 -1 -10 -4 4 0 6 0 -1 -20 2 -1 -2 16 -4 0 8 -1 -5 9 -10 4 -6 -10 3 -8 10 5 -8 8 -9 -9 17 3 -3 1 -5 -2 -6 2 20 -9 4 -9 -6 0 9 -3 -2 1 -9 -2 12 4 -4 -3 -2 -2 5 19 2 0 -1 0 6 13 1 6 -25 8 -10 -6 19 -8 10 -3 |

Table 3.11 The decrypted message after the first step of the decrypting process.

Table 3.11 shows the decrypted message **a** with N equivalent to 167. Because Bob is computing **a** modulo q, the coefficients of **a** has been set to lie between -q/2 and q/2. This is a very important step before Bob does the next step, which is to compute the polynomial **b** as follows:

$$\mathbf{b} = \mathbf{a} \text{ (mod p)}.$$

| The decrypted message a = *f**e mod q (N = 167) |
| --- |
| -1 0 0 1 1 1 -1 1 1 1 1 0 -1 1 -1 0 1 1 1 1 1 -1 -1 -1 -1 0 0 0 1 -1 1 1 0 0 -1 0 -1 1 0 1 0 1 0 1 1 0 -1 0 1 1 0 0 1 0 -1 0 -1 0 1 1 0 - 1 0 -1 -1 0 1 0 -1 -1 0 -1 1 0 -1 -1 0 0 -1 1 1 -1 1 1 0 1 1 -1 -1 -1 0 -1 0 0 0 -1 1 -1 -1 -1 0 -1 1 1 0 0 0 1 |

Table 3.12 The polynomial **b** value in the decrypting process.

After this step, Bob reduces each of the coefficients of **a** by modulo p. Now Bob uses his another private key, which is polynomial $\mathbf{F_p}$ to compute:

$$\mathbf{c = F_p * b} \text{ (modulo p).}$$

| The decrypted message c = $F_p$*b mod p (N = 167) |
| --- |
| -1 0 0 1 1 1 -1 1 1 1 1 0 -1 1 -1 0 1 1 1 1 1 -1 -1 -1 -1 0 0 0 1 -1 1 1 0 0 -1 0 -1 1 0 1 0 1 0 1 1 0 -1 0 1 1 0 0 1 0 -1 0 -1 0 1 1 0 - 1 0 -1 -1 0 1 0 -1 -1 0 -1 1 0 -1 -1 0 0 -1 1 1 -1 1 1 0 1 1 -1 -1 -1 0 -1 0 0 0 -1 1 -1 -1 -1 0 -1 1 1 0 0 0 1 |

Table 3.13 The decrypted message values.

The polynomial **c** should be Alice's original message **m**. The details will be given in the next section. By comparing the decrypted values in Table 3.13 with the original message **m** in Table 3.9, we notice that the recovered message **c** is exactly the same as message **m**.

Table 3.14 lists a speed comparison between NTRU and ECC PKCS during the decryption process. It is noted that the speed of the NTRU PKCS is approximately one order faster than that of the ECC PKCS at both moderate and high security levels. While at the highest security level, the decryption speed of the NTRU PKCS is 6 times faster than that of the ECC PKCS.

| **A speed comparison between NTRU and ECC in the decryption scheme** | | | |
|---|---|---|---|
| **PKCS Name** | **Decryption (ms)** | **NTRU vs. ECC** | **Security Level** |
| NTRU 167 | 2.75 | 9.46 | Moderate |
| ECC 112 | 26 | | |
| NTRU 263 | 6.9 | 9.7 | High |
| ECC168 | 67 | | |
| NTRU 503 | 18.90 | 6.3 | Highest |
| ECC196 | 119 | | |

Table 3.14 A comparison of decryption rate between ECC and NTRU.

### 3.6 The Lattice Based PKCS Verification Scheme

Alice's encrypted message **e** is: $\mathbf{e} = \mathbf{r}*\mathbf{h} + \mathbf{m}$ (modulo q). At first, Bob doesn't know the values of **r** and **m** at first. His first step is to compute $f*\mathbf{e}$ and reduce the coefficients by modulo q. Bob's public key **h** was actually created by multiplying $p\mathbf{F}_q*\mathbf{g}$ and reducing its coefficients modulo q. So, although Bob doesn't know **r** and **m**, after he computes $\mathbf{a} = f*\mathbf{e}$ (modulo q), he is, in fact, doing the following computation:

$\mathbf{a} = f*\mathbf{e}$ (modulo q)

$= f*(\mathbf{r}*\mathbf{h} + \mathbf{m})$ (modulo q) [since $\mathbf{e} = \mathbf{r}*\mathbf{h} + \mathbf{m}$ (modulo q)]

$= f*(\mathbf{r}*p\mathbf{F}_q*\mathbf{g} + \mathbf{m})$ (modulo q) [since $\mathbf{h} = p\mathbf{F}_q*\mathbf{g}$ (modulo q)]

$= p\mathbf{r}*\mathbf{g} + f*\mathbf{m}$ (modulo q) [since $f*\mathbf{F}_q = 1$ (modulo q)]

Now, when we check every coefficient of these parameters, we'll find the polynomials **r**, **g**, $f$, and **m** all have very small coefficients (either 0's, 1's and -1's ). This means that the coefficients of the products $\mathbf{r}*\mathbf{g}$ and $f*\mathbf{m}$ will also be quite small; at least in a comparison with q. Likewise, the prime p is also quite small as compared to q. This implies a proper selection of the parameters. The coefficients of the polynomial $p\mathbf{r}*\mathbf{g} + f*\mathbf{m}$ will lie between -q/2 and q/2.

In other words, when Bob computes **a** by first multiplying $f$ and **e** and then reducing the coefficients modulo q, the polynomial **a** he ends up with is exactly equal to the summation of polynomials $p\mathbf{r}*\mathbf{g}$ and $f*\mathbf{m}$. Thus, when Bob next reduces the

coefficients of **a** modulo p to form the polynomial **b**, he is really reducing the coefficients of pr*g + *f**m modulo p. So the value of **b** he ends up with is equal to:

$$\mathbf{b} = f*\mathbf{m} \text{ (modulo p)}.$$

However, at this stage, Bob still doesn't know the value of **m**, but he now knows the value of **b**. So his final step is to multiply **b** by $\mathbf{F}_p$ and use the fact that the value of $\mathbf{F}_p*f$ is equivalent to 1 (modulo p) to compute

$$\mathbf{c} = \mathbf{F}_p * \mathbf{b} = \mathbf{F}_p * f*\mathbf{m} = \mathbf{m} \text{ (modulo p)},$$

which allows him eventually to recover Alice's message **m**.

## 3.7 The Almost Inverse Algorithm

In the lattice based NTRU PKCS, the creation of public and private key pairs often requires finding the inverse of a polynomial $f(x)$ modulo a prime. The "Almost Inverse Algorithm" of Schroeppel, Orman, O'Malley and Spatscheck [SOOS95] gives an efficient way to compute the inverse of the polynomial a(X) in the ring (Z/pZ)[X]/(m(X)) provided that the greatest common devisor (gcd) of a(X) and m(X) is equivalent 1 and m(0) is equivalent to 1.

The basic idea of the Almost Inverse Algorithm is the same as the polynomial inversion as we described in Chapter 2. It starts from the following formula:

$$b(X) * f(X) + c(X) * g(X) \equiv 1 \bmod M$$

where M is a prime polynomial. The Almost Inverse Algorithm is initialized with:

- b(X) = 1;
- c(X) = 0;
- $f$(X) as the source polynomial;
- g(X) = M = prime polynomial;
- k is an integer, k=0; and
- b(X) and c (X) are polynomials.

To find an inverse of the source polynomial, the algorithm uses the following steps:

**The Inversion Algorithm in (Z/pZ)[X]/($X^N$-1)**

**Input:** $f$(X) = a(X), p (prime);

**Output:** b(X) $\equiv f(X)^{-1}$ (mod p);

---

1. b (X) = 1; c (X) = 0; $f$ (X) = a (X); g (X) = $X^N$-1; k = 0;
2. while ( $f_0$ = 0 ) {
3. $f$(X) = $f$(X)/X;
4. c (X) = c (X)*X;
5. k = k + 1;
6. }
7. if ( deg ( $f$(X) = 0 ) {
8. b(X) = $f_0^{-1}$ b (X) ( mod p )
9. return $X^{N-k}$ b(X) ( mod $X^N$-1)
10. }
11. if ( deg ( $f$(X) < deg ( g (X) ) ) {
12. tmp1 = $f$(X); $f$(X) = g (X); g(X) = tmp1;
13. tmp2 = b (X); b(X) = c (X); c(X) = tmp2;
14. }
15. u = $f_0$ * $g_0^{-1}$ ( mod p )
16. $f$(X) = $f$(X) – u*g (X) ( mod p )
17. b(X) = b (X) – u*c(X) ( mod p )
18. goto 2

---

In step 1 of this algorithm, we have the initial polynomials with $f$(X) = a (X) and g(X) = m(X) = $X^N$-1. If we represent these two polynomials as a vector, we have ($f$, g) =

(a, m). If this vector multiplies the following transformations matrices, we obtain the following results:

$$E=\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}\Rightarrow (f,g)E=(f,g)\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}=(g,f)$$

$$S=\begin{pmatrix}X^{-1} & 0\\ 0 & 1\end{pmatrix}\Rightarrow (f,g)S=(f,g)\begin{pmatrix}X^{-1} & 0\\ 0 & 1\end{pmatrix}=(X^{-1}f,g)$$

$$M_u=\begin{pmatrix}1 & 0\\ -u & 1\end{pmatrix}\Rightarrow (f,g)M_u=(f,g)\begin{pmatrix}1 & 0\\ -u & 1\end{pmatrix}=(f-ug,g)$$

These transformations are presented in the inversion algorithm.

Step 3 is a shift operation of $f$(X). Every item in the polynomial $f$(X) is divided by X, and the constant term in $f$(X) is 0. In step 12, when the degree of $f$(X) is less than that of of g(X), we exchange $f$(X) and g(X). This is done by a vector ($f$, g) multiplied by a matrix E. Then, step 16 is the ($f$, g) vector multiplied by the matrix $M_u$. Because in every loop, we guarantee that the degree of $f$ (X) is no less than the degree of g(X), we reduce the total degree of deg($f$) + deg(g) by at least 1 after $f$(X) is divided by x. Eventually, $f$(X) becomes a constant. The loop runs, at most, deg(a) + deg(m) times.

If we use three symbols $T_1$, $T_2$, $T_3$ to represent the matrix E, S and $M_u$, the Almost Inverse algorithm will produce a sequence such as follows:

$$(a, m)\, T_1T_2T_3...T_{r-1}\, T_r = (\lambda, *),$$

where $\lambda$ is a non-zero number modulo p. Because the matrix S has $X^{-1}$ as an entry, the coefficients of the product $T_1T_2T_3...T_{r-1}\ T_r$ are not polynomials. Let $k$ be the number of times matrix S appears in the product $T_1T_2T_3...T_{r-1}\ T_r$, then $X^kT_1T_2T_3...T_{r-1}T_r$ has coefficients that are polynomials as follows:

$$X^kT_1T_2T_3...T_r = \begin{pmatrix} a' & * \\ m' & * \end{pmatrix}$$

If we multiplying $(a, m)$ on the left side, the expression yields:

$$(aa'+mm', *) = (a, m)\begin{pmatrix} a' & * \\ m' & * \end{pmatrix} = (a, m)X^kT_1T_2T_3...T_r = X^k(\lambda, *),$$

Finally we have:

$$aa' \equiv \lambda X^k \pmod{m}.$$

If we could find some values for a' to make:

$$\lambda^{-1}X^{-k}aa' \equiv \lambda^{-1}X^{-k}\lambda X^k \pmod{m}$$

$$\text{or}\quad \lambda^{-1}X^{-k}aa' \equiv 1 \pmod{m}$$

which means:

$$\lambda^{-1}X^{-k}(a^{-1}a)a' \equiv 1 \cdot a^{-1} \pmod{m}$$

We could finally obtain:

$$a^{-1} \equiv \lambda^{-1} X^{-k} a' (\mathrm{mod}\, m)$$

The question is how our algorithm will find this value $a'$ ? If, when we are applying the transformations $T_1 T_2 T_3 \ldots T_r$ in the algorithm starting from (a, m), we are applying the same transformations starting from (b, c) = (1, 0), except that in place of $S = \begin{pmatrix} X^{-1} & 0 \\ 0 & 1 \end{pmatrix}$, we instead applies $XS = \begin{pmatrix} 1 & 0 \\ 0 & X \end{pmatrix}$. Because S has been used k times, at the end of the algorithm, the value of (b, c) is:

$$(b, c) = (1, 0) X^{k} T_1 T_2 T_3 \ldots T_r = (1,0) \begin{pmatrix} a' & * \\ m' & * \end{pmatrix} = (a', *).$$

In other words, at the end of the algorithm, b has a value satisfying

$$ab \equiv \lambda X^{k} (\mathrm{mod}\, m).$$

Because the value of λ is simply f0, the constant term of $f$ actually equals to $f$ at this stage of the algorithm. Also, $X^{-k}$ is equal to $X^{N-k}$ when we are working modulo $X^{N}-1$. Thus, we finally get:

$$a^{-1} \equiv \lambda^{-1} X^{-k} b \, (\mathrm{mod}\, m)$$

In the lattice based PKCS, the creation of public/private key pairs sometimes also need to find the inverse of a polynomial $f(X)$ modulo a prime power, in particular, a power of 2. When an inverse is determined modulo a prime p, the Newton iteration could be applied to compute rapidly the inverse modulo powers $p^r$, as described in the following:

Inversion Algorithm in $(\mathbb{Z}/p^r\mathbb{Z})[X]/X^N-1)$

Input: $a(X)$, $b(X) \equiv a(X)^{-1} \pmod{p}$, $p$ (a prime), $r$;

Output: $b(X) \equiv a(X)^{-1} \pmod{p^r}$

---

```
1.    q = p;
2.    while ( q < p^r){
3.          q = q^2
4.          b(X) = b(X) ( 2 – a (X)b(X)) ( mod q )
5.    }
```

---

The above algorithm converges double exponentially; it requires only about $\log_2(r)$ steps to find the inverse of $a(X)$ modulo $p^r$ based on the known inverse modulo p.

# Chapter 4

# The Lattice Based Digital Signature Scheme

## 4.1 General Introduction of the Lattice Based Digital Signature Scheme

The concept of digital signature was introduced in 1976 by Diffie and Hellman [DH76B]. Digital signature appears to have been independently discovered by Merkle [MER78] [MER79]. The RSA signature scheme, discovered by Rivest, Shamir, and Adleman [RIVE78], was the first practical signature scheme based on the public key techniques. The Digital Signature Algorithm (DSA) is attributed to Kravitz [KRA93], and was proposed as a Federal Information Processing Standard (FIPS) in August of 1991 by the U.S. National Institute of Science and Technology. It became the Digital Signature Standard (DSS) in May 1994, as specified in FIPS 186 [FIPS94].

The lattice based polynomial authentication and signature scheme is based on the hard mathematical problem of finding a binary polynomial $f(X)$ that takes on prescribed values $f(S)$ mod q are at a given collection of a set:

$$S = \{ \alpha_0, \alpha_1, \alpha_2, \ldots \alpha_{N-1} \}$$

This is an equivalent problem to solve the closest vector problem in a certain lattice. As the Signer publishes the set of values $f(\alpha_i)$ as his/her public key, the verifier can use the set S and signer's public key to verify the signature of the message.

The security level of RSA was given by factoring integers using the Number Field Sieve (NFS) algorithm to break the RSA scheme, and the security level of the ECC was given by finding elliptic curve discrete logarithms when applying the Pollard Rho Method (PRM) to break the ECC scheme [HPS00A]. The best way known to attack the lattice based polynomial digital signature scheme is to use the LLL lattice reduction algorithm discovered by Lenstra, Lenstra and Lovasz [LLL82]. Similar to the RSA and ECC, the security level of the lattice based polynomial digital signature scheme was given by the estimated breaking time when the improved LLL algorithm is applied to it.

| PKCS | Security Level | Estimated Breaking Time |
|---|---|---|
| **RSA** | 512 bits | $10^5$ MIPS-years |
| **NTRU** | N = 167 | $10^6$ MIPS-years |
| **RSA** | 1024 bits | $10^{12}$ MIPS-years |
| **NTRU** | N = 263 | $10^{14}$ MIPS-years |
| **RSA** | 2048 bits | $10^{21}$ MIPS-years |
| **NTRU** | N = 503 | $10^{35}$ MIPS-years |

Table 4.1 A comparison of security level between RSA and NTRU PKCS.

In Table 4.1, the comparison is based on a time estimation that is approximately how long it would take a single machine to break a single key by using the best-known attack method for the RSA and the NTRU cryptography system.

Table 4.2 demonstrates the approximate equivalent security level of NTRU as compared with RSA and ECC.

| | NTRU | RSA | ECC |
|---|---|---|---|
| Moderate Security | N = 167 | RSA 512 | ECC 110 |
| High Security | N = 263 | RSA 1024 | ECC 168 |
| Highest Security | N = 503 | RSA 2048 | ECC 216 |

Table 4.2 The comparison of security level among NTRU, RSA, and ECC PKCS.

## 4.2. Parameters of the NTRU Digital Signature Scheme

A lattice based polynomial digital signature scheme depends on the choice of a prime number q and another number N. The number N represents the highest power of the polynomial $f(\alpha_i)$. The relationship between N and q is N = q–1. Besides the parameters q and N, the lattice based polynomial signature scheme requires a number of other parameters. The parameters and their corresponding definitions in the scheme are listed as follows.

| | |
|---|---|
| **N** | The polynomials in the truncated polynomial ring have degree N-1 |
| **q** | The polynomials have coefficients modulo q (the number q must be prime, and the number N must equal N=q-1) |
| $\mathbf{d_f}$ | The private key $f = (f_1, f_2)$ consists of two polynomials, each of which has $d_f$ coefficients equal to each of 1 and –1. |
| $\mathbf{d_g}$ | The commitment $\mathbf{g} = (\mathbf{g}_1, \mathbf{g}_2)$ consists of two polynomials, each of which has $d_g$ coefficients equal to each of 1 and –1. |
| $\mathbf{d_c}$ | The challenge $\mathbf{c} = (\mathbf{c}_{11},\mathbf{c}_{12},\mathbf{c}_{21},\mathbf{c}_{22})$ consists of four polynomials, each of which has $d_c$ coefficients equal to each of 1 and -1 |
| **S** | A set of values $\{\alpha_1,\alpha_2,...,\alpha_{N/2}\}$ modulo q. If $\alpha$ is in S, then $\alpha^{-1}$ (mod q) is also in S. |
| $\mathbf{B_h}$ | A bound for the norm of the polynomial h |

The foundation of the latticed based polynomial authentication and signature scheme is the ring **R**. It consists of all truncated polynomials of degree N-1 having coefficients modulo q, i.e.,

$$f(X) = a_0 + a_1X + a_2X^2 + a_3X^3 + \ldots + a_{N-2}X^{N-2} + a_{N-1}X^{N-1} \pmod q.$$

The definitions and properties of the polynomial ring have been described in Chapter 2. For more details, please see [KRO98], [LEV90], [ORE76], and [ROS93].

There are two important operations in this truncated polynomial: Addition and multiplication. Polynomials are added in the usual way. They are also multiplied more-or-less as usual, except that $X^N$ is replaced by 1, $X^{N+1}$ is replaced by X, $X^{N+2}$ is replaced by $X^2$, and so on.

For example, if q = 11, N = 10, and we have two polynomials as follows:

$$f_1(X) = 1 + 3X - 4X^2 + X^3 - 2X^4 + X^7 + 5X^9 \pmod{11},$$

$$f_2(X) = 2X - X^3 + 6X^4 - X^5 - 4X^7 + 11X^8 \pmod{11}.$$

The addition and multiplication operations yield the following results:

$$f_1(X)+f_2(X) = 1+5X-4X^2+4X^4-X^5-3X^7+11X^8+5X^9 \pmod{11}.$$

$$= 1+5X-4X^2+4X^4-X^5-3X^7+5X^9 \pmod{11}.$$

$$f_1(X)*f_2(X) = 2X-X^3+6X^4-X^5-4X^7+11X^8+6X^2-3X^4+18X^5-3X^6-12X^8+33X^9$$
$$-8X^3+4X^5-24X^6+4X^7+16X^9-44X^0+2X^4-X^6+6X^7-X^8-4X^0+11X^1$$
$$-4X^5+2X^7-12X^8+2X^9+8X^1-22X^2+2X^8-X^0+6X^1-X^2-4X^4+11X^5$$
$$+10X^0-5X^2+30X^3-5X^4-20X^6+55X^7 \pmod{11}$$
$$= (10-1-4-44) + (2+11+8+6)X + (6-22-1-5)X^2 + (-1-8+30)X^3$$
$$+ (6-3+2-4-5)X^4 + (-1+18+4-4+11)X^5 + (-3-24-1-20)X^6$$
$$+ (-4+4+6+2+55)X^7 + (11-12-1-12+2)X^8 + (33+16+2)X^9 \pmod{11}$$
$$= -39+27X-22X^2+21X^3-4X^4+28X^5-48X^6+63X^7-12X^8+51X^9 \pmod{11}$$
$$= -6+5X+10X^3-4X^4+6X^5-4X^6+8X^7-X^8+7X^9 \pmod{11}$$

### 4.3 Hard Problem of the Lattice Based Digital Signature

Identifying hard computational problems that are amenable for cryptographic use is a very important task. Although hard computational problems seem to be all around us, only very few of those problems were found to be useful for cryptography. In fact, after two decades of research in cryptography, the vast majority of the public-key

cryptosystems still depend on either the hardness of integer factorization or the hardness of extracting discrete logarithms. Moreover, it has often been the case that algorithmic advance in one of these problems was then applied to the other one as well.

The hard problem underlying the security of the lattice based digital signature scheme is to find a binary polynomial $f(x)$ that takes on prescribed values $f(\alpha)$ mod q at a given collection of numbers $\alpha = \{ \alpha_1, \alpha_2, \alpha_3, \alpha_4, \ldots, \alpha_n \}$. Table 4.3 lists four practical public key cryptosystems that are based on different mathematically hard problems, and the best-known algorithms for solving the underlying mathematical problems and breaking systems.

| System | Underlying Problem | Best Known Attack |
|---|---|---|
| **RSA** | Factor large numbers | Number Field Sieve |
| **El Gamal** | Discrete logarithm | Index Calculus |
| **ECC** | Elliptic curve discrete log | Pollard Rho |
| **NTRU** | Find a short vector in lattice | LLL lattice reduction |

Table 4.3 The hard problems and known attacks for most popular PKCS.

### 4.4 Secure Hash Algorithm

A message digest algorithm transforms an arbitrary length data into a "message digest" of fixed size. This process is known as "hashing". "Hashing" is a one-way transformation, in which the resulting message digest (or hash) cannot be used to reveal

the original data. A message digest algorithm will always produce the same message digest for the same input pieces of data.

A lot of algorithms have been written on the design of one-way hash functions. Table 4.4 lists some characteristics of the most popular hash algorithms. For more details, please see [MMO85], [JMM85], [JUE87], [NY89], [MER90], [DAM90], [BD92] and [PRE94].

| Name | Hash Length | Encryption Speed(KB/Sec) |
|---|---|---|
| Davies-Meyer | 64 | 9 |
| GOST Hash | 256 | 11 |
| HAVAL(3 Passes) | Variable | 168 |
| HAVAL(3 Passes) | Variable | 118 |
| HAVAL(5 Passes) | Variable | 95 |
| MD2 | 128 | 23 |
| MD4 | 128 | 236 |
| MD5 | 128 | 174 |
| RIPE-MD | 128 | 182 |
| SHA | 160 | 75 |

Table 4.4 The most popular hash algorithms and their characteristics.

Secure Hash Algorithm, SHA-1, is applied for computing a condensed representation of a message or a data file. When a message of any length less than $2^{64}$ bits is an input, the SHA-1 produces a 160-bit output called a "message digest." The message digest can then be used as input to the Digital Signature Algorithm that generates or verifies the signature for the message. Signing the message digest rather than the message itself often improves the efficiency of the process because the message digest's size is usually much smaller than that of the message. In addition, the hash algorithm used by the verifier of a digital signature must be the same as that was used by the creator of the digital signature.

The SHA-1 is called secure because it is computationally infeasible to find a message that corresponds to a given message digest, or to find two different messages that produce the same message digest. Any change to a message in transit will, with very high probability, result in a different message digest. Correspondingly, the signature will fail to verify. The SHA-1 is based on principles similar to those used by Professor Ronald L. Rivest of the Massachusetts Institute of Technology when designing the "MD4" message digest algorithm, and is closely modeled after that algorithm.

The Secure Hash Algorithm is required for use combined with the Digital Signature Algorithm as specified in the Digital Signature Standard, and whenever a secure hash algorithm is required for federal applications. The message digest, produced by the SHA-1, is used during generation of a digital signature for the message. The SHA-1 is also used to compute a message digest for the received version of the message during the process of verifying the signature.

The SHA-1 is designed to have the following properties: it is computationally infeasible to find a message that corresponds to a given message digest, or to find two different messages that produce the same message digest.

Figure 4.1 and Figure 4.2 demonstrate the digital signature generation and verification process by applying the SHA-1 message digest algorithm, respectively.

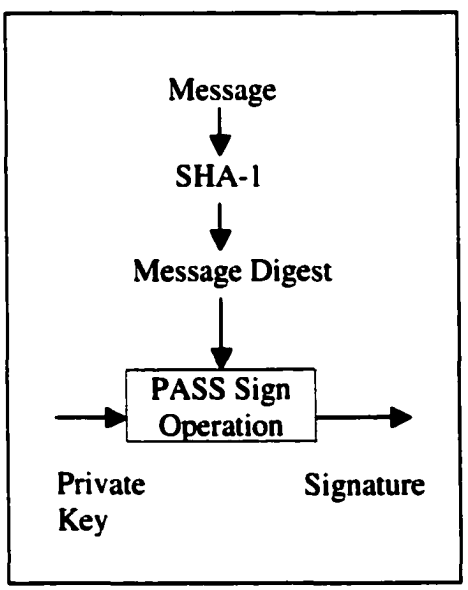



Figure 4.1 The digital signature generation process.

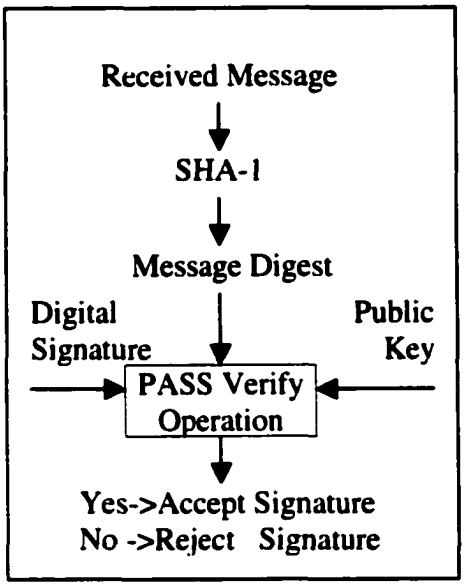



Figure 4.2 The digital signature verification process.

## 4.5 Key Creation Operation

### 4.5.1 Selection of the Public Key Set S

Let a set of values, $S = \{\alpha_1, \alpha_2, \ldots \alpha_{N/2}\} \pmod q$, is one of the public keys of the signer. the signer will publish all of the values of set S to the public. For every value of set S, if $\alpha_i \pmod q$ is in set S, then $\alpha_i^{-1} \pmod q$ must be also in set S.

The follows are the Java style pseudo-code written for finding values of set S:

**Input:** positive integer i.

**Output:** $v = i^{-1} \pmod q$

```
Vector S = new Vector( ); // Create an empty Vector object.

for ( int i = N/4; i < N*3/4; i++){
        int v ← constInv(i, q);
        if ( (v < q) && ( v > 0 ) ){
                S.add ( new Integer( i ) );
                S.add ( new Integer( v ) );
        }
}

int constInv(const int a, const int q)
{
        int sw, m, n, p0, p1, p2, r, qq;
        n ← a; // a is the const in polynomial->a0;
        m ← q; // q is the prime in polynomial ring;
        while(n<0) n ← q + a;
                sw ← 1;   p0 ← 1;
                p1 ← m/n; r  ← m % n;
                qq ← p1;

        while( r != 0 )
                sw ← -sw;  m ← n;   n ← r;
                qq ← m/n; r  ← m%n;
                p2 ← qq*p1 + p0;
                p0 ← p1; p1 ← p2;
        return sw * p0;

}
```

The method constInv(i, q) is designed for finding $i^{-1}$ (mod q) for integer i by using the Euclidean algorithm. We have described it in details in Chapter 2.

Table 4.5 shows an example of set S when q equivalent to 769. From the first row of data, we see that:

$$2 * 385 = 770 \equiv 1 \bmod 769.$$

$$51 * 573 = 29223 \equiv 1 \bmod 769.$$

$$100 * 546 = 54600 \equiv 1 \bmod 769.$$

$$145 * 297 = 43065 \equiv 1 \bmod 769.$$

Once the signer obtains all the values of set S, he/she will use all these values in the signing process. These values will be published to the public. Then the Verifier will use these values in the verification process later on.

### 4.5.2 The Private Key f Selection Process

The private key of the lattice based digital signature scheme includes two random generated polynomials, $f_1$ and $f_2$. From the parameters explanation, we know both $f_1$ and $f_2$ have $d_f$ of their coefficients equal to 1 and –1. All the rest of their coefficients are equal to 0. In our implementation, we use a vector as our data structure for both $f_1$ and $f_2$ because a vector is more flexible than a one-dimensional array.

| i | $i^{-1}$ | i | $i^{-1}$ | i | $i^{-1}$ | i | $i^{-1}$ |
|---|---|---|---|---|---|---|---|
| 385 | 2 | 573 | 51 | 546 | 100 | 297 | 145 |
| 513 | 3 | 281 | 52 | 434 | 101 | 265 | 148 |
| 355 | 13 | 413 | 54 | 525 | 104 | 364 | 150 |
| 564 | 15 | 206 | 56 | 520 | 105 | 494 | 151 |
| 470 | 18 | 358 | 58 | 370 | 106 | 350 | 156 |
| 423 | 20 | 378 | 59 | 245 | 113 | 529 | 157 |
| 293 | 21 | 353 | 61 | 398 | 114 | 503 | 159 |
| 535 | 23 | 354 | 63 | 455 | 120 | 406 | 161 |
| 562 | 26 | 268 | 66 | 286 | 121 | 394 | 162 |
| 412 | 28 | 264 | 67 | 561 | 122 | 558 | 164 |
| 282 | 30 | 502 | 72 | 283 | 125 | 261 | 165 |
| 536 | 33 | 530 | 74 | 420 | 130 | 227 | 166 |
| 475 | 34 | 298 | 80 | 317 | 131 | 571 | 167 |
| 235 | 36 | 347 | 82 | 451 | 133 | 325 | 168 |
| 291 | 37 | 454 | 83 | 319 | 135 | 312 | 175 |
| 425 | 38 | 495 | 87 | 311 | 136 | 485 | 176 |
| 531 | 42 | 201 | 88 | 426 | 139 | 300 | 182 |
| 465 | 43 | 553 | 89 | 390 | 140 | 374 | 183 |
| 402 | 44 | 326 | 92 | 417 | 142 | 212 | 185 |
| 565 | 49 | 215 | 93 | 242 | 143 | 492 | 186 |
| 323 | 50 | 435 | 99 | 251 | 144 | 366 | 187 |

Table 4.5 The values of set S with q equivalent to 769.

| The private key $f_1$ | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | -1 | 1 | 1 | 0 | 0 | -1 | 0 | 1 | 1 | 0 | 0 | -1 | -1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | -1 | 0 | 1 | -1 | 0 | 1 | 0 | -1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | -1 | 1 | 1 | -1 | 0 | 1 | 1 | 0 | 1 | 1 | -1 | 0 | 0 | 0 | -1 | 0 | -1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | -1 | 1 | 0 | -1 | 0 | 1 | -1 | -1 | 0 | 1 | -1 | 0 | 0 | -1 |
| 0 | -1 | 1 | -1 | -1 | 1 | 0 | -1 | 0 | -1 | -1 | 1 | 1 | 1 | 1 | -1 | 1 | -1 | 1 | -1 |
| 1 | 0 | 1 | -1 | -1 | 0 | 1 | -1 | -1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 | -1 | -1 | 1 | 1 |
| -1 | 1 | -1 | 1 | 1 | -1 | 1 | 0 | 0 | -1 | 1 | -1 | 0 | 1 | 1 | -1 | -1 | -1 | 0 | 0 |
| 1 | -1 | 1 | 1 | -1 | 0 | -1 | -1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | -1 | 0 | 0 | -1 | 0 |
| -1 | -1 | 0 | -1 | 0 | 0 | 1 | 1 | -1 | 1 | 1 | 1 | -1 | 0 | -1 | 1 | 0 | 1 | 0 | -1 |
| 0 | 0 | -1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | -1 | 0 | 0 | 1 | -1 | 0 | 0 | 1 | 1 |
| 1 | -1 | -1 | -1 | 0 | 0 | -1 | 0 | -1 | -1 | -1 | 0 | -1 | 1 | -1 | 1 | -1 | 0 | 1 | 0 |
| 0 | 0 | -1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | -1 |
| -1 | 0 | 1 | 1 | 0 | 0 | -1 | -1 | -1 | 0 | 1 | 0 | 1 | 0 | 1 | -1 | 1 | 0 | -1 | 0 |
| 0 | -1 | -1 | 1 | 0 | -1 | 1 | -1 | -1 | 1 | 1 | -1 | -1 | 1 | -1 | 0 | 1 | 0 | 0 | -1 |
| 0 | 1 | 0 | 1 | 1 | -1 | 0 | 1 | 1 | 1 | 1 | 1 | -1 | -1 | -1 | 0 | -1 | -1 | -1 | -1 |
| 1 | 1 | 1 | 0 | 0 | -1 | 0 | 1 | -1 | 0 | 0 | 1 | 0 | 1 | -1 | 0 | 1 | -1 | 0 | 0 |
| 0 | 0 | -1 | 1 | 1 | 0 | -1 | 0 | -1 | -1 | -1 | -1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | -1 |
| 1 | -1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | -1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | -1 | 0 | -1 | -1 | 0 | -1 | 1 | 1 | 1 | 0 | -1 | 0 | 1 | -1 | -1 | 1 | 1 |
| 0 | 1 | 0 | -1 | 0 | 0 | -1 | -1 | 1 | 1 | 0 | 0 | 0 | -1 | -1 | 1 | -1 | -1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | -1 | -1 | 0 | 0 | 1 | 1 | -1 | 1 | -1 | 0 | 1 | 0 |
| -1 | 0 | -1 | -1 | 1 | -1 | -1 | 0 | -1 | 0 | 1 | -1 | 0 | 1 | 1 | -1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | -1 | 0 | 1 | 0 | 0 | 0 | 0 | -1 | -1 | 0 | 1 | 0 | -1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | -1 | -1 | -1 | 0 | -1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | -1 | -1 | 1 |
| 1 | -1 | 1 | -1 | 0 | -1 | 0 | 0 | -1 | -1 | -1 | -1 | 0 | 1 | -1 | 0 | -1 | -1 | 0 | -1 |
| -1 | 1 | 1 | 1 | 1 | -1 | 0 | 1 | 0 | -1 | -1 | 1 | 0 | -1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | -1 | -1 | 0 | 1 | 0 | -1 | 0 | 1 | -1 | 1 | -1 | -1 | 1 | 0 | 0 | -1 |
| 1 | 0 | 0 | 0 | 1 | -1 | 0 | -1 | 1 | 0 | 0 | 1 | 1 | -1 | -1 | -1 | 1 | -1 | 1 | 1 |
| 0 | -1 | 1 | 1 | -1 | -1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | -1 | -1 | 0 | 1 | -1 |
| -1 | 0 | 1 | 0 | 1 | -1 | -1 | -1 | 1 | -1 | -1 | 1 | 0 | 1 | 1 | 1 | -1 | -1 | -1 | -1 |
| 1 | -1 | 1 | 0 | 1 | 1 | 0 | -1 | 0 | -1 | -1 | -1 | 0 | 0 | -1 | 0 | -1 | -1 | 1 | -1 |
| 0 | -1 | 0 | -1 | -1 | 1 | -1 | 1 | -1 | 0 | 1 | 0 | -1 | 1 | -1 | 0 | 0 | -1 | 0 | 0 |
| 1 | 0 | 0 | -1 | 1 | 1 | -1 | -1 | 0 | 1 | 0 | 1 | 1 | -1 | 1 | -1 | 0 | 0 | -1 | -1 |
| 1 | 1 | 0 | 0 | 1 | 0 | -1 | 0 | -1 | 0 | 1 | 1 | -1 | 1 | 1 | 0 | -1 | 0 | -1 | 0 |
| -1 | -1 | -1 | -1 | 1 | 1 | -1 | -1 | 1 | 1 | -1 | 0 | 1 | -1 | 0 | 1 | -1 | 1 | 1 | -1 |
| 0 | 0 | 1 | 1 | -1 | -1 | 1 | -1 | -1 | 1 | 0 | 1 | 1 | -1 | -1 | 0 | 1 | -1 | 1 | -1 |
| 1 | -1 | 0 | 1 | 1 | -1 | -1 | 0 | 0 | 0 | 0 | 0 | 1 | -1 | 1 | 0 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 1 | 1 | -1 | 1 | 0 | 1 | -1 | 0 | 1 | -1 | 0 | 1 | 1 | 0 | 0 | -1 |
| 1 | -1 | -1 | 0 | 1 | -1 | -1 | -1 | | | | | | | | | | | | |

Table 4.6 The values of private key $f_1$ when q is equivalent to 769.

| The private key $f_2$ | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | -1 | -1 | 0 | -1 | 1 | 1 | 0 | 0 | 1 | -1 | 1 | 1 | 0 | 0 | 0 | -1 | 1 | 0 | -1 |
| 0 | 0 | 0 | -1 | 1 | 1 | -1 | 0 | 0 | 1 | 0 | -1 | 1 | 0 | -1 | -1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 | 1 | 1 | -1 | 0 | -1 | -1 | 1 | 1 | 0 |
| 1 | -1 | 0 | 1 | -1 | -1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | -1 | 1 | -1 | 1 | -1 | 0 | 1 |
| -1 | 1 | 0 | 0 | 0 | -1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | -1 | -1 | 1 | 1 | 1 |
| -1 | -1 | 0 | -1 | -1 | 1 | -1 | 0 | 1 | 1 | -1 | 1 | 0 | 1 | 1 | 1 | 1 | -1 | 0 | 0 |
| -1 | -1 | -1 | -1 | 0 | -1 | 1 | 0 | -1 | 0 | 0 | -1 | -1 | 1 | 0 | 0 | -1 | -1 | 0 | 1 |
| -1 | 0 | 0 | 0 | 1 | -1 | -1 | -1 | 0 | -1 | -1 | 0 | 1 | -1 | -1 | 1 | 1 | 1 | -1 | 0 |
| -1 | 1 | -1 | -1 | 1 | 1 | 0 | -1 | -1 | 1 | -1 | 1 | 0 | 0 | 0 | 1 | -1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | -1 | 1 | 1 | -1 | -1 | 0 | 1 | 1 | 1 | 0 | 0 | -1 | 0 | 0 |
| 1 | 0 | -1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | -1 | -1 | 1 | -1 | -1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | -1 | -1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | -1 | 1 | 0 | -1 |
| 1 | -1 | 0 | 0 | 1 | 1 | 0 | -1 | 1 | 1 | -1 | -1 | 0 | 1 | 1 | 1 | 0 | -1 | 1 | 1 |
| 1 | 1 | 1 | 0 | -1 | 1 | 1 | 0 | 0 | -1 | -1 | 0 | -1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | -1 | 0 | 1 | -1 | 0 | -1 | 1 | -1 | -1 | -1 | -1 | -1 | 0 | 1 | 0 | -1 | -1 | 1 | 1 |
| -1 | 1 | 0 | 1 | -1 | 1 | -1 | 1 | 0 | -1 | 1 | -1 | 0 | 0 | 0 | -1 | 1 | -1 | 0 | -1 |
| -1 | 0 | -1 | 0 | 0 | 0 | 1 | 0 | 0 | -1 | 1 | 0 | 1 | -1 | 1 | -1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | -1 | 0 | 0 | 1 | 0 | -1 | 1 | 0 | 1 | -1 | -1 | 0 | 0 | 1 | 0 | 1 |
| -1 | -1 | 0 | -1 | 0 | 1 | -1 | -1 | 1 | 0 | -1 | 0 | 0 | 0 | -1 | 1 | 0 | 0 | 1 | -1 |
| 0 | 0 | 1 | 1 | -1 | -1 | 0 | 1 | 1 | 1 | 1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 0 | 0 | -1 | -1 | 1 | 0 | -1 | 1 | -1 | 0 | -1 | -1 | -1 | 1 | -1 | -1 | 0 |
| -1 | 1 | 0 | 1 | -1 | 0 | 0 | 1 | 0 | 0 | -1 | 0 | -1 | 0 | 0 | 0 | -1 | 0 | 1 | -1 |
| 0 | 1 | -1 | -1 | -1 | 0 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | -1 | 1 | 0 | -1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | -1 |
| 0 | -1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | -1 | -1 | 0 | -1 | 1 | -1 | 0 | 1 | 0 | 1 |
| -1 | -1 | 0 | 1 | -1 | -1 | 0 | 0 | 0 | -1 | 1 | 1 | 0 | 1 | 1 | 1 | -1 | -1 | 0 | -1 |
| 1 | -1 | -1 | 0 | 0 | 1 | -1 | 1 | 0 | 0 | 1 | 0 | 0 | -1 | -1 | 0 | 1 | -1 | -1 | 1 |
| 1 | -1 | -1 | 1 | 1 | -1 | 0 | -1 | -1 | -1 | 1 | -1 | 0 | 0 | 1 | 0 | 0 | -1 | 0 | -1 |
| 0 | 1 | -1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | -1 | 0 | 0 | 1 | 1 |
| -1 | 1 | 1 | 1 | -1 | 0 | -1 | 1 | -1 | 0 | 1 | -1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| -1 | 1 | 1 | 0 | 0 | -1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | -1 | 1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 0 | 0 | 0 | -1 | 1 | -1 | 1 | -1 | 1 | -1 | 1 | 0 | 0 | -1 | 1 | -1 |
| 1 | 1 | 0 | -1 | -1 | 0 | -1 | 1 | -1 | 1 | 1 | 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | -1 | 1 | 0 | 0 | -1 | 1 | 1 | -1 |
| 1 | 1 | 1 | 1 | -1 | 1 | -1 | 0 | -1 | -1 | 0 | 0 | 0 | -1 | 0 | -1 | -1 | 0 | -1 | 0 |
| 0 | 0 | 0 | -1 | 0 | 0 | 1 | -1 | -1 | -1 | -1 | 0 | 1 | 1 | 1 | 0 | -1 | -1 | 1 | 1 |
| -1 | -1 | 0 | -1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | -1 | -1 | 0 | -1 | 1 | 1 | 0 | 1 | 1 |
| 0 | -1 | 0 | 0 | 0 | -1 | 1 | -1 | -1 | -1 | 0 | 0 | 0 | 0 | -1 | -1 | -1 | -1 | -1 | 0 |
| 1 | 0 | -1 | 1 | -1 | 1 | 1 | 1 | | | | | | | | | | | | |

Table 4.7 The values of private key $f_2$ when q is equivalent to 769.

Table 4.6 and Table 4.7 give the $f_1$ and $f_2$ private key when q is equal to 769. Table 4.8 gives the value of the public key $f$(S).

| The key $f$(S) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 116 | 474 | 704 | 540 | 758 | 733 | 542 | 147 | 733 | 616 |
| 697 | 15 | 122 | 295 | 361 | 300 | 491 | 660 | 411 | 658 |
| 235 | 553 | 515 | 469 | 666 | 723 | 300 | 112 | 635 | 392 |
| 385 | 586 | 0 | 339 | 738 | 335 | 180 | 115 | 616 | 343 |
| 505 | 228 | 411 | 612 | 137 | 730 | 369 | 740 | 277 | 551 |
| 553 | 235 | 228 | 282 | 20 | 672 | 657 | 14 | 245 | 190 |
| 249 | 511 | 624 | 665 | 413 | 543 | 509 | 536 | 219 | 745 |
| 266 | 240 | 300 | 361 | 317 | 12 | 738 | 647 | 59 | 644 |
| 212 | 249 | 761 | 139 | 25 | 86 | 432 | 265 | 467 | 94 |
| 603 | 609 | 242 | 57 | 69 | 615 | 535 | 330 | 556 | 668 |
| 299 | 539 | 514 | 688 | 209 | 417 | 286 | 442 | 472 | 322 |
| 341 | 723 | 686 | 473 | 747 | 95 | 757 | 526 | 25 | 384 |
| 373 | 18 | 143 | 548 | 514 | 297 | 560 | 494 | 495 | 669 |
| 539 | 299 | 726 | 228 | 564 | 180 | 461 | 691 | 301 | 228 |
| 576 | 64 | 181 | 578 | 328 | 78 | 360 | 47 | 497 | 115 |
| 409 | 742 | 615 | 69 | 223 | 185 | 576 | 229 | 659 | 442 |

Table 4.8 The values of public key $f$(S)

### 4.5.3 The Private Key g Selection Process

The signer randomly selects two polynomial $\mathbf{g}_1$ and $\mathbf{g}_2$, which have $d_g$ of their coefficients equal to each of 1 and -1, and all of their other coefficients equal to 0.

Table 4.9 and Table 4.10 show the private key $g_1$ and $g_2$ when q equals to 769.

| The private key $g_i$ | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | -1 | 0 | 0 | 1 | -1 | -1 | 1 |
| 0 | -1 | 1 | 0 | -1 | 1 | 1 | 0 | 1 | 0 | -1 | -1 | 0 | -1 | 1 | 1 | -1 | 0 | 1 | 1 |
| 0 | 0 | -1 | -1 | -1 | 0 | -1 | 0 | 0 | 1 | -1 | -1 | 1 | 1 | -1 | 1 | 0 | 0 | 0 | 0 |
| 0 | -1 | -1 | 1 | 1 | 0 | -1 | 0 | -1 | 1 | 0 | 1 | -1 | 1 | -1 | -1 | 1 | -1 | -1 | -1 |
| 1 | -1 | -1 | 1 | 1 | 0 | 0 | -1 | 0 | -1 | -1 | -1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| -1 | 1 | 0 | -1 | -1 | 0 | 0 | -1 | -1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | -1 |
| -1 | 0 | -1 | 1 | 0 | 1 | 0 | 1 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | -1 | 0 | 1 | 0 | -1 |
| 0 | 1 | 1 | 0 | -1 | 1 | 1 | 0 | -1 | -1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | -1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | -1 | 0 | 0 | 0 | 1 | -1 | 1 | -1 | -1 | 0 | 1 | 0 | 0 | 1 |
| -1 | 1 | 1 | -1 | 1 | 1 | -1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | -1 | 1 | 0 | 0 |
| 1 | 1 | -1 | -1 | 0 | -1 | 1 | 0 | 1 | 0 | 1 | -1 | -1 | -1 | -1 | 1 | 0 | -1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | -1 | 0 | 0 | 1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 | 1 | -1 | 0 | 0 |
| -1 | -1 | -1 | 0 | 1 | 1 | -1 | -1 | -1 | -1 | 0 | 1 | 1 | 0 | -1 | -1 | 0 | -1 | 0 | 1 |
| 1 | 0 | 1 | -1 | 1 | 0 | 0 | -1 | -1 | -1 | -1 | 1 | 0 | 0 | 1 | -1 | 1 | 1 | -1 | 0 |
| -1 | 0 | -1 | -1 | 0 | 0 | 1 | -1 | -1 | 0 | -1 | -1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| -1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | -1 | 0 | 0 | -1 | 1 | -1 | -1 | -1 | -1 | 0 | 0 | -1 |
| 1 | -1 | 0 | 1 | -1 | 1 | 1 | -1 | 0 | 1 | -1 | 0 | -1 | -1 | 1 | 0 | 1 | -1 | 0 | 1 |
| 0 | 1 | 0 | -1 | -1 | 1 | 1 | -1 | 0 | 0 | 0 | 1 | 1 | -1 | -1 | 1 | 1 | 0 | -1 | 1 |
| 0 | 1 | -1 | 1 | 0 | 0 | 0 | -1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | -1 |
| -1 | -1 | -1 | 0 | -1 | 1 | 1 | 1 | -1 | 0 | 0 | 0 | -1 | -1 | 0 | 0 | -1 | 0 | -1 | 1 |
| 0 | -1 | 1 | 0 | 0 | 0 | -1 | 0 | -1 | 1 | -1 | -1 | 1 | -1 | -1 | 0 | 1 | -1 | 0 | -1 |
| -1 | -1 | 0 | 1 | -1 | 0 | -1 | -1 | -1 | 0 | 0 | 1 | 0 | 0 | -1 | -1 | 0 | 0 | -1 | 1 |
| -1 | 1 | 0 | 0 | 0 | 1 | 1 | -1 | 0 | 1 | 1 | -1 | 1 | 1 | 1 | 1 | -1 | -1 | 1 | 0 |
| 0 | -1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | -1 | -1 | -1 | 0 | 0 | -1 | 0 | -1 | 0 |
| 0 | -1 | -1 | -1 | -1 | -1 | -1 | 0 | 1 | -1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | -1 | 0 | 0 | 0 | -1 | -1 | 1 | 0 | -1 | 0 | 1 | -1 | -1 | 1 | 0 | 1 |
| -1 | 1 | -1 | 1 | 0 | 1 | 0 | -1 | -1 | 1 | 1 | -1 | 0 | 1 | 1 | 0 | -1 | 1 | 1 | -1 |
| 1 | 0 | -1 | 1 | 0 | -1 | 1 | 0 | 1 | 1 | 1 | -1 | 0 | 1 | 0 | 1 | -1 | 0 | 1 | 0 |
| 0 | -1 | 0 | -1 | 0 | -1 | -1 | 0 | -1 | 0 | -1 | 0 | 0 | -1 | -1 | 1 | 1 | 0 | 1 | 1 |
| -1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 | -1 | 1 | 1 | -1 | 0 | 1 | 1 | -1 | -1 | 1 | 0 |
| 0 | -1 | 0 | 0 | 0 | 1 | -1 | 0 | -1 | 0 | -1 | 0 | 0 | 1 | 1 | -1 | 1 | 0 | 1 | -1 |
| 1 | 1 | 1 | 0 | -1 | -1 | -1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | -1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | -1 | 0 | 0 | 1 | 1 | 0 | -1 | 1 | 0 | 0 | -1 | 1 | 1 |
| 1 | -1 | 0 | 0 | 1 | 1 | 0 | -1 | -1 | 0 | -1 | -1 | 0 | 1 | 1 | 0 | -1 | 1 | -1 | 1 |
| 1 | 1 | 1 | 1 | 1 | -1 | -1 | 1 | -1 | 1 | 1 | -1 | 1 | 1 | 0 | 0 | -1 | 1 | -1 | 1 |
| 1 | -1 | 1 | 0 | 1 | -1 | 1 | 1 | 1 | -1 | 1 | 1 | 1 | 1 | 0 | 1 | -1 | 1 | -1 | 1 |
| 0 | 0 | 1 | 1 | -1 | -1 | 1 | 1 | 0 | 0 | -1 | -1 | -1 | -1 | -1 | 1 | 1 | 0 | -1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | -1 | -1 | 0 | 0 | -1 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | 0 | -1 |
| 0 | 0 | 1 | -1 | -1 | 0 | 1 | 0 | | | | | | | | | | | | |

Table 4.9 The values of private key $g_i$ with q equivalent to 769.

| The private key $g_2$ | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | -1 | 0 | 1 | 0 | -1 | -1 | 0 | -1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | -1 | 1 | 0 | -1 |
| -1 | -1 | -1 | -1 | -1 | 0 | -1 | -1 | 1 | 1 | -1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| -1 | -1 | -1 | -1 | 1 | -1 | 0 | -1 | 0 | 0 | 0 | 1 | 0 | -1 | 0 | 0 | 1 | 0 | 0 | -1 |
| 0 | 1 | 0 | -1 | -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | 0 | 0 | 1 | -1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | -1 | -1 | 1 | -1 | -1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | -1 | 0 | -1 | -1 | 0 | -1 | -1 | 1 | -1 | 0 | 1 | 0 | 0 | 0 | 0 | -1 |
| -1 | 1 | -1 | 1 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 1 | 1 | -1 | 1 | 0 | -1 | -1 | -1 | 1 |
| 0 | 0 | -1 | 0 | 1 | 0 | 0 | -1 | 0 | 1 | 0 | 0 | 1 | -1 | 0 | 1 | -1 | -1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | -1 | -1 | 1 | -1 | -1 | 1 | -1 | 1 | 0 | 1 | -1 | 1 | 0 |
| 0 | -1 | 1 | 1 | -1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | -1 | 1 | -1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | -1 | -1 | -1 | 0 | -1 | 0 | 0 | -1 | 1 | 1 | -1 | 0 | -1 | 0 | -1 |
| -1 | 1 | 0 | -1 | -1 | -1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | -1 | 1 | 1 | 0 | 0 | 1 |
| -1 | 0 | -1 | 1 | 1 | 0 | -1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | -1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | -1 | -1 | 1 | 0 | 0 |
| -1 | 0 | 0 | -1 | 0 | 0 | -1 | 1 | -1 | 1 | 0 | 0 | -1 | -1 | 1 | -1 | -1 | 1 | 1 | 1 |
| 1 | 1 | 1 | -1 | 0 | 1 | -1 | 1 | 1 | 0 | -1 | 0 | -1 | 0 | 1 | 1 | 0 | 1 | 1 | -1 |
| 1 | -1 | 1 | -1 | -1 | 1 | -1 | -1 | 0 | -1 | 0 | 1 | -1 | 0 | -1 | 1 | -1 | -1 | 1 | 1 |
| 1 | 0 | 0 | 1 | -1 | 0 | -1 | 0 | -1 | 0 | 1 | -1 | 0 | 0 | 0 | -1 | 1 | -1 | -1 | 1 |
| 0 | 0 | 1 | 0 | -1 | -1 | -1 | 0 | 1 | 0 | 0 | 0 | 0 | -1 | 0 | -1 | -1 | 0 | -1 | 0 |
| 0 | -1 | -1 | 1 | 0 | -1 | 0 | 0 | 1 | 1 | -1 | -1 | 0 | 0 | 0 | 0 | 0 | 1 | -1 | 0 |
| -1 | 0 | 1 | -1 | -1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | -1 | 1 | 1 | 0 | 1 | 0 | -1 | -1 |
| -1 | 0 | 0 | 1 | -1 | 0 | 1 | 1 | 0 | -1 | 1 | 1 | 1 | 1 | 1 | -1 | 0 | 0 | -1 | -1 |
| 0 | 1 | 1 | 0 | 1 | -1 | 0 | -1 | 1 | 1 | -1 | 0 | -1 | 0 | 1 | 1 | 1 | -1 | -1 | 1 |
| 0 | 1 | 0 | -1 | 0 | 0 | 1 | 1 | -1 | 1 | -1 | 1 | 1 | -1 | 1 | -1 | 1 | 0 | 1 | 1 |
| -1 | 0 | 1 | 0 | 1 | -1 | -1 | -1 | 1 | 1 | -1 | 1 | 0 | -1 | -1 | -1 | -1 | 1 | 0 | 1 |
| -1 | 0 | -1 | -1 | 1 | 1 | 0 | 1 | -1 | 0 | -1 | 1 | 1 | 0 | 0 | 0 | -1 | -1 | 1 | 1 |
| 1 | 0 | -1 | 1 | 0 | -1 | -1 | -1 | 0 | -1 | -1 | 1 | 0 | 1 | 0 | -1 | -1 | -1 | 1 | 0 |
| 1 | -1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | -1 | 1 | 1 | -1 | -1 | 0 | 1 | 1 | -1 |
| -1 | 1 | -1 | 0 | -1 | 0 | 0 | -1 | 1 | 0 | 1 | -1 | 0 | 0 | 1 | 1 | 0 | -1 | 1 | -1 |
| -1 | -1 | -1 | -1 | -1 | 1 | -1 | -1 | 0 | 1 | -1 | 1 | 1 | -1 | 0 | 0 | -1 | 1 | 1 | 0 |
| 0 | 1 | -1 | -1 | 0 | 1 | -1 | 0 | 0 | 1 | 1 | 1 | 1 | -1 | 0 | -1 | -1 | 0 | -1 | -1 |
| 1 | 1 | 0 | -1 | -1 | -1 | 1 | 0 | 1 | -1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | -1 | -1 | 1 |
| -1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | -1 | -1 | 1 | 1 | -1 | 0 | -1 | 1 | -1 | 0 | -1 | -1 |
| -1 | 0 | 0 | -1 | -1 | 0 | 1 | 0 | 0 | 1 | -1 | 1 | 0 | 0 | 0 | 1 | 0 | -1 | 0 | 0 |
| 1 | 1 | -1 | -1 | -1 | 1 | 1 | -1 | -1 | 0 | 1 | 1 | 1 | 0 | 1 | -1 | 0 | -1 | 1 | 0 |
| -1 | 0 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | 0 | -1 | -1 | -1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | -1 | -1 | 1 | -1 | -1 | -1 | 0 | 0 | -1 | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 1 |
| 1 | 1 | 0 | 0 | 1 | -1 | 0 | 0 | -1 | 1 | 0 | 0 | 0 | 0 | -1 | 0 | 1 | 0 | -1 | -1 |
| 1 | 0 | 0 | 1 | -1 | 0 | 0 | 1 | | | | | | | | | | | | |

Table 4.10 The values of private key $g_2$ when q equivalent to 769.

The signer also computes the values of $g_1$ and $g_2$ for each of the elements of the set S. That is, $g(S) = (g_1(S), g_2(S)) = \{g_1(\alpha_1), g_1(\alpha_2), ..., g_1(\alpha_{N/2}), g_2(\alpha_1), g_2(\alpha_2), ..., g_2(\alpha_{N/2})\}$ (mod q). The polynomials $g_1$ and $g_2$ will be kept as the signer's secret key. Then the signer sends the set of values g(S) to the verifier. We call that the set g(S) is the signer's commitment.

An example of g(S) is shown in Table 4.11 as follows:

| The values of key g(S) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 20 | 564 | 538 | 215 | 80 | 74 | 4 | 567 | 319 | 383 |
| 566 | 311 | 477 | 241 | 187 | 31 | 503 | 422 | 99 | 178 |
| 274 | 85 | 219 | 37 | 559 | 475 | 457 | 379 | 280 | 657 |
| 571 | 687 | 400 | 600 | 667 | 709 | 241 | 548 | 112 | 761 |
| 443 | 594 | 586 | 333 | 147 | 288 | 66 | 638 | 295 | 78 |
| 85 | 274 | 615 | 457 | 425 | 603 | 25 | 508 | 698 | 496 |
| 752 | 300 | 30 | 428 | 235 | 177 | 273 | 147 | 423 | 102 |
| 596 | 108 | 31 | 187 | 490 | 145 | 433 | 561 | 87 | 749 |
| 251 | 463 | 698 | 630 | 613 | 688 | 472 | 429 | 76 | 82 |
| 153 | 98 | 512 | 142 | 279 | 351 | 708 | 454 | 27 | 103 |
| 59 | 696 | 647 | 744 | 88 | 514 | 579 | 712 | 328 | 252 |
| 580 | 392 | 594 | 703 | 684 | 257 | 18 | 283 | 509 | 125 |
| 71 | 710 | 565 | 629 | 48 | 469 | 2 | 141 | 127 | 200 |
| 696 | 59 | 82 | 684 | 504 | 671 | 219 | 182 | 53 | 304 |
| 126 | 642 | 257 | 643 | 249 | 61 | 494 | 764 | 228 | 757 |
| 392 | 4 | 351 | 279 | 732 | 11 | 533 | 252 | 51 | 209 |

Table 4.11 The values of key g(S)

In order to use PASS to create a digital signature, we assume the hash function SHA-1 has been applied and a 160-bit string output has been produced. Table 4.12 gives the 160-bit hash values for different but very similar text messages created by the SHA-1 algorithm. We notice that the hash values are completely different. We will use these

values in the formatting function, we described in the previous section, to get four polynomial $C_{11}$, $C_{12}$, $C_{21}$, $C_{22}$.

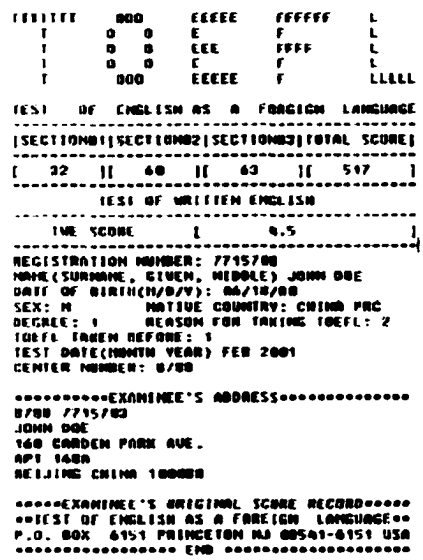
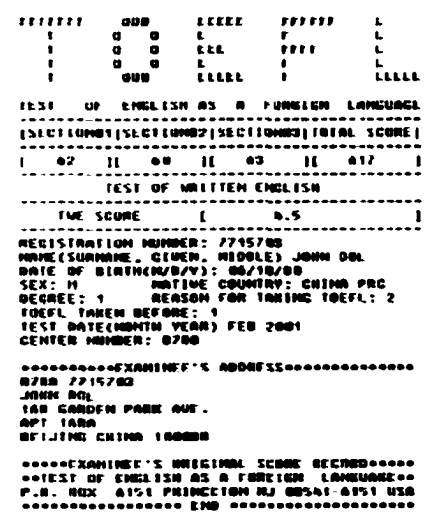
| The message | | The hash value of the message from SHA-1 |
|---|---|---|
|  The original text | ⇨ | 01010011 00101000<br>10000010 11000010<br>00110010 00011011<br>11011010 11001110<br>11101000 01011100<br>10101001 00101011<br>10010110 11111100<br>10110111 01100011<br>10101111 00110101<br>11111000 01100111 |
|  The modified text | ⇨ | 01101101 11101010<br>00111101 10011010<br>11101000 11110000<br>11010111 11100100<br>01011011 10110010<br>00110010 11100101<br>10001100 01011001<br>11101101 00111001<br>01100101 10001010<br>11001000 01101011 |

Table 4.12 Two similar texts have totally different hashing values created by the SHA-1 algorithm.

### 4.6 Applying the Format Function

After applying the SHA-1 algorithm, a 160-bit string of hash values was generated. Following this, we divided these 20 bytes into four groups. Each group has five bytes. We use $B_i$ to represent one byte of it. Then we apply the formatting function. four polynomials $C_{11}$, $C_{12}$, $C_{21}$, $C_{22}$ will be generated as follows:

$$C_{11} = \sum_{i=0}^{(lengthB)/4-1} | Bi | \pmod{q} \qquad C_{12} = \sum_{i=lengthB/4}^{(lengthB*2)/4-1} | Bi | \pmod{q}$$

$$C_{21} = \sum_{i=lennthB*2/4}^{(lenB*3)/4-1} | Bi | \pmod{q} \qquad C_{22} = \sum_{i=length*3/4}^{lengthB-1} | Bi | \pmod{q}$$

The follows list a pseudo-code for creating four polynomial $C_{11}$, $C_{12}$, $C_{21}$, and $C_{22}$ from the message digest.

**The formatting function algorithm:**

**Input:** 160-bit hashing value

**Output:** 4 polynomials $C_{11}$, $C_{12}$, $C_{21}$, and $C_{22}$

---

```
for (i = 0; i < 20; i++ ){
        B0 ← abs ( (int) hash[i] );
        B1 ← abs ( (int) hash[i+1] );
        B2 ← abs ( (int) hash[i+2] );
        B3 ← abs ( (int) hash[i+3] );
        B4 ← abs ( (int) hash[i+4] );
}
if CpolyBits = 2 then {
        highPower ← (B1 + B2 + B3 + B4) % q;
        lowPower ← B0 % q;
        if highPower = lowPower
                Right Shift lowPower by 1;
        if highPower < lowPower then {
```

```
                temp←highPower;
                highPower←lowPower;
                lowPower←temp;
        }
}

if CPolyBits = 4 then {
        FirstPower    ← B0% q;
        SecondPower ← (B1+B2) % q;
        if FirstPower = SecondPower
                Right Shift SecondPower by 1;

        ThirdPower   ← (B1+B2+B3) % q;
        if ThirdPower = FirstPower
                Right Shift SecondPower by 1;
        else if ThirdPower = SecondPower
                Right Shift SecondPower by 1;

        FourthPower  = (B1+B2+B3+B4) % q;
        if FourthPower  = FirstPower
                 Right Shift SecondPower by 1;
        else if FourthPower  = SecondPower
                 Right Shift SecondPower by 1;
        else if FourthPower  = ThirdPower
                 Right Shift SecondPower by 1;
        }
}
```

---

- The signer constructs the four polynomials $C_{ij} = (C_{11}, C_{12}, C_{21}, C_{22})$ by computing $C_{ij} = F(H(g(S)\|M))$. Here M is the message, H is the hash function and F is the formatting function, respectively.

Table 4.13 lists 160 bit hash values and the corresponding $C_{11}$, $C_{12}$, $C_{21}$, $C_{22}$ polynomials.

| Hash Value | $C_{ij}$ |
|---|---|
| 11101000<br>01111110<br>10011011<br>11101000<br>01010000 | 0000000000000000000000000-10000000000<br>00000000000000000000000000000000000000<br>00000000000000000000000000000000000000<br>00000000000000000000000000000000000000<br>00000000000000000000000000000000000000<br>00000000000000000000000000000000000000<br>00000000000000000000000000000000000000<br>00000000000000000000000000000000000000<br>00000000000000000000000000000000000000<br>00000000000000000000000000000001 |
| 11010111<br>00011110<br>00000111<br>01011011<br>10110110 | 00000000000000000000000000000000000000<br>0000000-100000000000000000000000000000<br>00000000000000000000000000000000000000<br>00000000000000000000000000000000000000<br>00000000000000000000000000000000000000<br>00000000000000000000000000000001 |
| 11010110<br>11110010<br>10000101<br>11011001<br>01001011 | 00000000000000000000000000000000000000<br>00000000-100000000000000000000000000000<br>00000000000000000000000000000000000000<br>00000000000000000000000000000000000000<br>00000000000000000000000000000000000000<br>00000000000000000000000000000000000000<br>00000000000000000000000000000000000000<br>0000000000000001 |
| 10000110<br>00001011<br>11101111<br>00110011<br>10001100 | 00000000000000000000000000000000000000<br>00000000000000000000000000000000000000<br>00000000000000000000000000000000000000<br>00000000000000000000-100000000000000<br>00000000000000000000000000000000000000<br>00000000000000000000000000000001 |

Table 4.13 160 bit hash values and their corresponding $C_{ij}$ polynomials.

The signer computes the response **h** to the polynomial **g** and the polynomial **c**, using his/her private key $f$, in the usual way:

$$\mathbf{h} = \mathbf{c}_{11} * f_1 * \mathbf{g}_1 + \mathbf{c}_{12} * f_1 * \mathbf{g}_2 + \mathbf{c}_{21} * f_2 * \mathbf{g}_1 + \mathbf{c}_{22} * f_2 * \mathbf{g}_2 .$$

The signer's signed message is then the message M followed by the signature ($f$(S), **g**(S),**h**).

Table 4.14 shows the signature value h.

| **The signature value h** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| -82 | -9 | -97 | -104 | -7 | 17 | -49 | 57 | 54 | 27 |
| -2 | -17 | 55 | -31 | -38 | 74 | 19 | 17 | 43 | 87 |
| -90 | 52 | -9 | -30 | -3 | -24 | -20 | -22 | 31 | -9 |
| 5 | -54 | 78 | -34 | 77 | -16 | -48 | 69 | -32 | 57 |
| -9 | 37 | 29 | 12 | 5 | -27 | 23 | -49 | 11 | -20 |
| 33 | 24 | -8 | -28 | -45 | 121 | -46 | 3 | -56 | 31 |
| -51 | -21 | 4 | -12 | 6 | -25 | 38 | -22 | -69 | -60 |
| 33 | -13 | -53 | 54 | 28 | 33 | 18 | 41 | -21 | 60 |
| -35 | -63 | -35 | -18 | 15 | -60 | 38 | 2 | 105 | 6 |
| 114 | 12 | 38 | 23 | -15 | -44 | -52 | -71 | -70 | -43 |
| 10 | -34 | 54 | 70 | -31 | 0 | -5 | 34 | -49 | 58 |
| -11 | -23 | -104 | 15 | -73 | -30 | 1 | 102 | 69 | 22 |
| -53 | -58 | -43 | 46 | -6 | 14 | -52 | -51 | -31 | 24 |
| -55 | 79 | -51 | -57 | -44 | 35 | -51 | 30 | 8 | 28 |
| 69 | 59 | 78 | -29 | -87 | -6 | -18 | -51 | -45 | -9 |
| -16 | 9 | 80 | -9 | -51 | -18 | 100 | -27 | -33 | -7 |
| -58 | 31 | -55 | -37 | -20 | 27 | -25 | -36 | -14 | 5 |
| 125 | 12 | -25 | -50 | -32 | 31 | 11 | 17 | 39 | -13 |
| 40 | -2 | -43 | -41 | -36 | -5 | -9 | -3 | 98 | -40 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 34 | -6 | -50 | 50 | 48 | -30 | 12 | 3 | -59 | 63 |
| 7 | -27 | 34 | 19 | 14 | 67 | 23 | -7 | 4 | -23 |
| 44 | -58 | -16 | 7 | -7 | 37 | -35 | -20 | 4 | 14 |
| -54 | 51 | -59 | -6 | -16 | -75 | 10 | 22 | -85 | 21 |
| -25 | -62 | 60 | -73 | 61 | -12 | 20 | 96 | -17 | 36 |
| 0 | 46 | 10 | 0 | 42 | 3 | 4 | -68 | -12 | -64 |
| 1 | -32 | -41 | -50 | 66 | -26 | 51 | 92 | 19 | 65 |
| 30 | 3 | 2 | -50 | -30 | -8 | 12 | 63 | -36 | -52 |
| 60 | 12 | -1 | -44 | -64 | 46 | -26 | 54 | -21 | 26 |
| -7 | 129 | 11 | 3 | 10 | -34 | -24 | 10 | -2 | 32 |
| -48 | 66 | -42 | -18 | 2 | 34 | -2 | 62 | 28 | -8 |
| 51 | 43 | -2 | 48 | 34 | -37 | 18 | 42 | -7 | 29 |
| -74 | 15 | -75 | 0 | -108 | 78 | -2 | 6 | 44 | -63 |
| 20 | 90 | -71 | -16 | 0 | 5 | -27 | 7 | 5 | 20 |
| -11 | 10 | -34 | -21 | 5 | -61 | 88 | 15 | 72 | 17 |
| -39 | 45 | 37 | 38 | -25 | 41 | 93 | -15 | 34 | -79 |
| -26 | -50 | 39 | -78 | 17 | -19 | 12 | -49 | 0 | -21 |
| -56 | 91 | -71 | -53 | 29 | 34 | 32 | 146 | 61 | 60 |
| -72 | 11 | -7 | -65 | -80 | 24 | -53 | -49 | -37 | 28 |
| -63 | 32 | -13 | 117 | 51 | -74 | -48 | -3 | -66 | 11 |
| -31 | -4 | 0 | -35 | -46 | 5 | 24 | -12 | -57 | -100 |
| 3 | 3 | -16 | -9 | 47 | -27 | 65 | 43 | 66 | -105 |
| 5 | 41 | -25 | 55 | -79 | 18 | 30 | -26 | 4 | 3 |
| 25 | -84 | -53 | 51 | 16 | 79 | 41 | 27 | 46 | 43 |
| 60 | -44 | -97 | 31 | 40 | 57 | -22 | -67 | 5 | 94 |
| -3 | -29 | -9 | -26 | 57 | 27 | -47 | 77 | 27 | 70 |
| -27 | -50 | -43 | 58 | 31 | 17 | -35 | -39 | -6 | 12 |
| 42 | -1 | -106 | 78 | 41 | 94 | -23 | -35 | -5 | -11 |
| 3 | 35 | 58 | -134 | -7 | -5 | -154 | -44 | -48 | -102 |
| 32 | -33 | -41 | -1 | 55 | 67 | -131 | 35 | 68 | -52 |
| 1 | -4 | -34 | 39 | -20 | -6 | 4 | -26 | -78 | -7 |
| -5 | -5 | -26 | -49 | 44 | -13 | 9 | -18 | 83 | -56 |
| 65 | -6 | -63 | -1 | -20 | -33 | 48 | 51 | 51 | 43 |
| 14 | -45 | -41 | 64 | 44 | 11 | -110 | 20 | 4 | 16 |
| 30 | 32 | 4 | 14 | -11 | 49 | -89 | 49 | -38 | 124 |
| 37 | -4 | -23 | -25 | 4 | -66 | 88 | -18 | 3 | 40 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| -45 | -7 | 85 | -3 | 86 | -21 | 58 | -20 | 83 | -39 |
| -82 | -72 | 41 | -63 | -60 | 2 | -43 | -22 | 66 | 23 |
| -40 | 48 | 70 | -13 | 1 | -26 | -55 | -5 | -83 | -104 |
| -108 | 5 | 127 | 50 | 34 | 85 | 44 | 64 | -152 | 3 |
| -65 | -58 | -89 | 0 | -52 | -57 | -26 | 41 | -13 | -21 |
| -64 | 32 | 9 | -52 | -8 | 15 | -43 | 25 | -114 | 59 |
| 4 | 76 | -30 | -2 | 106 | 53 | 71 | 51 | -56 | 15 |
| 66 | 15 | -62 | -47 | -18 | -43 | 2 | 47 | 7 | -9 |
| -13 | -15 | 52 | 20 | 48 | 4 | -26 | 36 | 15 | -55 |
| 19 | -8 | -39 | 43 | 17 | -78 | -96 | 34 | 13 | -60 |
| 40 | 62 | 54 | -56 | 70 | 85 | 94 | 27 | -4 | 30 |
| 90 | 69 | -95 | -2 | -41 | 0 | 9 | 12 | 75 | 70 |
| 15 | 61 | -3 | 45 | -22 | 9 | 23 | 48 | -19 | 43 |
| -35 | -19 | -60 | 84 | 30 | -3 | 55 | 16 | 3 | -14 |
| -65 | 11 | -106 | -77 | 80 | 85 | 39 | 44 | 53 | 3 |
| -37 | 112 | -14 | -18 | 24 | 36 | 157 | -42 | 10 | -6 |
| 17 | -21 | -39 | -59 | 52 | 4 | 63 | -60 | 112 | -13 |
| 17 | 81 | -23 | -86 | -79 | -8 | -26 | -114 | 18 | 4 |
| 27 | -5 | -64 | -46 | 5 | 111 | 10 | -4 | -55 | -34 |
| -22 | -12 | 24 | -11 | -9 | 3 | -61 | -118 | -23 | 9 |
| -33 | -14 | -59 | 30 | -86 | 95 | 29 | 89 | -6 | -49 |
| 39 | -57 | 44 | -49 | -106 | -11 | 24 | -2 | | |

Table 4.14 The signature value of h polynomial.

## 4.7 Verifying the Signature Process

To verify the signed message **M** and its signature, the Verifier computes the **C** polynomials from $f(S)$, $g(S)$ and the signed message **M** by using the publicly available hash function **H** (i.e., SHA-1 in our example) and formatting function **F**. He/she then

uses the signer's public key $f(\mathbf{S})$ to verify whether the signature value **h** was generated by the signer.

We describe the details of the verification process as follows.

1.) Verifier computes four polynomials $\mathbf{C_{ij}}$ from **g(S)** and **M** by using the publicly available hash function **H** and the formatting function **F**. Then use the polynomial **C** to computer $\mathbf{C_{ij}(S)}$. The following is the experimental result:

| **The 160-bit hash value from SHA-1** | |
|---|---|
| hash[00] = 0101 0011 | hash[10] = 1010 1001 |
| hash[01] = 0010 1000 | hash[11] = 0010 1011 |
| hash[02] = 1000 0010 | hash[12] = 1001 0110 |
| hash[03] = 1100 0010 | hash[13] = 1111 1100 |
| hash[04] = 0011 0010 | hash[14] = 1011 0111 |
| hash[05] = 0001 1011 | hash[15] = 0110 0011 |
| hash[06] = 1101 1010 | hash[16] = 1010 1111 |
| hash[07] = 1100 1110 | hash[17] = 0011 0101 |
| hash[08] = 1110 1000 | hash[18] = 1111 1000 |
| hash[09] = 0101 1100 | hash[19] = 0110 0111 |

Table 4.15 The hash values after applying SHA-1 during the verification process.

After we obtain the 160-bit hash values from the SHA-1 algorithm, we apply the formatting function **F** as described before. The four polynomials $\mathbf{C_{ij}}$ are created.

Table 4.16 lists the values of $\mathbf{C_{ij}}$ during the verification process.

| Hash Value | $C_{ij}$ |
|---|---|
| 11101000<br>01111110<br>10011011<br>11101000<br>01010000 | 0000000000000000000000000-1000000000<br>0000000000000000000000000000000000000<br>0000000000000000000000000000000000000<br>0000000000000000000000000000000000000<br>0000000000000000000000000000000000000<br>0000000000000000000000000000000000000<br>0000000000000000000000000000000000000<br>0000000000000000000000000000000000000<br>0000000000000000000000000000000000000<br>00000000000000001 |
| 11010111<br>00011110<br>00000111<br>01011011<br>10110110 | 0000000000000000000000000000000000000<br>000000-1000000000000000000000000000<br>0000000000000000000000000000000000000<br>0000000000000000000000000000000000000<br>0000000000000000000000000000000000000<br>00000000000000000000000000000001 |
| 11010110<br>11110010<br>10000101<br>11011001<br>01001011 | 0000000000000000000000000000000000000<br>0000000-1000000000000000000000000000<br>0000000000000000000000000000000000000<br>0000000000000000000000000000000000000<br>0000000000000000000000000000000000000<br>0000000000000000000000000000000000000<br>0000000000000000000000000000000000000<br>00000001 |
| 10000110<br>00001011<br>11101111<br>00110011<br>10001100 | 0000000000000000000000000000000000000<br>0000000000000000000000000000000000000<br>0000000000000000000000000000000000000<br>0000000000000000-1000000000000000<br>0000000000000000000000000000000000000<br>00000000000000000000000000000001 |

Table 4.16 $C_{ij}$ polynomials during the verification process.

2.) The verifier then uses the signer's public key **h** and the set S to calculate the value **h(S)**. Table 4.17 shows the experimental results of **h(S)** values during the verification process.

| The values of h(S) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 348 | 679 | 732 | 351 | 181 | 608 | 618 | 182 | 286 | 181 |
| 372 | 711 | 232 | 623 | 405 | 434 | 100 | 170 | 118 | 304 |
| 649 | 629 | 12 | 109 | 703 | 614 | 541 | 202 | 169 | 274 |
| 606 | 15 | 52 | 505 | 240 | 333 | 51 | 602 | 281 | 157 |
| 276 | 482 | 78 | 440 | 198 | 170 | 403 | 605 | 329 | 542 |
| 629 | 649 | 330 | 251 | 616 | 126 | 658 | 713 | 572 | 463 |
| 200 | 353 | 761 | 740 | 91 | 73 | 11 | 611 | 349 | 25 |
| 205 | 365 | 434 | 405 | 330 | 401 | 586 | 230 | 530 | 627 |

Table 4.17 The h(S) polynomial values during the verification process.

3.) The verifier then uses the signer's public key ***f*(S)** and **g(S)** to verify whether signature values **h** was generated by the signer, i.e., by someone with knowledge of the private key ***f***. This verifies the signer's signature on the message **M**. There are two conditions for signed message to pass the verification process.

$$(A)\ h_0^2 + h_1^2 + h_2^2 + \ldots + h_{N-1}^2 < (B_h * q)^2.$$

$$(B)\ h(s_i) = c_{11}(s_i)*f_1(s_i)*g_1(s_i) + c_{12}(s_i)*f_1(s_i)*g_2(s_i) + c_{21}(s_i)*f_2(s_i)*g_1(s_i) + c_{22}(s_i)*f_2(s_i)*g_2(s_i) \pmod q.$$

4.) The signer's signature value **h** will pass test (A) if the polynomials $f$, **g** and **c** all have very small coefficients. Thus, the polynomial:

$$\mathbf{h} = \mathbf{c}_{11} * f_1 * \mathbf{g}_1 + \mathbf{c}_{12} * f_1 * \mathbf{g}_2 + \mathbf{c}_{21} * f_2 * \mathbf{g}_1 + \mathbf{c}_{22} * f_2 * \mathbf{g}_2$$

will also have small coefficients. Therefore, with an appropriate choice of parameters (such as the norm bound $B_h$), **h** will certainly pass test (A). On the other hand, because **h** is equal to $\mathbf{c}_{11} * f_1 * \mathbf{g}_1 + \mathbf{c}_{12} * f_1 * \mathbf{g}_2 + \mathbf{c}_{21} * f_2 * \mathbf{g}_1 + \mathbf{c}_{22} * f_2 * \mathbf{g}_2$, the equality

$$\begin{aligned} \mathbf{h}(s_i) = {} & \mathbf{c}_{11}(s_i) * f_1(s_i) * \mathbf{g}_1(s_i) + \mathbf{c}_{12}(s_i) * f_1(s_i) * \mathbf{g}_2(s_i) \\ & + \mathbf{c}_{21}(s_i) * f_2(s_i) * \mathbf{g}_1(s_i) + \mathbf{c}_{22}(s_i) * f_2(s_i) * \mathbf{g}_2(s_i) \pmod q \end{aligned}$$

will be true for all values of a modulo q. In particular, it will be true for all a's in S, so the response will also pass test (B).

Table 4.18 lists the c*$f$*g values during the verification process.

| **The values of $\Sigma$ (c*$f$*g)** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 348 | 679 | 732 | 351 | 181 | 608 | 618 | 182 | 286 | 181 |
| 372 | 711 | 232 | 623 | 405 | 434 | 100 | 170 | 118 | 304 |
| 649 | 629 | 12 | 109 | 703 | 614 | 541 | 202 | 169 | 274 |
| 606 | 15 | 52 | 505 | 240 | 333 | 51 | 602 | 281 | 157 |
| 276 | 482 | 78 | 440 | 198 | 170 | 403 | 605 | 329 | 542 |
| 629 | 649 | 330 | 251 | 616 | 126 | 658 | 713 | 572 | 463 |
| 200 | 353 | 761 | 740 | 91 | 73 | 11 | 611 | 349 | 25 |
| 205 | 365 | 434 | 405 | 330 | 401 | 586 | 230 | 530 | 627 |

**Table 4.18 The c*$f$*g values during the verification process.**

### 4.8 The Lattice Based Signature Scheme for Wireless Security

More and more people are connecting to the Internet via wireless hand-held devices, requiring efficient security and technologies that provides a secure, hassle-free user experience. In the near future, you won't think twice about using your cell phone or PDAs to access att.com, J. P. Morgan Chase-Manhattan Bank or ConEdsion from beach or your backyard barbecue table. You'll exchange messages, pay monthly bills, and check market reports while commuting on train or waiting for a flight. Mobile commerce will be available all wireless, all the time.

M-commerce (the "M" stands for mobile) is growing by leaps and bounds as more companies and consumers discover the ease and efficiency of the wireless Internet access. Already being called the "new, new economy," the outlook for m-commerce is huge, with millions of wireless devices fast becoming the next gateway for billions of dollars in electronic transactions. The variety of mobile devices, both in use and in development, is astounding. Wireless phone manufacturers like Ericsson, Nokia and Motorola are marketing enhanced cellular phones with e-mail and Internet features. RIM and Motorola produce two-way pagers with Internet access capabilities. PDAs made by Compaq, Hewlett Packard, Palm are evolving from electronic schedulers to multi-purpose online media.

When the customer uses a wireless device like mobile phone and the Wireless Application Protocol (WAP), the privacy of the data is in fact not guaranteed in current version of WAP1.2.1. The WAP gateway constitutes a security hole since the data is transmitted in its original, un-encrypted form in side the gateway. What WAP fails to

provide is end-to-end security, which we define as a secure communication channel between the two parties on top of a potentially insecure network. In WAP, the gateway is the point between the two end points where the data may be compromised. Figure 4.3 demonstrates the current WAP solution for m-commerce.

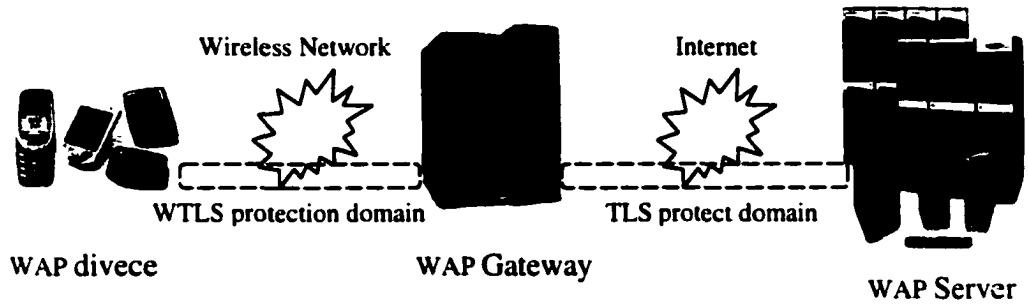



Figure 4.3 Security hole between wireless and server.

There are three major remedies to the breach of end-to-end security in WAP:

1. Putting the gateway inside the "vault".
2. Enabling Internet on the mobile device.
3. Application level security on top of WAP.

The first solution to the security problem conflicts with one of the fundamental purposes of the WAP gateway, namely to covert between two distinct protocol suites, one for the wireless network (WAP) and one for the traditional wired network (the Internet protocol suite, including HTTP and TCP).

The second approach is to re-design the WAP protocol to not use a gateway, and employ the existing Internet standards, including the transport protocol (TCP), for the entire wired and wireless part of a connection. By definition, this solves the security problem introduced by the gateway. Discarding the WAP gateway makes it possible to achieve the same high level of security for an m-commerce transaction as that of an e-commerce transaction on the ordinary web using full end-to-end encryption. Indeed, for WAP to discard the WAP gateway would turn the fully WAP-enabled mobile phone into an ordinary Internet device.

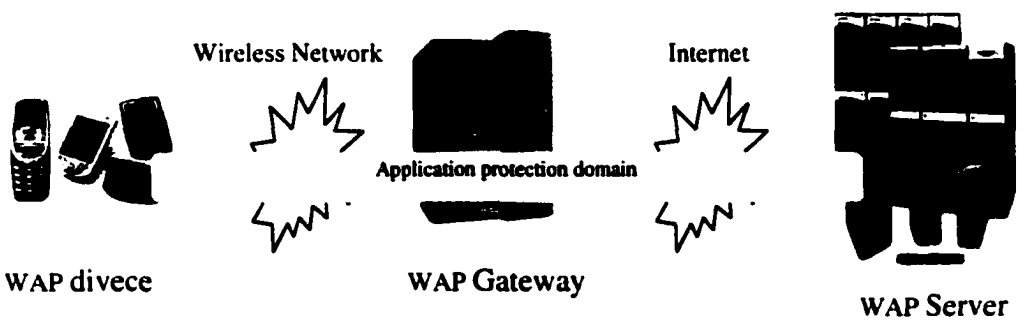



Figure 4.4 Applying secure application between wireless devices and server.

The third method to remove the security hole is to use end-to-end dedicated application between server and wireless devices. This solution will introduce security at a software layer above WAP, and considering WAP merely as a potentially insecure communication means. Instead of using WAP's security features, security is taken care of by means of dedicated software running at the two "ends", the wireless device and the e-merchant's web server (Figure 4.4). This solution requires infrastructural remedies to

convince the consumer about the security level available. Current solutions to the end-to-end security problem on the application layer still lack standardization and support from the WAP Forum.

Wireless devices are handicapped by their small physical size, restricted processing power, limited memory and finite power supply. Designed to be small and light, they currently cannot accommodate Pentium IV processors, a 20-GB hard drive and 256 MBs of RAM. Moreover, there is a limited bandwidth space for transmitting data and security protocols. Traditional enterprise security technologies like RSA and ECC are not practical for wireless devices. They are too large and too slow for the small wireless devices.

The NTRU PKCS delivers greater efficiencies and strong security in constrained platforms. From chapter 3 we can see that the NTRU PKCS is very fast while generating the private key but the RSA and ECC PKCS need a comparatively long period. Because of this reason, the NTRU Cryptosystem offers disposable keys, generated by embedded devices, that are disposable and created when users enter a PIN into their wireless device, producing an encryption key that's only good for a very short period. Once the transaction is complete, the key is discarded and a new key is generated for the next transaction. NTRU technology enables end-to-end secure service for the m-commerce, interactive banking, and mobile enterprise applications.

The NTRU PKCS will be the emerging standard for m-commerce and end-to-end transactional protection. We simulated the end-to-end digital signature generation /verification process on application secure level on our desktop computer, which has the

same CPU speed as the current PDAs. The simulation results are presented in the next section.

### 4.9 Simulation results of the Lattice Based Digital Signature for PDAs

The current (May 2001) palm-based products have 33-MHz processor with 8 MB RAM (Sony's Clie, Palm m505 etc.), and the latest class PDAs has 206-MHz processor with 32 MB memory (COMPAQ: iPAQ, please see http://www.compaq.com). Both of the mobile products have the ability to surf the web.

We use 200 MHz desktop computers to simulate the NTRU107 for the PDA. Table 4.19 lists the simulation results, which demonstrate that NTRU107 and NTRU167 are very fast and could be applied to palm-based products and PDAs directly.

| **Simulation Results of the lattice based signature scheme (Pentium Pro 200M)** | | | |
|---|---|---|---|
| **N=107** | **Second** | **N=167** | **Second** |
| $f$ = | 0.0007 | $f$ = | 0.0009 |
| g = | 0.0007 | g = | 0.0009 |
| $f$S = | 0.0231 | $f$S = | 0.062 |
| gS = | 0.025 | gS = | 0.060 |
| h = | 0.0073 | h = | 0.018 |
| Verification | 0.024 | Verification | 0.060 |

Table 4.19 Simulation results of the lattice based signature scheme.

As we know from chapter 3, the NTRU 107 and NTRU 167 are not very secure for the sensitive information, such as on-line banking or e-trade on the Internet by the wireless device. If we pursue the higher-level end-to-end security of information exchange by using NTRU 503 or NTRU 768, it'll take much more time. In Figure 4.5. we noticed that for wireless devices and the server, there exist two operations: signing and verifying. Because the wireless device has relatively weak computation capability compared with the server, the process will take much longer on the wireless divice side.

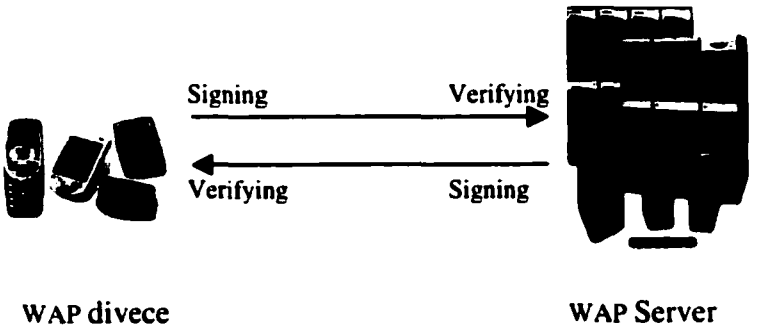



Figure 4.5 Unbalanced computation capabilities between wireless devices and the server.

When applying higher-level security like the NTRU 503 or NTRU 769 for m-commerce, we need find a way to decrease the computation time on the wireless device side. By analyzing the lattice based digital signature scheme in the verifying process. we have.

$$\mathbf{h}(s_i) = \mathbf{c}_{11}(s_i)*\boldsymbol{f}_1(s_i)*\mathbf{g}_1(s_i) + \mathbf{c}_{12}(s_i)*\boldsymbol{f}_1(s_i)*\mathbf{g}_2(s_i) + \mathbf{c}_{21}(s_i)*\boldsymbol{f}_2(s_i)*\mathbf{g}_1(s_i) + \mathbf{c}_{22}(s_i)*\boldsymbol{f}_2(s_i)*\mathbf{g}_2(s_i) \pmod q.$$

This is the most timing consuming process in the verification process because there are 8 polynomial multiplication operations for verifying every number in set S. Suppose the wireless device only checks the condition of verification for a random selection of $s_i \in S$ (normally more than 100), the time for verification process will be dropped tremendously. The distribution of observed sample mean could be approximated by a normal distribution with large sample size. We can apply a statistic hypothesis test to determine the required sample size (less than the size of S) to achieve a high success probability for a verification success. In other words, suppose that the verifier checks (say) 40 values of $s_i$ and all of them pass the test. The probability of this happening at random is $2^{-40}$, so he can be fairly confident that every $s_i \in S$ will pass the verification test. By this way, we can greatly increase operating speed while decreasing memory requirements.

| **Verification options for wireless device and its rate (second)** | | | |
|---|---|---|---|
| | **$\hat{S} = 40$** | **$\hat{S} = 60$** | **$\hat{S} = 80$** |
| **N = 263** | 0.360 | 0.554 | 0.721 |
| **N = 503** | 1.275 | 1.899 | 2.540 |
| | | | |
| | **$\hat{S} = 90$** | **$\hat{S} = 120$** | **$\hat{S} = 180$** |
| **N = 503** | 3.965 | 5.638 | 7.521 |
| **N = 769** | 8.650 | 12.268 | 17.565 |

Table 4.20 The verification process speed with different sample size $\hat{S}$ on wireless device side.

Table 4.20 demonstrates an improvement of the verification process speed with selected sample size ($\hat{S}$) on the wireless device side.

## 4.10 The NTRU Authentication Scheme

The lattice based digital signature scheme could also be used as an authentication scheme to verify the identity during communication. As we described at the beginning of this chapter, this scheme is based on the hard mathematical problem of finding a binary polynomial $f(X)$ that takes on prescribed values $f(\alpha)$ mod q at a given collection of numbers $\alpha = \alpha_1, \alpha_2, \ldots, \alpha_n$. This problem is equivalent to solving the closest vector problem in a certain lattice; please see [HPS00A] for details. Briefly, the Prover publishes the set of values $f(\alpha_i)$ as her public key and she proves her identity by demonstrating that she possesses a binary polynomial taking those values. The main steps are listed as follows.

1. Public Parameters selection

All users agree on a prime number q and a set of distinct numbers $S = \{\alpha_1, \ldots, \alpha_{N/2}\}$ modulo q. Let N = q-1. All polynomials are of degree N-1. Polynomial multiplication uses the rule given at the beginning of this chapter, and all computations (except for verification step A) are performed modulo q. Appropriate quantities $A_h$, $B_h$ are also chosen to be used during the verification process.

2. Key Creation

The Prover selects a binary polynomial $f(X)$ . This polynomial is her private key. She publishes a set of values $f(S) = \{ f(\alpha) : \alpha \in S \}$ . This set of values is her public key.

3. Commitment

During the commitment step, the Prover selects another binary polynomial $g_1(X) \in R_q$. She computes and sends to the Verifier a set of values $g_1(S)$ .

4. Challenge

During the challenge step, the Verifier selects four extremely small polynomials $c_{11}(X)$, $c_{12}(X)$, $c_{21}(X)$ and $c_{22}(X)$ and sends them to the Prover. For security reasons, it is important that all these polynomials have no nonzero roots modulo q for values of X not in S .

5. Response

During the response step, the Prover selects a third binary polynomial $g_2(X)$ . She computes and sends to the Verifier the following polynomial:

$$h(X) = c_{11}(X)*f_1(X)*g_1(X) + c_{12}(X)*f_1(X)*g_2(X) + c_{21}(X)*f_2(X)*g_1(X) + c_{22}(X)*f_2(X)*g_2(X) \pmod q.$$

6. Verification

The Verifier performs the following two steps to verify the Prover's identity:

(A) The Verifier checks that the polynomial h(X) is moderately small by writing it as $h(X) = \Sigma a_i X^i$ and verifying the bound

$$\sum_{i=0}^{N-1} (a_i - A_h)^2 < B_h$$

where $A_h$ and $B_h$ are public quantities. Note that this computation is not done modulo q but is simply a sum of integers.

(B) For each $\alpha_i \in S$, the Verifier computes the quantity.

$$\mathbf{h}(\alpha_i) = \mathbf{c}_{11}(\alpha_i)*\boldsymbol{f}_1(\alpha_i)*\mathbf{g}_1(\alpha_i) + \mathbf{c}_{12}(\alpha_i)*\boldsymbol{f}_1(\alpha_i)*\mathbf{g}_2(\alpha_i) +$$

$$\mathbf{c}_{21}(\alpha_i)*\boldsymbol{f}_2(\alpha_i)*\mathbf{g}_1(\alpha_i) + \mathbf{c}_{22}(\alpha_i)*\boldsymbol{f}_2(\alpha_i)*\mathbf{g}_2(\alpha_i) \pmod q$$

and checks whether it is equal $h(\alpha_i) \pmod q$ . If the polynomial h passes tests (A) and (B), then the Verifier accepts the Prover's identity.

# Chapter 5

# An Application of the Lattice Based Digital Signature on the "Paperless" Report/Billing System

## 5.1 The Lattice Based Signature Scheme and "Paperless" Report/Billing Document Integrity

Many business applications can benefit from the lattice based public key cryptography system and the digital signature technology. Because of its high-speed operation, fast generation of "disposable keys," or the ability to generate a new independent security key for each transaction, the lattice based polynomial ring digital signature scheme is well suited to provide authentication and non-repudiation in the electronic-commerce. Such applications as banking record and payment billing systems, internet brokerage trades, business to business portals, postal applications and healthcare professionals in hospitals or clinical areas who are performing inquiries on sensitive patient's medical record information. With more and more of these activities being replaced by an electronic or "paperless" system, the authentication and integrity become more important than ever.

With the goal of accelerating the migration to a "paperless" electronic society, the National Institute of Standards and Technology (NIST) adopted the Digital Signature Standard (DSS) in 1994. This was followed by the banking standards of the American National Standards Institute (ANSI) adopting the DSS for electronic banking. As part of the application of the DSS, the United States Government is also planning a network of certification authorities, called the Public Key Infrastructure (PKI). Digital signatures with certificates will eventually be used for securing all electronic transactions for the government and be considered the enabling technology for conducting secure electronic commerce and verifying the integrity of the downloaded software, such as ActiveX components, JAVA applets, on the Internet.

## 5.2 An Application of the "Paperless" Report/Billing System

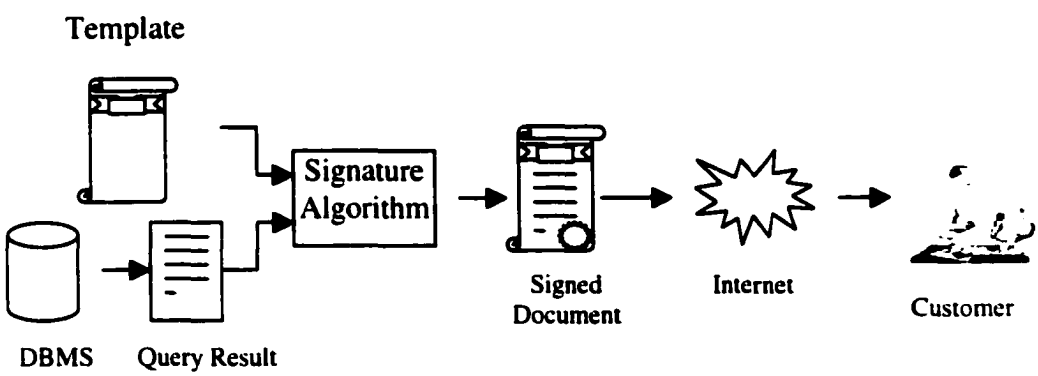



Figure 5.1 The structure of the "paperless" report/billing system.

The Educational Testing Service (ETS) is the world's largest private educational testing and measurement organization. A nonprofit company has dedicated to serving the

needs of individuals, educational institutions and agencies, and governmental bodies in 181 countries. ETS develops and annually administers more than 11 million tests worldwide. Every year, over 1 million people take the TOEFL (Test Of English as Foreign Language) and GRE (Graduate Record Exam) for their higher education. A lot of foreign governments and international companies measure their employer's English ability by using TOEFL scores from the ETS. The ETS needs many score reporting services for TOEFL and GRE examinees all over the world. By using the traditional paper report via post mail, an examinee in a foreign country could receive his/her score in about two mouths. For some foreign students who want to study in a US college, this months means they will wait longer before they make their final decision, and that's if the mail doesn't get lost during the delivery.

Receiving TOEFL and GRE score reports via the Internet by email is easier and faster than other approaches. By using the electronic score reporting system, or paperless score reporting system, examinees could ask ETS to send their scores, with digital authentication, via email after they see the results on the ETS website. The ETS delivery center can also send the score to the graduate school of the examinee choice.

### 5.3 The Structure of the "Paperless" Report/Billing System

From the figure 5.1 we can see that the whole reporting system can be divided into five steps:

- Query from DBMS.
- Query results data merge with the template file.
- Signing the merged data.
- Send signed document through the Internet.
- Verification process by the receiver.

We'll describe each step in the following sections.

### 5.3.1 DBMS in "Paperless" Report/Billing System

The system can use any relational DBMS in the current market. In our example, we use Microsoft Access as our database driver. The reason we use Access is because relational databases are very popular and have many advantages over other methods for the construction of data processing systems. Other reasons include:

- Massive installed base; the majority of existing database systems are relational;
- Strong theory: high normal forms together with integrity rules guarantee relational databases that are incapable of storing anomalous information;
- Wide range of high-quality implementations;
- Well-defined mathematical models for data manipulation based on relational algebra or relational calculus;
- Powerful query and manipulation languages such as SQL.

Relational databases systems allow complex data-processing problems to be solved with relatively simple, elegant solutions. Figure 5.2 shows the structure of the database ETS.mdb.

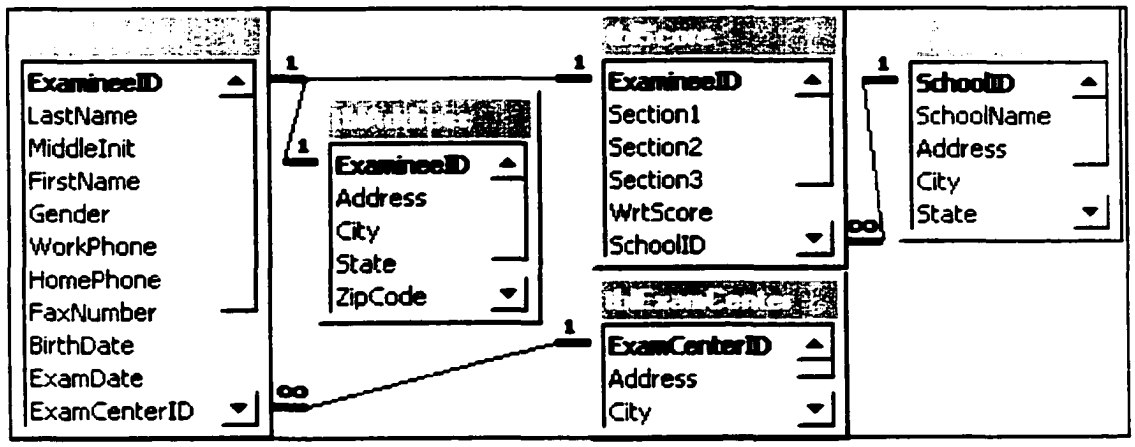



Figure 5.2 The database structure of the ETS "paperless" report/billing system.

The template file is a standard file for the "paperless" report/billing system. In our TOFEL reporting system, before we run the query, the template file looks like Figure 5.3.

```
TTTTTTT    000     EEEEE    FFFFFF    L
   T      0   0    E        F         L
   T      0   0    EEE      FFFF      L
   T      0   0    E        F         L
   T       000     EEEEE    F         LLLLL

TEST   OF  ENGLISH AS  A  FORGIGN  LANGUAGE
=========================================
|SECTION#1|SECTION#2|SECTION#3|TOTAL SCORE|
=========================================

=========================================
        TEST OF WRITTEN ENGLISH
=========================================

=========================================

*****EXAMINEE'S ORIGINAL SCORE RECORD*****
**TEST OF ENGLISH AS A FOREIGN  LANGUAGE**
P.O. BOX  6151 PRINCETON NJ 08541-6151 USA
****************** END ********************
```

Figure 5.3 The template file in the "paperless" report/billing system.

It is noted that the template file doesn't contain any information for a specific examinee before we merge it with the query result.

Figure 5.4 is the query result for Mr. John Doe, with a registration number equal to 7715703 for TOFEL exam.

```
|SECTION#1|SECTION#2|SECTION#3|TOTAL SCORE|
=========================================
[   32   ][   60   ][   63    ][   517    ]
=========================================
    TWE SCORE      [       4.5           ]
=========================================

REGISTRATION NUMBER: 7715703
NAME(SURNAME, GIVEN, MIDDLE) JOHN DOE
DATE OF BIRTH(M/D/Y): 06/18/80
SEX: M         NATIVE COUNTRY: CHINA PRC
DEGREE: 1      REASON FOR TAKING TOEFL: 2
TOEFL TAKEN BEFORE: 1
TEST DATE(MONTH YEAR) FEB 2001
CENTER NUMBER: 8788

**********EXAMINEE'S ADDRESS**************
8788 7715703
JOHN DOE
168 GARDEN PARK AVE.
APT 168A
BEIJING CHINA 100088
```

Figure 5.4 The query result in the "paperless" report/billing system

## 5.4 The Java Database Connectivity (JDBC)

JDBC technology is an API that lets you access virtually any tabular data source from the Java programming language [HC99]. It provides cross-DBMS connectivity to a wide range of SQL databases, and now, with the new JDBC API, it also provides access to other tabular data sources, such as spreadsheets or flat files. JDBC does three things:

- establish a connection with a database;
- send SQL statements;
- process the results.

Figure 5.5 demonstrates the JDBC structure in the "paperless" report/billing system.

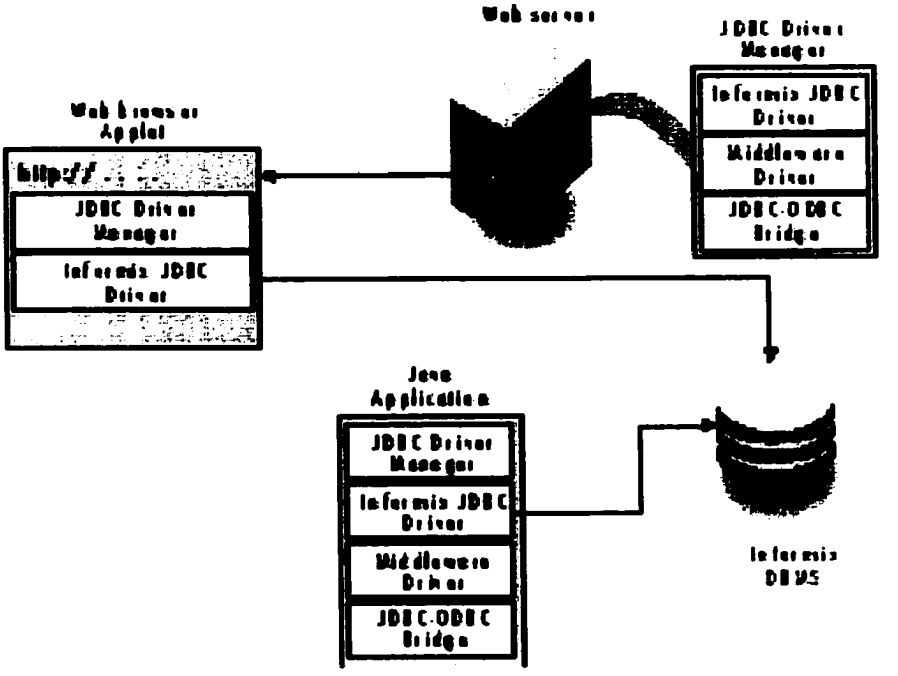



Figure 5.5 The JDBC connection structure.

The JDBC API allows developers to take advantage of the Java platform's "Write Once, Run Anywhere" capabilities for industrial strength, cross-platform applications that require access to enterprise data. With a JDBC technology-enabled driver, a developer can easily connect all corporate data, even in a heterogeneous environment.

The following pseudo-code demonstrates the main steps of the JDBC operations.

```
/* Demonstrate connection to a database and retrieval
   of some data.Then sign the data at once.
*/

public class QueryTest {

  public static void main(String args[]) {

    final String outfS    = "fS.txt";
    final String outgS    = "gS.txt";
    final String intemp   = "temp.txt";
    final String outhPoly = "hPoly.txt";

    //Input template file from "temp.txt"
```

```
        BufferedReader br_temp;
        PrintWriter    pw_fs, pw_gs, pw_hPoly;

        Open_files(outfS, outgS, intemp, outhPoly);

        try {
          Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

          Connection con = DriverManager.getConnection
            ("jdbc:odbc:jdbctest", "sml", "Querytest");

          Statement stmt = con.createStatement();

          ResultSet rs =
            stmt.executeQuery("SELECT values FROM Mytables");
          while (rs.next()) {
            String x = rs.getString("value");
            pw_result.println(x);
          }

          MergeData(br_temp, pw_result);

        } catch (Exception e) {
            System.out.println(e);
        }

        flushDataToFile();
        singMergedData();
        closeAllOpennedFiles();

      }
  }
```

The above pseudo-code has the following functions:

1. Open the template file and reads the data into a buffer.
2. Load the JDBC-ODBC driver.
3. Set up a connection to the named data-source.
4. Use the connection to create a statement.
5. Call executeQuery on the statement to retrieve a result set.
6. Iterates over all the tuples in result set, retrieving a particular attribute each time.
7. Close the connection.
8. Merge the template file with query results.
9. Create new query result file.
10. Sign the query result file.

The same format can be followed for any query statement that retrieves data from the source.

### 5.5 The Query Result and Its Signature

The following text is the query result file of the original text file "TOFELORG.txt".

```
TTTTTTT    000     EEEEE    FFFFFF    L
   T      0   0    E        F         L
   T      0   0    EEE      FFFF      L
   T      0   0    E        F         L
   T       000     EEEEE    F         LLLLL

TEST   OF  ENGLISH AS  A  FORGIGN  LANGUAGE
=========================================
|SECTION#1|SECTION#2|SECTION#3|TOTAL SCORE|
=========================================
[   32   ][   60   ][   63   ][   517    ]
=========================================
         TEST OF WRITTEN ENGLISH
=========================================
    TWE SCORE      [       4.5           ]
=========================================
REGISTRATION NUMBER: 7715703
NAME(SURNAME, GIVEN, MIDDLE) JOHN DOE
DATE OF BIRTH(M/D/Y): 06/18/80
SEX: M         NATIVE COUNTRY: CHINA PRC
DEGREE: 1      REASON FOR TAKING TOEFL: 2
TOEFL TAKEN BEFORE: 1
TEST DATE(MONTH YEAR) FEB 2001
CENTER NUMBER: B788

**********EXAMINEE'S ADDRESS**************
B788 7715703
JOHN DOE
168 GARDEN PARK AVE.
APT 168A
BEIJING CHINA 100088

*****EXAMINEE'S ORIGINAL SCORE RECORD*****
**TEST OF ENGLISH AS A FOREIGN  LANGUAGE**
P.O. BOX  6151 PRINCETON NJ 08541-6151 USA
****************** END ********************
```

Figure 5.6 The merged text "TOFELORG.txt"

After we apply SHA-1 algorithm on the "TOEFLORG.txt" file, the 160 bit hash values are listed in Table 5.1.

| 160 bit SHA-1 hash values of the file "TOEFLORG.txt" | |
|---|---|
| hash[00] = 0101 0011 | hash[10] = 1010 1001 |
| hash[01] = 0010 1000 | hash[11] = 0010 1011 |
| hash[02] = 1000 0010 | hash[12] = 1001 0110 |
| hash[03] = 1100 0010 | hash[13] = 1111 1100 |
| hash[04] = 0011 0010 | hash[14] = 1011 0111 |
| hash[05] = 0001 1011 | hash[15] = 0110 0011 |
| hash[06] = 1101 1010 | hash[16] = 1010 1111 |
| hash[07] = 1100 1110 | hash[17] = 0011 0101 |
| hash[08] = 1110 1000 | hash[18] = 1111 1000 |
| hash[09] = 0101 1100 | hash[19] = 0110 0111 |

Table 5.1 The hash values of the file "TOEFLORG.txt"

Table 5.2 lists the signature of the file after the sender signs the query result.

| The signature of the file "TOEFLORG.txt" | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| g(S) | | | | | | | | | |
| 10 | 422 | 341 | 278 | 110 | 553 | 57 | 206 | 83 | 462 |
| 147 | 110 | 611 | 182 | 711 | 196 | 15 | 513 | 567 | 368 |
| 718 | 171 | 385 | 704 | 60 | 48 | 80 | 671 | 692 | 84 |
| 491 | 328 | 188 | 354 | 318 | 20 | 440 | 203 | 350 | 46 |
| 33 | 541 | 626 | 139 | 128 | 743 | 369 | 411 | 613 | 52 |
| 171 | 718 | 377 | 138 | 77 | 363 | 404 | 702 | 765 | 251 |
| 369 | 198 | 541 | 647 | 50 | 429 | 394 | 665 | 399 | 1 |
| 136 | 243 | 196 | 711 | 35 | 531 | 674 | 486 | 651 | 282 |
| 548 | 69 | 161 | 118 | 403 | 548 | 752 | 540 | 244 | 496 |
| 235 | 673 | 638 | 653 | 750 | 608 | 595 | 108 | 98 | 440 |
| 180 | 198 | 531 | 461 | 276 | 58 | 731 | 389 | 192 | 4 |
| 100 | 624 | 711 | 26 | 158 | 477 | 560 | 759 | 673 | 280 |
| 136 | 542 | 133 | 421 | 444 | 709 | 279 | 391 | 496 | 240 |
| 198 | 180 | 384 | 116 | 673 | 274 | 717 | 715 | 549 | 457 |
| 553 | 471 | 539 | 28 | 66 | 212 | 213 | 718 | 507 | 449 |
| 51 | 381 | 608 | 750 | 429 | 290 | 4 | 245 | 346 | 126 |

| h(S) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| -64 | -82 | -46 | 49 | -41 | 83 | -23 | 33 | 29 | -24 |
| 7 | -36 | 35 | -22 | -19 | 52 | -37 | -4 | -50 | -50 |
| -93 | -37 | -20 | -1 | 39 | -77 | 15 | -77 | -24 | -63 |
| 50 | 0 | -22 | 46 | -24 | 68 | 70 | 2 | -46 | 33 |
| 14 | -22 | 50 | 45 | -55 | 9 | -36 | 52 | -98 | 34 |
| -60 | 53 | -4 | -37 | 50 | 144 | 50 | 93 | 14 | -22 |
| -55 | -11 | 18 | -31 | 62 | -78 | 11 | 126 | -53 | 66 |
| -129 | -38 | 34 | -3 | 44 | 24 | -2 | -27 | -32 | -50 |
| -26 | -7 | 73 | 4 | 44 | 93 | 79 | -48 | 78 | -6 |
| 85 | -18 | -57 | 56 | 12 | -7 | -71 | -81 | -27 | 9 |
| 60 | 37 | 20 | -58 | -36 | 9 | -1 | 18 | -71 | 84 |
| -25 | 6 | -11 | 1 | 39 | -15 | -42 | -55 | -20 | 36 |
| -50 | 12 | -25 | 32 | -53 | 38 | -25 | -26 | -111 | -60 |
| 49 | -98 | -13 | 16 | -6 | 102 | 54 | -97 | -12 | 40 |
| -23 | -44 | 27 | -20 | -120 | 27 | 96 | 26 | 3 | -5 |
| 11 | -34 | -47 | -7 | 38 | 31 | 34 | -5 | 53 | 16 |
| -126 | 84 | 1 | -4 | 19 | -35 | 67 | 8 | -6 | 72 |
| 21 | -46 | 54 | 3 | 20 | 76 | 28 | 62 | -78 | -55 |
| 0 | 22 | -44 | 31 | 1 | -71 | 66 | 50 | 7 | 0 |
| -42 | -58 | -7 | 21 | -24 | 22 | 93 | 19 | 49 | -54 |
| -51 | -73 | 75 | -73 | -46 | 18 | 13 | -5 | -51 | 70 |
| -11 | -29 | 34 | -36 | -37 | 10 | 23 | 82 | -14 | 51 |
| 26 | 114 | 5 | -73 | 88 | -47 | 24 | -45 | -5 | -58 |
| -81 | 25 | -50 | 123 | -55 | -36 | -33 | 64 | 34 | 5 |
| 31 | -62 | 36 | 2 | -21 | -10 | 28 | -37 | -60 | -136 |
| 11 | -90 | 31 | 11 | -12 | -10 | -40 | -45 | -1 | 34 |
| 57 | -36 | 63 | 71 | 27 | 25 | 6 | 39 | -34 | 8 |
| 63 | 16 | -43 | 14 | -28 | -29 | 35 | 38 | -14 | 50 |
| 10 | -11 | -35 | -19 | -6 | -25 | 88 | 86 | -63 | 5 |
| 17 | 21 | 26 | 0 | -120 | 17 | -10 | 10 | 114 | 1 |
| -8 | -17 | 57 | -55 | 9 | -69 | -61 | 55 | 38 | 25 |
| -38 | 7 | -55 | -8 | -12 | -63 | 25 | -45 | -85 | -3 |
| 13 | -31 | 9 | -24 | 3 | 10 | 7 | 55 | 35 | -49 |
| -13 | 15 | 18 | 53 | 11 | -53 | 12 | 31 | -66 | 74 |
| 45 | -22 | -33 | 6 | -84 | -48 | 57 | 26 | 73 | 50 |
| 95 | -115 | 28 | -7 | -45 | -85 | -61 | 19 | -51 | 89 |
| -101 | -24 | 75 | -23 | -9 | -51 | 4 | -80 | 57 | 8 |
| 0 | 50 | -58 | 12 | -26 | 58 | -123 | 24 | 41 | -32 |
| 49 | -104 | 0 | -44 | -11 | 17 | 10 | -61 | -29 | 65 |
| -24 | -81 | -95 | 84 | 73 | 50 | 80 | 49 | 47 | 75 |
| -34 | -20 | 49 | -23 | 6 | -59 | 43 | -21 | -13 | -11 |
| -19 | 7 | -31 | -21 | -21 | 5 | 30 | 42 | 78 | -87 |
| 40 | -11 | -94 | 7 | 67 | -2 | -43 | -10 | -70 | -41 |

| 49 | -23 | -59 | 69 | 31 | 43 | -3 | -5 | -25 | 16 |
|---|---|---|---|---|---|---|---|---|---|
| 27 | 61 | 19 | 37 | -67 | 11 | 13 | -19 | -33 | 76 |
| -33 | 73 | 7 | -112 | -73 | -32 | 124 | 17 | -25 | -55 |
| 28 | -17 | 28 | 44 | 21 | -52 | 17 | -31 | -36 | 67 |
| -19 | -13 | -9 | -55 | 93 | -19 | -15 | 118 | -69 | -11 |
| -23 | -8 | -76 | 168 | 1 | -66 | 7 | -22 | 15 | 25 |
| 108 | -42 | -24 | 23 | -41 | 53 | -53 | 30 | -87 | 23 |
| 0 | -6 | 27 | 28 | 69 | 13 | 21 | -19 | 14 | 59 |
| -19 | 31 | -50 | -45 | 21 | -8 | 38 | -80 | 17 | 31 |
| 11 | -19 | -19 | 55 | -99 | -77 | 30 | 7 | 29 | -21 |
| 129 | -24 | 42 | -22 | 25 | 13 | -15 | 22 | 40 | 32 |
| 17 | -9 | -88 | 92 | -95 | 9 | -17 | 58 | 2 | -14 |
| 98 | -31 | -55 | -61 | -68 | -44 | -136 | 120 | 22 | 20 |
| -22 | 12 | 44 | -11 | 28 | -141 | 140 | -44 | 0 | 2 |
| -10 | 37 | -46 | 46 | 68 | 74 | -44 | -14 | -38 | -99 |
| 49 | -38 | 168 | 0 | -31 | -18 | -13 | 91 | -47 | 2 |
| 35 | -61 | 51 | -51 | 57 | 4 | 51 | -2 | 27 | 62 |
| 9 | -35 | 2 | 78 | 49 | 39 | -62 | 57 | -56 | -59 |
| -61 | -50 | -2 | 30 | -47 | -58 | -27 | 21 | -59 | -37 |
| 76 | 37 | -30 | -8 | -36 | -8 | -56 | 53 | 23 | -51 |
| 49 | -44 | 28 | -4 | 20 | -1 | -43 | 16 | -11 | 7 |
| -37 | -35 | 93 | 10 | 65 | -8 | -5 | 40 | -55 | 17 |
| 41 | -4 | -47 | 62 | 46 | 23 | 54 | 13 | -35 | -5 |
| -109 | 65 | -69 | 18 | -33 | -47 | -21 | -7 | 92 | -44 |
| 59 | -36 | 62 | -64 | -5 | 7 | -98 | -12 | -72 | -29 |
| -39 | 90 | -12 | 25 | 35 | -104 | -15 | 3 | 27 | -52 |
| 128 | 24 | 16 | 36 | 44 | 11 | -120 | -76 | 5 | 32 |
| -41 | 58 | 6 | -98 | 12 | -3 | 46 | 39 | -17 | -81 |
| -100 | 79 | 6 | 99 | -51 | -19 | 10 | 43 | 36 | -20 |
| 60 | 32 | -27 | 8 | 33 | 75 | -26 | 54 | 27 | -90 |
| 13 | -104 | 54 | 9 | -3 | -13 | 12 | 12 | 40 | 75 |
| 0 | 12 | -13 | -65 | -8 | 23 | 3 | -33 | 25 | -59 |
| 2 | 125 | -63 | 45 | -12 | 8 | 48 | 14 | -7 | -25 |
| 65 | -21 | 1 | -74 | -5 | -32 | -3 | -20 | | |

Table 5.2 The signature of the file "TOEFLORG.txt"

When the examinee receives the file and the digital signature of the file through the Internet. He/she can verify the validity of the file by the verification process we have described in Chapter 4.

The verification results are listed in Table 5.3.

| The verification results of the file "TOFELORG.txt" | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| The values of h(S) are: | | | | | | | | | |
| 256 | 623 | 598 | 629 | 642 | 338 | 710 | 286 | 88 | 595 |
| 403 | 41 | 456 | 477 | 331 | 183 | 552 | 508 | 732 | 184 |
| 322 | 729 | 480 | 374 | 281 | 349 | 431 | 754 | 738 | 350 |
| 278 | 237 | 255 | 172 | 727 | 111 | 563 | 76 | 389 | 146 |
| 708 | 399 | 731 | 476 | 178 | 160 | 196 | 746 | 68 | 644 |
| 729 | 322 | 679 | 401 | 614 | 116 | 303 | 639 | 246 | 594 |
| 161 | 438 | 639 | 339 | 402 | 305 | 202 | 730 | 104 | 613 |
| 55 | 267 | 183 | 331 | 396 | 370 | 256 | 180 | 621 | 47 |
| The values of *f**g*c are: | | | | | | | | | |
| 256 | 623 | 598 | 629 | 642 | 338 | 710 | 286 | 88 | 595 |
| 403 | 41 | 456 | 477 | 331 | 183 | 552 | 508 | 732 | 184 |
| 322 | 729 | 480 | 374 | 281 | 349 | 431 | 754 | 738 | 350 |
| 278 | 237 | 255 | 172 | 727 | 111 | 563 | 76 | 389 | 146 |
| 708 | 399 | 731 | 476 | 178 | 160 | 196 | 746 | 68 | 644 |
| 729 | 322 | 679 | 401 | 614 | 116 | 303 | 639 | 246 | 594 |
| 161 | 438 | 639 | 339 | 402 | 305 | 202 | 730 | 104 | 613 |
| 55 | 267 | 183 | 331 | 396 | 370 | 256 | 180 | 621 | 47 |
| (A) The test pass Sigma(hi^2) < (Bh*q)^2 test!!! | | | | | | | | | |
| (B) The verification process is PASSED! | | | | | | | | | |

Table 5.3 The signature verification result of the file "TOEFLORG.txt"

During the transmission, if the file is modified, the modified file can't pass the verification process as it is described in the following example.

In the following file "TOFELMOD.txt", Mr. John Doe's TOFEL test results were modified. The grade of section#1 was changed from 32 to 62, and the total score was changed from 517 to 617.

```
TTTTTTT    000     EEEEE    FFFFFF    L
   T      0   0    E        F         L
   T      0   0    EEE      FFFF      L
   T      0   0    E        F         L
   T       000     EEEEE    F         LLLLL

TEST   OF  ENGLISH AS  A  FORGIGN  LANGUAGE
=========================================
|SECTION#1|SECTION#2|SECTION#3|TOTAL SCORE|
=========================================
[   62   ][   60   ][   63    ][   617    ]
=========================================
         TEST OF WRITTEN ENGLISH
=========================================
    TWE SCORE      [       4.5           ]
=========================================
REGISTRATION NUMBER: 7715703
NAME(SURNAME, GIVEN, MIDDLE) JOHN DOE
DATE OF BIRTH(M/D/Y): 06/18/80
SEX: M         NATIVE COUNTRY: CHINA PRC
DEGREE: 1      REASON FOR TAKING TOEFL: 2
TOEFL TAKEN BEFORE: 1
TEST DATE(MONTH YEAR) FEB 2001
CENTER NUMBER: B788

**********EXAMINEE'S ADDRESS**************
B788 7715703
JOHN DOE
168 GARDEN PARK AVE.
APT 168A
BEIJING CHINA 100088

*****EXAMINEE'S ORIGINAL SCORE RECORD*****
**TEST OF ENGLISH AS A FOREIGN  LANGUAGE**
P.O. BOX  6151 PRINCETON NJ 08541-6151 USA
****************** END ********************
```

Figure 5.7 The modified text "TOFELMOD.txt"

Table 5.4 lists the hash values of the modified file "TOEFLMOD.txt".

| 160 bits SHA-1 hash value of file "TOFELMOD.txt" | |
|---|---|
| hash[00] = 0110 1101 | hash[10] = 0011 0010 |
| hash[01] = 1110 1010 | hash[11] = 1110 0101 |
| hash[02] = 0011 1101 | hash[12] = 1000 1100 |
| hash[03] = 1001 1010 | hash[13] = 0101 1001 |
| hash[04] = 1110 1000 | hash[14] = 1110 1101 |
| hash[05] = 1111 0000 | hash[15] = 0011 1001 |
| hash[06] = 1101 0111 | hash[16] = 0110 0101 |
| hash[07] = 1110 0100 | hash[17] = 1000 1010 |
| hash[08] = 0101 1011 | hash[18] = 1100 1000 |
| hash[09] = 1011 0010 | hash[19] = 0110 1011 |

Table 5.4 The hash values of the file "TOEFLMOD.txt"

As we can see from the verification results in Table 5.5, the verification process failed.

| The verification results of the file "TOFELMOD.txt" | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| The values of h(S) | | | | | | | | | |
| 256 | 623 | 598 | 629 | 642 | 338 | 710 | 286 | 88 | 595 |
| 403 | 41 | 456 | 477 | 331 | 183 | 552 | 508 | 732 | 184 |
| 322 | 729 | 480 | 374 | 281 | 349 | 431 | 754 | 738 | 350 |
| 278 | 237 | 255 | 172 | 727 | 111 | 563 | 76 | 389 | 146 |
| 708 | 399 | 731 | 476 | 178 | 160 | 196 | 746 | 68 | 644 |
| 729 | 322 | 679 | 401 | 614 | 116 | 303 | 639 | 246 | 594 |
| 161 | 438 | 639 | 339 | 402 | 305 | 202 | 730 | 104 | 613 |
| 55 | 267 | 183 | 331 | 396 | 370 | 256 | 180 | 621 | 47 |

| The values of c*f*g in the verification process | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 224 | 246 | 43 | 382 | 357 | 687 | 9 | 756 | 332 | 710 |
| 733 | 547 | 337 | 169 | 127 | 646 | 37 | 289 | 541 | 126 |
| 82 | 343 | 299 | 232 | 369 | 620 | 763 | 667 | 123 | 660 |
| 662 | 722 | 626 | 599 | 481 | 481 | 718 | 693 | 215 | 760 |
| 101 | 546 | 24 | 244 | 627 | 482 | 449 | 629 | 694 | 564 |
| 343 | 82 | 376 | 76 | 201 | 643 | 209 | 330 | 226 | 646 |
| 245 | 168 | 174 | 576 | 580 | 681 | 23 | 640 | 564 | 528 |
| 457 | 260 | 646 | 127 | 593 | 555 | 528 | 665 | 241 | 193 |
| (A) The test pass Sigma(hi^2) < (Bh*q)^2 test!!! | | | | | | | | | |
| (B) Set Values hS[0] and cfg[0] doesn't match! hS[0] = 256 and cfg[0] = 224. Authentication Failed! The verification process is FAILED! | | | | | | | | | |

Table 5.5 The verification result of the file "TOEFLMOD.txt".

In order to help the examinees verify the "paperless" score report, ETS should publish their public key for the user with a digital certification. They can get the digital certification from the trusted third parties. Digital Certification is the process of the trusted third-party electronic verification of a sender's identity.

# Chapter 6

# Attacks on the Lattice Based PKCS

The lattice based NTRU PKCS, was first presented by J. Hoffstein, J. Pipher and J.H. Silverman at Crypto'96, received a great attention in the cryptographic community immediately. A lot of cryptographers have studied this significant new PKCS. In 1997. Coppersmith and Shamir stated the NTRU system was not reliably secure after they analyzed the first version of the NTRU using the lattice reduction method [CS97]. Because of this, the parameters of NTRU were cautiously chosen and the system was significantly improved. The preliminary version of the NTRU Signature Scheme (NSS) [HS00] was attacked by Mironov from the correlation between some bits of a signature and coefficients of a secret random polynomial [MIR00]. The new version of NSS [HPS00B] was analyzed and attacked recently by Nguyen and Stern [PS01]. We will describe these attack methods in the following sections.

## 6.1 The Lattice Reduction Attack on the Public Key

The lattice reduction method was originated by Shamir in 80's and greatly improved by Lenstra, Lenstra, and Lovasz [LLL82]. The latest improvement was made

by Schnorr and Euchner [SE91]. The goal of the lattice reduction is to find one or more "small" vectors in a given lattice. In theory the smallest vector can be found by an exhaustive search, but in practice this is not possible if the dimension is very large. Lattice reduction methods can be used by attacker to recover the private key $f$, or forge a valid false private key $f'$ from the public key. When the attacker uses this method as an off-line attack, he/she could also fabricate a valid public key h for a given signature or attack the message **m**.

The method starts by constructing a lattice for any polynomial $F \in R$, with a vector as its coefficients as follows: $(a_0, a_1, a_2, \ldots, a_{N-1})$. Similarly for any such vector or point in $Z^N$, one can take the polynomial built from these coefficients, reduce mod q and obtain an $F \in R$.

Let L be the lattice of all points in $Z^N$ such that the corresponding polynomial F satisfies

$$F(\alpha) \equiv 0 \pmod{q} \text{ for each } \alpha \equiv S.$$

We can easily check that L is a lattice and the determinant of L is $q^t$. As stated above, it's not too hard to find a polynomial $F' \in R$ so that $F'(\alpha) \equiv f(\alpha) \bmod q$ for all $\alpha \in S$. However, the probability of such an F' having small coefficients is very small. Suppose we find an F' with non-small coefficients and then search for a point $F \in L$ close to F'. Let's set $\mathbf{I} = F' - F$ if such an F is found. Then **I** will still have the right evaluations at $\alpha \bmod q$. If F is very close to F', then $|\mathbf{I}|$ will still be quite small.

The problem of finding an **I** that will give a good impersonation of private key f is thus reduced to find a point in a lattice as close as possible to a given point outside the lattice. This is a non-homogeneous version of the problem of finding a short vector in a lattice. This could also be converted into a homogeneous question in a similar lattice of one higher dimension. The probability of an attacker's success in a given period gets larger when the distance of the given point to the lattice gets smaller. The attacker's success probability gets poorer when the dimension value of the lattice gets bigger.

Table 6.1 shows the average time of recovering private key $f$ from $\{f(\alpha)\}$ by using the lattice reduction method. A sequence of primes q , N = q-1 and $d_f = [q/3]$ are used in the experiments [HLS99].

| q | N | T_average(second) |
|---|---|---|
| 101 | 100 | 171 |
| 109 | 108 | 226 |
| 113 | 112 | 366 |
| 149 | 148 | 3787 |
| 157 | 156 | 7443 |
| 173 | 172 | 32760 |
| 181 | 180 | 167021 |

Table 6.1 The average recovering time of the private $f$ from $f(\alpha)$.

The regression line for the average time as a function of N is:

$$\log (T) = 0.0750N - 2.661$$

By using the regression line to extrapolate the breaking time for larger values of N, we estimate the breaking time for a large number N as shown in Table 6.2.

| N | T(seconds) | T(MIPS-years) |
|---|---|---|
| 640 | $5.03*10^{19}$ | $3.19*10^{14}$ |
| 768 | $7.46*10^{23}$ | $4.73*10^{18}$ |
| 1152 | $2.44*10^{36}$ | $1.55*10^{31}$ |

Table 6.2 An estimation of breaking time for a large N.

## 6.2 The Zero-forced Lattice Attack

Alexander May has given an improved method for searching small vectors that have a comparatively large number of coordinates equal to 0 [MAY99]. These ideas lead to the notion of zero-forced lattices, in which one guesses that r particular coordinates of the target are 0, and forces them to be zero, thereby reducing the dimension of the lattice.

An NTRU private key contains two vectors:

$$f = (f_0, f_1, \ldots, f_{N-1}) \quad \text{and} \quad g = (g_0, g_1, \ldots, g_{N-1}).$$

The standard NTRU lattice $L^{NT}$ is the lattice of dimension 2N generated by the row vectors of a matrix with the following form, where $(h_0, \ldots, h_{N-1})$ is a known list of integers.

$$L^{NT} = \begin{pmatrix} \lambda & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{N-1} \\ 0 & \lambda & \cdots & 0 & h_1 & h_2 & \cdots & h_0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda & h_{N-1} & h_0 & \cdots & h_{N-2} \\ 0 & 0 & \cdots & 0 & q & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & q \end{pmatrix}$$

The constant λ is a balancing constant chosen to maximize the efficiency of the search for small vectors in the lattice. The attacker recognizes the lattice contains the relatively short vector v as follows

$$v = ( \lambda f_0, \lambda f_1, \ldots, \lambda f_{N-1}, g_0, g_1, \ldots, g_{N-1} ).$$

From the cyclical nature of the h portion of the matrix, $L^{NT}$ also contains the vectors obtained by shifting the *f* and g coordinates of v cyclically an equal amount. We can denote these shifted vectors by:

$$v^{(k)} = ( \lambda f_k, \lambda f_{k+1}, \ldots, \lambda f_{k-1}, g_k, g_{k+1}, \ldots, g_{k-1} ).$$

So $v = v^{(0)}$, and $L^{NT}$ contains all of the vectors $v^{(0)}, \ldots, v^{(N-1)}$. The attacker knows the h vector that is the NTRU public key. So he/she knows the lattice $L^{NT}$, and his/her goal is to recover the unknown vectors f and g from the lattice $L^{NT}$ or some similar lattice. According to [HPS98], the extensive experimental results show that the log time needed to find a target vector grows at least linearly in the dimension. This means we can get

$$\log(T) \geq A*N + B$$

for certain constants A and B.

Alexander May pointed out in [MAY99] that there is a very efficient method in using the lattice reduction algorithm to find the target vectors from a smaller dimensional lattice in which the specified coordinates are forced to equal to zero.

The first step in forming a Zero-Forced Lattice is to choose a set of indices.

$$J = \{ j_1, j_2, \ldots, j_r \} \text{ satisfying } 0 \leq j_1 < \ldots < j_r < N.$$

Then find the vector whose $j_{1st}, j_{2nd}, \ldots, j_{rth}$ coordinates are equal to zero. In order to do this, we need to find the original congruences that were used to form the $L^{NT}$ lattice:

$$f_0 h_j + f_1 h_{j+1} + \ldots + f_{N-1} h_{j-1} \equiv g_j \pmod{q}, \; 0 \leq j < N.$$

If $g_{j1}, \ldots, g_{jr}$ are equal to 0 in above formula, we can say r has linear relations modulo q for the $f_i$'s. Then we can very easily solve these r congruences for $f_{N-r}, \ldots, f_{N-1}$ in terms of $f_0, \ldots, f_{N-r-1}$ and substitute back into the remaining N-r congruences to get a new system of congruences

$$a_{0j} f_0 + a_{1j} f_1 + \ldots + a_{N-1-r,j} f_{N-1-r} \equiv g_j \pmod{q}, \; 0 \leq j < N, j \notin J.$$

In the above formula aij's are known quantities and the $f_k$'s and $g_j$'s are the unknown quantities. Notice that we now have a system of only N-r congruences in 2(N-r)

unknowns, so we define the Zero-Forced Lattice $L_J^{ZF}$ to be the lattice of dimension 2(N-r) spanned by the rows of the following matrix:

$$L_J^{ZF} = \begin{pmatrix} 1 & 0 & \cdots & 0 & & & & \\ 0 & 1 & \cdots & 0 & & & & \\ \vdots & \vdots & \ddots & \vdots & & a_{ij} & & \\ 0 & 0 & \cdots & 1 & & & & \\ 0 & 0 & \cdots & 0 & q & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & q \end{pmatrix}$$

The ij indices in the upper right hand part of the matrix over the range $0 \leq i < N - r$ and $0 \leq j < N$ with $j \notin J$.

To describe the target vectors in $L_J^{ZF}$, we let

$$v_J^{(k)} = (f_{i+k})_{0 \leq i < N-r} \,\|\, (g_{j+k})_{j \notin J}$$

where || indicates concatenation of the two vectors. In other words, $v_J^{(k)}$ is equal to $v^{(k)}$ with the last r coordinates removed from its f-part, and with the J-coordinates removed from the g-part. Note that $v_J^{(k)}$ will be in the lattice $L_J^{ZF}$ if, and only if, the J-coordinates of $v^{(k)}$ are all zero, because we have assumed that they were all zero in creating $L_J^{ZF}$. The fact that we have reduced the dimension of the search lattice will certainly speed up the search using the LLL algorithm or a similar lattice reduction

algorithm. On the other hand, the lattice $L_J^{ZF}$ won't contain any target vectors at all unless we have chosen a J for which some $v^{(k)}$ has all of its J-coordinates equal to zero. Of course, if r is large, it will take a lot of tries before one reaches a correct guess. More details about how the zero-forced lattices works, and how to estimate their effectiveness, is explained in [MAY99].

### 6.3 Passive attack from signature's interception

Ilya Mironov showed a passive attack for private key recovery only from the signature interception [MIR00].

In the NSS, a valid signature is [HS00]:

$$s = f*w \pmod q.$$

Its k[th] coefficient is

$$s[k] = \sum_{i=0}^{N-1} f[i]w[k-i] = \sum_{i=0}^{N-1} (f_1[i]w_1[k-i] + q_f f_2[i]w_1[k-i] + q_w f_1[i]w_2[k-i]) (\mathrm{mod}\, q).$$

The correlation coefficient is at least [MIR00]:

$$\rho(s[k], wl[k-ai]) \geq \frac{q_f / 4}{\sqrt{(\frac{1}{4}(K(q_f+1)^2 + (N-K)) + \frac{1}{4} K q_w^2)\frac{1}{4}}} \geq 0.42.$$

It is turned out that the correlation is very high. This means that when s[k] is large, $w1[k-a_i]$ is more likely to be one. If s[k] is small, then $w1[k-a_i]$ is zero with probability more than ½.

Because:

$$s \equiv f_1 * w_1 + q_f f_1 * w_1 + q_w f_1 * w_2 \pmod{q}$$

and the correlation between coefficients $s[k+a_i]$ and $w_1[k]$ is very high, the value of $w_1[k]$ can be guessed by observing $s[k+a_i], \ldots, s[k+a_k]$. When $w_1$ is known, the more subtle correlation between coefficients $f_1$ and s can be detected. Experimental results demonstrated that the NSS is insecure [MIR00].

# Chapter 7

# Conclusions and Future Work

## 7.1 Discussions and Conclusions

This research work has investigated the lattice based PKCS from key creations, encryption rate, decryption rate, signing process and verifying process. We found that the NTRU cryptosystem is considerably faster than ECC and RSA. We strongly believe that after Texas Instrument (TI) Co., Ltd. adopted the lattice based PKCS NTRU into its wireless devices, NTRU will be widely adopted in near future by the varieties applications.

From the investigative results, we notice that the claim "NTRU is 100 times faster than any competitor" could be replaced by more accurate performance description. Its key generation speed is about 4 to 7 times faster than the ECC PKCS. Its encryption speed varies from 27 to 44 times faster than ECC and the decryption speed varies from 6 to 10 times faster than ECC respectively depending on which security level is chosen. In Chapter 3, we demonstrate all the key creation rates, encryption rates, and decryption rates, as compared with ECC PKCS.

Because the lattice based PKCS generates the private key very fast, so could every encryption process use different session keys, and all previous keys needn't store in the machine. Disposable keys are generated so rapidly and efficiently that a new key can be produced for all or specific parts of a transaction. The mobile devices are limited in speed, processing power and storage space, which constrains the manipulation, storage, encryption and decryption of keys. Thus, the lattice based disposable key function makes the key management much easier.

Another very important advantage of the lattice based PKCS is that it needs a very low memory space. For example, when applying the NTRU107, the required memory is no more than 3K. This means that we could apply the lattice based PKCS on almost any embedded system. From our simulation results of the NTRU107 and NTRU167 for the PDAs, we notice that the lattice based digital signature scheme could be adopted directly on the current PDAs at application level with a high operation rate and sufficient security. This is a prompt remedy to remove the security hole in the WAP gateway.

The application in Chapter 5 shows the lattice based PKCS could be easily applied on "paperless" report/billing system with any popular DBMS. It could sign the document "on the fly." By using "disposable key," the private key management becomes very easy. This is very suitable in "heavy billing" companies such as the telephone company, gas company, or banks.

In short, the conclusions of this investigation are that the lattice based PKCS is a faster PKCS, and could replace RSA and ECC in many situations. It provides fast key

generation, allowing disposable keys. It offers more security options, and could easily scale to low-cost embedded devices.

### 7.2 Future Work

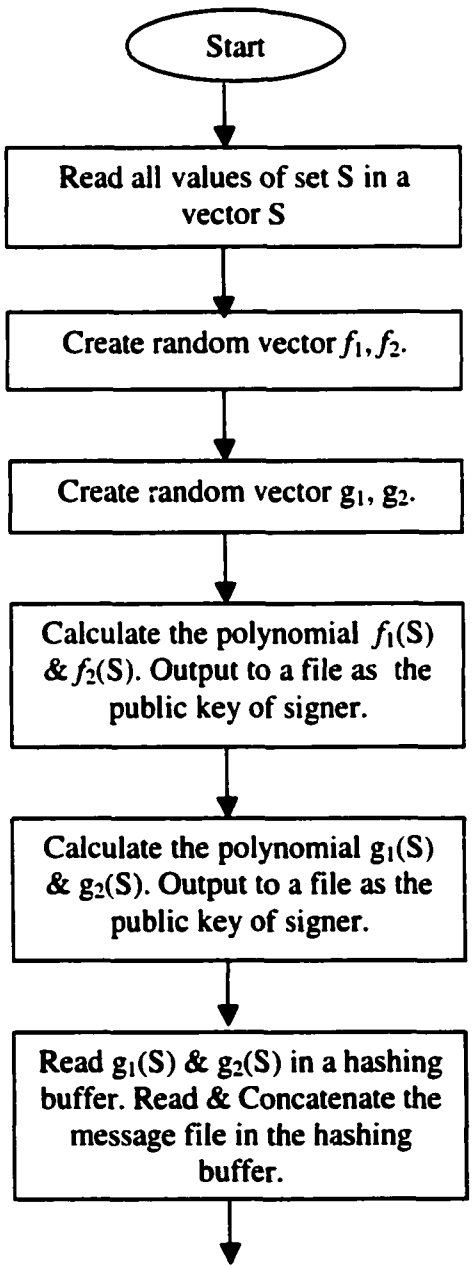
For future study, our research will focus on Business-to-Customer online billing systems that are based on DBMS and need heavy encryption and authentication. It may greatly reduce the cost of processing paper-based purchase orders and billings, including the time and staff involved to raise and approve a requisition. We will apply the lattice based digital signature scheme as a XML function for every transaction at application level security to enhance the security and authentication for wireless and embedded devices.

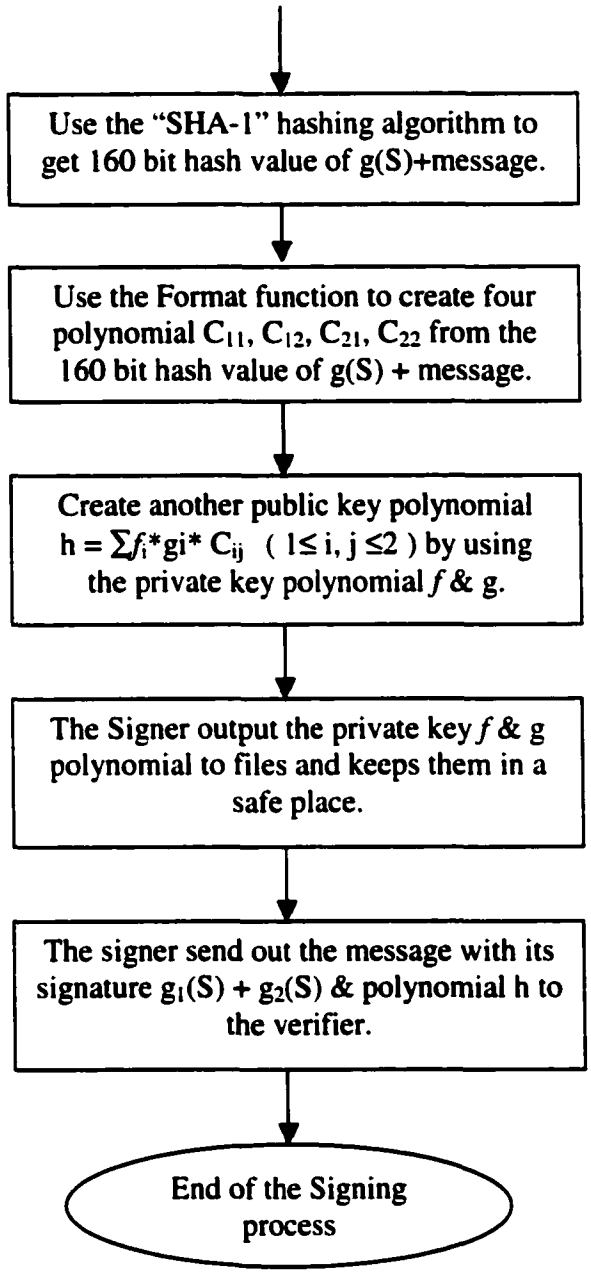
A lot of operations could shift to hardware to greatly improve the efficiency of the lattice based digital signature scheme. I'll explore the feasibility to design the special chipset for lattice based PKCS for wireless and embedded system like smartcard, cellular phone and PDAs in near future.

# Appendix A

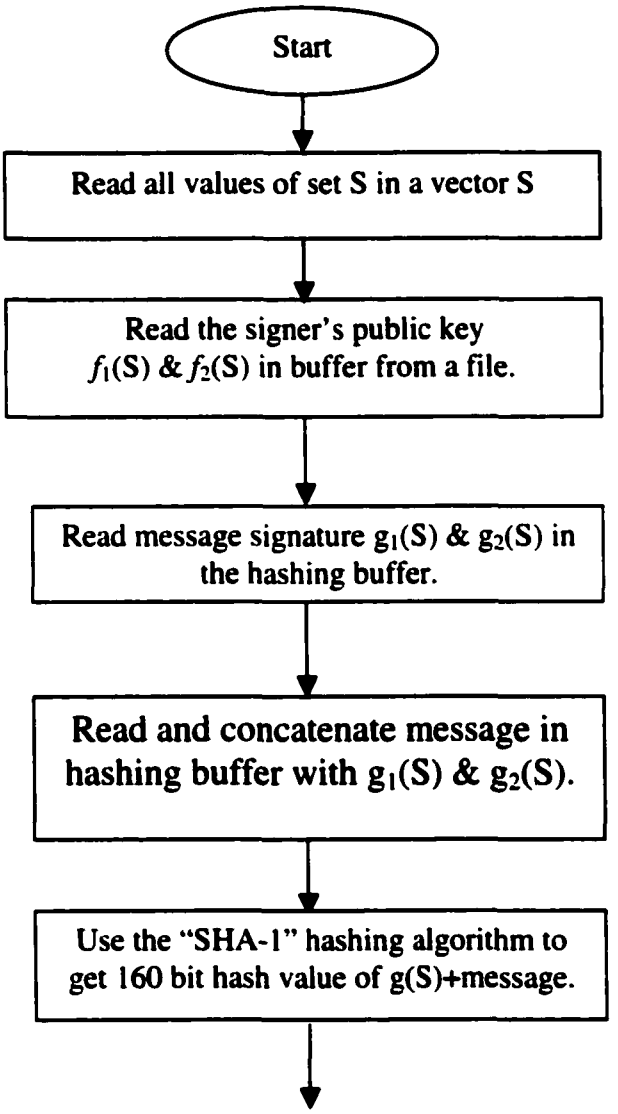
## The Digital Signature Signing and Verifying Process

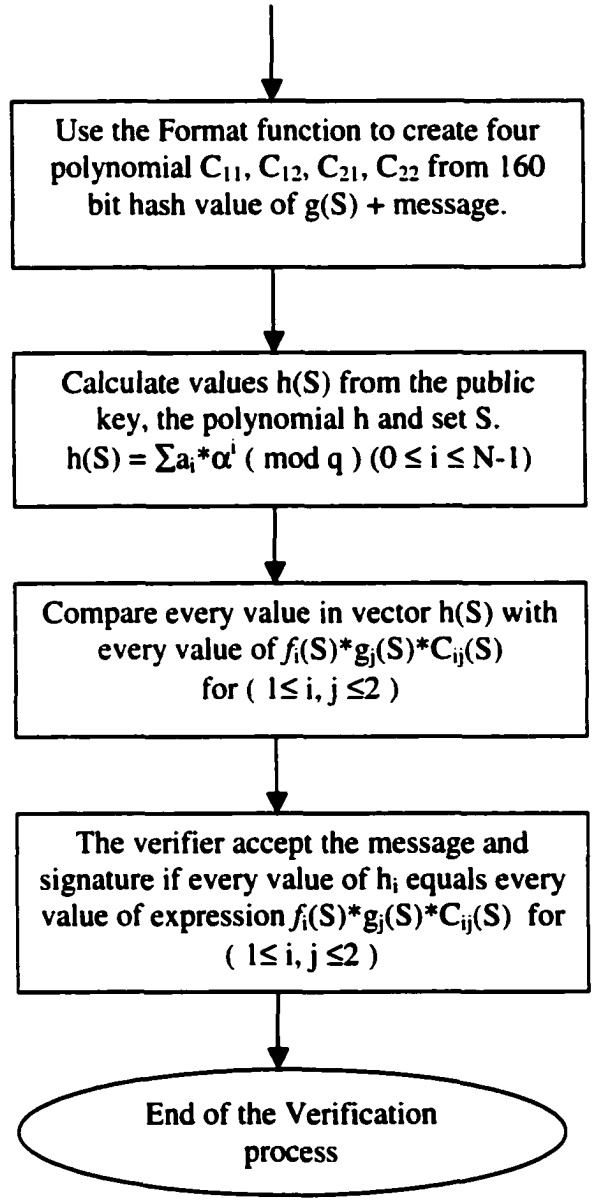
### A.1 The Signing Process for the Lattice Based PKCS





## A.2 The Verifying Process of the Lattice Based PKCS





# Appendix B

## Chinese Remainder Theorem

**B.1 Theorem CRT** (Chinese Remainder Theorem) Let $m, n \in \mathbb{N}$ be coprime. Then we have the isomorphism of rings:

$$\mathbb{Z}/mn\mathbb{Z} \cong \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}.$$

Restricting to units on both sides, we have the isomorphism of groups:

$$U_{mn} \cong U_m \times U_n.$$

Map $\mathbb{Z} \to \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$ by $c \to (c \bmod m, c \bmod n)$. This is a ring homomorphism, which is surjective by the previous Proposition, and has kernel $m\mathbb{Z} \cap n\mathbb{Z} = mn\mathbb{Z}$ (the last equality because gcd (m, n) = 1). The first result follows by the Isomorphism Theorem for ring homomorphisms. In the correspondence $(a, b) \leftrightarrow c$ we have $a \equiv c \pmod m$ and $b \equiv c \pmod n$, so gcd (c, mn) = 1 <==> gcd (c, m) = gcd (c, n) = 1 <=> gcd (a, m) = gcd (b, m) = 1, which gives the last bijection. More over, from the ring isomorphism we get an isomorphism of the groups of units, so $U_{mn} = U(\mathbb{Z}/mn\mathbb{Z}) \cong U(\mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}) \cong U(\mathbb{Z}/m\mathbb{Z}) \times U(\mathbb{Z}/n\mathbb{Z}) = U_m \times U_n$.

Both forms of the CRT extend to several moduli $m_1, m_2, \cdots, m_k$ provided that they are pairwise coprime.

The second part of the proposition has the following important corollary: φ is a **multiplicative function.**

**Proposition B.1** Let $m, n \in \mathbb{N}$ be coprime. Then $\varphi(mn) = \varphi(m)\,\varphi(n)$.

$$\varphi(mn) = |U_{mn}| = |U_m \times U_n| = |U_m| \times |U_n| = \varphi(m)\,\varphi(n).$$

**Corollary B.1** Let $m \in \mathbb{N}$ have prime factorization:

$$m = \prod_{i=1}^{k} p_i^{e_i}$$

where the $p_i$ are distinct primes and $e_i \geq 1$. Then:

$$\varphi(m) = \prod_{i=1}^{k} p_i^{e_i - 1}(p_i - 1) = m\prod_{i=1}^{k}\left(1 - \frac{1}{p_i}\right).$$

We can show the above by multiplicativity:

$$\varphi(m) = \prod_{i=1}^{k} \varphi(p_i^{e_i}),$$

and

$$\varphi(p_i^{e_i}) = p_i^{e_i - 1}(p_i - 1)\ .$$

The last part is just a rearrangement of the product; it has merit in that the exponents of the prime divisors of m do not appear explicitly.

**Examples:** (1). $\varphi(168) = \varphi(8)\,\varphi(3)\,\varphi(7)$ (splitting 168 into prime powers) = (8-4)(3-1)(7-1) = 4 x 2 x 6 = 48. Alternatively, ). $\varphi(168) = 168 - (1 - ½)(1 - 1/3)(1-1/7) = 168$ x ½ x 2/3 x 6/7 = 48.

(2). $\varphi(100) = \varphi(4)\,\varphi(25) = 2 \times 20 = 40$.

One more property of $\varphi(m)$ will be useful later.

**Proposition B.2** Let $m \in$ N. Then $\sum_{d \mid m} \varphi(d) = m$.

The sum here is over all positive divisors of $m$. For example, when $m$=12 we have:

$$12 = \varphi(1) + \varphi(2) + \varphi(3) + \varphi(4) + \varphi(6) + \varphi(12)$$
$$= 1 + 1 + 2 + 2 + 2 + 4.$$

To show why it works, consider the $m$ fractions $k/m$ for $0 \le k \le m$-1. Reduced to lowest terms they become a/d where $d|m$, $0 \le a \le d$-1, and gcd $(a, d) = 1$. So there are $\varphi(d)$ fractions with denominator $d$ for each divisor $d$ of $m$, giving the total as stated.

**B.2 Applications of CRT:** The CRT says that congruences to two coprime moduli are, in a sense, independent. Solving a general congruence to a general modulus reduces it to solving modulo prime powers, and then using CRT to "glue" the separate solutions together.

For example: solve $x^2 \equiv 1 (\text{mod } 91)$. Since 91 = 7 x 13, we first solve modulo 7 and modulo 13 separately, giving $x \equiv \pm 1 (\text{mod } 7)$ and $x \equiv \pm 1 (\text{mod } 13)$ by an earlier proposition, since 7 and 13 are prime. This gives four possibilities for modulo 91:

( +1 mod 7, + 1 mod 13 ) <-> ( +1 mod 91 )

( +1 mod 7, - 1 mod 13 ) <-> ( -27 mod 91 )

( -1 mod 7, + 1 mod 13 ) <-> ( +27 mod 91 )

( -1 mod 7, - 1 mod 13 ) <-> ( - 1 mod 91 )

So the solutions are $x \equiv \pm 1 (\text{mod } 91)$ and $x \equiv \pm 1 (\text{mod } 91)$. To solve the second and third we use the method given above: write $1 = 7a + 13b = 14 - 13$, then (1, -1) maps to $1(-13) - 1(14) \equiv -27 (\text{mod } 91)$.

Systematic study of various types of congruences now follows the following pattern.

First work modulo primes; this is easiest, since Z/pZ is a field. Then, somehow go from primes to prime powers. The process here is rather like taking successive decimal approximations to an ordinary equation, and we will come back to this at the end of the module. Finally, use the CRT to glue together the information from the separate prime powers.

# BIBLIOGRAPHY


[AKL83] S. Akl, *Digital Signatures: A Tutorial Survey.* Computer, February 1983.

[AMM97] L. Ammeraal, *STL for C++ Programmers*, New York, NY: John Wiley. 1997.

[ANS99] M. Anshel, *Constructing Public Key Cryptosystems Via Combinatorial Group Theory*, Mathmatic Research Letter 6, 287-291, November 5, 1999.

[BD92] F.Bauspiess and F. Damm, *Requirements for Cryptographic hash functions*, Computers & Security, v. 11, n. 5, Sep 1992, pp. 427-437.

[BUD97] T. Budd, *Data Structures in C++ using STL*, Addison-Wesley, 1997.

[CAM98] P. J. Cameron, *Introduction to algebra*, Oxford; New York: Oxford University Press, 1998.

[CFV86] G. D. Crown, Maureen H. Fenrick, Robert J. Valenza, *Abstract algebra*. New York: M. Dekker, c1986.

[COH93] H. Cohen, *A Course in Computational Algebraic Number Theory*, GTM 138,Springer-Verlag, Berlin, 1993.

[COH00] H. Cohen, *Advanced topics in computational number theory*, New York: Springer, c2000.

[CS97] D. Coppersmith, A. Shamir, *Lattice attacks on NTRU*, Eurocrypt 97

[CLR90] T. H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, MA, 1990.

[DAM90] I. B. Damgard, *A design principle for hash functions*, Advances in Cryptology-CRYPTO'88 Proceedings, Springer-Verlag, 1990, pp. 416-427.

[DD99] H. Deitel & P. Deitel, *Java How to Program*, 3rd Edition. Prentice Hall 1999.

[DD98] H. Deitel and P. Deitel, *C++ How to Program*, 2nd Edition, Prentice Hall. 1998.

[DH76A] W. Diffie and M.E. Hellman, *Multiuser cryptographic techniques*. Proceedings of AFIPS National Computer Conference, 109-112, 1976.

[DH76B] W. Diffie, M.E. Hellman, *New direction in cryptography*, IEEE Trans. On Information Theory 22(1976), 644-654.

[ECK95] B. Eckel, *Thinking in C++*, Prentice Hall, 1995.

[ELG85] T. ElGamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory, 31 (1985). 469-472.

[FIPS94] Federal Information Processing Standards (FIPS) Publication 186, *Digital signature standard*, U.S. Dept. of Commerce/N.I.S.T., National Technical Information Service, Springfield, Virginia, 1994.

[GGH96] O. Goldreich, S. Goldwasser, S. Halevi, *Public-key cryptosystems from lattice reduction problems*, MIT-Laboratory for Computer Science preprint, November 1996.

[HLS99] J. Hoffstein, D. Lieman, J.H. Silverman, *Polynomial Rings and Efficient Public Key Authentication*, CtypTEC'99, Hong Kong, City University of Hong Kong Press, 1999.

[HPS98] J. Hoffstein, J. Pipher, J.H. Silverman, *NTRU: A new high speed public key cryptosystem*, Lecture Notes in Computer Science 1423. Springer-Verlag, Berlin, 1998, 267-288.

[HPS00A] J. Hoffstein, J. Pipher, J.H. Silverman, *NTRU: A ring-based public key cryptosystem*, U.S. Patent#6081597. Jun. 27, 2000.

[HS00] J. Hoffstein, J.H. Silverman, *NSS: The NTRU signature scheme*. Preliminary version, preprint, August, 2000.

[HPS00B] J. Hoffstein, J. Pipher, J.H. Silverman, *NSS: The NTRU signature scheme, preprint, 2000.*

[HC99] C. S. Horstmann, Gary Cornell, *Core Java 2*, Prentice Hall. 1999.

[JUE87] R.R. Jueneman, *A high speed manipulation detection code*, Advances in Cryptology-CRYPTO'86 Proceedings, Springer-Verlag, 1987. pp. 327-346.

[JMM85] R. R. Jueneman, S.M. Matyas, and C.H. Meyer, *Message Authentication*. IEEE Communications Magazine, v. 23, n. 9, Sep 1985, pp.29-40.

[KAL92] B. Kaliski, *The MD2 message digest algorithm*, Internet Request for Comments 1319, April 1992.

[KOB94] N. Koblitz, *A course in Number Theory and Cryptography*. New York: Springer-Verlag, 1994.

[KOB99] N. Koblitz, *Algebraic aspects of cryptography*, Berlin; New York: Springer, c1998.

[KRA93] D. W. Kravitz, *Digital signature algorithm*, U.S. Patent#5,231,668, 27 Jul 1993.

[KRO98] R. Kumanduri, C. Romero, *Number Theory with Computer Applications*. Prentice Hall, 1998.

[LEV90] W. Leveque, *Elementary Theory of Numbers*. New York: Dover, 1990.

[LLL82] A. K. Lenstra, H.W. Lenstra, L. Lovsz, *Factoring polynomials with polynomial coefficients*, Math. Annalen 261(1982), 515-534.

[LLP98] S. B. Lippman, Josee Lajoie, *C++ Primer*, 3rd Edition, Addison-Wesley, 1998.

[LIP96] S. B. Lippman, *Inside the C++ Object Model*, Addison-Wesley, 1996.

[LP98] R. Lidl, Gunter Pilz, *Applied abstract algebra*, 2nd Edition, New York: Springer, c1998.

[MAMO85] S.M. Matyas, C.H. Meyer, and J. Oseas, *Generating Strong One-Way Functions*, IBM Technical Disclosure Bulletin, v. 27, n. 10A, Mar 1985, pp. 5658-5659.

[MAY99] A. May, *Cryptanalysis of NTRU*, preprint, February 1999.

[MB77] N. H. McCoy, Thomas R. Berger, *Algebra: groups, rings, and other topics*, Boston: Allyn and Bacon, c1977.

[MER78] R. C. Merkle, *Secure communications over insecure channels*. Communications of the ACM, 21(1978), 294-299.

[MER79] R. C. Merkle, *Secrecy, Authentication, and Public Key Systems*, UMI Research Press, Ann Arbor, Michigan, 1979.

[MER80] R. C. Merkle, *Protocols fro public key cryptosystems*, Proceedings of the 1980 IEEE Symposium on Security and Privacy, 122-134, 1980.

[MER90] R. C. Merkle, *One way hash functions and DES*, Advances in Cryptology-CRYPTO'89 Proceedings, Springer-Verlag, 1990, pp. 428-446.

[MEY92] S. Meyers, *Effective C++: 50 Specific Ways to Improve Your Programs and Designs*, Reading, MA: Addison-Wesley, 1992.

[MEY95] S. Meyers, *More Effective C++: 35 New Ways to Improve Your Programs and Designs*, Reading, MA: Addison-Wesley, 1995.

[MIR00] I. Mironov, *A Note on Cryptanalysis of the Preliminary Version of the NTRU Signature Scheme*, preprint, 2000.

[MPW92] C. Mitchell, F. Piper, and P. Wild, *Digital signatures*. Contemporary Cryptology: The Science of Information Integrity, 325-378, IEEE Press. 1992.

[NY89] M. Naor and M. Yung, *Universal one-way hash functions and their cryptographic application*, Proceedings of the 21st Annual ACM Symposium on the Theory of Computing, 1989, pp. 33-43.

[ORE76] O. Ore, *Number Theory and Its History*. New York: Dover 1976.

[POH97A] I. Pohl, *C++ Distilled: A Concise ANSI/ISO Reference and Style Guide*. Reading, MA: Addison-Wesley, 1997.

[POH97B] I. Pohl, *Object Oriented Programming Using C++*, Second Edition, Reading, MA:Addison-Wesley, 1997.

[PRE94] B. Preneel, *Cryptography hash functions*, European Transactions on Telecommunications, v 5, n. 4, Jul/Aug 1994, pp. 431-448.

[PS01] P. Q. Nguyen and Jacques Stern, *Preliminary Research Announcement: Cryptanalysis Of The NTRU Signature Scheme (NSS)*, Preprint, 2001. (http://www.di.ens.fr/~stern/nss.html)

[RAB78] M. O. Rabin, *Digitalized signature*, M.O. Rabin, Foundations of Secure Computation, 155-168, Academic Press, 1978.

[RAB79] M. O. Rabin, *Digitalized signatures and public key functions as intractable as factorization*, MIT/LCS/TR-212, MIT Laboratory for Computer Science, 1979.

[RIV92A] R. L. Rivest, *The MD4 message digest algorithm*, Internet Request for Comments 1321, April 1992.

[RIV92B] R. L. Rivest, *The MD5 message digest algorithm*, Internet Request for Comments 1321, April 1992.

[RIV91] R. L. Rivest, *The MD4 message digest algorithm*, Advances in Cryptology - CRYPTO'90(LNCS 537), 303-311, 1991.

[RIVE78] R. L. Rivest, *A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, 21(1978), 120-126.

[ROS93] K. Rosen, *Elementary Number Theory and its Applications*. Reading, MA: Addison-Wesley, 1993.

[SAL96] A. Salomaa, *Public key Cryptography*. New York: Springer-Verlag, 1996.

[SCHE91] B. Schneier, *One way hash functions*, Dr. Dobb's Journal, v. 16, n. 9, Sep 1991, pp. 148-151.

[SCHE96] B. Schneier, *Applied Cryptography*, 2nd Edition, John Wiley & Sons, Inc. 1996.

[SCHO91] C. Schnorr, *Efficient Signature for Smart Card.* Journal of Cryptology, No. 3, 1991.

[SE91] C. P. Schnorr, M. Euchner, *Proc. Fundamentals of computation theory*, Lecture Notes in Computer Science 529, p68-85, 1991.

[SCHO94] C. P. Schnorr, M. Euchner, *Lattice basis reduction: improved practical algorithms and solving subset sum problems*, Mathematical Programming 66(1994), 181-199.

[SED92] R. Sedgwick, *Algorithms in C++*, Reading, MA:Addison-Wesley, 1992.

[SIL97] J. H. Silverman, *A Friendly Introduction to Number Theory*, Prentice Hall,New Jersey, 1997.

[SIL99] J. H. Silverman, *Dimension-Reduced Lattices, Zero-Forced Lattices, and the NTRU Public Key Cryptosystem*, NTRU Technical Note 013, March 2, 1999.

[SMI97] R. Smith, *Internet Cryptography*. Reading, MA: Addison-Wesley, 1997.

[SOOS95] Schroeppel, Orman, O'Malley, and Sptascheck, *Fast key exchange with elliptic curve systems*, Advances in Cryptology-CRYPTO'95, Lecture

Notes in Computer Science 973, D. Coppersmith, ed., Springer-Verlag, New York, 1995, 43-56.

[STA98] W. Stallings, *Cryptography and Network Secority Principles and Practice*, Second Edition, Prentice Hall, 1998.

[STI95] D. R. Stinson, *Cryptography: Theory and Practice*, CRC Press, Boca Raton, Florida, 1995.

[STR91] B. Stroustrup, *The C++ Programming Language*, Second Edition, Addison-Wesley, 1991.

[VOM96] S. Vanstone, P. Van Oorschot, A. Menezes, *Handbook of Cryptography*, CRC Press, Boca Raton, 1996.

[WAL98] Wallace, D. A. R., *Groups, rings, and fields*, London; New York: Springer, c1998.

[ZPS93] Y. Zheng, J. Pieprzyk, and J. Seberry, *HAVAL - A one way hashing algorithm with variable length of output*, Advances in Crytology - AUSCRYPT'92 Proceedings, Springer-Verlag, 1993, pp. 83-104.