

The NTRU Cryptosystem: Implementation and Comparative Analysis

Semester Project

ECE 646, Fall 2001
Professor Kris Gaj
George Mason University

Author:
Rodney D'Souza

Abstract

The NTRU cryptosystem is a relatively new public key cryptographic algorithm that was first introduced in 1996. The main advantage of this algorithm is that it runs much faster than conventional public key algorithms such as RSA. This speed advantage is especially large in key generation, which is often the most important part of public key cryptography. This paper introduces the ideas behind NTRU and presents results from a implementation in Java. The results include a comparison with RSA in terms of key generation, encryption and decryption speeds.

1. Introduction

Public key (PK) cryptography, also known as asymmetric cryptography, is more often used today in the areas of key exchange and digital signatures than in encryption and decryption of large amounts of data. Perhaps the main reason why today's PK systems such as RSA are not used for bulk encryption/decryption is that they are orders of magnitude slower than symmetric key systems. Nonetheless, key exchange and digital signature are applications that are very important and are currently best performed by a Public Key Infrastructure (PKI).

Since the introduction of RSA in 1977 it has been the only widely used public key encryption algorithm. In the mid-to-late 1980s, Elliptic Curve Cryptography (ECC) was introduced and has since been slowly gaining acceptance. It has recently also begun to be standardized. Other algorithms, such as ElGamel and braid systems have also enjoyed limited success. The subject of this paper, the NTRU algorithm, is a proprietary algorithm patented by NTRU Cryptosystems which claims to dramatically reduce one of the big problems in PK cryptography, namely speed. For this project, the algorithm was implemented in Java and compared to RSA on the basis of speed of three operations: key generation, encryption and decryption.

The rest of this paper is organized as follows: Section 2 introduces the concepts behind the NTRU system and briefly describes the key generation, encryption and decryption methodologies. Section 3 describes the author's software implementation of the NTRU algorithm for this project. Section 4 compares the NTRU implementation to the RSA algorithm on the basis of speed and security. Section 5 draws some conclusions about NTRU from the results of the implementation and gives some suggestions of how this project could be expanded upon for further analysis.

2. The NTRU Algorithm

2.1 Basic Algorithm

The NTRU algorithm is based on embedding messages in a polynomial ring, R . The ring is defined such that the multiplication operation of the polynomials is wrapped around the degree ("size") of the polynomial rather than expanding the degree of the polynomial. Further, each coefficient of the polynomial is an integer that is reduced modulo certain parameters, after every math operation. The notation for the ring is given as :

$$R = \mathbb{Z}[X]/(X^N - 1)$$

where \mathbb{Z} represents the set of integers and N is 1 more than the degree of the polynomial.

A full mathematical explanation is beyond the scope of this paper and the reader is referred to the literature for an in depth analysis of NTRU cryptography.

NTRU Cryptosystems has published several versions of the NTRU algorithm, of which two basic algorithms were implemented for this project. A detailed explanation of the basic algorithm, including an illustrative example and a proof of why it works is given in [1]. Next a brief explanation of the algorithm is presented.

Key creation is as follows: Bob chooses 2 random polynomials f and g that are in the defined ring R and that whose inverses exist in the ring modulo key parameters p and q . These inverses are denoted F_p and F_q respectively, and need to be computed for the chosen f . If the inverse of either does not exist another f is chosen and the process is repeated. Next Bob generates the public key as the polynomial h :

$$h = F_q * g \text{ mod } q$$

where g is also a randomly chosen polynomial in R . Note that f and g are “randomly” chosen in R but have a specific amount of coefficients that are 0, +1 and -1. The private key is the polynomial f along with the inverses F_p and F_q . N , p , and q are considered parameters of the algorithm rather than part of either key.

Encryption is as follows: Alice, wishing to send a binary message m to Bob, begins by randomly choosing a polynomial r that is in R . She then performs the encryption by computing

$$e = pr * h + m \text{ mod } q.$$

Note that scalar multiplication of integer p with polynomial r is simply multiplication of each coefficient of r with p . Multiplication of r with h is wrapped around the ring size, as stated above.

Decryption is as follows: Bob begins to decrypt the message e by first computing

$$a = f * e \text{ mod } q$$

and then reducing the coefficients of a to be between $-q/2$ to $q/2$. Then he obtains the message m by computing

$$m = a * F_p,$$

where F_p is part of his private key and is the multiplicative inverse of $f \text{ mod } p$, as derived in the key generation.

N	The polynomials in the ring R have degree $N-1$. (Non-secret)
q	The large modulus to which each coefficient is reduced. (Non-secret)
p	The small modulus to which each coefficient is reduced. (Non-secret)
f	A polynomial that is the private key.
g	A polynomial that is used to generate the public key h from f (Secret but discarded after initial use)
h	The public key, also a polynomial
r	The random “blinding” polynomial (Secret but discarded after initial use)
d_f	f has d_f coefficients equal to 1 and d_f-1 coefficients equal to -1
d_g	g has d_g coefficients equal to 1 and d_g equal to -1
d_r	r has d_r coefficients equal to 1 and d_r equal to -1

Table 1. NTRU keys and parameters

Table 1 lists the parameters of the basic algorithm along with a brief explanation of each. Table 2 lists parameter values for various levels of security as reported by NTRU Cryptosystems. The NTRU algorithm is actually probabilistic in nature; i.e. there

is a small chance of decryption failure. With the appropriate choice of parameters the decryption probability can be made to be on the order of 10^{-25} or less. The values listed in Table 2 yield such small decryption failure rates. These parameters are purported to be used in NTRU Cryptosystems commercial applications. A more detailed explanation of the security levels listed in the table is given in Section 4.

	N	q	p	df	dg	dr
Low Security	107	64	3	15	12	5
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 2 - NTRU parameters for various levels of security

2.2 Key Size

It is sometimes desirable to express the public and private key sizes in bits. The formula for key size is simply the number of bits required to store the coefficient for each term and for each polynomial in the key, multiplied by the number of terms in the polynomial. For example, the public key size for $N=167$ and $q=128$ is $167 * \log_2 128 = 1169$ bits. The private key usually involves storing both f and F_p and therefore is twice as large. However, the faster key generation variation of NTRU does not require the storage of and thus the private key size is the same as the public key size.

2.3 NTRU Variations

There are several variations on the theme of NTRU, but all essentially follow the basic algorithm as described above. The variations primarily involve optimization techniques for speed, message mapping to the ring, and minor security issues. The reader is referred to [3][4][6] for a discussion of these and other issues. It is somewhat unclear as to which version of NTRU has received the most scrutiny by the cryptographic community and only time will tell as to which variations are more secure than others.

A variation of the basic algorithm as discussed above involves choosing the private key polynomial f . If a small polynomial F is chosen first as a random polynomial, then f can be computed simply as $f = F + 1 \pmod p$. This is a simple addition of $1 \pmod p$, which is significantly less time consuming than computing F from f using an inverse polynomial calculation. In addition, it saves the second multiplication during decryption since the inverse of $f \pmod p$ is exactly 1 (the multiplicative identity), thereby reducing the decryption time by about half. This basic variation is included in the implementation for this project.

2.4 Message Encoding and Padding

A brief description of the possible mapping of messages into the polynomial ring is instructive and provided here. As described in [13], with $p = 3$, each coefficient of the polynomial can be coded as $-1, 0,$ or 1 and therefore a 416 bit message can be coded into ring with $N=263$. A ternary coding of messages is not natural for most software uses and therefore a binary mapping of $\{0,1\}$, though less efficient, was used in the implementation presented in this paper. A variation of NTRU which takes p as the

polynomial $p = 2+X$, instead of integer $p=3$ also uses binary coding instead of trinary coding.

As an example of message encoding, let a message m be equal to 10101101_2 with $N = 8$. The polynomial ring representation of m would be

$$1 + X^2 + X^4 + X^5 + X^6 + X^8$$

Padding of messages is recommended for security just as in RSA and therefore OAEP type padding is currently recommended. The implementation for this paper only provides padding enough to recover the message and does not provide security. However, this can easily be accomplished by extending the code as needed.

2.5 Security Note

NTRU is a proprietary cryptosystem and has not been in existence for a long time. It is therefore unclear as to how secure the algorithm is. Although the basic algorithm remains the same, several variations have emerged over the years and the recommended parameters have also been changing. Currently, it is unclear as to which variations have been analyzed by third parties. Originally, parameters were changed due to cryptanalysis by third parties (e.g. by May [15]), but more recently the reason for the changes is a bit unclear.

For example, originally the values for N for the 3 levels of security as listed in Table 2 were 107, 167, and 503. Around the year 2000, this was changed to 167, 263, and 503 and in December 2001, the values changed to 251, 347 and 503. To a critical reader, the continuous changing of these parameters may suggest an unstable cryptographic system.

3. Project Software Implementation of NTRU

3.1 Implementation Procedures and Algorithm Variations Implemented

For this project the author chose to implement the basic algorithm in Java in order to compare the algorithm to other PK algorithms, specifically RSA. In addition, a slightly faster version of key generation was implemented, which consequently also doubles decryption speed. The encryption and decryption methods were written to accept arrays of binary data (8-bit blocks), which allowed encryption of arbitrary data. To be consistent with other similar java libraries, encryption and decryption of files is included as an example program rather than as part of the set of library functions. This allows use of the library for encryption of files without restricting it from other uses such as encrypting and decrypting of communications link data.

The two variations of the NTRU algorithm included in the implementation represent the basic algorithm and a variation that most significantly increases speed variation. Problems were encountered while implementing the basic algorithm due to a lack of detailed documentation as to how implement the mathematical operations and how to choose parameters such that decryption works with high probability. A similar problem occurred when implementing the faster algorithm involving $f = F + 1 \pmod p$. Here, it was discovered that depending on how F was chosen (this was not documented by NTRU Cryptosystems), the execution trials could yield a high decryption failure rate. In the java implementation, F was chosen so as to yield low decryption failure rate, as observed empirically by testing large files.

3.2 Implementation modules

Table 3 lists the classes in the implementation, which can be included as a library in jar format or as individual classes. NtruKeyGenerator and NtruCipher are the only classes that are directly created by the end user. The remaining classes are used by the aforementioned two classes. The classes were tested using test methods in each class. Additionally, the cipher was tested by reading in binary files, performing encryption/decryption and then testing the output and input files using the system utility 'fc' for any differences. With the parameters given in Table 2 from NTRU Cryptosystems, no decryption failure was encountered using the software to encrypt/decrypt large binary files. Note that test vectors for NTRU are not publicly available at this time, and were therefore not used.

Class/file	Description
Polynomial	Implements the algebra for the polynomial ring R, including multiplication, addition, inverse computation, and modulo
NtruPublicKey	Holds public key h, as well as, parameters p,q, and N
NtruPrivateKey	Holds private key f and F_p , as well as, parameters p,q, and N
NtruKeyGenerator	Generates keys for the four common security levels. Includes two versions of key generation
NtruCipher	Implements the encryption and decryption methods
RandomPolyGenerator	Generates polynomials with specified number of +1 and -1 coefficients
PolynomialByteMapper	Converts binary representations of byte values into the polynomial coefficients and vice versa. Handles padding.

Table 3. Java Implementation modules

3.4 Implementation Design and Optimizations

The main design philosophy in the creation of these classes was to implement well-abstracted classes, as well as, code optimized for speed. Method invocations were avoided where possible by declaring commonly used variables 'public' as opposed to using traditional get/set methods. Object creation and memory allocation was also closely controlled where possible. The code could probably be speeded up slightly by further optimizations but it is likely that these will give varying results depending on the operating system and hardware platform.

The implementation includes some but not all the optimizations for speed as found in the literature. The reason for this lies mostly in the limited amount of time available in a semester time frame. Examples of optimizations that were included are the "Almost Inverse" algorithm for finding the inverse of a polynomial in the ring and the faster key generation and decryption variation of the basic algorithm. An example of an optimization not included is an algorithm for polynomial multiplication that reduces the order of operations from N^2 to $\frac{3}{4}N^2$. Since multiplication takes up a significant portion of

encryption, decryption and key generation times, optimizing the multiplication is worthwhile enhancement if any further work is done with the implementation.

Java has numerous options in programming constructs, compilation and runtime that increase speed, compared to the default, interpreted model that is usually considered. Some of these were mentioned above as programming optimizations. Further speed can be obtained by compiling the Java code to native instructions using a compiler such as GCJ [19]. With a native binary compilation the code should attain approximately the same speed as a C++ implementation.

3.5 Problems encountered

The implementation of NTRU began by first understanding the basic algorithm and mathematical structures i.e. the truncated polynomial ring. Since there were no freely available implementations in any programming language, the code was written from scratch instead. Pseudocode for multiplication and inverse operations is provided by NTRU Cryptosystems, but the pseudocode is difficult to understand and much trial and error was needed before the correct software code was obtained. Once the basic polynomial ring operations of addition, subtraction, multiplication, and inverse were written the cryptosystem operations of key generation, encryption and decryption were straightforward to obtain.

Another unfortunate stumbling block was discovering that encryption/decryption would only work with carefully selected parameters, e.g. those in Table 1. The NTRU Cryptosystems tutorials and literature seem to present the parameters as security parameters rather than decryption failure probability parameters.

4. Empirical Speed Comparison of NTRU to RSA

4.1 PK systems and RSA

Before comparing NTRU to RSA a brief discussion of the speed metric relating to the three basic operations of key generation, encryption and decryption would be beneficial: Public key cryptography is often not used for encrypting large amounts of data and therefore encryption and decryption speeds are generally not considered as important as in symmetric key cryptography. Therefore, for this discussion we will refer to each of the three operations as having roughly equal significance.

For RSA, key generation time is mostly due to the generation of the 2 primes, P and Q. For software implementations of RSA, the software must emulate the large binary numbers as structures or objects since the microprocessor is typically only 32 or 64 bits long. The Java language has a built-in class `java.lang.BigInteger` than implements arbitrary sized integer numbers. The class also includes prime number generation methods. The RSA implementations considered for use in this study make use of this class for key generation, encryption and decryption.

4.2 RSA implementations used

After examining a few public domain Java implementations of RSA, one was chosen that yielded the best performance in terms of speed. That implementation is from

Cryptix [20]. The Cryptix RSA implementation does not offer any padding scheme, which caused decryption of the last block to yield incorrect plaintext (for messages/files that were not a multiple of the block size). However, this was not a concern with regard to the scope of this project (speed comparison). Running without padding in the RSA algorithm is the appropriate way to compare to the NTRU implementation because the latter does not include significant padding either. The reader should note that the Cryptix algorithm always used $e = F4$ (Fermat's number 4) and uses the Chinese Remainder Theorem (CRT) to speed up decryption.

4.3 NTRU and RSA Comparability

Theoretical comparisons for the PK operations between NTRU and other PK systems can be found in the literature, including order of operations, message expansion, and key size. Here the empirical results observed while using the NTRU implemented done for this project are discussed.

Comparing NTRU and RSA can most conveniently be done by comparing key sizes that give the same security level. Comparing security levels across algorithms is subjective, one method being based on estimated breaking times of each algorithm for various key sizes. As such NTRU167 is compared to RSA512, NTRU263 is compared to RSA1024 and NTRU503 is compared to RSA2048. These equivalencies are reported by NTRU Cryptosystems and are based on running times of lattice based attacks. Table 4 shows the operation speeds run on a 666Mhz Pentium III with 256 MB RAM, running JDK 1.3.1 while encrypting/decrypting a 280K file.

	Key Generation	Encryption	Decryption
NTRU167	90 msecs	21 secs	20 secs
RSA512	5 secs	3.5 secs	34 secs
NTRU263	150 msecs	31 secs	30 secs
RSA1024	10 secs	5.8 secs	100 secs
NTRU503	420 msecs	56 secs	55 secs
RSA2048	50 secs	10.8 secs	345 secs

Table 4. Key generation, encryption and decryption times using 280K file on 666Mhz Pentium III

4.4 Key Generation Speed Comparison

Table 4 shows that NTRU yields significantly faster key generation times than RSA. The reason for this is that prime number generation in RSA takes a significant amount of computational operations. The prime numbers were generated to a degree of certainty of $1 - 2^{-100}$. RSA's use of the generation of large primes with a 32-bit processor became obvious here. The numbers shown for NTRU are for the faster variation of the algorithm, i.e. with faster key generation and decryption times. NTRU does perform many operations (of order N^2), but each of these is a series of simple integer math instructions using relatively small numbers; specifically all numbers on the order of mod q .

Although not shown here, it should be not surprising that it was found that the faster key generation algorithm that does only inverse of $f \bmod q$, instead of two inverses in the basic algorithm, runs almost twice as fast as the latter.

4.5 Encryption and Decryption Speed Comparison

The decryption time advantage for NTRU over RSA shows the advantage of NTRU's use of small integer values over RSA's large integer values. It can also be seen that as the key size increases, NTRU gains increasing advantage in performance. The fastest variations of both algorithms were used and in order to make the comparison fair.

The encryption time for RSA is shown to be significantly faster than NTRU. The reason for this is the small amount of modulo exponentiation operations needed when using F_4 as the public exponent e . A similar result should be expected if e were to be set to be 3. Some of the NTRU vs. RSA comparisons in the literature use e to be a random large number, on the order of the size of modulus. However, this option does not appear to be available in the Cryptix RSA implementation and was therefore omitted.

Conclusions about encryption speeds for RSA using random large values for e can easily be obtained by knowledge of RSA encryption/decryption methods and looking at the RSA decryption speed. Specifically with RSA using a random public exponent e , we should expect encryption time to be about twice as long as that of the decryption time since decryption uses the Chinese Remainder Theorem. Therefore, for encryption, we should expect NTRU to have a speed advantage to be about twice that of its advantage observed in decryption.

5. Conclusions and Topics for Further Study

This project implemented the NTRU algorithm in software so that the claims of fast PK cryptography could be tested. The software implementation showed that NTRU was indeed faster than RSA, but did not reveal speed advantages that were orders of magnitude for encryption and decryption that were being claimed. While a more optimized version of NTRU would yield faster times, it is doubted that these optimizations would yield an increase of even two times the current implementation using the same computer language and hardware. Implementation and comparison in a different language may yield different results and the same could be said for hardware implementations.

In order to improve on the implementation, further optimizations of the algorithm which could be done which were not possible attainable during to time limits of this project. The most promising optimization is to use a faster algorithm for polynomial multiplication since it is the most time consuming of operations involved.

This study could be extended by comparing NTRU against other PK systems such as ECC, ElGamal and more recent systems such as braid group cryptography. While this study focused on speed of the operations, other factors should be considered such as key size, message expansion and perhaps most importantly the actual security provided by NTRU

References:

- [1] William Thomson, NTRU - A Public Key Cryptosystem with A Review of Comparative Metrics Between NTRU, RSA and ECC
- [2] Ntru Cryptosystems, The NTRUEncrypt Public Key Cryptosystem (Basic Tutorial)
- [3] Ntru Cryptosystems, The NTRUEncrypt Public Key Cryptosystem : Advanced Topics
- [4] Ntru Cryptosystems, The NTRUEncrypt Public Key Cryptosystem : Further Topics in Fast Implementation
- [5] Commutative NTRU: Pseudo-code implementation (Technical Note #001)
- [6] J. Hoffstein, J. Silverman, Optimizations for NTRU
- [7] Ntru Cryptosystems, Efficient Conversions from Mod q to Mod p (Technical Note #008)
- [8] Ntru Cryptosystems, High-Speed Multiplication of (Truncated) Polynomials (Technical Note #009)
- [9] Ntru Cryptosystems, Invertibility in Truncated Polynomial Rings (Technical Note #009)
- [10] Ntru Cryptosystems, Estimated Breaking Times for NTRU Lattices (Technical Note # 012)
- [11] Ntru Cryptosystems, Almost Inverses and Fast NTRU Key Creation (Technical Note # 014)
- [12] J. Hoffstein, D. Lieman, J. Pipher, J.H. Silverman, NTRU: A Public Key Cryptosystem
- [13] Karu and Loikkanen, Practical Comparison of Fast Public-key Cryptosystems, 2000.
- [14] Silverman, The NTRU Public Key Cryptosystem, Operating Characteristics and Comparison with RSA, ElGamel, and ECC Cryptosystems
- [15] A. May, Cryptanalysis of NTRU, 1999.
- [16] D. Coopersmith, A. Shamir, Lattice Attacks on NTRU, Eurocrypt 1997.
- [17] Joseph Silverman, Hard Problems and Backdoors for NTRU and Other PKCS's (Technical Report # 005).

[18] Peer Review and Independent Scrutiny of the NTRU Public Key Cryptosystem, NTRU.COM Technical Papers,

[19] GNU GCJ Java Compiler, <http://www.gnu.org/software/gcc/java/index.html>

[20] Cryptix Java Cryptography Implementations <http://www.cryptix.com/>