Capabilities and Performance of Java/C++/JNI Implementations of NTRU Public Key Cryptosystem

Krishnapriya Kadati,
Karthika Bhimavarapu and
George Koodarappally

History of NTRU

- Relatively new Public Key Cryptosystem
- First introduced in CRYPTO ’96
- Formally published in 1998
- Algorithms patented by NTRU Cryptosystem, Inc.

Introduction to NTRU Public Key Cryptosystem

- NTRU is a recent entrant to the family of public key cryptosystems. NTRU is based on algebraic operations on truncated polynomial rings and modular reduction.

- NTRU can be mainly described using three integer parameters (N, p, q) where N is used to determine the degree of the polynomials, p and q are two integers which are used for modular reduction.

Mathematical Background

- An element F in the ring $R=\mathbb{Z}[X]/(X^N-1)$

F€R $F=\sum_{i=0}^{N-1} F_i x^i=[F_0, F_1, \ldots, F_{N-1}]$

Lg, Lf, Lr, Lm are four set of polynomials of degree N-1 that are used to define sample set for g, f, r,m and these sample spaces are of the form

$$L(d_1, d_2)=\left\{ F\in R: \begin{matrix} F \text{ has } d_1 \text{ coefficients equal } 1, \\ d_2 \text{ coefficients equal } -1, \text{the rest } 0 \end{matrix} \right\}$$

Mathematical Background….contd

- The basic operations involved include addition, multiplication and modular reduction of the polynomial coefficients.
- Addition is performed by simple addition and Multiplication is performed by cyclic convolution product given by $F * G = H$ with

$$H_k = \sum_{i=0}^{k} F_i G_{k-i} + \sum_{i=k+1}^{N-1} F_i G_{N+k-i} = \sum_{i+j \equiv k \,(mod\ N)} F_i G_j$$

Key Generation

- Two polynomials f and g of degree N-1 are selected from sets Lf and Lg. The inverses are computed using a modified Extended Euclidean's Algorithm and they are denoted by Fq and Fp where $F_q * f \equiv 1 (mod\ q)$ and $F_p * f \equiv 1 (mod\ p)$

The public key h is computed as $h \equiv F_q * g (mod\ q)$

The private key is nothing but the polynomial f

Encryption

- The sender encrypts a message *m*, a polynomial of degree *N-1* by selecting a random polynomial *r* and using the receiver's public key *h* to calculate *e* as follows $e \equiv pr * h + m \pmod{q}$
- Here ‘e’ is the encrypted message.

Decryption

- To decrypt the message the receiver must compute a temporary product *a* using the encrypted message *e* and his private key *f*.

$$a \equiv f * e \pmod{q}$$

The original message is recovered by multiplying the temporary product *a* with the precomputed inverse of *f* modulo *p*, F_p.

$$F_p * a \pmod{p}$$

NTRU Security Levels

Security level	N	p	q	L_f (d_1,d_2)	L_g (d_1,d_2)	L_r (d_1,d_2)
Moderate	107	3	64	15,14	12,12	5,5
High	167	3	128	61,60	20,20	18,18
Highest	503	3	256	216,215	72,72	55,55

NTRU Operations

- StarMultiply
- RandPoly
- Inverse_Poly_Fq
- Inverse_Poly_Fp
- CreateKey
- Encode
- Decode

StarMultiply

- Performs the polynomial multiplication to generate cyclic convolution product

- StarMultiply(int a[ ], int b[ ], int c[ ], int N, int M)

RandPoly

- Generates a random Polynomial of degree N-1 with NumOnes values 1 and NumNegOnes values -1.

- RandPoly(int r[ ], int N, int NumOnes, int NumNegOnes)

Inverse_Poly_Fp

- Generates the Inverse of f modulo p using Extended Euclidean algorithm.

- Inverse_Poly_Fp(int a[ ], int Fp[ ], const int N, int p)

Inverse_Poly_Fq

- Generates the Inverse of f modulo q using Extended Euclidean algorithm. q must be a power of an arbitrary prime p. It first determines the inverse modulo a prime p and then uses Newton iteration method to determine inverse modulo q.

- Inverse_Poly_Fq(int a[ ], int Fq[ ], int N, int q)

CreateKey

- Creates the NTRU public key h and the Inverses of f modulo p and f modulo q.

- CreateKey(int N, int q, int p, int f[ ], int g[ ], int h[ ], int Fp[ ], int Fq[ ])

Encode

- Encryptes a message polynomial m and generates the encrypted message polynomial e

- Encode(int N, int q, int r[ ], int m[ ], int h[ ], int e[ ])

Decode

- Decodes an encrypted message e and generates a decrypted plaintext polynomial d

- Decode(int N, int q, int p, int f[ ], int Fp[ ], int e[ ], int d[ ])

Java Implementation

- Simple straigtforward implementation with one class NTRUMath with all the necessary methods

- Java 2 Standard Edition (J2SE Build 1.4.2)
- JCreator 3.5 LE

C++ Implementation

- Simple straigtforward implementation with one class NTRUMath with all the necessary methods

- Dev-C++ from Bloodshed Software
- Visual C++ .NET from Microsoft

JNI/C++ Implementation

- New Java class created to generate header for native functions
- New C++ class created to implement the native functions from above header
- Interfaced with the original C++ class with modifications
- Compiled into Windows DLL for call by Java JNI

Hardware

- Intel Pentium M Centrino
- 1.6 GHz
- 512 Mb RAM

- Other machines also used to compare findings

Testing

- Tested for each implementation
- Tested for each security level
- Iterated each function 1,000 times to get average timing in milliseconds

Java Performance

Security Level	Key Creation (msec)	Encode (msec)	Decode (msec)
Medium	6.53	0.38	0.751
High	17.465	0.921	1.833
Highest	175.793	9.323	16.835

C++ Performance (Dev-C++)

Security Level	Key Creation (msec)	Encode (msec)	Decode (msec)
Medium	5.718	0.34	0.701
High	15.161	0.842	1.682
Highest	150.206	7.561	15.372

C++ Implementation (Visual C++)

Security Level	Key Creation (msec)	Encode (msec)	Decode (msec)
Medium	5.668	0.34	0.691
High	15.071	0.831	1.673
Highest	150.326	7.531	15.051

JNI/C++ Performance

Security Level	Key Creation (msec)	Encode (msec)	Decode (msec)
Medium	5.608	0.351	0.681
High	14.992	0.861	1.683
Highest	150.737	7.701	15.312

Key Generation

Security Level	Java	C++	JNI/C++
Moderate	0.751	0.701	0.681
High	1.833	1.682	1.683
Highest	16.835	15.372	15.312

Encode

Security Level	Java	C++	JNI/C++
Moderate	0.38	0.34	0.351
High	0.921	0.842	0.861
Highest	9.323	7.561	7.701

Decode

Security Level	Java	C++	JNI/C++
Moderate	0.751	0.701	0.681
High	1.833	1.682	1.683
Highest	16.835	15.372	15.312

Conclusion

- Encryption and Decryption speeds are comparable between C++ and Java implementations
- Key Generation is faster in C++ than in Java
- NTRU uses simple integer addition, multiplication and modular operations

Further Work

- Incorporate various optimizations published
- Study the NTRUSign signature creation and verification algorithms

Acknowledgements

- Prof. Kris Gaj and Ashraf AbuSharekh for their invaluable guidance in getting us over several bottlenecks, especially polynomial inverse calculations