Technical Overview And Applications

Daniel Bailey

Product Manager

Embedded Systems

dbailey@ntru.com

NTRU Public Key Cryptography

- Part I: An Overview of NTRU
  - NTRU – Fast, Small and Secure
  - Why Is NTRU So Fast?
  - How Fast is NTRU?
  - Super Fast NTRU Key Creation
  - NTRU Disposable Keys
  - NTRU: Easy to Implement in Constrained Environments
  - History of the NTRU Public Key Cryptosystem

NTRU – Fast, Small and Secure

The NTRU Public-Key Cryptosystem features super fast cryptography:

- NTRU C code optimized for portability, NOT optimized for speed:
  - NTRU encrypt/decrypt runs up to 475 times faster than RSA/ECC on servers (at RSA-1024 security level – higher levels give NTRU an even BIGGER advantage!)
  - Encrypts 40-50 times faster than hand-optimized (for speed) RSA, ECC on constrained devices
  - Decrypts 333 times faster than hand-optimized (for speed) RSA on constrained devices

The NTRU Public-Key Cryptosystem features super fast key generation:

- NTRU key generation 30-1000+ times faster than RSA, ECC
- NTRU sign/verify up to 100 times faster than RSA or ECC
- Key generation is 1-3 orders of magnitude faster than RSA, ECC

NTRU is even FASTER on 8-bit chips or DSPs:

- NTRU NEVER requires a coprocessor, even on 8-bit devices!
- Up to 2000 times faster than RSA on DSPs

Why is NTRU so Fast?

NTRU's basic operation is *convolution product* of two vectors of small numbers:

$$[a_0,\ldots,a_{N-1}] * [b_0,\ldots,b_{N-1}] = [c_0,\ldots,c_{N-1}]$$

With

$$c_k = a_0b_k + a_1b_{k-1} + a_2b_{k-2} + \ldots + a_{N-2}b_{k+2} + a_{N-1}b_{k+1} .$$

Convolution products can be computed very rapidly using Karatsuba multiplication or Fast Fourier Transforms.

RSA's basic operation is *modular multiplication* of big numbers:

$a * b \pmod N$ with $N=p*q$.

ECC's (elliptic curve cryptosystem) basic operation is *addition of points on an elliptic curve* :

$P + Q \pmod p$ with P and Q points on E.

Why is NTRU so Fast?

System	Basic Operation
NTRU	Convolution Product
RSA	Modular Multiplication
ECC	Elliptic Curve Addition

- NTRU and ECC basic operations take approximately the same time (basic operations for RSA are a little faster).
- How many basic operations are needed to encrypt and decrypt?

System	Number of Operations Required	
	Encrypt	Decrypt
NTRU	1	2
RSA	17	~1,000
ECC	~160	~160

How Fast is NTRU on Servers? (RSA 1024 Level Security)

Function	Units	NTRU 251	RSA 1024	NTRU Advantage
Encrypt	Blocks/sec	22727	1280	17 to 1
Decrypt	Blocks/sec	10869	110	99 to 1
Sign	Sigs/sec	1242	106	11.7 to 1
Verify	Sigs/sec	3450	1851	1.86 to 1

Function	Units	NTRU 251	ECC 163	NTRU Advantage
Encrypt	Blocks/sec	22727	48	475 to 1
Decrypt	Blocks/sec	10869	55	197 to 1
Sign	Sigs/sec	1242	57	21 to 1
Verify	Sigs/sec	3450	68	50 to 1

NTRU's NERI toolkit vs. Wei Dai Toolkit (for RSA, ECC) with e=65537 (800 MHz Pentium III)

How Fast is NTRU on Constrained Devices? (RSA 1024 Level Security)

Function	Units	NTRU 251	RSA 1024	NTRU Advantage
Encrypt	Blocks/sec	21	0.5	42 to 1
Decrypt	Block/sec	12	0.036	333 to 1
Sign	Sigs/sec	2	0.036	55 to 1
Verify	Sigs/sec	3.1	0.5	6.2 to 1

Function	Units	NTRU 251	ECC 163	NTRU Advantage
Encrypt	Blocks/sec	21	0.4	52.5 to 1
Decrypt	Block/sec	12	1.3	9 to 1
Sign	Sigs/sec	2	1.3	1.5 to 1
Verify	Sigs/sec	3.1	0.5	6.2 to 1

NTRU's NERI portable toolkit vs. published results (for ECC, RSA) specialized for the Palm device with e=65537.

Super Fast NTRU Key Creation

- NTRU key creation is *lightning fast*
- All NTRU keys are *fully independent*

	Key Size	Create Key Pair	Key Pairs/ Second	NTRU Advantage
NTRU	1757	1.11 ms	900	
RSA	1024	560 ms	1.8	505 to 1
ECC GF(p)	163	4.42 ms	111.5	3.98 to 1

Timing at 800MHz. NTRU uses NERI5.6 toolkit. RSA and ECC use Crypto++ package.

Note:
- Independent ECC keys need random elliptic curves.
- ECC key generation of independent keys is slower than RSA.
- All NTRU keys are completely independent.

NTRU Disposable Keys
A New Public Key Paradigm

- NTRU's fast key creation enables new public key paradigms
  - **Content Protection**
    - Encrypt audio and video with a different key for every few seconds of content
  - **eCommerce / mCommerce**
    - Use independent keys for every transaction
  - **E-mail**
    - Master Key/Disposable Key Protocol saves storage and increases security

Easy to Implement in Constrained Environments

- NTRU is:
  - Easy to program
  - Easy to build into hardware
  - Ideal for Digital Signal Processors (DSPs)
- NTRU Requires:
  - Less memory (RAM) in software
  - Less storage in software
  - Fewer gates in hardware
- NTRU easily fits into:
  - Low power smart cards
  - Handheld devices
  - Cellular telephones
  - Set top boxes

… without sacrificing speed!

History of NTRU

- Developed by team of cryptographer/mathematicians
  - J. Hoffstein, J. Pipher, J. Silverman (1994-1996)
- Presented by J. Hoffstein at CRYPTO ’96
- Immediate feedback from top cryptographers (Coppersmith, Hastad, Odlyzko, Shamir,…) used to set appropriate security parameters
- Ongoing research by experts in lattices and cryptography (Phong, Stern, Schnorr, …) reaffirms NTRU’s security
- New IEEE P1363.1 standard based on NTRU
- New CEES Efficient Embedded Security Standard based on NTRU

NTRU Public-Key Cryptography

- Part II: How NTRU Works
  - NTRU Multiplication
  - Small Polynomials and Polynomials Mod q
  - NTRU Public Parameters
  - NTRU Key Creation
  - NTRU Encryption
  - NTRU Decryption
  - Why Does NTRU Work?
  - NSS Key Creation
  - NSS Signing/Verifying
  - NSS Encoding/Masking
  - Why does NSS Work?

NTRU Multiplication

NTRU uses polynomials $a_0+a_1x+a_2x^2+\ldots+a_{N-1}x^{N-1}$

Always multiply NTRU polynomials using the extra rule

$$x^N = 1.$$

Example with N=4 (Extra Rule: $x^4 = 1$)

$(x^3+2x-1)*(3x^3-x^2+x+2)$ $= 3x^6-x^5+7x^4-3x^3+3x^2+3x-2$

$= 3x^2-x + 7 - 3x^3+3x^2+3x-2$

$= -3x^3+6x^2+2x+5$

NTRU Multiplication

The product of

$$f(x) = a_0+a_1x+a_2x^2+\ldots+a_{N-1}x^{N-1} \quad \text{and}$$
$$g(x) = b_0+b_1x+b_2x^2+\ldots+b_{N-1}x^{N-1}$$

Is

$$h(x) = f(x)*g(x) = c_0+c_1x+c_2x^2+\ldots+c_{N-1}x^{N-1}$$

With

$$c_k = a_0b_k + a_1b_{k-1} + a_2b_{k-2} + \ldots + a_{N-1}b_{k+1}.$$

- The coefficient vector of the product h(x) = f(x)*g(x) is the *convolution product* of the coefficient vectors of f(x) and g(x).
- The convolution product can be computed very rapidly using Fast Fourier Transforms or Karatsuba multiplication in approximately N*log(N) steps.

Small Polynomials and Polynomials Mod q

NTRU uses polynomials whose coefficients are all –1, 0, or 1, where the number of 1's and –1's may be specified. We will call these small polynomials, since they have small coefficients.

The coefficients of NTRU polynomials may be reduced modulo a number q. That means that the coefficients are replaced with their remainders after being divided by q.

Example.

$$5x^3 - 11x^2 + 4x + 6 = -x^3 + x^2 + x \pmod 3$$

NTRU Public Parameters

The NTRU Public Key Cryptosystem depends on three public parameters:

N, p, q

Typical values for these parameters, with approximate equivalent RSA security levels, are:

NTRU			RSA Level
N	p	q	(bits)
251	3	128	1024
347	3	128	2048
503	3	256	4096

NTRU Key Creation

Bob chooses two small polynomials f(x) and g(x).
Bob computes inverses for f(x) modulo p and modulo q. This means that Bob computes polynomials $F_1(x)$ and $F_2(x)$ so that

$$F_1(x)*f(x) = 1 \pmod{p} \quad \text{and} \quad F_2(x)*f(x) = 1 \pmod{q}.$$

Finding these inverses is very fast using the Euclidean algorithm.
Bob computes the product

$$h(x) = p*F_2(x)*g(x) \pmod{q}.$$

Bob's Private Key: the pair f(x) and $F_1(x)$
Bob's Public Key: the polynomial h(x)

NTRU Encryption

Alice's plaintext is a polynomial

$$m(x) \quad \text{modulo } p.$$

Alice randomly chooses a small polynomial $r(x)$.

Alice uses Bob's public key $h(x)$ to compute

$$e(x) = r(x)*h(x) + m(x) \quad (\text{mod } p).$$

Alice's encrypted message: $e(x)$

NTRU Decryption

Bob uses his private key f(x) and Alice's ciphertext e(x) to compute

$$a(x) = f(x)*e(x) \pmod{q}.$$

Bob chooses the coefficients of a(x) to lie between -q/2 and q/2.

Bob uses the other part of his private key $F_1(x)$ to compute

$$b(x) = F_1(x)*a(x) \pmod{p}.$$

The polynomial b(x) is equal to Alice's plaintext message m(x).

Why Does NTRU Work?

a	= f * e	(mod q)	f is Bob's private key
	= f * (r*h + m)	(mod q)	r is the message blinding value
	= f * (r*p*g*F_2 + m)	(mod q)	p*g*F_2 is Bob's public key
	= p*r*g + f*m	(mod q)	since f*F_2 = 1 (mod q).

All of the polynomials r, g, f, m are small, so the polynomial

$$p*r*g + f*m$$

will have coefficients between -q/2 and q/2. Therefore the polynomial a(x) is exactly equal to p*r*g + f*m. Then

$$F_1*a = F_1*(p*r*g + f*m) = F_1*f*m = m \pmod p$$

because $F_1*f = 1 \pmod p$.

NSS Key Creation

Bob chooses two small polynomials f(x) and g(x).
Bob computes inverses for f(x) modulo p and modulo q. This means that Bob computes polynomials $F_1(x)$ and $F_2(x)$ so that

$$F_1(x)*f(x) = 1 \pmod{p} \quad \text{and} \quad F_2(x)*f(x) = 1 \pmod{q}.$$

Finding these inverses is very fast using the Euclidean algorithm.
Bob computes the product

$$h(x) = p*F_2(x)*g(x) \pmod{q}.$$

Bob's Private Key: the polynomial f(x)
Bob's Public Key: the polynomial h(x)

NSS Signing/Verifying

First, use a hash function to produce a message digest m(x) from a document D.

Construct a small polynomial w(x) that encodes m(x) mod p.

The signature is $s(x) = f(x)*w(x) \pmod q$.

To verify the signature, compute

$$t = h(x)*s(x) = g(x)*w(x) \pmod q,$$

and verify that

1) s(x) and t(x) look like products of small polynomials
2) s(x) and t(x) are tied to m(x) mod p.

NSS Encoding/Masking

Choose private key polynomials f(x) and g(x) in the form

$f(x) = u(x) + pF(x)$ and $g(x) = u(x) + pG(x)$

where u(x), F(x) and G(x) are random small polynomials.

The polynomial w(x) used in the signature $s(x) = f(x)*w(x) \pmod q$ is called the encoding/masking polynomial. It does two things:

- w(x) encodes information about m(x) mod p;
- w(x) masks f(x) and g(x) within s(x) and t(x), making it difficult for an attacker to extract information.

The encoding/masking polynomial w(x) is constructed as:

1) Choose small polynomials w1(x) and w2(x)
2) w2(x) random, and w1(x) chosen to conceal mod q reductions.
3) Compute $w0(x) = (u(x)^{-1}*m(x) \bmod p) + (u(x)^{-1} *w1(x) \bmod p)$.
4) Set $w(x) = w0(x) + pw2(x)$.

Why Does NSS Work?

$$
\begin{aligned}
s &= f * w && \pmod q \\
&= (u + pF) * (w0 + pw2) && \pmod q \\
&= u*w0 + pu*w2 + pF*w && \pmod q
\end{aligned}
$$

$$
\begin{aligned}
t &= g * w = h * s && \pmod q \\
&= (u + pG) * (w0 + pw2) && \pmod q \\
&= u*w0 + pu*w2 + pG*w && \pmod q
\end{aligned}
$$

All of the polynomials u, G, F, m are small, and

$$u*w0 = u*(u(x)^{-1}*m + u(x)^{-1}*w1) = m + w1 \pmod p.$$

Mod p, the ith coefficient of s and t is the ith coefficient of m, unless

1) the ith coefficient of m has been altered, or
2) the ith coefficient of f*w or g*w has been reduced mod q

This mod p correlation ties the signature s(x) to the digest m(x).

NTRU Public Key Cryptography

- Part III: Security
  - The NTRU/NSS Hard Problem
  - NTRU versus ECC – Speed and Security
  - Lattices and the Short Vector Problem
  - Brief History of Lattice Problems
  - NTRU Lattices and Testing NTRU Security
  - Security Comparison – Time to Break

The NTRU/NSS Hard Problem

The hard problem underlying NTRU and NSS is the

Short Vector Problem

in lattices of high dimension

System	Hard Problem	Best Solution Method
NTRU	Short vector problem	LLL lattice reduction
RSA	Integer factorization	Number field sieve
ECC	Elliptic curve discrete log	Pollard rho
DH	Discrete logarithm	Index calculus

Best Known Methods to Break:

- NTRU and ECC are exponential (very slow)
- RSA and DH are subexponential (faster)

NTRU versus ECC – Speed and Security

To improve the speed of ECC, people use a special elliptic curve.

<u>ECC Using One Special Elliptic Curve</u>
Advantage – improved speed
(but NTRU is still much faster!)
Disadvantage -- possibly decreased security

<u>ECC Using Many (Random) Elliptic Curves</u>
Advantage -- probable increased security
Disadvantage -- very slow

The NTRU Advantage
- Every NTRU key has its own random lattice.
- NTRU keys are always completely independent.

Lattices and the Short Vector Problem

Take a collection of vectors

$\mathbf{V}_1 = [a_1,\ldots,a_n]$ … $\mathbf{V}_n = [c_1,\ldots,c_n]$.

A lattice L of dimension n is the set of all linear combinations

$L = \{ z_1\mathbf{V}_1+\ldots+z_n\mathbf{V}_n : z_1,\ldots,z_n \text{ are integers}\}$.

- The *Shortest Vector Problem* (SVP) is to find the shortest nonzero vector in the lattice L.
- The *Closest Vector Problem* (CVP) is to find the vector in the lattice L that is closest to a given vector **W**.
- If the dimension n of the lattice is large, both SVP and CVP are very difficult to solve.
- The hard problems underlying NTRU are the SVP and the CVP.

Brief History of Lattice Problems

- Lattices, SVP, and CVP have been extensively studied for more than 100 years (Hermite 1870s, Minkowski 1890s,…).
- Best computational tool was developed by Lenstra, Lenstra, and Lovasz (LLL algorithm) in early 1970s.
- Improvements to LLL are due to Schnorr, Euchner, Horner, others in 1980s.
- Algorithms to find small vectors in lattices have been extensively studied because they have applications to many areas outside of cryptography, including physics, combinatorics, number theory, computer algebra,….
- Contrast this with integer factorization (RSA) and elliptic curve discrete logarithms (ECC), where the only applications are to cryptography.

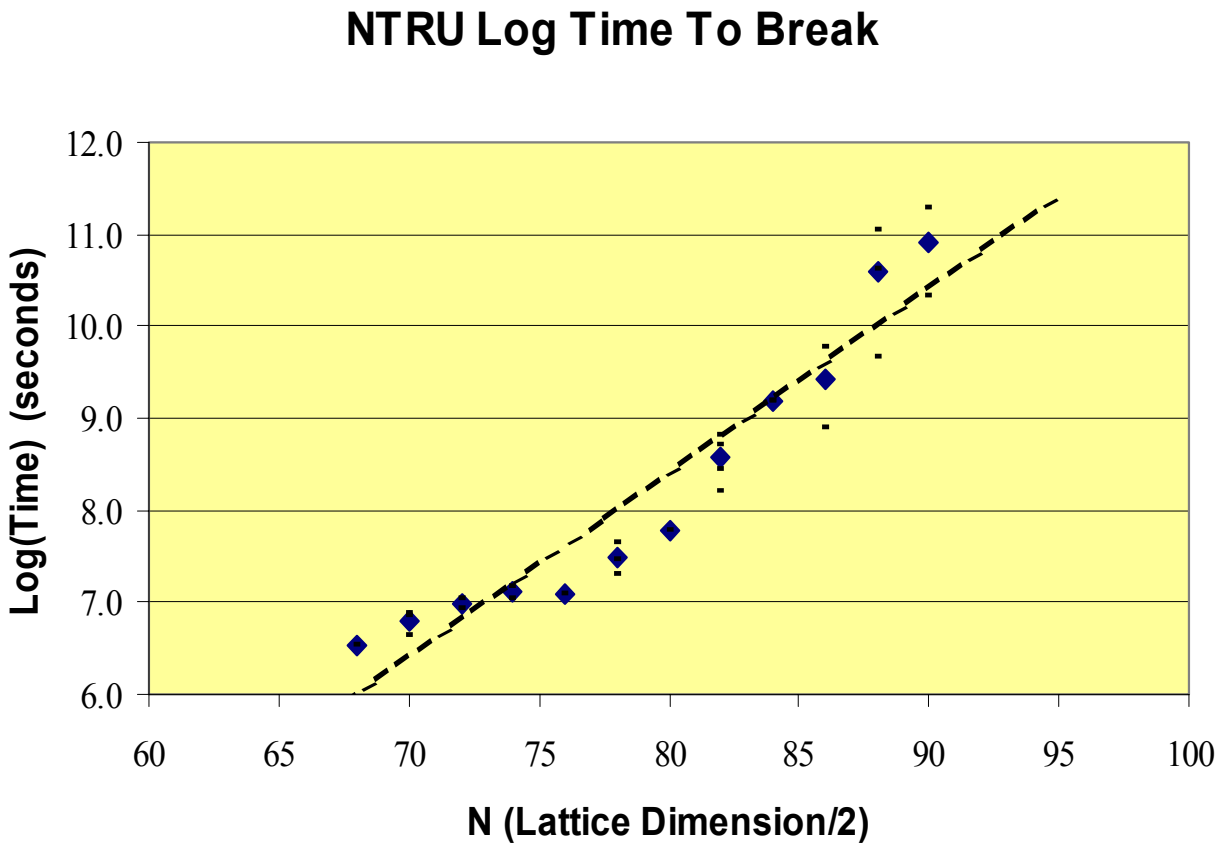
NTRU Lattices and Testing NTRU Security

- The NTRU lattice has dimension 2N.
- The NTRU (public) key has length approximately N*log(N/2) bits.

N	251	347	503
Lattice Dimension	502	694	1006
Key Length (bits)	1757	2429	4024

- Earlier lattice based cryptosystems (knapsack type, Ajtai-Dwork, Goldreich-Goldwasser-Halevi) had dimension N and key length approximately N^2 bits.
- Earlier systems were impractical for lattices of dimension 500 to 1000 and insecure for lattices of dimension 100 to 200.
- NTRU is practical and secure using lattices of dimension 500 to 1000.

Computing NTRU Security Levels



Computing NTRU Security Levels

Extrapolation Line:

$\log(T) = 0.174*N - 5.598$

Coefficient of Correlation: 0.958

Extrapolated Breaking Time:

N = 251

Log(T) = 37.694

T = $3.48*10^{16}$ seconds = $4.4*10^{11}$ MIPS-years

[Conversion at 400 MHz: T seconds ≈ $1.27*10^{-5}*T$ MIPS-years]

NTRU Security: Time to Break

Cryptographic System	Block/Public Key Size (Bits)	Processing Time (MIPS-Years)
NTRU 251 (RSA 1024 Security Level)	1757	$> 10^{11}$
NTRU 347 (RSA 2048 Security Level)	2429	$> 10^{21}$
NTRU 503 (RSA 4096 Security Level)	4024	$> 10^{33}$

Technical Overview And Applications

Daniel Bailey

Product Manager

Embedded Systems

dbailey@ntru.com