NTRUEncrypt and pqNTRUSign

Zhenfei Zhang @ NTRU team

April 12, 2018
NTRU
One of the first lattice based cryptosystems; 20 years old.

Through the years we heard

- It doesn’t have security proof!
- It only focuses on practicality!
- It uses an ad-hoc ring!
- It uses a sparse trinary polynomial!
- It has decryption errors!
How lattice based encryption should have been developed - Vadim Lyubashevsky
An alternate universe

What if NTRU was not proposed 22 years ago?
An alternate universe

- What if NTRU was not proposed 22 years ago?
- We wouldn’t have seen the failure of NTRUSign.
What if NTRU was not proposed 22 years ago?
We wouldn’t have seen the failure of NTRUSign.
Luckily, we still have FALCON.
An alternate universe
What if NTRU was not proposed 22 years ago?

- RLWE, 10 → RLWE based
- LWE, 06 → SS-NTRU, 11 → NTRU based
- SS-NTRU, 11 → RLWR, 12 → RLWR based
An alternate universe
What if NTRU was not proposed 22 years ago, but now?

Earth 1
- It doesn’t have security proof!
- It only focuses on practicality!
- It uses an ad-hoc ring!
- It uses a sparse trinary polynomial!
- It has decryption errors!

Earth 2
- It stems from a provable secure design;
- and is practical!
- Ring is not restricted to $x^{2^p} + 1$!
- It uses a sparse trinary polynomial!
- Decrypt errors are negligible!
An alternate universe
What if NTRU was not proposed 22 years ago, but now?

Earth 1
- It doesn’t have security proof!
- It only focuses on practicality!
- It uses an ad-hoc ring!
- It uses a sparse trinary polynomial!
- It has decryption errors!

Earth 2
- It stems from a provable secure design;
- and is practical!
- Ring is not restricted to $x^{2^p} + 1$!
- It uses a sparse trinary polynomial!
- Decrypt errors are negligible!

NTRU APPEARS more popular if it wasn’t invented 22 years ago!

What about (provable) security?
- Just find parameters secure from BKZ (+ sieving)
- We did it with (R)-LWE based KEX anyway …
An alternate universe
What if NTRU was not proposed 22 years ago?

Let’s do a clean slate comparison

- NTRU uses a trapdoored lattice; RLWE/RLWR uses a generic lattice
- NTRU relies on uSVP - unique shortest vector is sparse trinary;
- Practical RLWE/RLWR rely on BDD - distance vector MAY be sparse trinary;
- The rest are all tunable parameters (in practice)
  - Both can be instantiated with the same ring; same noise distribution

Fundamental difference: Trapdoor

- NTRU lattices are more useful in PKE and Signatures
- RLWE/RLWR have the advantages in KEX
NTRU lattice

NTRU assumption

- Decisional: given two small ring elements $f$ and $g$; it is hard to distinguish $h = f/g$ from a uniformly random ring element;
- Computational: given $h$, find $f$ and $g$.

NTRU lattice with unique shortest vectors $(g, f)$

$\begin{bmatrix} qI_N & 0 \\ H & I_N \end{bmatrix} := \begin{bmatrix} q & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & q & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & q & 0 & 0 & \cdots & 0 \\ h_0 & h_1 & \cdots & h_{N-1} & 1 & 0 & \cdots & 0 \\ h_{N-1} & h_0 & \cdots & h_{N-2} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ h_1 & h_2 & \cdots & h_0 & 0 & 0 & \cdots & 1 \end{bmatrix}$
**Enc** \((h = g/f, p = 3, R, m \in \{-1, 0, 1\}^N)\)

- Find a random ring element \(r\);
- Compute \(e = p \times r \cdot h + m\);

**Dec** \((f, p = 3, R, e)\)

- Compute \(c = e \cdot f = p \times r \cdot g + m \cdot f\);
- Reduce \(c \mod p = m \cdot f \mod p\)
- Recover \(m = c \cdot f^{-1} \mod p\)
NTRUEncrypt
A CCA-2 secure encryption scheme based on NTRU assumption

Enc \((h = g/f, p = 3, f \equiv 1 \mod p, R, m \in \{-1, 0, 1\}^N)\)

- Find a random ring element \(r\);
- Compute \(e = p \times r \cdot h + m\);

Dec \((f \equiv 1 \mod p, p = 3, R, e)\)

- Compute \(c = e \cdot f = p \times r \cdot g + m \cdot f\);
- Reduce \(c \mod p = m \cdot f \mod p = m\)
NTRUEncrypt
A CCA-2 secure encryption scheme based on NTRU assumption

Enc \( (h = g/f, p = 3, f \equiv 1 \mod p, R, m \in \{-1, 0, 1\}^k) \)
- Find a random string \( b; r = \text{hash}(h|b) \)
- \( m' = r \otimes \langle m|b \rangle \)
- Compute \( e = p \times r \cdot h + m' \);

Dec \( (f \equiv 1 \mod p, g, p = 3, R, e) \)
- Compute \( c = e \cdot f = p \times r \cdot g + m' \cdot f \);
- Reduce \( c \mod p = m' \cdot f \mod p = m' \)
- Compute \( r' = p^{-1} \times (c - m' \cdot f) \cdot g^{-1} \)
- Extract \( m, b \) from \( m' \otimes r' \), compute \( r = \text{hash}(h|b) \);
- Output \( m \) if \( r = r' \).
Modular Lattice Signatures

The core idea

- Given a lattice $\mathcal{L}$ with a trapdoor $T$, a message $m$, find a vector $v$
  - $v \in \mathcal{L}$
  - $v \equiv \text{hash}(m) \mod p$

- Can be instantiated via any trapdoored lattice
  - SIS, R-SIS, etc

- pqNTRUSign is an efficient instantiation using the NTRU lattice
pqNTRUSign

Sign \((f, g, h = g/f, p = 3, R, m)\)

- Hash message into a “mod \(p\)” vector \(\langle v_p, u_p \rangle = hash(m|h)\)
- Repeat with rejection sampling:
  - Sample \(v_0\) from certain distribution; compute \(v_1 = p \times v_0 + v_p\)
  - Find a random lattice vector \(\langle v_1, u_1 \rangle = v_1 \cdot \langle l, h \rangle\)
    - “\(v\)-side” meets the congruent condition.
  - Micro-adjust “\(u\)-side” using trapdoor \(f\) and \(g\)
    - Compute \(a = (u_1 - u_p) \cdot g^{-1} \mod p\)
    - Compute \(\langle v_2, u_2 \rangle = a \cdot \langle p \times f, g \rangle\)
    - Compute \(\langle v, u \rangle = \langle v_1, u_1 \rangle + \langle v_2, u_2 \rangle\)
- Output \(v\) as signature

Remark

\[ v = v_1 + v_2 = (p \times v_0 + v_p) + p \times a \cdot f = p \times (v_0 + a \cdot f) + v_p \]
pqNTRUSign

Verify \((h, p = 3, R, m, \nu)\)

- Hash message into a “mod \(p\)” vector \(\langle \nu_p, u_p \rangle = hash(m|h)\)
- Reconstruct the lattice vector \(\langle \nu, u \rangle = \nu \cdot \langle l, h \rangle\)
- Check \(\langle \nu_p, u_p \rangle = hash(m|h)\)
- Public key security: recover $f$ and $g$ from $h$;
- Forgery: as hard as solving an approx.-SVP in an intersected lattice;
- Transcript security - achieved via rejection sampling.
Forgery: as hard as solving an approx.-SVP in an intersected set:
\[ \mathcal{L}' := \mathcal{L}_h \cap (p\mathbb{Z}^{2N} + \langle v_p, u_p \rangle) \]

- \[ \det(\mathcal{L}_h \cap p\mathbb{Z}^{2N}) = p^{2N}q^N \quad \rightarrow \quad \text{Gaussian heuristic length} \]
  \[ = \sqrt{\frac{p^2q^{N}}{\pi e}} \]
- Target vector length \[ \| \langle v, u \rangle \| \leq \sqrt{2N}q^2 \]
- Approx.-SVP with root Hermite factor \[ \gamma = \sqrt{\frac{q\pi e}{2p^2}}^{\frac{1}{\dim}} = \left( \frac{q\pi e}{2p^2} \right)^{\frac{1}{4N}} \]
Consider \( b := v_0 + a \cdot f \)

- “large” \( v_0 \) drawn from uniform or Gaussian;
- “small” \( a \) drawn from sparse trinary/binary;
- sparse trinary/binary \( f \) is the secret.

RS on \( b \)

- \( b \) follows certain publicly known distribution independent from \( f \);
- for two secret keys \( f_1, f_2 \) and a signature \( b \), one is not able to tell which key signs \( b \).
### Performance

<table>
<thead>
<tr>
<th>Param</th>
<th>PK size</th>
<th>CTX size</th>
<th>KeyGen</th>
<th>Encryption</th>
<th>Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntrukem-743</td>
<td>8184 bits</td>
<td>8184 bits</td>
<td>1017 $\mu$s</td>
<td>140 $\mu$s</td>
<td>210 $\mu$s</td>
</tr>
<tr>
<td>ntrupke-743</td>
<td>8184 bits</td>
<td>8184 bits</td>
<td>990 $\mu$s</td>
<td>121 $\mu$s</td>
<td>195 $\mu$s</td>
</tr>
</tbody>
</table>

**Table:** NTRUEncrypt

<table>
<thead>
<tr>
<th>Param</th>
<th>PK size</th>
<th>RSig size</th>
<th>KeyGen</th>
<th>Signing</th>
<th>Verifying</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian-1024</td>
<td>16384 bits</td>
<td>$\approx$ 11264 bits</td>
<td>47.8 ms</td>
<td>120 ms</td>
<td>0.96 ms</td>
</tr>
<tr>
<td>Uniform-1024</td>
<td>16384 bits</td>
<td>16384 bits</td>
<td>48.9 ms</td>
<td>289 ms</td>
<td>0.97 ms</td>
</tr>
</tbody>
</table>

**Table:** pqNTRUSign
Feedback we have received so far

Bugs in the code
- Mask function was incorrectly implemented for NTRUEncrypt with Gaussian secret
- Gauss sampler took smaller deviation than required for NTRUEncrypt with Gaussian secret
- Rejection sampling on $ag$ is missing for pqNTRUSign

Mistakes in the algorithm
- Parameter for the bound of $\nu$-side was incorrect

Signature simulations
- Attacker learns more information on the lattice vs simulator
- Can be fixed via message randomization or deterministic signing.