Introduction to lattice-based KEMs

May 4, 2022
<table>
<thead>
<tr>
<th>Row Labels</th>
<th>Key Exchange</th>
<th>Signature</th>
<th>Grand Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Braids</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Chebychev</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Codes</td>
<td>19</td>
<td>5</td>
<td>24</td>
</tr>
<tr>
<td>Finite Automata</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Hash</td>
<td></td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Hypercomplex Numbers</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Isogeny</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Lattice</td>
<td>24</td>
<td>4</td>
<td>28</td>
</tr>
<tr>
<td>Mult. Var</td>
<td>6</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>Rand. walk</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>RSA</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td><strong>Grand Total</strong></td>
<td><strong>57</strong></td>
<td><strong>23</strong></td>
<td><strong>80</strong></td>
</tr>
</tbody>
</table>
All the way back in 2016...

“We’re indebted to Erdem Alkim, Léo Ducas, Thomas Pöppelmann and Peter Schwabe, the researchers who developed “New Hope”, the post-quantum algorithm that we selected for this experiment.”

All the way back in 2016...

“Key Agreement using the ‘NewHope’ lattice-based algorithm detailed in the New Hope paper, and LUKE (Lattice-based Unique Key Exchange), an ISARA speed-optimized version of the NewHope algorithm.”

https://www.isara.com/isara-radiate/
All the way back in 2016...

“The deployed algorithm is a variant of “New Hope”, a quantum-resistant cryptosystem”

Learning with errors (LWE)

• Given uniform $A \in \mathbb{Z}_q^{k \times \ell}$
• Given “noise distribution” $\chi$
• Given samples $As + e$, with $e \leftarrow \chi$
Learning with errors (LWE)

• Given uniform $A \in \mathbb{Z}_q^{k \times \ell}$
• Given “noise distribution” $\chi$
• Given samples $As + e$, with $e \leftarrow \chi$
• Search version: find $s$
• Decision version: distinguish from uniform random
• Given uniform $A \in \mathbb{Z}_q^{k \times \ell}$
• Given samples $\lceil A_s \rceil_p$, with $p < q$
Learning with rounding (LWR)

• Given uniform $A \in \mathbb{Z}_q^{k \times \ell}$
• Given samples $\lfloor As \rfloor_p$, with $p < q$
• Search version: find $s$
• Decision version: distinguish from uniform random
Using structured lattices

- Problem with LWE-based cryptosystems: public-key size
- Only NIST candidate exclusively using standard LWE: FrodoKEM
Using structured lattices

• Problem with LWE-based cryptosystems: public-key size
• Only NIST candidate exclusively using standard LWE: FrodoKEM
• Idea to solve this: allow structured matrix $A$, e.g.,

\[
A = \begin{pmatrix}
\alpha_1 & \alpha_2 & \cdots & \alpha_n \\
\beta_1 & \beta_2 & \cdots & \beta_n \\
\end{pmatrix}
\]
• Problem with LWE-based cryptosystems: public-key size
• Only NIST candidate exclusively using standard LWE: FrodoKEM
• Idea to solve this: allow structured matrix $A$, e.g.,
  • NewHope: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$; $n$ a power of 2, $q$ prime
Using structured lattices

- Problem with LWE-based cryptosystems: public-key size
- Only NIST candidate exclusively using standard LWE: FrodoKEM
- Idea to solve this: allow structured matrix $A$, e.g.,
  - NewHope: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$; $n$ a power of 2, $q$ prime
  - NTRU: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n - 1)$; $n$ prime, $q$ a power of 2
• Problem with LWE-based cryptosystems: public-key size
• Only NIST candidate exclusively using standard LWE: FrodoKEM
• Idea to solve this: allow structured matrix $A$, e.g.,
  • NewHope: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$; $n$ a power of 2, $q$ prime
  • NTRU: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n - 1)$; $n$ prime, $q$ a power of 2
  • NTRU Prime: work in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n - X - 1)$; $q$ prime, $n$ prime
Using structured lattices

• Problem with LWE-based cryptosystems: public-key size
• Only NIST candidate exclusively using standard LWE: FrodoKEM
• Idea to solve this: allow structured matrix \( A \), e.g.,
  • NewHope: work in \( \mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1) \); \( n \) a power of 2, \( q \) prime
  • NTRU: work in \( \mathcal{R}_q = \mathbb{Z}_q[X]/(X^n - 1) \); \( n \) prime, \( q \) a power of 2
  • NTRU Prime: work in \( \mathcal{R}_q = \mathbb{Z}_q[X]/(X^n - X - 1) \); \( q \) prime, \( n \) prime
  • Kyber/Saber: use small-dimension matrices and vectors over \( \mathcal{R}_q = \mathbb{Z}_q[X]/(X^{256} + 1) \)
Using structured lattices

- Problem with LWE-based cryptosystems: public-key size
- Only NIST candidate exclusively using standard LWE: FrodoKEM
- Idea to solve this: allow structured matrix \( \mathbf{A} \), e.g.,
  - NewHope: work in \( \mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1) \); \( n \) a power of 2, \( q \) prime
  - NTRU: work in \( \mathcal{R}_q = \mathbb{Z}_q[X]/(X^n - 1) \); \( n \) prime, \( q \) a power of 2
  - NTRU Prime: work in \( \mathcal{R}_q = \mathbb{Z}_q[X]/(X^n - X - 1) \); \( q \) prime, \( n \) prime
  - Kyber/Saber: use small-dimension matrices and vectors over \( \mathcal{R}_q = \mathbb{Z}_q[X]/(X^{256} + 1) \)
- Perform arithmetic on (vectors of) polynomials instead of vectors/matrices over \( \mathbb{Z}_q \)
How to build a KEM?

<table>
<thead>
<tr>
<th>Alice (server)</th>
<th>Bob (client)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s, e \xleftarrow{$} \chi$</td>
<td>$s', e' \xleftarrow{$} \chi$</td>
</tr>
<tr>
<td>$b \leftarrow as + e$</td>
<td>$b \rightarrow u \leftarrow as' + e'$</td>
</tr>
</tbody>
</table>

Alice has $v = us = ass' + e's$

Bob has $v' = bs' = ass' + es'$

- Secret and noise polynomials $s, s', e, e'$ are small
- $v$ and $v'$ are *approximately* the same
### How to build a KEM, part 2

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s, e \leftarrow \chi$</td>
<td>$s', e' \leftarrow \chi$</td>
</tr>
<tr>
<td>$b \leftarrow as + e$</td>
<td>$(b)$</td>
</tr>
<tr>
<td>$v' \leftarrow us$</td>
<td>$(u)$</td>
</tr>
<tr>
<td>$a \leftarrow Parse(XOF(seed))$</td>
<td></td>
</tr>
<tr>
<td>$u \leftarrow as' + e'$</td>
<td></td>
</tr>
<tr>
<td>$v \leftarrow bs'$</td>
<td></td>
</tr>
</tbody>
</table>

**This is LPR encryption, written as KEM (except for generation of $a$).**
How to build a KEM, part 2

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$seed \leftarrow {0, 1}^{256}$</td>
<td>$s', e' \leftarrow \chi$</td>
</tr>
<tr>
<td>$a \leftarrow \text{Parse}(	ext{XOF}(seed))$</td>
<td>$a \leftarrow \text{Parse}(	ext{XOF}(seed))$</td>
</tr>
<tr>
<td>$s, e \leftarrow \chi$</td>
<td>$s', e' \leftarrow \chi$</td>
</tr>
<tr>
<td>$b \leftarrow as + e$</td>
<td>$a \leftarrow \text{Parse}(	ext{XOF}(seed))$</td>
</tr>
<tr>
<td>$v' \leftarrow us$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$v \leftarrow bs'$</td>
</tr>
</tbody>
</table>
### How to build a KEM, part 2

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{seed} \leftarrow {0, 1}^{256}$</td>
<td>$\text{s}', \text{e}' \leftarrow \chi$</td>
</tr>
<tr>
<td>$\text{a} \leftarrow \text{Parse} (\text{XOF} (\text{seed}))$</td>
<td>$\text{a} \leftarrow \text{Parse} (\text{XOF} (\text{seed}))$</td>
</tr>
<tr>
<td>$\text{s}, \text{e} \leftarrow \chi$</td>
<td>$\text{u} \leftarrow \text{as}' + \text{e}'$</td>
</tr>
<tr>
<td>$\text{b} \leftarrow \text{as} + \text{e}$</td>
<td>$\text{v} \leftarrow \text{bs}'$</td>
</tr>
<tr>
<td>$\text{v}' \leftarrow \text{us}$</td>
<td>$\text{k} \leftarrow {0, 1}^n$</td>
</tr>
<tr>
<td>$\rightarrow$</td>
<td>$\text{k} \leftarrow \text{Encode} (\text{k})$</td>
</tr>
<tr>
<td>$\leftarrow \text{c}$</td>
<td>$\text{c} \leftarrow \text{v} + \text{k}$</td>
</tr>
</tbody>
</table>
### How to build a KEM, part 2

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>seed ← $\{0, 1\}^{256}</code></td>
<td><code>s’, e’, e'' ← $\chi</code></td>
</tr>
<tr>
<td><code>a ← Parse(XOF(seed))</code></td>
<td><code>a ← Parse(XOF(seed))</code></td>
</tr>
<tr>
<td><code>s, e ← $\chi</code></td>
<td><code>s’, e’, e'' ← $\chi</code></td>
</tr>
<tr>
<td><code>b ← as + e</code></td>
<td><code>a ← Parse(XOF(seed))</code></td>
</tr>
<tr>
<td><code>v’ ← us</code></td>
<td><code>c ← v + k</code></td>
</tr>
</tbody>
</table>
How to build a KEM, part 2

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>seed $\leftarrow {0, 1}^{256}$</td>
<td></td>
</tr>
<tr>
<td>$a \leftarrow \text{Parse}(\text{XOF}(\text{seed}))$</td>
<td>$s', e', e'' \leftarrow \chi$</td>
</tr>
<tr>
<td>$s, e \leftarrow \chi$</td>
<td></td>
</tr>
<tr>
<td>$b \leftarrow as + e$</td>
<td>$a \leftarrow \text{Parse}(\text{XOF}(\text{seed}))$</td>
</tr>
<tr>
<td></td>
<td>$u \leftarrow as' + e'$</td>
</tr>
<tr>
<td></td>
<td>$v \leftarrow bs' + e''$</td>
</tr>
<tr>
<td></td>
<td>$k \leftarrow {0, 1}^n$</td>
</tr>
<tr>
<td>$v' \leftarrow us$</td>
<td>$k \leftarrow \text{Encode}(k)$</td>
</tr>
<tr>
<td>$k' \leftarrow c - v'$</td>
<td>$c \leftarrow v + k$</td>
</tr>
</tbody>
</table>
How to build a KEM, part 2

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$seed \leftarrow {0, 1}^{256}$</td>
<td>$s', e', e'' \leftarrow \chi$</td>
</tr>
<tr>
<td>$a \leftarrow \text{Parse}(\text{XOF}(seed))$</td>
<td>$a \leftarrow \text{Parse}(\text{XOF}(seed))$</td>
</tr>
<tr>
<td>$s, e \leftarrow \chi$</td>
<td>$s', e', e'' \leftarrow \chi$</td>
</tr>
<tr>
<td>$b \leftarrow as + e$</td>
<td>$a \leftarrow \text{Parse}(\text{XOF}(seed))$</td>
</tr>
<tr>
<td>$v' \leftarrow us$</td>
<td>$u \leftarrow as' + e'$</td>
</tr>
<tr>
<td>$k' \leftarrow c - v'$</td>
<td>$v \leftarrow bs' + e''$</td>
</tr>
<tr>
<td>$\mu \leftarrow \text{Extract}(k')$</td>
<td>$k \leftarrow \text{Encode}(k)$</td>
</tr>
<tr>
<td>$k \leftarrow \text{Encode}(k)$</td>
<td>$k \leftarrow \text{Encode}(k)$</td>
</tr>
<tr>
<td>$v' \leftarrow us$</td>
<td>$c \leftarrow v + k$</td>
</tr>
<tr>
<td>$\mu \leftarrow \text{Extract}(k)$</td>
<td>$\mu \leftarrow \text{Extract}(k)$</td>
</tr>
</tbody>
</table>
### How to build a KEM, part 2

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>seed</strong> ← ({0, 1}^{256})</td>
<td><strong>s’, e’, e’’</strong> ← (\chi)</td>
</tr>
<tr>
<td>a←Parse(XOF(seed))</td>
<td>a←Parse(XOF(seed))</td>
</tr>
<tr>
<td>s, e ← (\chi)</td>
<td><strong>(b, seed)</strong> →</td>
</tr>
<tr>
<td>b←as + e</td>
<td>a←Parse(XOF(seed))</td>
</tr>
<tr>
<td>s’, e’, e’’ ← (\chi)</td>
<td><strong>u←as’ + e’</strong></td>
</tr>
<tr>
<td><strong>v’←us</strong></td>
<td><strong>v←bs’ + e’’</strong></td>
</tr>
<tr>
<td><strong>k’←c − v’</strong></td>
<td><strong>k ← {0, 1}^n</strong></td>
</tr>
<tr>
<td><strong>μ←Extract(k’)</strong></td>
<td><strong>μ←Extract(k</strong></td>
</tr>
</tbody>
</table>

This is LPR encryption, written as KEM (except for generation of a)
• Encoding in LPR encryption: map $n$ bits to $n$ coefficients:
  • A zero bit maps to 0
  • A one bit maps to $q/2$
• Idea: Noise affects low bits of coefficients, put data into high bits
• Encoding in LPR encryption: map \( n \) bits to \( n \) coefficients:
  • A zero bit maps to 0
  • A one bit maps to \( q/2 \)

• Idea: Noise affects low bits of coefficients, put data into high bits

• Decode: map coefficient into \([−q/2, q/2]\)
  • Closer to 0 (i.e., in \([−q/4, q/4]\)): set bit to zero
  • Closer to \(±q/2\): set bit to one
From passive to CCA security

- The base scheme does not have active security
- Attacker can choose arbitrary noise, learns $s$ from failures
From passive to CCA security

- The base scheme does not have active security
- Attacker can choose arbitrary noise, learns $s$ from failures
- Fujisaki-Okamoto transform (sketched):

<table>
<thead>
<tr>
<th>Alice (Server)</th>
<th>Bob (Client)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Gen()</strong>:</td>
<td><strong>Enc(seed, b)</strong>:</td>
</tr>
<tr>
<td>$\text{pk, sk} \leftarrow \text{KeyGen()}$</td>
<td>$x \leftarrow {0, \ldots, 255}^{32}$</td>
</tr>
<tr>
<td>seed, $b \leftarrow \text{pk}$</td>
<td>$k, \text{coins} \leftarrow \text{SHA3-512}(x)$</td>
</tr>
</tbody>
</table>

Dec($s, (u, v)$):

- $x' \leftarrow \text{Decrypt}(s, (u, v))$
- $k', \text{coins}' \leftarrow \text{SHA3-512}(x')$
- $u', v' \leftarrow \text{Encrypt}((\text{seed}, b), x', \text{coins}')$
- **verify if** $(u', v') = (u, v)$
Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters $q$ and $p = 3$
Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters $q$ and $p = 3$
- **Keygen:**
  - Find $f, g \in \mathcal{R}_q$ and $f_q = f^{-1} \mod q$, $f_p = f^{-1} \mod p$
  - public key: $h = pf_qg$, secret key: $(f, f_p)$
Design space 0: The NTRU approach

• Historically first: NTRU
• Use parameters $q$ and $p = 3$
• Keygen:
  • Find $f, g \in \mathcal{R}_q$ and $f_q = f^{-1} \mod q$, $f_p = f^{-1} \mod p$
  • public key: $h = p f_q g$, secret key: $(f, f_p)$
• Encrypt:
  • Map message $m$ to $m \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  • Sample random small-coefficient polynomial $r \in \mathcal{R}_q$
  • Compute ciphertext $e = r \cdot h + m$
Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters $q$ and $p = 3$
- **Keygen:**
  - Find $f, g \in \mathcal{R}_q$ and $f_q = f^{-1} \mod q$, $f_p = f^{-1} \mod p$
  - Public key: $h = pf_qg$, secret key: $(f, f_p)$
- **Encrypt:**
  - Map message $m$ to $m \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  - Sample random small-coefficient polynomial $r \in \mathcal{R}_q$
  - Compute ciphertext $e = r \cdot h + m$
- **Decrypt:**
  - Compute $v = f \cdot e$
Design space 0: The NTRU approach

• Historically first: NTRU
• Use parameters $q$ and $p = 3$

• **Keygen:**
  • Find $f, g \in \mathcal{R}_q$ and $f_q = f^{-1} \mod q$, $f_p = f^{-1} \mod p$
  • public key: $h = pf_q g$, secret key: $(f, f_p)$

• **Encrypt:**
  • Map message $m$ to $m \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  • Sample random small-coefficient polynomial $r \in \mathcal{R}_q$
  • Compute ciphertext $e = r \cdot h + m$

• **Decrypt:**
  • Compute $v = f \cdot e = f \cdot (r \cdot h + m)$

• Advantages/Disadvantages compared to LPR:
  • Asymptotically weaker than Ring-LWE approach
  • Slower keygen, but faster encryption/decryption
Design space 0: The NTRU approach

• Historically first: NTRU
• Use parameters $q$ and $p = 3$

• **Keygen:**
  - Find $f, g \in \mathcal{R}_q$ and $f_q = f^{-1} \mod q$, $f_p = f^{-1} \mod p$
  - public key: $h = pf_qg$, secret key: $(f, f_p)$

• **Encrypt:**
  - Map message $m$ to $\mathbf{m} \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  - Sample random small-coefficient polynomial $\mathbf{r} \in \mathcal{R}_q$
  - Compute ciphertext $\mathbf{e} = \mathbf{r} \cdot h + \mathbf{m}$

• **Decrypt:**
  - Compute $\mathbf{v} = f \cdot \mathbf{e} = f \cdot (\mathbf{r} \cdot h + \mathbf{m}) = f(\mathbf{r} \cdot (pf_qg) + \mathbf{m})$
Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters $q$ and $p = 3$

**Keygen:**
- Find $f, g \in \mathcal{R}_q$ and $f_q = f^{-1} \mod q$, $f_p = f^{-1} \mod p$
- public key: $h = pf_qg$, secret key: $(f, f_p)$

**Encrypt:**
- Map message $m$ to $m \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
- Sample random small-coefficient polynomial $r \in \mathcal{R}_q$
- Compute ciphertext $e = r \cdot h + m$

**Decrypt:**
- Compute $v = f \cdot e = f \cdot (r \cdot h + m) = f(r \cdot (pf_qg) + m) = prg + f \cdot m$
Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters $q$ and $p = 3$

**Keygen:**
- Find $f, g \in \mathcal{R}_q$ and $f_q = f^{-1} \mod q$, $f_p = f^{-1} \mod p$
- public key: $h = pf_q g$, secret key: $(f, f_p)$

**Encrypt:**
- Map message $m$ to $m \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
- Sample random small-coefficient polynomial $r \in \mathcal{R}_q$
- Compute ciphertext $e = r \cdot h + m$

**Decrypt:**
- Compute $v = f \cdot e = f \cdot (r \cdot h + m) = f(r \cdot (pf_q g) + m) = prg + f \cdot m$
- Compute $m = v \cdot f_p \mod p$
Design space 0: The NTRU approach

- Historically first: NTRU
- Use parameters $q$ and $p = 3$
- **Keygen:**
  - Find $f, g \in \mathcal{R}_q$ and $f_q = f^{-1} \mod q$, $f_p = f^{-1} \mod p$
  - public key: $h = pf_q g$, secret key: $(f, f_p)$
- **Encrypt:**
  - Map message $m$ to $m \in \mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$
  - Sample random small-coefficient polynomial $r \in \mathcal{R}_q$
  - Compute ciphertext $e = r \cdot h + m$
- **Decrypt:**
  - Compute $v = f \cdot e = f \cdot (r \cdot h + m) = f(r \cdot (pf_q g) + m) = prg + f \cdot m$
  - Compute $m = v \cdot f_p \mod p$
- Advantages/Disadvantages compared to LPR:
  - Asymptotically weaker than Ring-LWE approach
  - Slower keygen, but faster encryption/decryption
Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - $q$ typically either prime or a power of two
  - $f$ typically of degree between 512 and 1024
Design space 1: What ring?

- Structured lattice-based schemes use ring $R_q = \mathbb{Z}_q[X]/f$
  - $q$ typically either prime or a power of two
  - $f$ typically of degree between 512 and 1024
- First option: $q = 2^k, f = (X^n - 1), n$ prime (NTRU)
Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - $q$ typically either prime or a power of two
  - $f$ typically of degree between 512 and 1024
- **First option**: $q = 2^k, f = (X^n - 1), n$ prime (NTRU)
- **Second option**: $q = 2^k, f = (X^n + 1), n = 2^m$ (Saber)
- Third option:
- Fourth option:
- Fifth option:
- Sixth option:

No proof that any option is more or less secure

NTRU Prime advertises “less structure” in their $R_q$

NewHope and Kyber have fastest (NTT-based) arithmetic
Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - $q$ typically either prime or a power of two
  - $f$ typically of degree between 512 and 1024
- First option: $q = 2^k$, $f = (X^n - 1)$, $n$ prime (NTRU)
- Second option: $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)
- Third option: $q = 2^k$, $f = \Phi_{n+1}$, $n + 1$ prime (Round5)
Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - $q$ typically either prime or a power of two
  - $f$ typically of degree between 512 and 1024
- **First option**: $q = 2^k, f = (X^n - 1), n$ prime  (NTRU)
- **Second option**: $q = 2^k, f = (X^n + 1), n = 2^m$  (Saber)
- **Third option**: $q = 2^k, f = \Phi_{n+1}, n + 1$ prime  (Round5)
- **Fourth option**: $q$ prime, $f = (X^n + 1) = \Phi_{2n}, n = 2^m$  (NewHope, Kyber, LAC)
- **Sixth option**: ThreeBears works on large integers instead of polynomials
- No proof that any option is more or less secure
- NTRU Prime advertises “less structure” in their $\mathcal{R}_q$
Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - $q$ typically either prime or a power of two
  - $f$ typically of degree between 512 and 1024
- **First option:** $q = 2^k$, $f = (X^n - 1)$, $n$ prime (NTRU)
- **Second option:** $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)
- **Third option:** $q = 2^k$, $f = \Phi_{n+1}$, $n + 1$ prime (Round5)
- **Fourth option:** $q$ prime, $f = (X^n + 1) = \Phi_{2n}$, $n = 2^m$ (NewHope, Kyber, LAC)
- **Fifth option:** $q$ prime, $f = (X^n - X - 1)$ irreducible, $n$ prime (NTRU Prime)
Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - $q$ typically either prime or a power of two
  - $f$ typically of degree between 512 and 1024
- **First option:** $q = 2^k, f = (X^n - 1), n$ prime  (NTRU)
- **Second option:** $q = 2^k, f = (X^n + 1), n = 2^m$  (Saber)
- **Third option:** $q = 2^k, f = \Phi_{n+1}, n + 1$ prime  (Round5)
- **Fourth option:** $q$ prime, $f = (X^n + 1) = \Phi_{2n}, n = 2^m$  (NewHope, Kyber, LAC)
- **Fifth option:** $q$ prime, $f = (X^n - X - 1)$ irreducible, $n$ prime  (NTRU Prime)
- **Sixth option:** ThreeBears works on large integers instead of polynomials

- No proof that any option is more or less secure
- NTRU Prime advertises "less structure" in their $\mathcal{R}_q$
- NewHope and Kyber have fastest (NTT-based) arithmetic
Design space 1: What ring?

• Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  • $q$ typically either prime or a power of two
  • $f$ typically of degree between 512 and 1024
• First option: $q = 2^k, f = (X^n - 1), n$ prime (NTRU)
• Second option: $q = 2^k, f = (X^n + 1), n = 2^m$ (Saber)
• Third option: $q = 2^k, f = \Phi_{n+1}, n+1$ prime (Round5)
• Fourth option: $q$ prime, $f = (X^n + 1) = \Phi_{2n}, n = 2^m$ (NewHope, Kyber, LAC)
• Fifth option: $q$ prime, $f = (X^n - X - 1)$ irreducible, $n$ prime (NTRU Prime)
• Sixth option: ThreeBears works on large integers instead of polynomials
• No proof that any option is more or less secure
Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - $q$ typically either prime or a power of two
  - $f$ typically of degree between 512 and 1024
- **First option**: $q = 2^k$, $f = (X^n - 1)$, $n$ prime (NTRU)
- **Second option**: $q = 2^k$, $f = (X^n + 1)$, $n = 2^m$ (Saber)
- **Third option**: $q = 2^k$, $f = \Phi_{n+1}$, $n + 1$ prime (Round5)
- **Fourth option**: $q$ prime, $f = (X^n + 1) = \Phi_{2n}$, $n = 2^m$ (NewHope, Kyber, LAC)
- **Fifth option**: $q$ prime, $f = (X^n - X - 1)$ irreducible, $n$ prime (NTRU Prime)
- ** Sixth option**: ThreeBears works on large integers instead of polynomials
- No proof that any option is more or less secure
- NTRU Prime advertises “less structure” in their $\mathcal{R}_q$
Design space 1: What ring?

- Structured lattice-based schemes use ring $\mathcal{R}_q = \mathbb{Z}_q[X]/f$
  - $q$ typically either prime or a power of two
  - $f$ typically of degree between 512 and 1024
- **First option**: $q = 2^k, f = (X^n - 1), n$ prime (NTRU)
- **Second option**: $q = 2^k, f = (X^n + 1), n = 2^m$ (Saber)
- **Third option**: $q = 2^k, f = \Phi_{n+1}, n + 1$ prime (Round5)
- **Fourth option**: $q$ prime, $f = (X^n + 1) = \Phi_{2n}, n = 2^m$ (NewHope, Kyber, LAC)
- **Fifth option**: $q$ prime, $f = (X^n - X - 1)$ irreducible, $n$ prime (NTRU Prime)
- **Sixth option**: ThreeBears works on large integers instead of polynomials
- No proof that any option is more or less secure
- NTRU Prime advertises “less structure” in their $\mathcal{R}_q$
- NewHope and Kyber have fastest (NTT-based) arithmetic
Design space 2: module vs. ring?

• “Traditionally”, work directly with elements of $\mathcal{R}_q$ (“Ring-LWE”)
• Alternative: Module-LWE (MLWE):
  • Choose smaller $n$, e.g., $n = 256$ (Kyber, Saber, ThreeBears)
  • Work with small-dimension matrices and vectors over $\mathcal{R}_q$
Design space 2: module vs. ring?

- “Traditionally”, work directly with elements of $\mathcal{R}_q$ (“Ring-LWE”)
- Alternative: Module-LWE (MLWE):
  - Choose smaller $n$, e.g., $n = 256$ (Kyber, Saber, ThreeBears)
  - Work with small-dimension matrices and vectors over $\mathcal{R}_q$
- MLWE encrypts shorter messages than Ring-LWE
Design space 2: module vs. ring?

- "Traditionally", work directly with elements of $\mathcal{R}_q$ ("Ring-LWE")
- Alternative: Module-LWE (MLWE):
  - Choose smaller $n$, e.g., $n = 256$ (Kyber, Saber, ThreeBears)
  - Work with small-dimension matrices and vectors over $\mathcal{R}_q$
- MLWE encrypts shorter messages than Ring-LWE
- MLWE eliminates some of the structure of Ring-LWE
Design space 2: module vs. ring?

• “Traditionally”, work directly with elements of $\mathcal{R}_q$ (“Ring-LWE”)
• Alternative: Module-LWE (MLWE):
  • Choose smaller $n$, e.g., $n = 256$ (Kyber, Saber, ThreeBears)
  • Work with small-dimension matrices and vectors over $\mathcal{R}_q$
• MLWE encrypts shorter messages than Ring-LWE
• MLWE eliminates some of the structure of Ring-LWE
• MLWE can very easily scale security (change dimension of matrix):
  • Optimize arithmetic in $\mathcal{R}_q$ once
  • Use same optimized $\mathcal{R}_q$ arithmetic for all security levels
Design space 3: what noise?

• Need to sample noise (for LWE schemes) and small secrets
• More noise means
  • more security from the underlying hard problem
  • higher failure probability of decryption
Design space 3: what noise?

- Need to sample noise (for LWE schemes) and small secrets
- More noise means
  - more security from the underlying hard problem
  - higher failure probability of decryption
- Three main choices to make:
  - **Narrow or wide noise**
    - Narrow noise (e.g., in \{-1, 0, 1\}) not conservative
    - Wide noise requires larger $q$ (or more failures)
    - Larger $q$ means larger public key and ciphertext

LWE or LWR
- LWE considered more conservative (independent noise)
- LWR easier to implement (no noise sampling)
- LWR allows more compact public key and ciphertext

Fixed-weight noise or not?
- Fixed-weight noise needs random permutation (sorting)
- Naive implementations leak secrets through timing
- Advantage of fixed-weight: easier to bound (or eliminate) decryption failures
Design space 3: what noise?

• Need to sample noise (for LWE schemes) and small secrets
• More noise means
  • more security from the underlying hard problem
  • higher failure probability of decryption
• Three main choices to make:
  • Narrow or wide noise
    • Narrow noise (e.g., in \{-1, 0, 1\}) not conservative
    • Wide noise requires larger $q$ (or more failures)
    • Larger $q$ means larger public key and ciphertext
  • LWE or LWR
    • LWE considered more conservative (independent noise)
    • LWR easier to implement (no noise sampling)
    • LWR allows more compact public key and ciphertext
Design space 3: what noise?

• Need to sample noise (for LWE schemes) and small secrets
• More noise means
  • more security from the underlying hard problem
  • higher failure probability of decryption
• Three main choices to make:
  • Narrow or wide noise
    • Narrow noise (e.g., in \{-1, 0, 1\}) not conservative
    • Wide noise requires larger $q$ (or more failures)
    • Larger $q$ means larger public key and ciphertext
  • LWE or LWR
    • LWE considered more conservative (independent noise)
    • LWR easier to implement (no noise sampling)
    • LWR allows more compact public key and ciphertext
  • Fixed-weight noise or not?
    • Fixed-weight noise needs random permutation (sorting)
    • Naive implementations leak secrets through timing
    • Advantage of fixed-weight: easier to bound (or eliminate) decryption failures
Design space 4: allow failures?

- Can avoid decryption failures entirely (NTRU, NTRU Prime)
- Advantage:
  - Easier CCA security transform and analysis
- Disadvantage:
  - Need to limit noise (or have larger $q$)
Design space 4: allow failures?

- Can avoid decryption failures entirely (NTRU, NTRU Prime)
- Advantage:
  - Easier CCA security transform and analysis
- Disadvantage:
  - Need to limit noise (or have larger $q$)
- For passive-security-only can go the other way:
  - Allow failure probability of, e.g., $2^{-30}$
  - Reduce size of public key and ciphertext
Design space 4: allow failures?

- Can avoid decryption failures entirely (NTRU, NTRU Prime)
- Advantage:
  - Easier CCA security transform and analysis
- Disadvantage:
  - Need to limit noise (or have larger $q$)
- For passive-security-only can go the other way:
  - Allow failure probability of, e.g., $2^{-30}$
  - Reduce size of public key and ciphertext
- Active (CCA) security needs negligible failure probability
Design space 5: public parameters?

• “Traditional” approach to choosing \( a \) in LWE/LWR schemes:
  
  “Let \( a \) be a uniformly random . . . ”
Design space 5: public parameters?

• “Traditional” approach to choosing \( a \) in LWE/LWR schemes:
  “Let \( a \) be a uniformly random…”

• Before NewHope: real-world approach: generate fixed \( a \) once
“Traditional” approach to choosing $a$ in LWE/LWR schemes: “Let $a$ be a uniformly random...”

Before NewHope: *real-world* approach: generate fixed $a$ once

What if $a$ is backdoored?

Parameter-generating authority can break key exchange

“Solution”: Nothing-up-my-sleeves (involves endless discussion!)
Design space 5: public parameters?

• “Traditional” approach to choosing \( a \) in LWE/LWR schemes:
  “Let \( a \) be a uniformly random . . .”

• Before NewHope: real-world approach: generate fixed \( a \) once

• What if \( a \) is backdoored?

• Parameter-generating authority can break key exchange

• “Solution”: Nothing-up-my-sleeves (involves endless discussion!)

• Even without backdoor:
  • Perform massive precomputation based on \( a \)
  • Use precomputation to break all key exchanges
  • Infeasible today, but who knows. . .
  • Attack in the spirit of Logjam
Design space 5: public parameters?

• “Traditional” approach to choosing $a$ in LWE/LWR schemes:
  “Let $a$ be a uniformly random . . .”

• Before NewHope: real-world approach: generate fixed $a$ once

• What if $a$ is backdoored?

• Parameter-generating authority can break key exchange

• “Solution”: Nothing-up-my-sleeves (involves endless discussion!)

• Even without backdoor:
  • Perform massive precomputation based on $a$
  • Use precomputation to break all key exchanges
  • Infeasible today, but who knows . . .
  • Attack in the spirit of Logjam

• Solution in NewHope: Choose a fresh $a$ every time

• Server can cache $a$ for some time (e.g., 1h)

• All NIST PQC candidates now use this approach
• Ring-LWE/LWR schemes work with polynomials of $> 256$ coefficients
• “Encrypt” messages of $> 256$ bits
• **Need to encrypt** only 256-bit key
• Question: How do we put those additional bits to use?
• Answer: Use error-correcting code (ECC) to reduce failure probability
Design space 6: error-correcting codes?

- Ring-LWE/LWR schemes work with polynomials of > 256 coefficients
- “Encrypt” messages of > 256 bits
- **Need to encrypt** only 256-bit key
- Question: How do we put those additional bits to use?
- Answer: Use error-correcting code (ECC) to reduce failure probability
- NewHope: very simple threshold decoding
Design space 6: error-correcting codes?

- Ring-LWE/LWR schemes work with polynomials of > 256 coefficients
- “Encrypt” messages of > 256 bits
- Need to encrypt only 256-bit key
- Question: How do we put those additional bits to use?
- Answer: Use error-correcting code (ECC) to reduce failure probability
- NewHope: very simple threshold decoding
- LAC, Round5: more advanced ECC
  - Correct more errors, obtain smaller public key and ciphertext
  - More complex to implement, in particular without leaking through timing
Design space 7: CCA security?

• Ephemeral key exchange does not need CCA security
• Can offer passively secure version
• Protocols will combine this with signatures for authentication
Design space 7: CCA security?

- Ephemeral key exchange does not need CCA security
- Can offer passively secure version
- Protocols will combine this with signatures for authentication
- **Advantages:**
  - Higher failure probability → more compact
  - Simpler to implement, no CCA transform
  - More flexibility for secret/noise generation
- **Disadvantages:**
  - Less robust (will somebody reuse keys?)
  - More options (CCA vs. CPA): easier to make mistakes
Ephemeral key exchange does not need CCA security
Can offer passively secure version
Protocols will combine this with signatures for authentication

**Advantages:**
- Higher failure probability → more compact
- Simpler to implement, no CCA transform
- More flexibility for secret/noise generation

**Disadvantages:**
- Less robust (will somebody reuse keys?)
- More options (CCA vs. CPA): easier to make mistakes
Design space 8: CCA transforms

- General Fujisaki-Okamoto principle is the same for most KEMs (exception: NTRU)
- Tweaks to FO transform:
  - Hash public-key into coins: multitarget protection (for non-zero failure probability)
• General Fujisaki-Okamoto principle is the same for most KEMs (exception: NTRU)

• Tweaks to FO transform:
  • Hash public-key into coins: multitarget protection (for non-zero failure probability)
  • Hash public-key into shared key: KEM becomes contributory
Design space 8: CCA transforms

- General Fujisaki-Okamoto principle is the same for most KEMs (exception: NTRU)
- Tweaks to FO transform:
  - Hash public-key into coins: multitarget protection (for non-zero failure probability)
  - Hash public-key into shared key: KEM becomes contributory
  - Hash ciphertext into shared key: more robust (?)

How to handle rejection?
- Return special symbol (-1): explicit
- Return $H(s, C)$ for secret $s$: implicit

As of round 2, no proposal uses explicit rejection
- Would break some security reduction
- More robust in practice (return value always 0)
Design space 8: CCA transforms

• General Fujisaki-Okamoto principle is the same for most KEMs (exception: NTRU)

• Tweaks to FO transform:
  • Hash public-key into coins: multitarget protection (for non-zero failure probability)
  • Hash public-key into shared key: KEM becomes contributory
  • Hash ciphertext into shared key: more robust (?)

• How to handle rejection?
  • Return special symbol (return -1): explicit
  • Return H(s, C) for secret s: implicit
• General Fujisaki-Okamoto principle is the same for most KEMs (exception: NTRU)
• Tweaks to FO transform:
  • Hash public-key into coins: multitarget protection (for non-zero failure probability)
  • Hash public-key into shared key: KEM becomes contributory
  • Hash ciphertext into shared key: more robust (?)
• How to handle rejection?
  • Return special symbol (return -1): explicit
  • Return H(s, C) for secret s: implicit
• As of round 2, no proposal uses explicit rejection
  • Would break some security reduction
  • More robust in practice (return value alwas 0)
Summary

- Lattice-based KEMs offer best overall performance in the PQ world
- Many tradeoffs between
  - Security (including passive vs. active)
  - Failure rate
  - Size
  - Speed
- More information about NIST PQC:
  - [https://pqc-wiki.fau.edu/](https://pqc-wiki.fau.edu/)
Exercise: the Wookie encapsulation mechanism

Download [https://cryptojedi.org/wookie.tar.gz](https://cryptojedi.org/wookie.tar.gz)
Slides at [https://cryptojedi.org/latticekems.pdf](https://cryptojedi.org/latticekems.pdf)

- CPA-secure “LPR KEM”, see slide 7
- Work in polynomial ring \( \mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1) \)
- Parameters \( q = 4096, n = 1024 \)
- Centered binomial noise with \( k = 8 \)
- “Messages” have \( n \) bits \( \Rightarrow \) trivial encoding (see slide 8)
Exercise: the Wookie encapsulation mechanism

Download [https://cryptojedi.org/wookie.tar.gz](https://cryptojedi.org/wookie.tar.gz)
Slides at [https://cryptojedi.org/latticekems.pdf](https://cryptojedi.org/latticekems.pdf)

- CPA-secure “LPR KEM”, see slide 7
- Work in polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$
- Parameters $q = 4096, n = 1024$
- Centered binomial noise with $k = 8$
- “Messages” have $n$ bits $\Rightarrow$ trivial encoding (see slide 8)

1. Implement arithmetic in $\mathcal{R}_q$ (file `poly.c`)
2. Implement the Wookie KEM (file `kem.c`)
Exercise: the Wookie encapsulation mechanism

Download [https://cryptojedi.org/wookie.tar.gz](https://cryptojedi.org/wookie.tar.gz)
Slides at [https://cryptojedi.org/latticekems.pdf](https://cryptojedi.org/latticekems.pdf)

- CPA-secure “LPR KEM”, see slide 7
- Work in polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$
- Parameters $q = 4096$, $n = 1024$
- Centered binomial noise with $k = 8$
- “Messages” have $n$ bits $\Rightarrow$ trivial encoding (see slide 8)

1. Implement arithmetic in $\mathcal{R}_q$ (file poly.c)
2. Implement the Wookie KEM (file kem.c)

- make builds various unit tests in test/ subdirectory
- Running test.sh in test/ subdirectory runs all tests
Centered binomial noise with $k = 8$

- Let $HW(b)$ be the Hamming weight of a byte $b$
Let $\text{HW}(b)$ be the Hamming weight of a byte $b$.

To sample one coefficient $p[i]$ of a polynomial in $\mathcal{R}_q$:

- Sample two uniformly random bytes $a$ and $b$.
- Set $p[i] = \text{HW}(a) - \text{HW}(b)$. 
Centered binomial noise with \( k = 8 \)

- Let \( \text{HW}(b) \) be the Hamming weight of a byte \( b \)
- To sample one coefficient \( p[i] \) of a polynomial in \( \mathcal{R}_q \):
  - Sample two uniformly random bytes \( a \) and \( b \)
  - Set \( p[i] = \text{HW}(a) - \text{HW}(b) \)
- Resulting coefficient will be in \( \{-8, \ldots, 8\} \)
- Sampling a polynomial needs \( 2n = 2048 \) uniformly random bytes
Some remarks

- Software skeleton assumes Linux system
- Need basic build tools (\texttt{make, gcc, ...}) installed:
  \begin{verbatim}
  apt install build-essential
  \end{verbatim}
- Some unit tests and \texttt{test.sh} script assume Sage to be installed
  \begin{verbatim}
  apt install sagemath
  \end{verbatim}
- Can also download pre-compiled binaries of Sage:
  \url{https://doc.sagemath.org/html/en/installation/binary.html}