CRYSTALS–Kyber

Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, Damien Stehlé

authors@pq-crystals.org
https://pq-crystals.org/kyber
August 23, 2019
Kyber.CPAPKE: LPR encryption or “Noisy ElGamal”

\[ s, e \leftarrow \chi \]
\[ sk = s, pk = t = As + e \]
\[ r, e_1, e_2 \leftarrow \chi \]
\[ u \leftarrow A^T r + e_1 \]
\[ v \leftarrow t^T r + e_2 + Enc(m) \]
\[ c = (u, v) \]

\[ m = Dec(v - s^T u) \]
Kyber.CPAPKE: LPR encryption or “Noisy ElGamal”

\[ s, e \leftarrow \chi \]
\[ sk = s, pk = t = As + e \]
\[ r, e_1, e_2 \leftarrow \chi \]
\[ u \leftarrow A^T r + e_1 \]
\[ v \leftarrow t^T r + e_2 + Enc(m) \]
\[ c = (u, v) \]
\[ m = Dec(v - s^T u) \]

Kyber.CCAKEM: CCA-secure KEM via tweaked FO transform

- Use implicit rejection
- Hash public key into seed and shared key
- Hash ciphertext into shared key
- Use Keccak-based functions for all hashes and XOF
• Use MLWE instead of LWE or RLWE
• Use MLWE instead of LWE or RLWE
• Use $\mathcal{R} = \mathbb{Z}_q[X]/(X^{256} + 1)$ with $q = 7681$
• Use MLWE instead of LWE or RLWE

• Use $\mathcal{R} = \mathbb{Z}_q[X]/(X^{256} + 1)$ with $q = 7681$

• Use centered binomial noise
Reminder: Kyber in Round 1

- Use MLWE instead of LWE or RLWE
- Use $\mathcal{R} = \mathbb{Z}_q[X]/(X^{256} + 1)$ with $q = 7681$
- Use centered binomial noise
- Generate $A$ via $\text{XOF}(\rho)$ ("NewHope style")
• Use MLWE instead of LWE or RLWE
• Use $\mathcal{R} = \mathbb{Z}_q[X]/(X^{256} + 1)$ with $q = 7681$
• Use centered binomial noise
• Generate $A$ via $\text{XOF}(\rho)$ (“NewHope style”)
• Compress ciphertexts (round off least-significant bits)
• Use MLWE instead of LWE or RLWE

• Use $\mathcal{R} = \mathbb{Z}_q[X]/(X^{256} + 1)$ with $q = 7681$

• Use centered binomial noise

• Generate $\mathbf{A}$ via XOF($\rho$) (“NewHope style”)

• Compress ciphertexts (round off least-significant bits)

• Compress public keys
“We note that a potential issue is that the security proof does not directly apply to Kyber itself, but rather to a modified version of the scheme which does not compress the public key.”

—NIST IR 8240
1. Remove the public-key compression
   • Proof now applies to Kyber itself
   • However, bandwidth requirement increases
Main changes in round 2

1. Remove the public-key compression
   - Proof now applies to Kyber itself
   - However, bandwidth requirement increases
2. Reduce parameter $q$ to 3329
   - Bandwidth requirement decreases
3. Update ciphertext-compression parameters
### Main changes in round 2

#### Kyber sizes, round 1 vs. round 2

<table>
<thead>
<tr>
<th>Kyber</th>
<th>$k = 2$, level 1</th>
<th>$k = 3$, level 3</th>
<th>$k = 4$, level 5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>round 1, sizes in bytes</strong></td>
<td><strong>round 2, sizes in bytes</strong></td>
<td><strong>round 2, sizes in bytes</strong></td>
<td><strong>round 2, sizes in bytes</strong></td>
</tr>
<tr>
<td><strong>pk:</strong></td>
<td>736</td>
<td>800</td>
<td>1440</td>
</tr>
<tr>
<td><strong>ct:</strong></td>
<td>800</td>
<td>736</td>
<td>1504</td>
</tr>
<tr>
<td><strong>pk:</strong></td>
<td>1088</td>
<td>1184</td>
<td>1568</td>
</tr>
<tr>
<td><strong>ct:</strong></td>
<td>1152</td>
<td>1088</td>
<td>1568</td>
</tr>
</tbody>
</table>

**Kyber512 ($k = 2$, level 1)**

**Kyber768 ($k = 3$, level 3)**

**Kyber1024 ($k = 4$, level 5)**
Main changes in round 2

1. Remove the public-key compression
   - Proof now applies to Kyber itself
   - However, bandwidth requirement increases
2. Reduce parameter $q$ to 3329
   - Bandwidth requirement decreases
3. Update ciphertext-compression parameters
4. Update the specification of the NTT (inspired by NTTRU)
   - Even faster polynomial multiplication
Main changes in round 2

1. Remove the public-key compression
   - Proof now applies to Kyber itself
   - However, bandwidth requirement increases
2. Reduce parameter $q$ to 3329
   - Bandwidth requirement decreases
3. Update ciphertext-compression parameters
4. Update the specification of the NTT (inspired by NTTRU)
   - Even faster polynomial multiplication
5. Reduce noise parameter to $\eta = 2$
   - Faster noise sampling
Main changes in round 2

1. Remove the public-key compression
   - Proof now applies to Kyber itself
   - However, bandwidth requirement increases

2. Reduce parameter $q$ to 3329
   - Bandwidth requirement decreases

3. Update ciphertext-compression parameters

4. Update the specification of the NTT (inspired by NTTRU)
   - Even faster polynomial multiplication

5. Reduce noise parameter to $\eta = 2$
   - Faster noise sampling

6. Represent public key in NTT domain
   - Save several NTT computations
Kyber is fast

<table>
<thead>
<tr>
<th>Kyber512 ($k = 2$, level 1)</th>
<th>Sizes (in Bytes)</th>
<th>Haswell Cycles (AVX2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sk:</td>
<td>1632</td>
<td>gen:</td>
</tr>
<tr>
<td>pk:</td>
<td>800</td>
<td>enc:</td>
</tr>
<tr>
<td>ct:</td>
<td>736</td>
<td>dec:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kyber768 ($k = 3$, level 3)</th>
<th>Sizes (in Bytes)</th>
<th>Haswell Cycles (AVX2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sk:</td>
<td>2400</td>
<td>gen:</td>
</tr>
<tr>
<td>pk:</td>
<td>1184</td>
<td>enc:</td>
</tr>
<tr>
<td>ct:</td>
<td>1088</td>
<td>dec:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kyber1024 ($k = 4$, level 5)</th>
<th>Sizes (in Bytes)</th>
<th>Haswell Cycles (AVX2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sk:</td>
<td>3168</td>
<td>gen:</td>
</tr>
<tr>
<td>pk:</td>
<td>1568</td>
<td>enc:</td>
</tr>
<tr>
<td>ct:</td>
<td>1568</td>
<td>dec:</td>
</tr>
</tbody>
</table>
Kyber is fast and small

<table>
<thead>
<tr>
<th>Kyber512 ($k = 2$, level 1)</th>
<th>Stack usage (in Bytes)</th>
<th>Cortex-M4 Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>gen:</td>
<td>2952</td>
<td>gen: 513992</td>
</tr>
<tr>
<td>enc:</td>
<td>2552</td>
<td>enc: 652470</td>
</tr>
<tr>
<td>dec:</td>
<td>2560</td>
<td>dec: 620946</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kyber768 ($k = 3$, level 3)</th>
<th>Stack usage (in Bytes)</th>
<th>Cortex-M4 Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>gen:</td>
<td>3848</td>
<td>gen: 976205</td>
</tr>
<tr>
<td>enc:</td>
<td>3128</td>
<td>enc: 1146021</td>
</tr>
<tr>
<td>dec:</td>
<td>3072</td>
<td>dec: 1094314</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kyber1024 ($k = 4$, level 5)</th>
<th>Stack usage (in Bytes)</th>
<th>Cortex-M4 Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>gen:</td>
<td>4360</td>
<td>gen: 1574351</td>
</tr>
<tr>
<td>enc:</td>
<td>3584</td>
<td>enc: 1779192</td>
</tr>
<tr>
<td>dec:</td>
<td>3592</td>
<td>dec: 1708692</td>
</tr>
</tbody>
</table>
What are we benchmarking, really?

- More than 50% of the cycles are spent in Keccak
  - Many conservative choices in FO transform
  - Use SHAKE-128 to as XOF
  - Generally, Keccak is not very fast in software

Long-term solution: hardware-accelerated Keccak

Short-term problem:
- Benchmarks of lattice-based KEMs are really benchmarks of symmetric crypto
- Risk to make wrong decision about lattice design from "symmetrically tainted" benchmarks
- Maybe just a small problem, because lattice-based KEMs are all fast enough

Better to decide based on:
- size/bandwidth
- RAM/ROM footprint and gate count in HW
- simplicity
- how conservative designs are
- cost of SCA protection
What are we benchmarking, really?

- More than 50% of the cycles are spent in Keccak
  - Many conservative choices in FO transform
  - Use SHAKE-128 to as XOF
  - Generally, Keccak is not very fast in software
- Long-term solution: hardware-accelerated Keccak
What are we benchmarking, really?

- More than 50% of the cycles are spent in Keccak
  - Many conservative choices in FO transform
  - Use SHAKE-128 to as XOF
  - Generally, Keccak is not very fast in software
- Long-term solution: hardware-accelerated Keccak
- Short-term problem:
  - Benchmarks of lattice-based KEMs are really benchmarks of symmetric crypto
  - Risk to make wrong decision about lattice design from “symmetrically tainted” benchmarks
What are we benchmarking, really?

- More than 50% of the cycles are spent in Keccak
  - Many conservative choices in FO transform
  - Use SHAKE-128 to as XOF
  - Generally, Keccak is not very fast in software
- Long-term solution: hardware-accelerated Keccak
- Short-term problem:
  - Benchmarks of lattice-based KEMs are really benchmarks of symmetric crypto
  - Risk to make wrong decision about lattice design from “symmetrically tainted” benchmarks
- Maybe just a small problem, because lattice-based KEMs are all fast enough
What are we benchmarking, really?

- More than 50% of the cycles are spent in Keccak
  - Many conservative choices in FO transform
  - Use SHAKE-128 to as XOF
  - Generally, Keccak is not very fast in software
- Long-term solution: hardware-accelerated Keccak
- Short-term problem:
  - Benchmarks of lattice-based KEMs are really benchmarks of symmetric crypto
  - Risk to make wrong decision about \textit{lattice} design from “symmetrically tainted” benchmarks
- Maybe just a small problem, because lattice-based KEMs are all fast enough
- Better to decide based on
  - size/bandwidth
  - RAM/ROM footprint and gate count in HW
  - simplicity
  - how conservative designs are
  - cost of SCA protection
90s crypto (AES, SHA-2) is accelerated in HW!
<table>
<thead>
<tr>
<th>Kyber cycles</th>
<th>Kyber-90s cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kyber512 ($k = 2$, level 1)</td>
<td>Kyber cycles</td>
</tr>
<tr>
<td>gen: 29100</td>
<td>gen: 15792</td>
</tr>
<tr>
<td>enc: 46196</td>
<td>enc: 26612</td>
</tr>
<tr>
<td>dec: 39410</td>
<td>dec: 22248</td>
</tr>
<tr>
<td>Kyber768 ($k = 3$, level 3)</td>
<td>Kyber cycles</td>
</tr>
<tr>
<td>gen: 57340</td>
<td>gen: 25632</td>
</tr>
<tr>
<td>enc: 78692</td>
<td>enc: 39976</td>
</tr>
<tr>
<td>dec: 68620</td>
<td>dec: 33744</td>
</tr>
<tr>
<td>Kyber1024 ($k = 4$, level 5)</td>
<td>Kyber cycles</td>
</tr>
<tr>
<td>gen: 81244</td>
<td>gen: 38164</td>
</tr>
<tr>
<td>enc: 109584</td>
<td>enc: 57280</td>
</tr>
<tr>
<td>dec: 97280</td>
<td>dec: 50360</td>
</tr>
</tbody>
</table>
https://pq-crystals.org/kyber