PQC Standardization
A Vendor’s Perspective

Mike Hamburg

10 April 2024
Recap

Standardization Process → Portfolio → Implementation
Outline

- Standardization Process
- Portfolio
- Implementation
Outline: divergence between Kyber and Dilithium

<table>
<thead>
<tr>
<th></th>
<th>Kyber/ML-KEM</th>
<th>Dilithium/ML-DSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lattice Based</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>NTT</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>NTT-friendly primes</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>12 bits</td>
<td></td>
<td>23 bits</td>
</tr>
<tr>
<td>Incomplete NTT</td>
<td></td>
<td>Complete NTT</td>
</tr>
<tr>
<td>Pairwise-pointwise Mul</td>
<td></td>
<td>Pointwise Mul</td>
</tr>
<tr>
<td>SHAKE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binomial Sampling,</td>
<td></td>
<td>Uniform Sampling,</td>
</tr>
<tr>
<td>Rejection Sampling</td>
<td></td>
<td>Rejection Sampling</td>
</tr>
</tbody>
</table>
Implementation
Rambus Quantum-Safe Engine

Host CPU

Host App

RAM

NVM

System Bus (AHB)

AHB Subordinate I/F

Host Interface

RISC-V CPU w/ QSE FW

Prog. Memory

Data RAM

Compute

QSE-IP-86

SHA-3

SHAKE

Uniform

Rejection

Binomial

NTT, NTT-1

Pairwise

Vector Ops

Scratchpad

Poly/Vector

Sampler

Hash

IRQ
Number-Theoretic Transform

Forward: Cooley-Tukey (CT)  Inverse: Gentleman-Sande (GS)
The Cost of Arithmetic Diversity

From

Designing an architecture for 32-bit modulus

to

Designing an architecture for 12-bit and 23-bit moduli

- Design complexity
- Development cost
- Verification cost
- Area cost
- Critical path
Reconfigurable Butterflies: State of the art [1]

• KaLi [1]:
  • 1x 23-bit Butterfly for Dilithium → 2x 12-bit butterflies for Kyber
  • 2x 23-bit Butterfly for Dilithium → 1x Pairwise-Pointwise (Karatsuba) mult

More efficient Butterfly Unit

- 2N-bit wide CT/GS butterfly operation
- 2N-bit wide multiplication, addition, subtraction, multiply-accumulate, ...
- 4X N-bit wide CT/GS butterfly operations in parallel
- 4X N-bit wide multiplication, addition, subtraction, multiply-accumulate, ...
- N-bit wide 2X2 Karatsuba polynomial multiplication

😊 Optimized for ASIC
😊 More efficient use of HW area
😊 More efficient use of Memory BW
NTT: Tricky Memory Pattern

Memory configuration at the beginning of Round 1

<table>
<thead>
<tr>
<th>Memory configuration at the beginning of Round 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>:</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>:</td>
</tr>
<tr>
<td>31</td>
</tr>
</tbody>
</table>

Memory re-ordering [2]

Multiple Banks [3]

NTT State-of-the-art [2]

4 Kyber butterflies working sequentially to perform 2 NTT layers on 4 coefficients

🤔 Requires specific memory layout in each round
➡️ Requires re-ordering of coefficients
😊 Kyber last NTT layer uses only ½ of HW

A 4N-bit wide datapath that can compute

- 1, 2 or 3 NTT layers
- On 16 Kyber coefficients (4 x 4N-bit)

0. Read A
\((a_3, a_2, a_1, a_0)\)

1. Read B
\((a_{L/2+3}, a_{L/2+2}, a_{L/2+1}, a_{L/2})\)

2. Read C
\((a_{L/4+3}, a_{L/4+2}, a_{L/4+1}, a_{L/4})\)

3. Read D
\((a_{3L/4+3}, a_{3L/4+2}, a_{3L/4+1}, a_{3L/4})\)

4. \(\ldots\)

5. \(\overset{\text{BFLY}(A',B')}\rightarrow (C',D')\)

6. \(\overset{\text{BFLY}(A'',C'')}\rightarrow (B'',D'')\)

7. Write A"

8. Write B"

9. Write C"

Write D"
More Efficient NTT Datapath

- Fully utilizes memory bandwidth
  - Each word is only read/written once per NTT layer

- No special memory layout required
  - 4N-bit words contain sequential coefficients
    - ex: (a3,a2,a1,a0)

- Efficiently deals with odd # NTT layers Kyber
  - Use a fused round of 3 NTT layers
  - Improves performance by 12.5%
  - Reduces memory reads/writes
What we liked

😊 Everything Kyber & Dilithium have in common (LWE, NTT, SHAKE, ...)

😊 NTT-friendly primes ➔ efficient Montgomery (and Barrett) reduction

😊 No need to store Matrix A ➔ stream SHAKE outputs into arithmetic
  - This is important for memory usage
What we didn’t like are less excited about

😊 Arithmetic Diversity
  • Different sizes of moduli
  • Incomplete vs complete NTT
  • Pairwise-pointwise vs pointwise Mul

😊 Lots of variations of Sampling

😊 FO-transform provides large side-channel attack surface

😊 Frequent XOF calls is problematic for module separation / system level integration

😊 Probabilistic runtimes make it difficult to test for timing leaks
  😞 Also difficult to handle in fixed-vs-random TVLA testing

❌ Floating-point arithmetic (FALCON)
A view on standardization efforts so far

😊 The open structure of the standardization effort is excellent to build trust

😊 The selected algorithms have been thoroughly studied and earned their trust
  • SIKE was broken before it was selected – the process worked as desired
  • We still recommend deploying in a hybrid with ECC

😊 ML-KEM and ML-DSA make a good default choice, even in HW

😊 SLH-DSA works well with ML-KEM / ML-DSA in HW (hash core reuse)
A view on standardization efforts so far

😊 Number of candidates put strain on academic HW research
  • Still no masking countermeasure for Falcon / floating point
    • Breaking news! [3]
  • Research on fault attacks still in early stage

😢 Last minute changes are bad for adoption

😡 Test vector are needed earlier, certification for first products now
  Adoption timelines are outside NIST’s purview but support from NIST is needed

Recommendations for the Remaining PQC efforts

Security must always come first but once that’s done, we suggest to:

1. Try to limit arithmetic diversity
   - HW customers want support for all algorithms -- better optimize area for all algorithms together than optimizing individual algorithms
   - Example: if possible, reuse ML-DSA / ML-KEM moduli even if it costs a little performance

2. Limit memory complexity to that of ML-DSA / ML-KEM

3. Avoid constructions like FO-transform that increase side-channel / FI attack surface
   - Of course, this is not always practical
Recommendations for Future Standardizations

Request [KEM, Signature] pairs where possible (e.g., lattices)
- Facilitates component reuse
- Reduces area overhead
- Reduces development & verification overhead

Look to combine single submissions into pairs after, e.g., 2nd round, based on arithmetic commonalities

Similar for other types of standardizations where multiple primitives are considered
Thank you