PQC Standardization

A Vendor’s Perspective

Mike Hamburg

10 April 2024

Recap

Standardization Process → Portfolio → Implementation

Outline

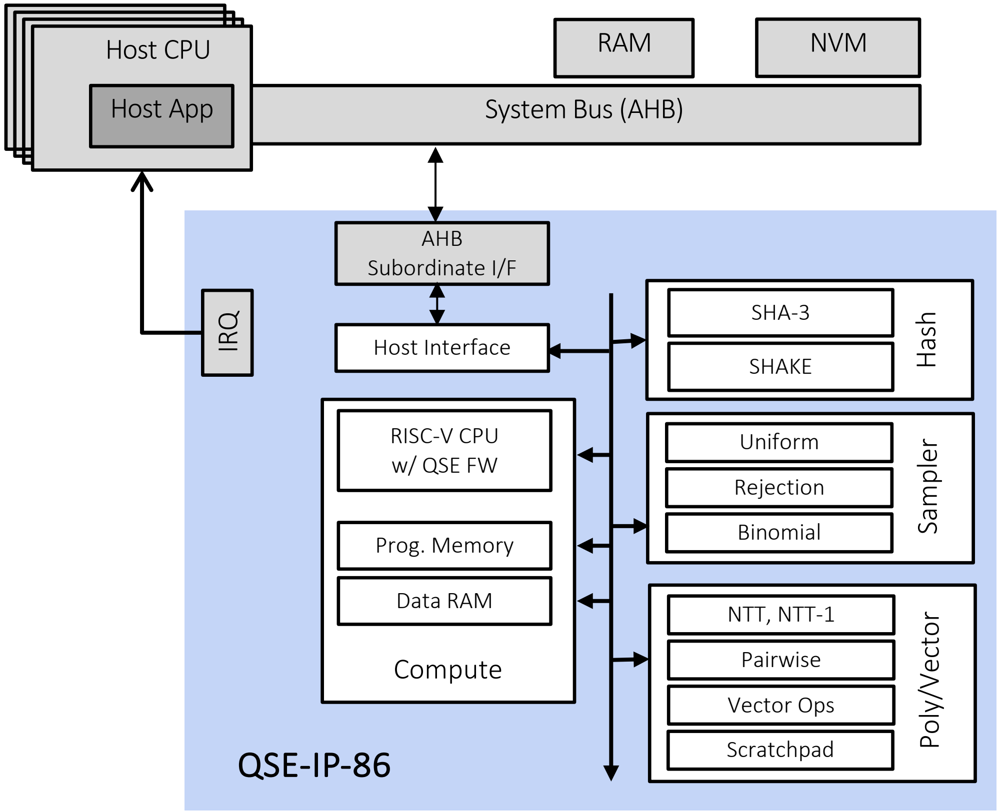
Standardization Process | Portfolio | Implementation

Outline: divergence between Kyber and Dilithium

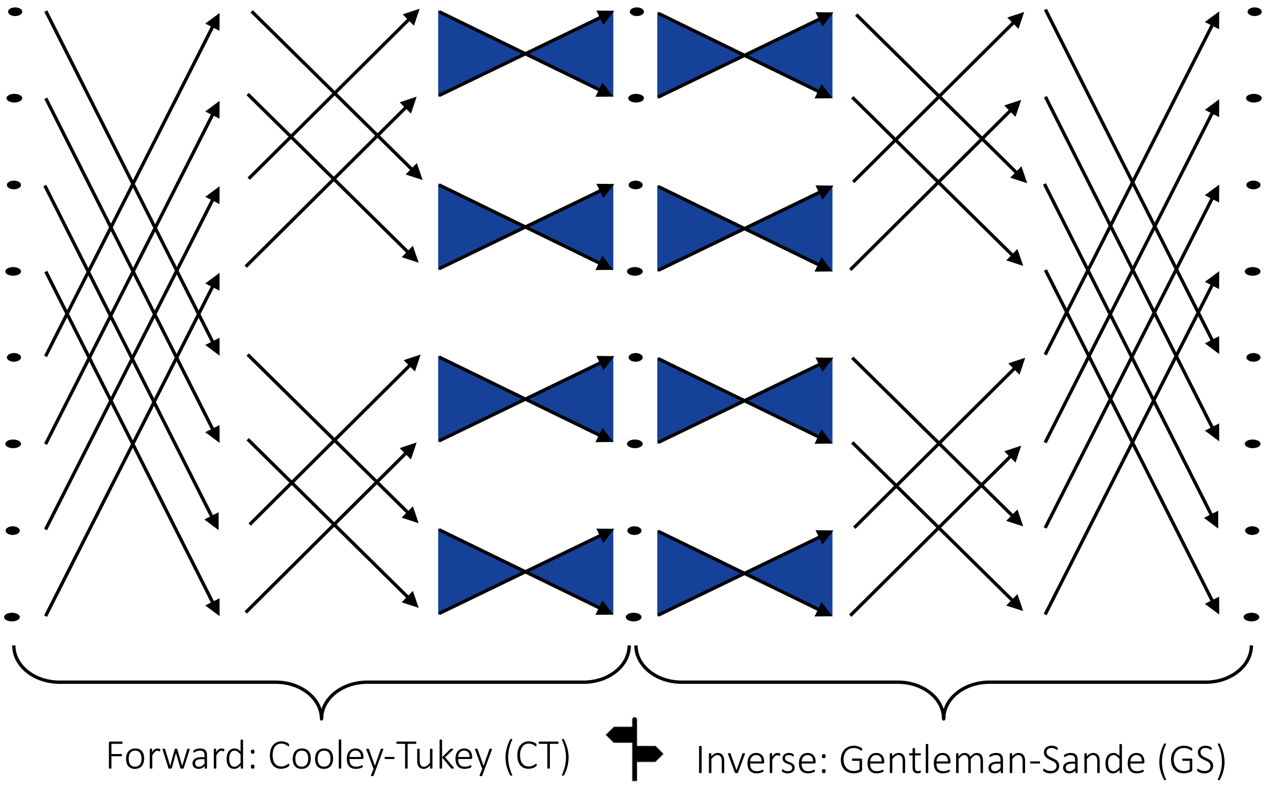
<table>
<tr><th>Kyber/ML-KEM</th><th>Dilithium/ML-DSA</th></tr>
<tr><td colspan="2">Lattice Based ✓</td></tr>
<tr><td colspan="2">NTT ✓</td></tr>
<tr><td colspan="2">NTT-friendly primes ✓</td></tr>
<tr><td>12 bits</td><td>23 bits</td></tr>
<tr><td>Incomplete NTT</td><td>Complete NTT</td></tr>
<tr><td>Pairwise-pointwise Mul</td><td>Pointwise Mul</td></tr>
<tr><td colspan="2">SHAKE ✓</td></tr>
<tr><td>Binomial Sampling,
Rejection Sampling</td><td>Uniform Sampling,
Rejection Sampling</td></tr>
</table>

Implementation

Rambus Quantum-Safe Engine

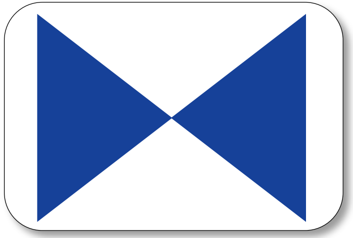


Number-Theoretic Transform



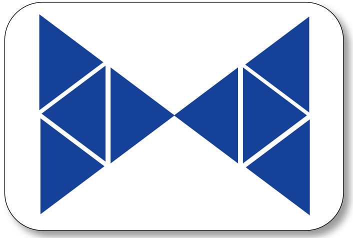
The Cost of Arithmetic Diversity

From

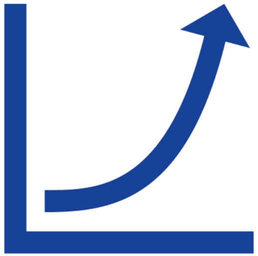


Designing an architecture for 32-bit modulus

to



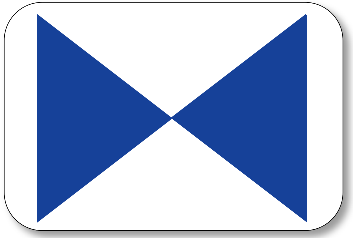
Designing an architecture for 12-bit and 23-bit moduli



- Design complexity
- Development cost
- Verification cost
- Area cost
- Critical path

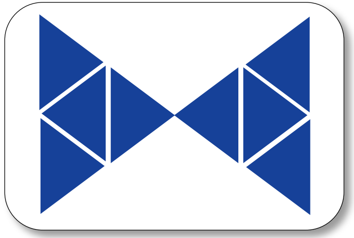
Reconfigurable Butterflies: State of the art [1]

- 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



[1] Aikata, Ahmet Can Mert, Malik Imran, Samuel Pagliarini, Sujoy Sinha Roy: **KaLi: A Crystal for Post-Quantum Security Using Kyber and Dilithium.** IEEE Trans. Circuits Syst. I Regul. Pap. 70(2): 747-758 (2023)

More efficient Butterfly Unit



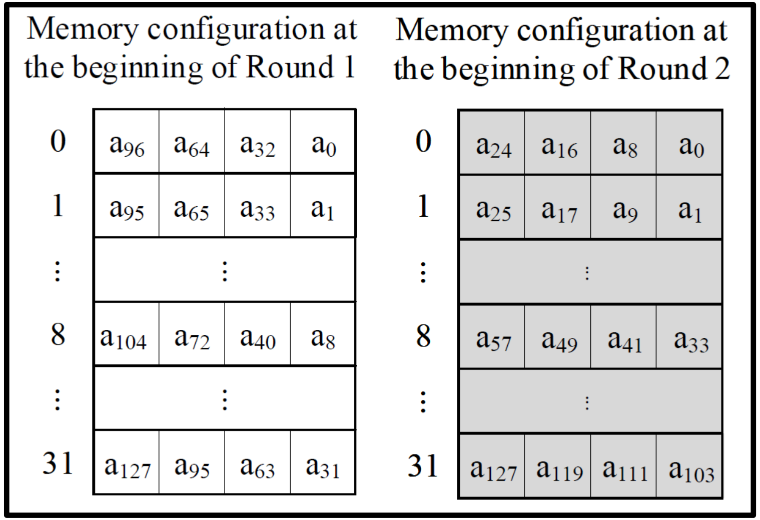
- ✓ 2N-bit wide CT/GS butterfly operation
- ✓ 2N-bit wide multiplication, addition, subtraction, multiply-accumulate, …
- ✓ 4X N-bit wide CT/GS butterfly operations in parallel
- ✓ 4X N-bit wide multiplication, addition, subtraction, multiply-accumulate, …
- ✓ N-bit wide 2X2 Karatsuba polynomial multiplication

☺ Optimized for ASIC

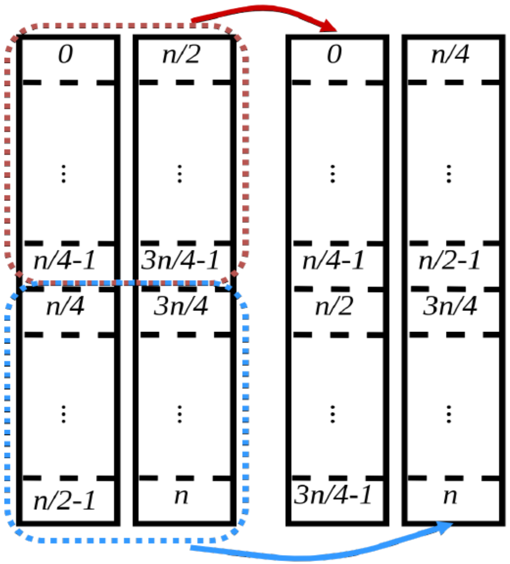
☺ More efficient use of HW area

☺ More efficient use of Memory BW

NTT: Tricky Memory Pattern



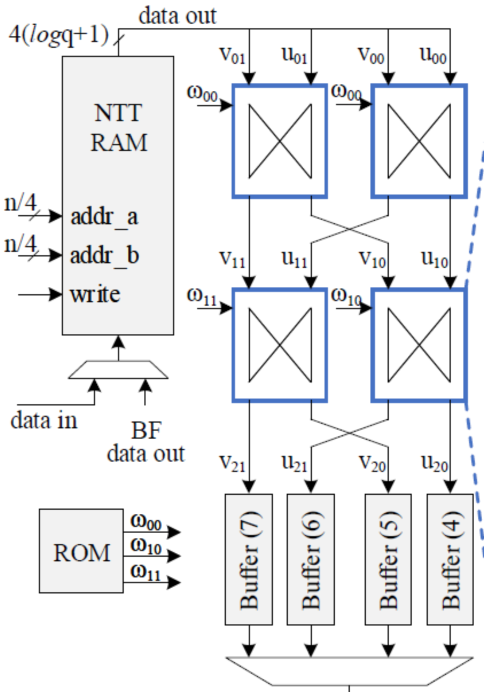
Memory re-ordering [2]



Multiple Banks [3]

[2] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, Mehran Mozaffari Kermani: **High-Speed NTT-based Polynomial Multiplication Accelerator for Post-Quantum Cryptography.** ARITH 2021: 94-101
[3] Ferhat Yaman, Ahmet Can Mert, Erdinç Öztürk, Erkay Savas: **A Hardware Accelerator for Polynomial Multiplication Operation of CRYSTALS-KYBER PQC Scheme.** DATE 2021: 1020-1025

NTT State-of-the-art [2]

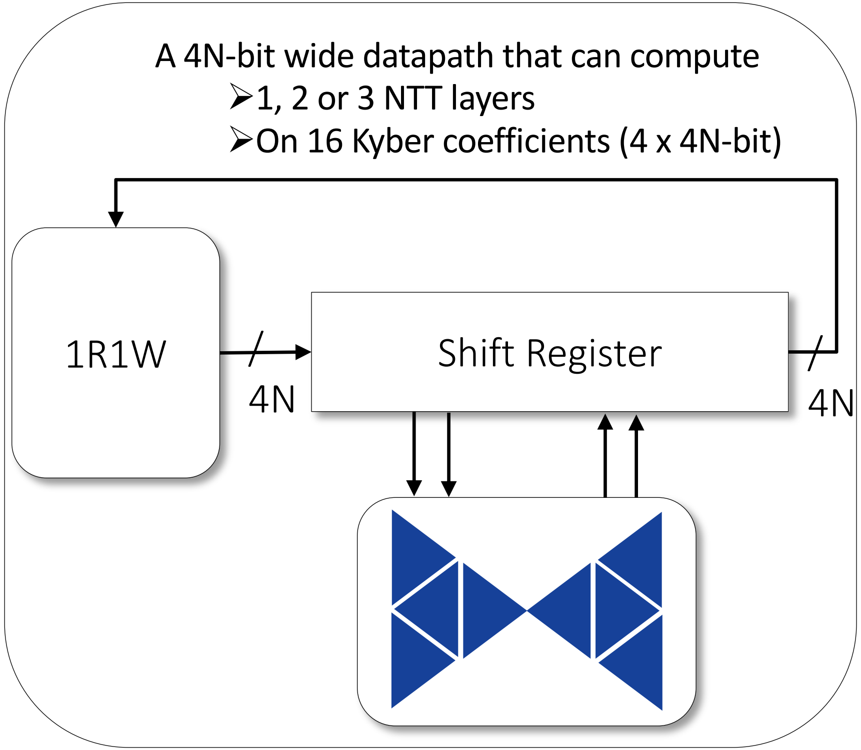


4 Kyber butterflies working sequentially to perform 2 NTT layers on 4 coefficients

😐 **Requires specific memory layout in each round**
→ Requires re-ordering of coefficients
😐 **Kyber last NTT layer uses only ½ of HW**

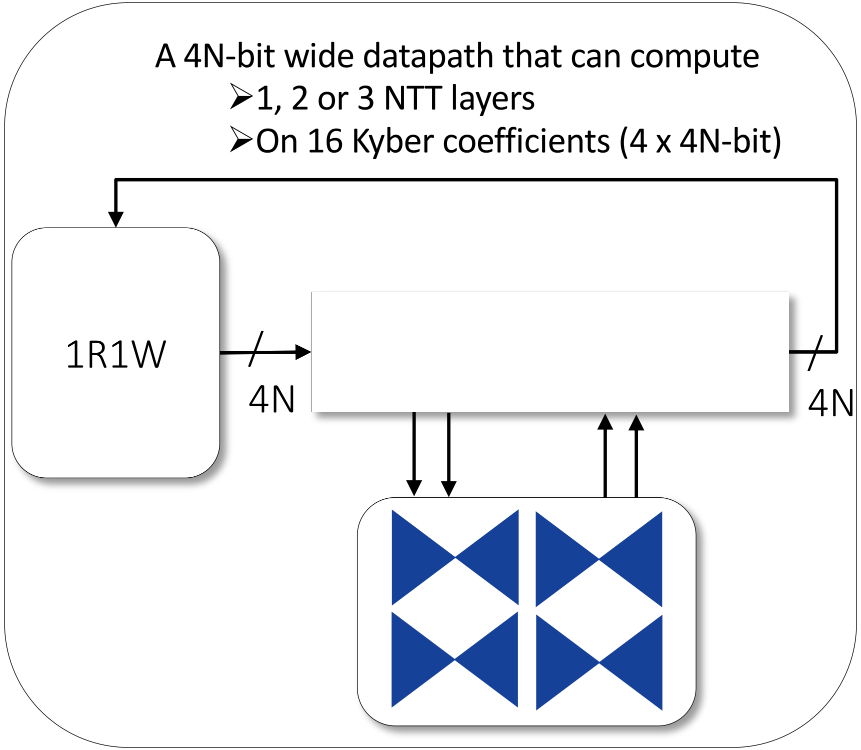
[2] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, Mehran Mozaffari Kermani: **High-Speed NTT-based Polynomial Multiplication Accelerator for Post-Quantum Cryptography.** ARITH 2021: 94-101

More Efficient NTT Datapath



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 ($a_{3L/4+3},a_{3L/4+2},a_{3L/4+1},a_{3L/4}$)	(A’,B’)←BFLY(A,B)	
4	…	(C’,D’)←BFLY(C,D)	
5		(A”,C”)←BFLY(A’,C’)	
6		(B”,D”)←BFLY(B’,D’)	Write A”
7		…	Write B”
8			Write C”
9			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

Portfolio

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)

Standardization Process

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

[3] Keng-Yu Chen, Jiun-Peng Chen: **Masking Floating-Point Number Multiplication and Addition of Falcon: First- and Higher-order Implementations and Evaluations.** IACR Transactions on Cryptographic Hardware and Embedded Systems, 2024(2), 276–303. https://doi.org/10.46586/tches.v2024.i2.276-303

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., 2^{nd} round, based on arithmetic commonalities

Similar for other types of standardizations where multiple primitives are considered


Thank you