Fast Quantum-Safe Cryptography on IBM Z

Basil Hess, Jonathan Bradbury
IBM Research Europe
IBM Systems

NIST Third PQC Conference, 9 June 2021
What most people think a mainframe is
What a mainframe is today: IBM z15

** BASIC FEATURES**
- 64-bit CPUs
- Big-Endian
- CISC architecture
- Application compatible back to IBM 360
- 12-core CPU chip
- 5.2 GHz clock frequency
- Large Caches
  - 128K L1
  - 8M L2
  - 256M Shared L3

92 of the top 100 worldwide banks

10 out of 10 of the world's largest insurers

23 out of 25 of the world's largest airlines

BANK

10

out of 10 of the top 100 worldwide banks

23

out of the top 100 worldwide banks

23

out of 10 of the world's largest insurers

23

out of 25 of the world's largest airlines

3
IBM Z - Cryptographic Acceleration

<table>
<thead>
<tr>
<th>Symmetric Ciphers</th>
<th>Hashing/XOF</th>
<th>Random Number Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithms</strong></td>
<td><strong>Block Modes</strong></td>
<td><strong>SP800-90A Hash DRBG using SHA512</strong></td>
</tr>
<tr>
<td>DES</td>
<td>ECB</td>
<td>SP800-90B True Random Number</td>
</tr>
<tr>
<td>TDES (2 and 3 key)</td>
<td>CBC</td>
<td></td>
</tr>
<tr>
<td>AES128</td>
<td>CFB</td>
<td></td>
</tr>
<tr>
<td>AES192</td>
<td>CTR</td>
<td></td>
</tr>
<tr>
<td>AES256</td>
<td>OFB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XTS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GCM (AEAD)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SHA256</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SHA512</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SHA3-256</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SHA3-384</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SHA3-512</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SHAKE128</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SHAKE256</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GHASH</td>
<td></td>
</tr>
</tbody>
</table>
IBM Z - SIMD Vector Instructions

32 128-bit vector registers

VMSL (Vector Multiply Sum Logical)
- Two 56-bit multiplications
- Optional multiplication by 2
- Full 128-bit addition

VA (Vector Add), VS (Vector Subtract)
- Full 128-bit unsigned addition / subtraction
- With and without carry/borrow in/out

VML (Vector Multiply Low)
VMH (Vector Multiply High)
VPERM (Vector Permute)
VSLDB (Vector Shift Left)
VMRL (Vector Merge)
ANDC (Vector And with Complement)
Optimizing SIKE and Dilithium on Z

**SIKE**
ISOGENY-BASED KEM

- Finite field arithmetic ($F_p$)
  - Multiplication
  - Montgomery reduction using special form of $p$
  - Addition, Subtraction
- Quadratic extension field arithmetic ($F_{p^2}$)
  - Elliptic curve arithmetic
  - Isogeny computation
  - SIDH/SIKE

Primes from 434-bit to 751-bit length

**DILITHIUM**
LATTICE-BASED SIGNATURE SCHEME

- Modular Multiplication
- NTT for arithmetic in polynomial ring
- Generating long random sequences from seeds: using SHAKE or AES256-CTR
- Sampling

Polynomial ring $\mathbb{Z}_q[X]/(X^{256} + 1)$

$q = 2^{23} - 2^{13} + 1$
SIKE – $F_p$ optimizations

1. Step
Limbification: Split element to 2x56-bit limbs stored in 128-bit vectors (Load, VPERM and VSLD)

2. Step
Multiplication using Comba’s method: VMSL, VADD $t_0, \ldots, t_{14}$ each at least 112 bit (for SIKEp434)

3. Step
Normalization $t_0, \ldots, t_{14}$ (> 112-bit to 56-bit limbs) (VADD and VSLD)

4. Step
Delimbification 56-bit limbs to 64-bit digits (Store)

Multiplication

**Squaring** benefits from many multiplications appearing twice, which is free with VMSL

Montgomery **reduction** uses multiplication by $p + 1$, which is optimized because $\lambda$ least significant 56 bit limbs in SIKE are zero (e.g. $\lambda_{p434} = 3$)

**Addition / Subtraction** using 128-bit addition / subtraction chain

**Speedup** compared to unrolled plain C version:
3.0 x – 3.8 x (multiplication), 3.0 x – 4.2 x (reduction)
SIKE - $F_{p^2}$ optimizations

Quadratic extension field as $F_{p^2} = F_p(i)$ with $i^2 + 1 = 0$

**MULTIPLICATION**

$$a \cdot b = c = (a_0b_0 - a_1b_1) + ((a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1) \cdot i$$

**SQUARING**

$$a^2 = (a_0^2 - a_1^2) + (2a_0a_1) \cdot i$$

Speedup to simple version: 1.37 x (p434) to 1.59 x (p751)

Speedup to simple version: 1.15 x (p434) to 1.22 x (p751)
SIKE results

Implementation based on SIKE optimized library (version 3.3)

Overall speedup compared to baseline implementation: factor 4 to 5.

Fastest reported performance metrics compared to SIKE 3rd round optimized/additional implementations (except SIKEp503).

Speedup increases with larger parameter sizes, and better 56-bit limb utilization (best in SIKEp434 and SIKEp610).

<table>
<thead>
<tr>
<th>Scheme</th>
<th>KeyGen</th>
<th>Encaps</th>
<th>Decaps</th>
<th>total (Encaps + Decaps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIKEp434</td>
<td>Portable C</td>
<td>22'771</td>
<td>36'807</td>
<td>39'089</td>
</tr>
<tr>
<td>This work</td>
<td>5'233 (1.01 ms)</td>
<td>8'676 (1.67 ms)</td>
<td>9'141 (1.76 ms)</td>
<td>17'818 (3.43 ms)</td>
</tr>
<tr>
<td>Speedup</td>
<td>4.4 x</td>
<td>4.2 x</td>
<td>4.3 x</td>
<td>4.3 x</td>
</tr>
<tr>
<td>SIKEp503</td>
<td>Portable C</td>
<td>34'442</td>
<td>57'364</td>
<td>60'663</td>
</tr>
<tr>
<td>This work</td>
<td>8'200 (1.58 ms)</td>
<td>13'915 (2.68 ms)</td>
<td>14'763 (2.84 ms)</td>
<td>28'667 (5.51 ms)</td>
</tr>
<tr>
<td>Speedup</td>
<td>4.2 x</td>
<td>4.1 x</td>
<td>4.1 x</td>
<td>4.1 x</td>
</tr>
<tr>
<td>SIKEp610</td>
<td>Portable C</td>
<td>61'783</td>
<td>113'745</td>
<td>114'270</td>
</tr>
<tr>
<td>This work</td>
<td>12'428 (2.39 ms)</td>
<td>23'338 (4.49 ms)</td>
<td>23'400 (4.50 ms)</td>
<td>46'738 (8.99 ms)</td>
</tr>
<tr>
<td>Speedup</td>
<td>5.0 x</td>
<td>4.9 x</td>
<td>4.9 x</td>
<td>4.9 x</td>
</tr>
<tr>
<td>SIKEp751</td>
<td>Portable C</td>
<td>110'838</td>
<td>179'540</td>
<td>193'048</td>
</tr>
<tr>
<td>This work</td>
<td>21'908 (4.21 ms)</td>
<td>37'700 (7.25 ms)</td>
<td>37'560 (7.22 ms)</td>
<td>75'260 (14.47 ms)</td>
</tr>
<tr>
<td>Speedup</td>
<td>5.1 x</td>
<td>4.8 x</td>
<td>5.1 x</td>
<td>5.0 x</td>
</tr>
</tbody>
</table>

Performance in $10^3$ cycles, on IBM z15 LPAR, 5.2 GHz. Linux on Z
Modular multiplication $VA \cdot VB$ with centered reduction to range $-\frac{q-1}{2} \leq r' \leq \frac{q-1}{2}$.

Vectors $VA$, $VB$, and $VT$ contain four 32-bit elements each.

- VMLF $VTL$, $VA$, $VB$
- VMHF $VTH$, $VA$, $VB$
- VMLF $VT$, $VTL$, $VQINV$
- VMHF $VT$, $VT$, $VQ$
- VSF $VT$, $VT$, $VTH$

Modular multiplication used in NTT and inverse-NTT, possible to perform 4 levels of NTT without reloading registers.

- 14x speedup for NTT
- 32x speedup for inverse-NTT

(compared to C reference implementation)
Dilithium:
Keccak, AES256 and Sampling

**SHA3/SHAKE**
- Supported since z14 (2017).
- High single digit GB/s for long hashing.
- Most hashing generates 840 bytes for SHAKE128.
- The Keccak state is 400 bytes to load and store, so the overhead to start and stop the accelerator is high.

**AES256-CTR**
- Supported since z196 (2010), further improved in z14 (2017) with hardware IV+counter generation.
- Encrypt/decrypt performance at ~12GB/s for long enough inputs.
- Increased initial hash to output 64 more bytes decreasing the number of calls.

**SAMPLING**
- Vectorized sampling is similar to the AVX2 optimization, sampling 4 values at the time: Approx. 6.5x speedup compared to the reference implementation.
Dilithium results

<table>
<thead>
<tr>
<th>Dilithium</th>
<th>Keygen (median us)</th>
<th>Sign (median us)</th>
<th>Verify (median us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dilithium2-ref</td>
<td>131.70 us</td>
<td>596.70 us</td>
<td>146.90 us</td>
</tr>
<tr>
<td>Dilithium2 (this work)</td>
<td>20.00 us</td>
<td>48.70 us</td>
<td>17.90 us</td>
</tr>
<tr>
<td>Dilithium2 Speedup</td>
<td>6.59 x</td>
<td>12.25 x</td>
<td>8.21 x</td>
</tr>
<tr>
<td>Dilithium2aes-ref</td>
<td>238.70 us</td>
<td>757.50 us</td>
<td>236.90 us</td>
</tr>
<tr>
<td>Dilithium2aes (this work)</td>
<td>16.30 us</td>
<td>42.80 us</td>
<td>14.70 us</td>
</tr>
<tr>
<td>Dilithium2aes Speedup</td>
<td>14.64 x</td>
<td>17.70 x</td>
<td>16.12 x</td>
</tr>
<tr>
<td>Dilithium3-ref</td>
<td>233.30 us</td>
<td>1005.90 us</td>
<td>234.20 us</td>
</tr>
<tr>
<td>Dilithium3 (this work)</td>
<td>46.00 us</td>
<td>80.60 us</td>
<td>27.50 us</td>
</tr>
<tr>
<td>Dilithium3 Speedup</td>
<td>5.07 x</td>
<td>12.48 x</td>
<td>8.52 x</td>
</tr>
<tr>
<td>Dilithium3aes-ref</td>
<td>454.40 us</td>
<td>1323.50 us</td>
<td>395.00 us</td>
</tr>
<tr>
<td>Dilithium3aes (this work)</td>
<td>38.70 us</td>
<td>70.70 us</td>
<td>21.60 us</td>
</tr>
<tr>
<td>Dilithium3aes Speedup</td>
<td>11.74 x</td>
<td>18.72 x</td>
<td>18.29 x</td>
</tr>
<tr>
<td>Dilithium5-ref</td>
<td>336.30 us</td>
<td>1123.40 us</td>
<td>358.10 us</td>
</tr>
<tr>
<td>Dilithium5 (this work)</td>
<td>51.30 us</td>
<td>103.50 us</td>
<td>45.10 us</td>
</tr>
<tr>
<td>Dilithium5 Speedup</td>
<td>6.56 x</td>
<td>10.85 x</td>
<td>7.94 x</td>
</tr>
<tr>
<td>Dilithium5aes-ref</td>
<td>693.90 us</td>
<td>1569.40 us</td>
<td>666.50 us</td>
</tr>
<tr>
<td>Dilithium5aes (this work)</td>
<td>39.30 us</td>
<td>88.10 us</td>
<td>34.10 us</td>
</tr>
<tr>
<td>Dilithium5aes Speedup</td>
<td>17.66 x</td>
<td>17.81 x</td>
<td>19.55 x</td>
</tr>
</tbody>
</table>

Overall speedup compared to baseline implementation: factor 6 to 20

Performance of Keccak-based Dilithium comes close to AES-based version (on platforms without Keccak acceleration, the gap is bigger)

Further Keccak speed improvements expected in future generations of Z

Implementation based on PQCRYSTALS code base (round 3 submission)

Performance in microseconds, on IBM z15 LPAR, 5.2 GHz, Linux on Z
Resources

- IBM Z Principles of Operation (ISA reference)

- IBM LinuxOne Community Cloud
Thank You!