MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

# Towards ML-KEM & ML-DSA on OpenTitan

Amin Abdulrahman[1] Felix Oberhansl[2] Hoang Nguyen Hien Pham[3,4]
Jade Philipoom[5,6] Peter Schwabe[1,7] Tobias Stelzer[2] Andreas Zankl[2,8]

August 23, 2024

[1] Max Planck Institute for Security and Privacy (MPI-SP)

[2] Fraunhofer Institute for Applied and Integrated Security (AISEC)

[3] BULL SAS [4] Université Grenoble Alpes

[5] zeroRISC [6] OpenTitan

[7] Radboud University [8] Technical University of Munich (TUM)

# Introduction

# Asymmetric Cryptography at Risk


Figure 1: IBM Quantum System One[1]

[1] https://www.flickr.com/photos/ibm_research_zurich/51248690716/

# Post-Quantum Cryptography

- July 2016: NIST Post-Quantum Cryptography (PQC) project

[2] [SAB+22]
[3] [LDK+22]
[4] [HBD+22]
[5] [PFH+22]

# Post-Quantum Cryptography

- July 2016: NIST Post-Quantum Cryptography (PQC) project
- July 2022: NIST announces schemes for standardization

[2] [SAB+22]
[3] [LDK+22]
[4] [HBD+22]
[5] [PFH+22]

# Post-Quantum Cryptography

- July 2016: NIST Post-Quantum Cryptography (PQC) project
- July 2022: NIST announces schemes for standardization
  - **KEM**: Kyber[2]
  - **Digital Signatures**: Dilithium[3], SPHINCS$^{+}$[4], Falcon[5]

---

[2] [SAB+22]
[3] [LDK+22]
[4] [HBD+22]
[5] [PFH+22]

# Post-Quantum Cryptography

- July 2016: NIST Post-Quantum Cryptography (PQC) project
- July 2022: NIST announces schemes for standardization
  - **KEM**: *Kyber*[2]
  - **Digital Signatures**: *Dilithium*[3], SPHINCS$^{+}$[4], Falcon[5]

---

[2] [SAB+22]
[3] [LDK+22]
[4] [HBD+22]
[5] [PFH+22]

# Post-Quantum Cryptography

- July 2016: NIST Post-Quantum Cryptography (PQC) project
- July 2022: NIST announces schemes for standardization
  - **KEM**: *Kyber*[2]
  - **Digital Signatures**: *Dilithium*[3], SPHINCS+[4], Falcon[5]
- Draft standards: FIPS 203 (ML-KEM) and FIPS 204 (ML-DSA)

[2] [SAB+22]
[3] [LDK+22]
[4] [HBD+22]
[5] [PFH+22]

# Post-Quantum Cryptography

- July 2016: NIST Post-Quantum Cryptography (PQC) project
- July 2022: NIST announces schemes for standardization
  - **KEM**: *Kyber*[2]
  - **Digital Signatures**: *Dilithium*[3], SPHINCS$^{+}$[4], Falcon[5]
- Draft standards: FIPS 203 (*ML-KEM*) and FIPS 204 (*ML-DSA*)
- PQC is going real-world: Integration into TLS by Google, Cloudflare, and Mozilla

[2] [SAB+22]
[3] [LDK+22]
[4] [HBD+22]
[5] [PFH+22]

# What’s inside?

- Vastly more complex than traditional, asymmetric cryptography

# What's inside?

- Vastly more complex than traditional, asymmetric cryptography
- Common thread: Hashing and polynomial arithmetic
  - Heavy usage of SHAKE and SHA-3
  - Polynomial arithmetic over $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$, where $n = 256$, $q = 3329$ for ML-KEM, and $q = 8380417$ for ML-DSA

$\rightarrow$ Coefficients are "small" integers

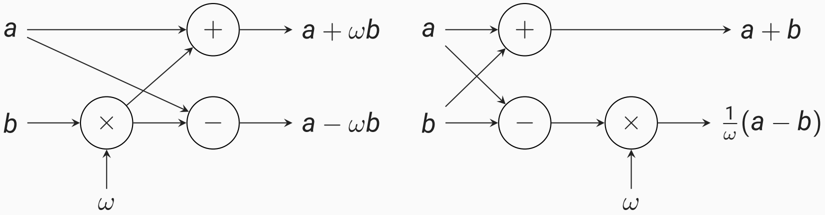
# What's inside?

- Vastly more complex than traditional, asymmetric cryptography
- Common thread: Hashing and polynomial arithmetic
  - Heavy usage of SHAKE and SHA-3
  - Polynomial arithmetic over $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$, where $n = 256$, $q = 3329$ for ML-KEM, and $q = 8380417$ for ML-DSA

  $\rightarrow$ Coefficients are "small" integers
- Efficient polynomial arithmetic achieved through NTT-based multiplication
  - Variant of the discrete Fourier transform over finite fields

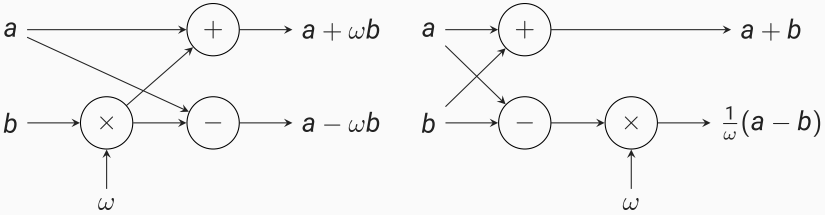
# NTT-based polynomial multiplication

- Fast NTT with Cooley-Tukey (CT) or Gentleman-Sande (GS) FFT algorithms in $\mathcal{O}(n \log n)$
  - Divide & Conquer approach
  - $\log n$ divide-steps with 128 parallel “butterfly” operations on coefficient pair



# NTT-based polynomial multiplication

- Fast NTT with Cooley-Tukey (CT) or Gentleman-Sande (GS) FFT algorithms in $\mathcal{O}(n \log n)$
  - Divide & Conquer approach
  - $\log n$ divide-steps with 128 parallel "butterfly" operations on coefficient pair




→ Central operations: Addition, subtraction, (modular) multiplication with a constant

# OpenTitan OTBN

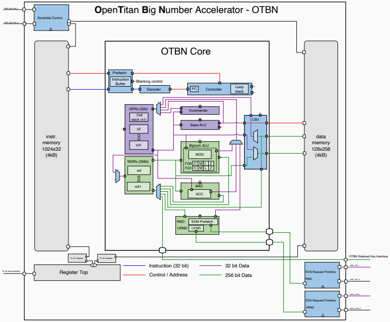



Figure 2: OTBN block architecture[6]

[1]https://opentitan.org/book/hw/ip/otbn/doc/otbn_blockarch.svg

# OpenTitan OTBN

- Reduced RV32 instruction set

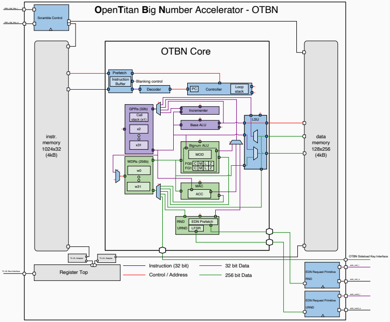



Figure 2: OTBN block architecture[6]

[1] https://opentitan.org/book/hw/ip/otbn/doc/otbn_blockarch.svg

# OpenTitan OTBN

- Reduced RV32 instruction set
- Big number instruction set

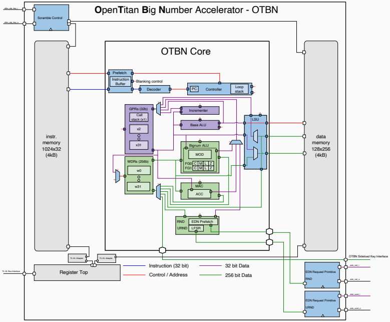



Figure 2: OTBN block architecture[6]

[1] https://opentitan.org/book/hw/ip/otbn/doc/otbn_blockarch.svg

# OpenTitan OTBN

- Reduced RV32 instruction set
- Big number instruction set
- 32 256-bit wide WDRs

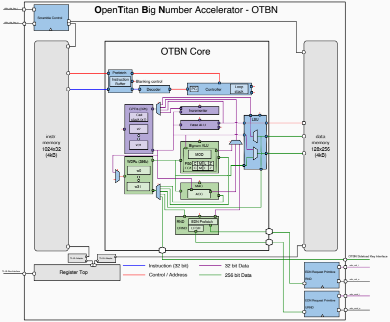



Figure 2: OTBN block architecture[6]

[1] https://opentitan.org/book/hw/ip/otbn/doc/otbn_blockarch.svg

# OpenTitan OTBN

- Reduced RV32 instruction set
- Big number instruction set
- 32 256-bit wide WDRs
- bn.addm /bn.subm

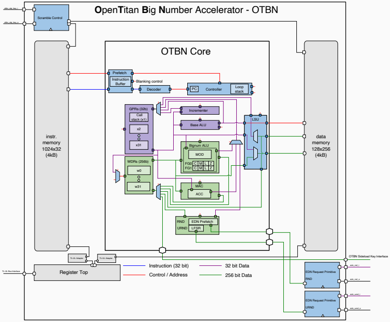



Figure 2: OTBN block architecture[6]

[1]https://opentitan.org/book/hw/ip/otbn/doc/otbn_blockarch.svg

# OpenTitan OTBN

- Reduced RV32 instruction set
- Big number instruction set
- 32 256-bit wide WDRs
- `bn.addm` / `bn.subm`
- 64 × 64-bit multiply-acc. unit

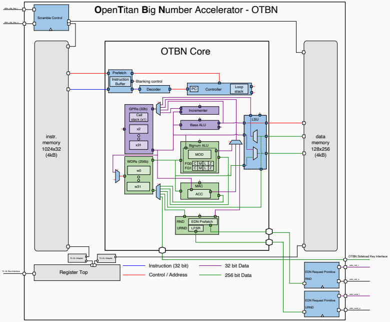



Figure 2: OTBN block architecture[6]

[1] https://opentitan.org/book/hw/ip/otbn/doc/otbn_blockarch.svg

# Implementing ML-KEM & ML-DSA on OTBN

# Approach

Establish a baseline for ML-KEM and ML-DSA on OTBN

- Using state-of-the-art implementation techniques
- Optimization tailored to the architecture
  - $\rightarrow$ Make use of WDRs whenever possible
  - $\rightarrow$ Leverage `bn.addm`/`bn.subm`
- Only modification required: More memory

# Profiling on OTBN

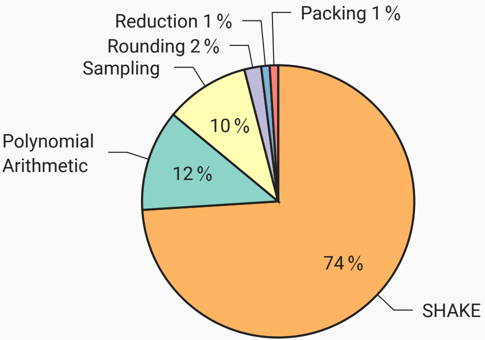



Figure 3: Profiling of ML-DSA-65 verification on OTBN.

# Extensions to OTBN

# Leveraging the KMAC Block

- High-speed, high-security Keccak accelerator available
  - 4 cycles/round
  - Masked
- Implementation in the OTBN Python simulator by Philipoom[7]
- Interfacing through WSRs
- We call OTBN with KMAC $\text{OTBN}^{\text{KMAC}}$

[7] https://github.com/jadephilipoom/opentitan/commit/e86be3446204f439c41c142b077a4ca8b449b1c9

# Profiling on $\text{OTBN}^{\text{KMAC}}$

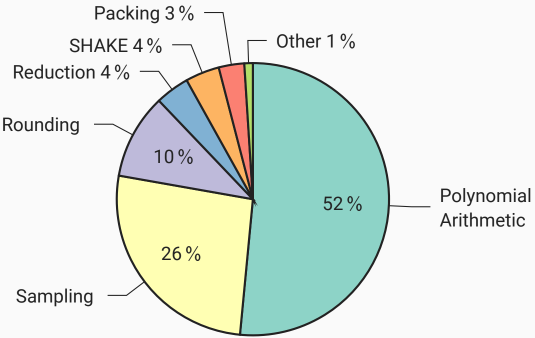



Figure 4: Profiling of ML-DSA-65 verification on $\text{OTBN}^{\text{KMAC}}$.

# Challenges

```
bn.and coeffa, coeffsa, consts >> 192 /* Mask out coeffs from buffer*/
bn.and coeffb, coeffsb, consts >> 192

/* Plantard multiplication: Twiddle * coeffb */
bn.mulqacc.wo.z coeffb, coeffb.0, twiddle.0, 192 /* (coeffb*R) */
bn.add          coeffb, consts, coeffb >> 160    /* +1 */
bn.mulqacc.wo.z coeffb, coeffb.1, consts.2, 0    /* *q */
bn.rshi         wtmp, consts, coeffb >> 32       /* >> d */
/* Butterfly */
bn.subm coeffb, coeffa, wtmp
bn.addm coeffa, coeffa, wtmp

bn.rshi coeffsa, coeffa, coeffsa >> 32 /* Shift results to buffer */
bn.rshi coeffsb, coeffb, coeffsb >> 32
```

Listing 1: CT butterfly on OTBN.

# Challenges

```
bn.and coeffa, coeffsa, consts >> 192 /* Mask out coeffs from buffer*/
bn.and coeffb, coeffsb, consts >> 192

/* Plantard multiplication: Twiddle * coeffb */
bn.mulqacc.wo.z coeffb, coeffb.0, twiddle.0, 192 /* (coeffb*R) */
bn.add          coeffb, consts, coeffb >> 160    /* +1 */
bn.mulqacc.wo.z coeffb, coeffb.1, consts.2, 0    /* *q */
bn.rshi         wtmp, consts, coeffb >> 32       /* >> d */
/* Butterfly */
bn.subm coeffb, coeffa, wtmp
bn.addm coeffa, coeffa, wtmp

bn.rshi coeffsa, coeffa, coeffsa >> 32 /* Shift results to buffer */
bn.rshi coeffsb, coeffb, coeffsb >> 32
```

Listing 1: CT butterfly on OTBN.

- High data-movement overhead
- No SIMD capabilities
- Generally: ISA not made for arithmetic on small integers

# Speeding-up polynomial arithmetic

## Approach

Make better use of 256-bit wide registers.
$\rightarrow$ Interpret WDRs as vectors of smaller elements, e.g., $16 \times 16$-bit or $8 \times 32$-bit.

# bn.addv, bn.subv, bn.mulv

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| w1 | $a_0$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ |

$$\{+, -, \times\}$$

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| w2 | $a_8$ | $a_9$ | $a_{10}$ | $a_{11}$ | $a_{12}$ | $a_{13}$ | $a_{14}$ | $a_{15}$ |

Figure 5: Vectorized (modular) arithmetic: `bn.{addv,subv,mulv}{m}`

# bn.addv, bn.subv, bn.mulv

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| w1 | $a_0$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ |

$$\{+, -, \times\}$$

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| w2 | $a_8$ | $a_9$ | $a_{10}$ | $a_{11}$ | $a_{12}$ | $a_{13}$ | $a_{14}$ | $a_{15}$ |

Figure 5: Vectorized (modular) arithmetic: `bn.{addv,subv,mulv}{m}`

## Example

```
bn.mulvm.8S w0, w1, w2
```

# bn.trn1, bn.trn2

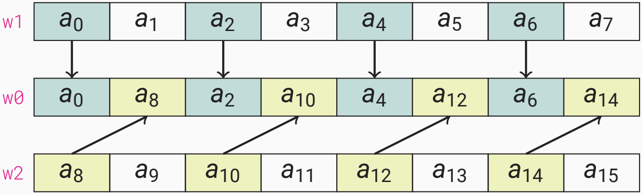



Figure 6: Vector transpose (odd/even indices): `bn.trn1`, `bn.trn2`

# bn.trn1, bn.trn2

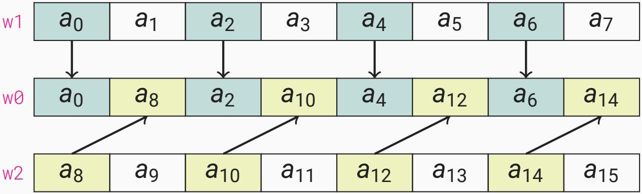



Figure 6: Vector transpose (odd/even indices): `bn.trn1`, `bn.trn2`

## Example

```
bn.trn1.8S w0, w1, w2
```

# bn.shv

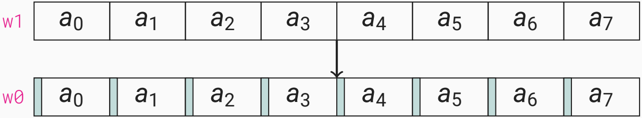



Figure 7: Vectorized {right,left} bit shift: bn.shv

# bn.shv

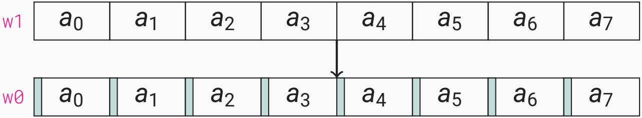



Figure 7: Vectorized {right,left} bit shift: `bn.shv`

## Example

```
bn.shv.8S w0, w1 >> 3
```

# Impact of the Extensions

## Evaluating $\text{OTBN}^{\text{KMAC}}_{\text{Ext.}}$

# Butterfly on $\text{OTBN}^{\text{KMAC}}_{\text{Ext.}}$

```
bn.mulvm.l.8S tmp, vec8, twiddles, 0
bn.subvm.8S   vec8, vec0, tmp
bn.addvm.8S   vec0, vec0, tmp
```

Listing 2: CT butterfly on $\text{OTBN}^{\text{KMAC}}_{\text{Ext.}}$.

# Profiling on OTBN$^{KMAC}$

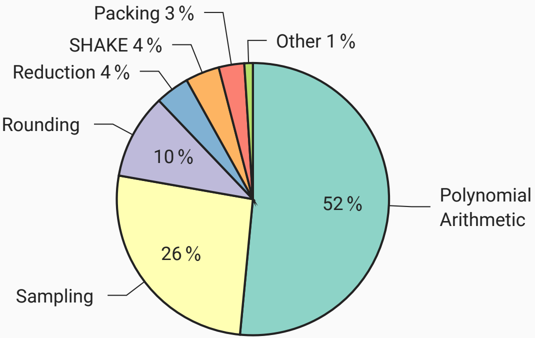



Figure 4: Profiling of ML-DSA-65 verification on OTBN$^{KMAC}$.

# Profiling on $OTBN^{KMAC}_{Ext.}$

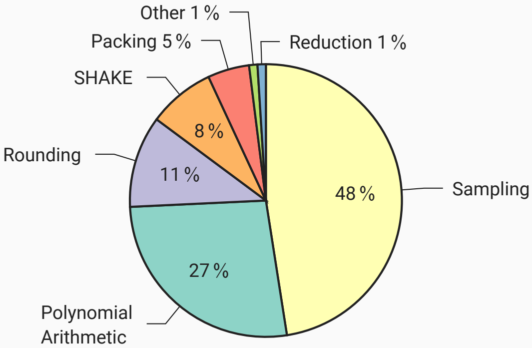



Figure 8: Profiling of ML-DSA-65 verification on $OTBN^{KMAC}_{Ext.}$.

# Full Scheme Benchmarks

Table 1: ML-DSA-65 full scheme benchmarks. All numbers refer to cycles. Median result was selected, if given. 10 000 iterations for our measurements.

| Platform | Key Gen. | | Sign | | Verify | |
|---|---|---|---|---|---|---|
| OTBN | 2 190 278 | (×8.39) | 4 490 766 | (×6.44) | 2 107 440 | (×8.22) |
| $\text{OTBN}^{\text{KMAC}}$ | 438 154 | (×1.68) | 1 842 696 | (×2.64) | 493 307 | (×1.92) |
| $\text{OTBN}^{\text{KMAC}}_{\text{Ext.}}$ | 261 000 | (×1.00) | 697 203 | (×1.00) | 256 327 | (×1.00) |
| OpenTitan [SOSK23][b,c] | — | — | — | — | 1 488 526 | (×5.81) |
| Skylake [LDK+22][a] | 154 308 | (×0.59) | 342 708 | (×0.49) | 154 622 | (×0.60) |
| Cortex-M4 [HAZ+24][a] | 2 390 080 | (×9.16) | 4 878 759 | (×7.00) | 2 289 269 | (×8.93) |

a Own benchmarks.
b Including modified variant of OTBN, parts of the execution on Ibex Core.
c Round 3 DILITHIUM.

# Conclusion

Five new instruction classes:

- `bn.addv{m}{.8S,.16H}`
- `bn.subv{m}{.8S,.16H}`
- `bn.mulv{m}{.8S,.16H,.l}`
- `bn.trn1{.2Q,.4D,.8S,.16H}`, `bn.trn2{.2Q,.4D,.8S,.16H}`
- `bn.shv{.8S,.16H}`

# Conclusion

Five new instruction classes:

- `bn.addv{m}{.8S,.16H}`
- `bn.subv{m}{.8S,.16H}`
- `bn.mulv{m}{.8S,.16H,.l}`
- `bn.trn1{.2Q,.4D,.8S,.16H}`, `bn.trn2{.2Q,.4D,.8S,.16H}`
- `bn.shv{.8S,.16H}`

## Result

Longer critical path, but: We achieve speed-ups of a **factor of 6 to 9** with only **11% area overhead** for OTBN and not even **2% for Top-Earlgrey**.

# Bibliography i


[HAZ+24] Junhao Huang et al. "Revisiting Keccak and Dilithium Implementations on ARMv7-M". In: *IACR TCHES* 2024.2 (2024), pp. 1–24. DOI: 10.46586/tches.v2024.i2.1-24.

[HBD+22] Andreas Hülsing et al. *SPHINCS+*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. National Institute of Standards and Technology, 2022.

[LDK+22] Vadim Lyubashevsky et al. *CRYSTALS-DILITHIUM*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. National Institute of Standards and Technology, 2022.

# Bibliography ii


[PFH+22] Thomas Prest et al. *FALCON*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. National Institute of Standards and Technology, 2022.

[SAB+22] Peter Schwabe et al. *CRYSTALS-KYBER*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. National Institute of Standards and Technology, 2022.

# Bibliography iii

[SOSK23] Tobias Stelzer et al. “Enabling Lattice-Based Post-Quantum Cryptography on the OpenTitan Platform”. In: *Proceedings of the 2023 Workshop on Attacks and Solutions in Hardware Security*. ASHES ’23. https://dl.acm.org/doi/10.1145/3605769.3623993. New York, NY, USA: Association for Computing Machinery, Nov. 2023, pp. 51–60. ISBN: 9798400702624. DOI: 10.1145/3605769.3623993. (Visited on 11/30/2023).