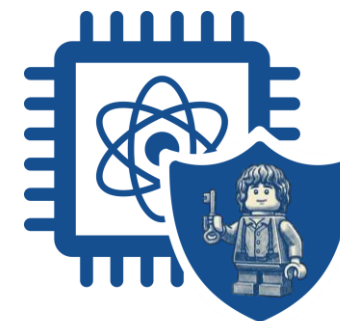


Accelerating FrodoKEM in Hardware



Sanjay Deshpande^{*}, Patrick Longa[✠], and Jakub Szefer^{*}

^{*}Northwestern University, Evanston, USA

[✠]Microsoft Research, Redmond, USA



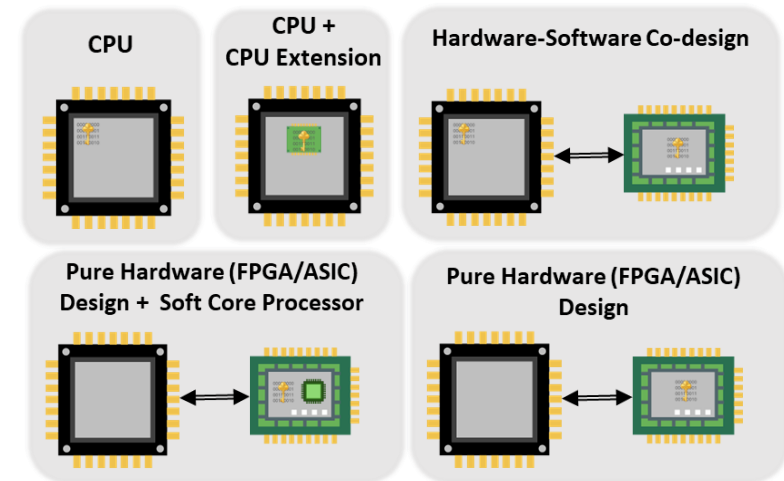
Northwestern
University



Microsoft

Motivation

- FrodoKEM is a Post-Quantum Key Encapsulation Mechanism that is standardized by ISO
 - Support from governmental agencies e.g., German BSI
- Hardware evaluation becomes a critical aspect of readiness for real-world deployment
- Existing hardware implementations (FPGA):
 - Light-weight [HOKG18]
 - High performance
 - Doesn't comply with the specification [HMOR21] (or)
 - Supports only one parameter set or only one primitive [DGA25] (or)
 - No support for performance tunability [YWZL26]
- We propose a design targeting FPGA and ASIC that is
 - **Unified** –supports keygen, encap, and decap
 - **Multi-parameter support at runtime** – all 12 parameter sets
 - **Performance tunable**



The FrodoKEM scheme



FrodoKEM

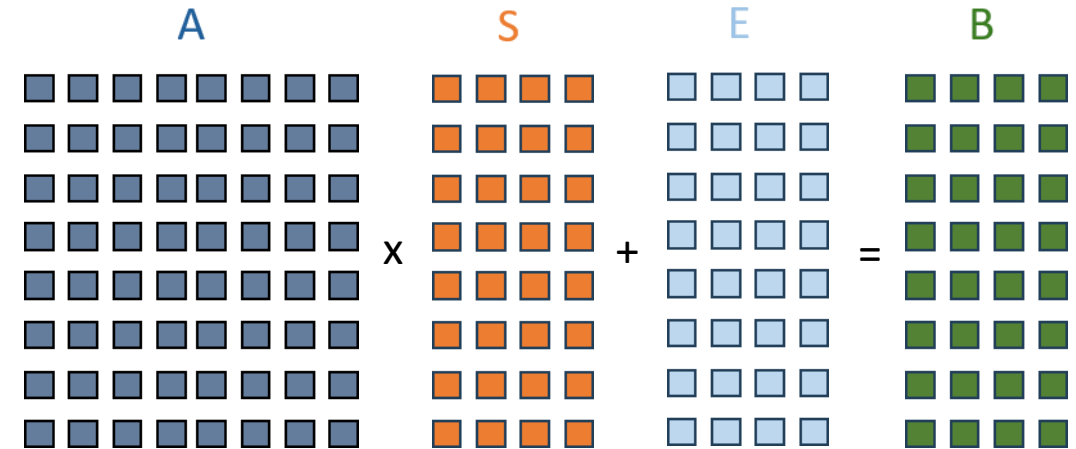
- FrodoKEM is based on the plain learning with errors (LWE) problem
 - It uses generic, algebraically unstructured lattices (no use of NTT)
 - Security - more conservative than ML-KEM

- Parameters:

- Integer matrix dimensions: $n, n^{\bar{}}$
- Modulus $q = 2^D$
- Symmetric Error Distribution: χ

- Setup:

- Generate pseudorandom matrix $A \leftarrow U(\mathbb{Z}_q^{n \times n})$,
- Sample from the error distribution $S \leftarrow \chi^{n \times n^{\bar{}}}, E \leftarrow \chi^{n \times n^{\bar{}}}$



$$B \leftarrow A \times S + E$$

A and B are public

S is a secret

Given A and B finding S is hard!

- pseudorandom matrix
- Error distribution matrix
- Error distribution matrix



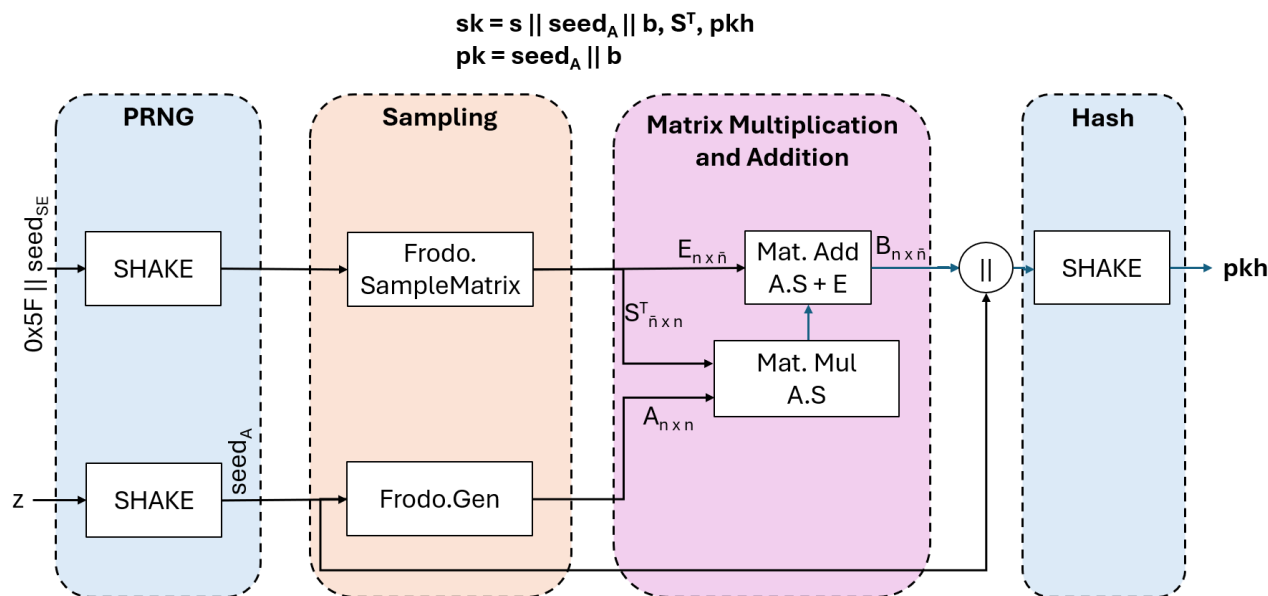
Key generation algorithm

Algorithm 1 FrodoKEM Key Generation (KeyGen)

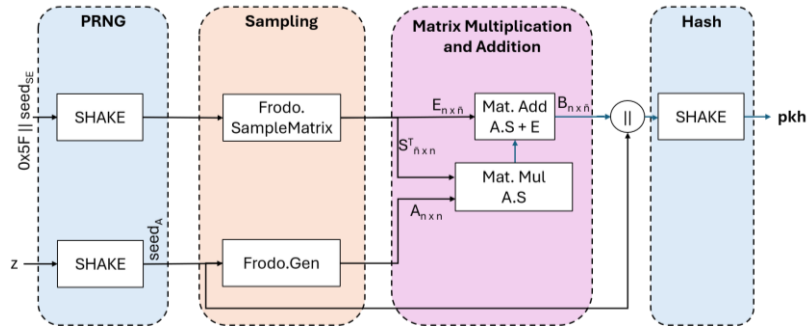
Input: –

Output: Public key $pk = (seed_A || b)$, secret key $sk = (s || seed_A || b || S^T || pkh)$

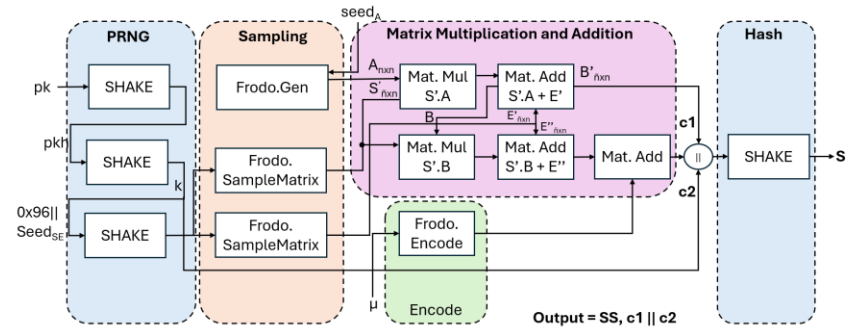
- 1: Choose uniformly random seeds s , $seed_{SE}$, and z of bitlengths len_{sec} , len_{SE} , and len_A , respectively
 - 2: Generate pseudorandom seed $seed_A \leftarrow \text{SHAKE}(z, len_A)$
 - 3: Generate the matrix $A \leftarrow \text{Gen}(seed_A)$
 - 4: Generate pseudorandom bit string $(r^{(0)}, r^{(1)}, \dots, r^{(2n\bar{n}-1)}) \leftarrow \text{SHAKE}(0x5F || seed_{SE}, 32n\bar{n})$
 - 5: Sample error matrix $S^T \leftarrow \text{SampleMatrix}((r^{(0)}, r^{(1)}, \dots, r^{(n\bar{n}-1)}), \bar{n}, n)$
 - 6: Sample error matrix $E \leftarrow \text{SampleMatrix}((r^{(n\bar{n})}, r^{(n\bar{n}+1)}, \dots, r^{(2n\bar{n}-1)}), n, \bar{n})$
 - 7: Compute $B \leftarrow AS + E$
 - 8: Compute $b \leftarrow \text{Pack}(B)$
 - 9: Compute $pkh \leftarrow \text{SHAKE}(seed_A || b, len_{sec})$
 - 10: Return public key $pk \leftarrow seed_A || b$ and secret key $sk \leftarrow s || seed_A || b || S^T || pkh$
-



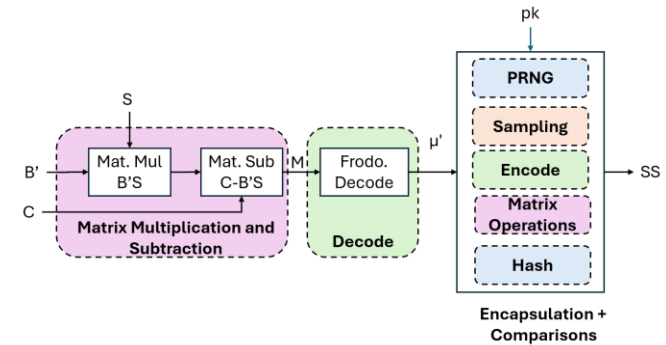
FrodoKEM primitives and Parameter sets



Key Generation



Encapsulation



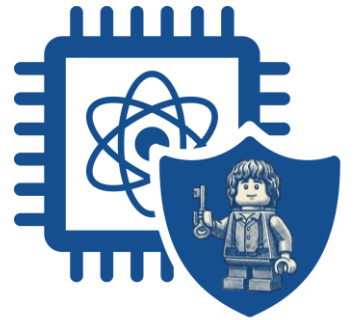
Decapsulation

Parameter Set	q	q -size (bits)	χ -size (bits)	n	$\tilde{n}$
FrodoKEM-640	32,768	15	5	640	8
FrodoKEM-976	65,536	16	5	976	8
FrodoKEM-1344	65,536	16	4	1344	8

- q – modulus for matrix elements in A
- $n, \tilde{n}$ – matrix dimensions
- χ -size – element size in error matrices S and E

- Parameter sets
 - Each parameter set has two variants:
 - AES-based
 - SHAKE-based
 - Each of these have an ephemeral variant
- Depending on the variant, $A_{n \times n}$ is either sampled from AES or SHAKE

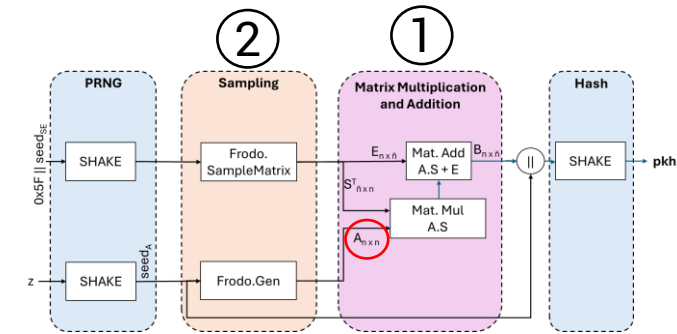
Prior Work



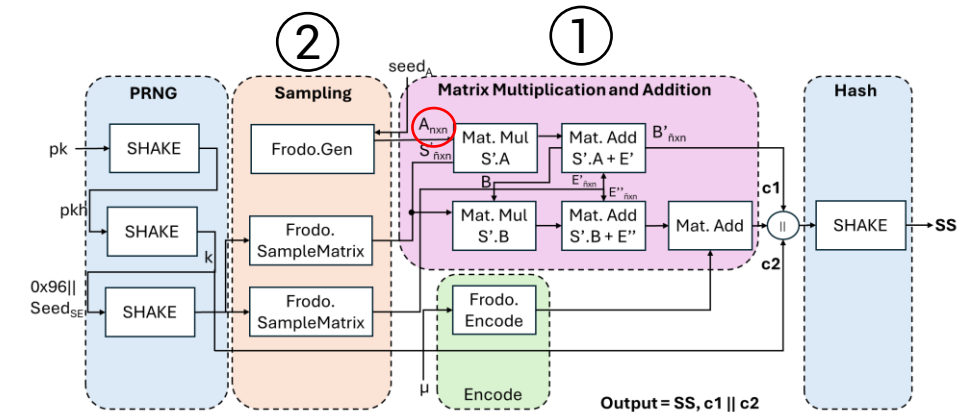
Two main bottlenecks

- Hardware [HOKG18, HMOR21, YWZL26] and microcontroller implementations [BBC+23] target two main challenges:
 - Performance:** How do we speed up the large matrix multiplications?
 - [HMOR21, YWZL26] vectorizes the dot products inside the matrix multiplication ([HMOR21] with Trivium used as PRNG – not specification compliant)
 - Memory:** How do we avoid storing the large matrices?
 - [BBC+23] generates matrices **A** and **S** on the fly
- What's missing:** No prior design offers tunable performance. If we scale up compute, where does the bottleneck move?

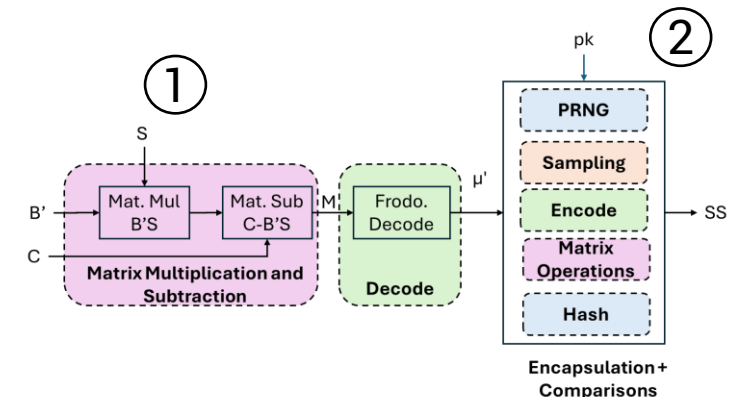
Keygen



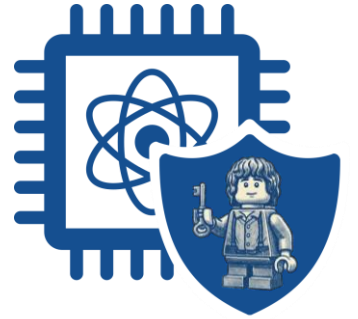
Encap



Decap



Design Considerations



Generating ' $A_{n \times n}$ ' on-the-fly

- $A_{n \times n}$ is sampled from a PRNG row-wise
- Only one row of ' A ' is generated and stored in memory
- Key Generation
 - $A.S$
- Encapsulation
 - $S'.A$
- Decapsulation
 - $S'.A$

$$\begin{pmatrix} A_{00} & A_{01} & \dots & A_{0n-1} \\ A_{10} & A_{11} & \dots & A_{1n-1} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n-10} & A_{n-11} & \dots & A_{n-1n-1} \end{pmatrix}$$

Parameter Set	q-size (bits)	χ -size (bits)	n	$\bar{n}$	Matrix A Size (bits)	Matrix S Size (bits)
FrodoKEM-640	15	5	640	8	6,144,000	25,600
FrodoKEM-976	16	5	976	8	15,241,216	39,040
FrodoKEM-1344	16	4	1,344	8	28,901,376	43,008



Traditional matrix multiplication

A

$$\begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{pmatrix}$$

S

$$\begin{pmatrix} S_{00} & S_{01} \\ S_{10} & S_{11} \\ S_{20} & S_{21} \\ S_{30} & S_{31} \end{pmatrix}$$

AxS

$$\begin{pmatrix} A_{00}S_{00}+A_{01}S_{10}+A_{02}S_{20}+A_{03}S_{30} & A_{00}S_{01}+A_{01}S_{11}+A_{02}S_{21}+A_{03}S_{31} \\ A_{10}S_{00}+A_{11}S_{10}+A_{12}S_{20}+A_{13}S_{30} & A_{10}S_{01}+A_{11}S_{11}+A_{12}S_{21}+A_{13}S_{31} \\ A_{20}S_{00}+A_{21}S_{10}+A_{22}S_{20}+A_{23}S_{30} & A_{20}S_{01}+A_{21}S_{11}+A_{22}S_{21}+A_{23}S_{31} \\ A_{30}S_{00}+A_{31}S_{10}+A_{32}S_{20}+A_{33}S_{30} & A_{30}S_{01}+A_{31}S_{11}+A_{32}S_{21}+A_{33}S_{31} \end{pmatrix}$$

AxS – 'A' generated on the fly

- 'S' is stored in memory
- Matrix dimensions in this example: A - 4x4, S - 4x2

A

$$\begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{pmatrix}$$

S

$$\begin{pmatrix} S_{00} & S_{01} \\ S_{10} & S_{11} \\ S_{20} & S_{21} \\ S_{30} & S_{31} \end{pmatrix}$$

AxS

$$\begin{pmatrix} A_{00}S_{00}+A_{01}S_{10}+A_{02}S_{20}+A_{03}S_{30} & A_{00}S_{01}+A_{01}S_{11}+A_{02}S_{21}+A_{03}S_{31} \\ A_{10}S_{00}+A_{11}S_{10}+A_{12}S_{20}+A_{13}S_{30} & A_{10}S_{01}+A_{11}S_{11}+A_{12}S_{21}+A_{13}S_{31} \\ A_{20}S_{00}+A_{21}S_{10}+A_{22}S_{20}+A_{23}S_{30} & A_{20}S_{01}+A_{21}S_{11}+A_{22}S_{21}+A_{23}S_{31} \\ A_{30}S_{00}+A_{31}S_{10}+A_{32}S_{20}+A_{33}S_{30} & A_{30}S_{01}+A_{31}S_{11}+A_{32}S_{21}+A_{33}S_{31} \end{pmatrix}$$

SxA - 'A' generated on the fly

- 'S' is stored in memory
- Matrix dimensions in this example: S - 2x4, A - 4x4

$$\begin{matrix} & \mathbf{S} \\ \begin{pmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{10} & S_{11} & S_{12} & S_{13} \end{pmatrix} & \begin{matrix} & \mathbf{A} \\ \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \end{pmatrix} \end{matrix} \end{matrix}$$

SxA

$$\begin{pmatrix} S_{00}A_{00}+S_{01}A_{10}+S_{02}A_{20}+S_{03}A_{30} & S_{00}A_{01}+S_{01}A_{11}+S_{02}A_{21}+S_{03}A_{31} & S_{00}A_{02}+S_{01}A_{12}+S_{02}A_{22}+S_{03}A_{32} & S_{00}A_{03}+S_{01}A_{11}+S_{02}A_{21}+S_{03}A_{31} \\ S_{10}A_{00}+S_{11}A_{10}+S_{12}A_{20}+S_{13}A_{30} & S_{10}A_{01}+S_{11}A_{11}+S_{12}A_{21}+S_{13}A_{31} & S_{10}A_{02}+S_{11}A_{12}+S_{12}A_{22}+S_{13}A_{32} & S_{10}A_{03}+S_{11}A_{11}+S_{12}A_{21}+S_{13}A_{31} \end{pmatrix}$$



Partial multiplications

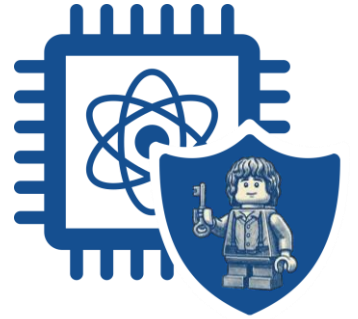
- Based on number of multipliers available at one time
- Let 'T' be the number of multipliers
- E.g., T=2

$$\mathbf{A} = \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{pmatrix}$$

$$\mathbf{S} = \begin{pmatrix} S_{00} & S_{01} \\ S_{10} & S_{11} \\ S_{20} & S_{21} \\ S_{30} & S_{31} \end{pmatrix}$$

$$\mathbf{A.S} = \begin{pmatrix} A_{00}S_{00}+A_{01}S_{10}+A_{02}S_{20}+A_{03}S_{30} & A_{00}S_{01}+A_{01}S_{11}+A_{02}S_{21}+A_{03}S_{31} \\ A_{10}S_{00}+A_{11}S_{10}+A_{12}S_{20}+A_{13}S_{30} & A_{10}S_{01}+A_{11}S_{11}+A_{12}S_{21}+A_{13}S_{31} \\ A_{20}S_{00}+A_{21}S_{10}+A_{22}S_{20}+A_{23}S_{30} & A_{20}S_{01}+A_{21}S_{11}+A_{22}S_{21}+A_{23}S_{31} \\ A_{30}S_{00}+A_{31}S_{10}+A_{32}S_{20}+A_{33}S_{30} & A_{30}S_{01}+A_{31}S_{11}+A_{32}S_{21}+A_{33}S_{31} \end{pmatrix}$$

Hardware Implementation



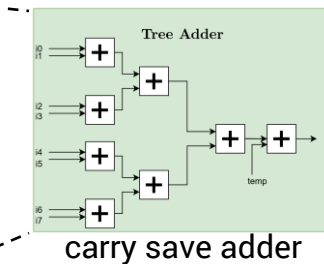
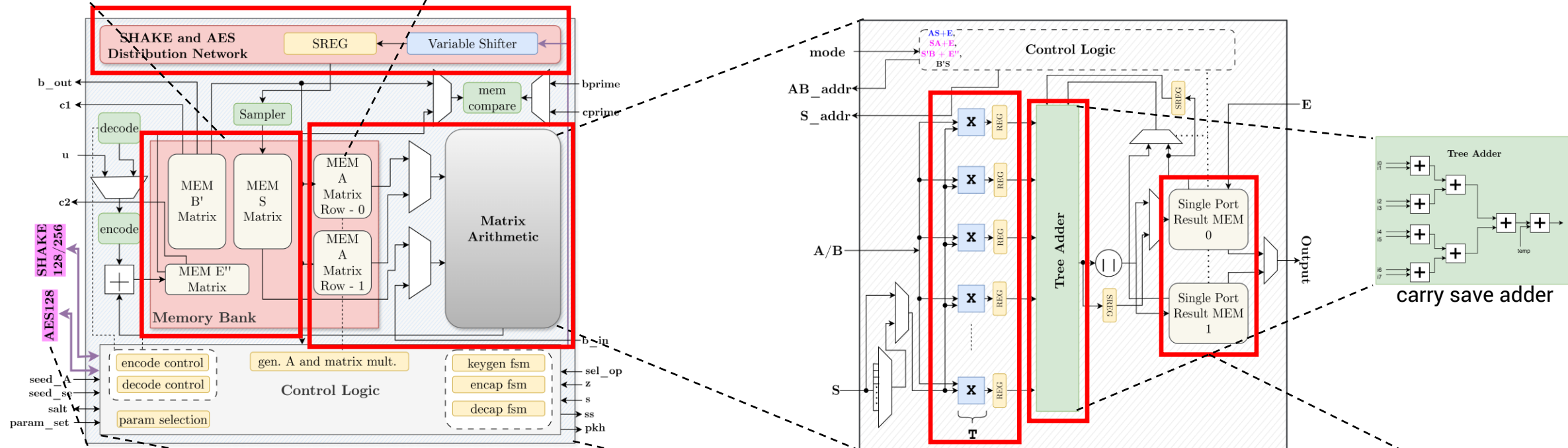
Hardware design plan

- Unified design that supports all (twelve) parameter sets at runtime
 - AES-variant
 - SHAKE-variant
- Performance parameter 'T' is used for scaling the number of dot products as synthesis-time parameter
 - This will scale the time and area
- Target Platforms
 - FPGA (Artix-7 xc7a200t)
 - ASIC (NANGATE45)
- Development and Synthesis Tools
 - AMD Vivado (FPGA)
 - OpenRoad flow (ASIC)

Hardware architecture

$S_{n \times n}$ and $E_{n \times n}$ are sampled in advance and stored

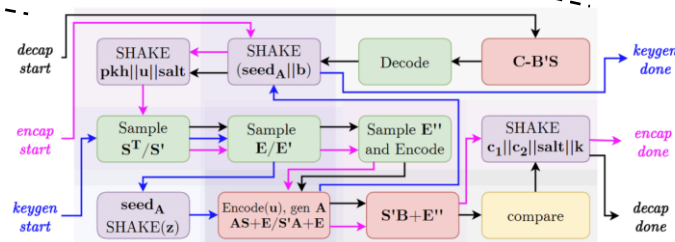
- $A_{n \times n}$ is streamed in row-wise
- Two rows of A are stored



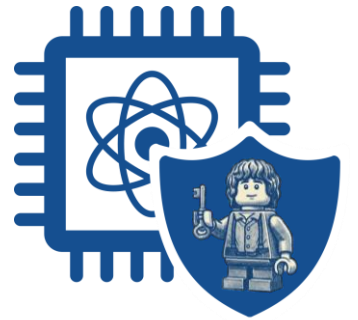
E is stored in Result Memory

- SHAKE config:
- $T=16$: 1 round per cycle
 - $T=32$: 2 rounds per cycle
 - $T=64$: 3 rounds per cycle

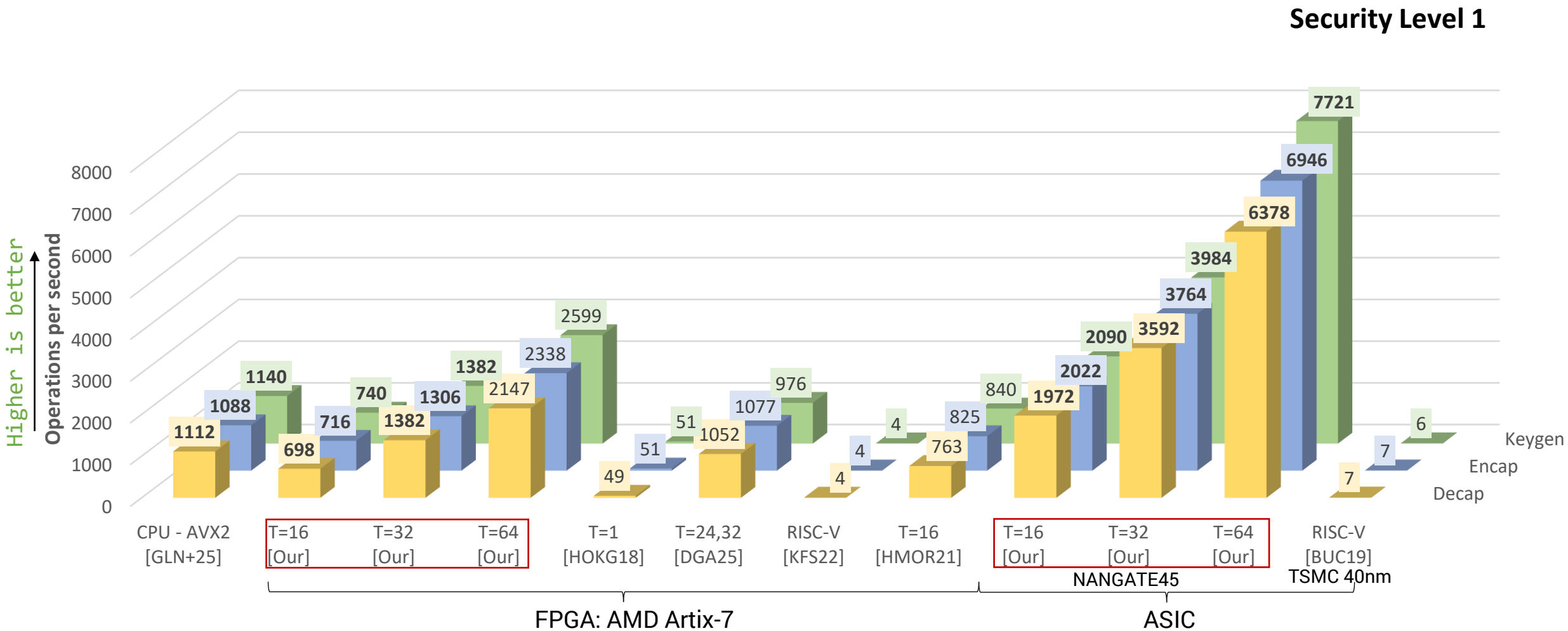
AES config: fully unrolled



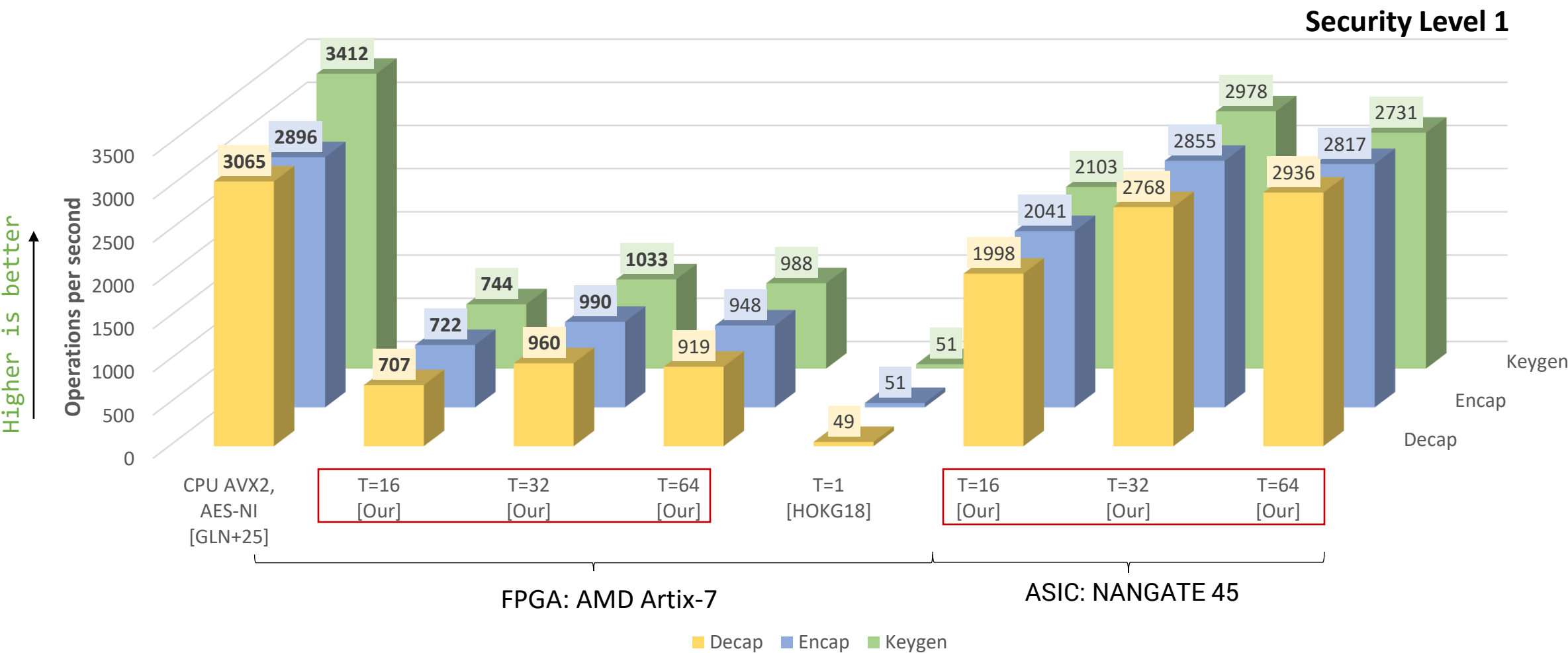
Evaluation



Performance comparison – SHAKE-based FrodoKEM

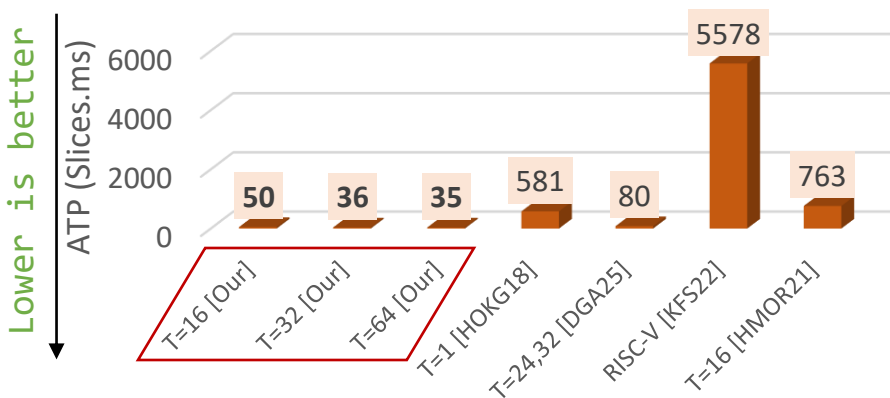


Performance comparison – AES-based



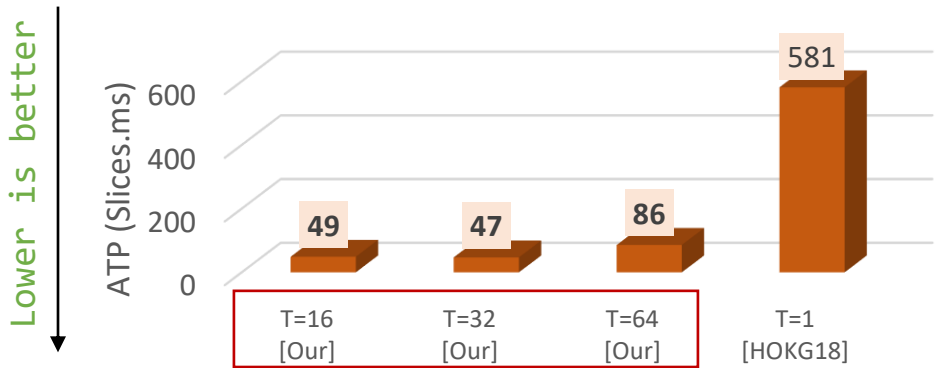
Performance comparison – Area Time Product (ATP)

FPGA – SHAKE-based FrodoKEM



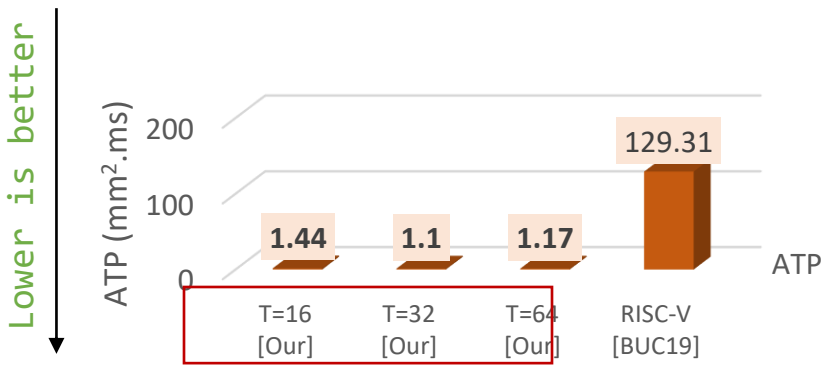
FPGA – AES-based FrodoKEM

FPGA: AMD Artix-7



$$\text{ATP} = (\text{LUT}/4 + \text{DSP} \times 100 + \text{BRAM} \times 128) \times (\text{Time}_{\text{KG}} + \text{Time}_{\text{ENC}} + \text{Time}_{\text{DEC}})$$

ASIC – SHAKE-based FrodoKEM



$$\text{ATP} = (\text{Area}_{\text{Logic}} + \text{Area}_{\text{Memory}}) \times (\text{Time}_{\text{KG}} + \text{Time}_{\text{ENC}} + \text{Time}_{\text{DEC}})$$



Side-channel security

- **Constant-time by design:** loop bounds and control flow depend only on public parameters, and $q = 2^k$ makes modular reduction simple and safe [ABD+25b]
- **First-order masking at zero overhead:** decapsulation ($M = C - B'S$) is masked using idle multipliers in the Matrix Arithmetic module, following [HMOR21]
- **Future work:** full protection also needs masked decode and key generation, plus empirical validation (e.g., TVLA)

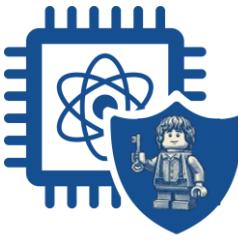
Key takeaways

- **Fully standard-compliant FrodoKEM hardware** for both FPGA and ASIC, supporting all 12 parameter sets
- **Synthesis-time parallelism (T)** lets designers trade area for throughput to fit their target
- **State-of-the-art results:** $T = 32$ and $T = 64$ outperform all prior implementations in both throughput (ops/s) and time-area product
- **Efficient scaling:** for SHAKE-based FrodoKEM, $\sim 4\times$ parallelism yields near-linear speedup at less than $2\times$ area, on both FPGA and ASIC
- At high parallelism, the bottleneck shifts from matrix multiplication to PRNG-based matrix generation

References

- [HOKG18] Howe et al., "Standard Lattice-Based Key Encapsulation on Embedded Devices," TCHES 2018
- [BUC19] Banerjee et al., "Sapphire: A Configurable Crypto-Processor for Post-Quantum Lattice-Based Protocols," TCHES 2019
- [HMOR21] Howe et al., "Exploring Parallelism to Improve the Performance of FrodoKEM in Hardware," JCN 2021
- [KFS22] Karl et al., "Hardware Accelerated FrodoKEM on RISC-V," DDECS 2022
- [BBC+23] Bos et al., "Enabling FrodoKEM on Embedded Devices," TCHES 2023
- [DGA25] Düzyol et al., "Can FrodoKEM Run in a Millisecond? FPGA Says Yes!," ePrint 2025/1312
- [GLN+25] Glabush et al., "FrodoKEM: A CCA-Secure Learning with Errors Key Encapsulation Mechanism," IACR CiC 2025
- [ABD+25] Alkim et al., "FrodoKEM Preliminary Standardization Proposal (submitted to ISO)," Sept. 2025
- [YWZL26] Wang et al., "A High-Performance and Reconfigurable Hardware Architecture for FrodoKEM," TCHES 2026
- AI-Generated Art: Lego Frodo with Key, created with Google Gemini





Thank you!



Link to the paper



Northwestern
University



Microsoft

<https://sanjaydeshpande.phd/>