Optimised Lattice-Based Key Encapsulation in Hardware

James Howe[†], Marco Martinoli[‡], Elisabeth Oswald[‡], and Francesco Regazzoni[*]

[†]PQShield, UK; [‡]University of Bristol, UK; and [*]ALaRI Institute, Switzerland

NIST's Second PQC Standardization Conference
24 August 2019

Presentation Outline

I Background

- (i) FrodoKEM and updates
- (ii) Current state-of-the-art in PQC hardware
- (iii) Keccak as a seed expander

II Optimising FrodoKEM's Throughput

- (i) What's different?
- (ii) First-order masking
- (iii) Optimising FrodoKEM in Hardware

III Results and Conclusions

- (i) Comparisons of FrodoKEM Encaps
- (ii) Comparisons of FrodoKEM Decaps
- (iii) Graphical representation of results

IV References

FrodoKEM

FrodoKEM primer:

- FrodoKEM is a lattice-based KEM.
- It bases its hardness on the (conservative) LWE problem.
- Performs well desite using unstructured lattices.

FrodoKEM updates:

- FrodoKEM makes it to round 2!
- Adds a new parameter set $(n = 1344)$ for NIST level 5 security.
- Changed PRNG / seed expander from cSHAKE to SHAKE.
- Slightly changed the error distribution parameter for FrodoKEM-640.

FrodoKEM

FrodoKEM is *still* comprised of a number of key modules:

- Matrix-matrix multiplication, of sizes $n = 640,\ 976,\ \text{and } 1344$.
- Uniform and Gaussian error generation.
- Random oracles via SHAKE for CCA security.

As well as a number of subsidiary operations:

- Matrix packing (and unpacking) to vectors.
- Message encoding and decoding.
- Parsing vectors and bit-strings.

FrodoKEM

FrodoKEM is *still* comprised of a number of key modules:

- Matrix-matrix multiplication, of sizes $n = 640,\ 976,$ and 1344.
- Uniform and Gaussian error generation.
- Random oracles via SHAKE for CCA security.

As well as a number of subsidiary operations:

- Matrix packing (and unpacking) to vectors.
- Message encoding and decoding.
- Parsing vectors and bit-strings.

How does FrodoKEM compare to other PQC in hardware?

PQC in Hardware to date

- Code-based designs have large KeyGen / decryption, but fast encryption.
- Isogeny-based also have large overall designs, but seem to be a lot slower.
- Lattice-based designs nicely balance area/performance across all operations.

Table 1: PQC on FPGA, results taken from `pqczoo.com`.

	Cryptographic Implementation	**Device**	**LUT**	**FF**	**Slice**	**DSP**	**BRAM**	**MHz**	**Thr-Put**
H	SPHINCS-256 (Total) [ACZ18]	Kin-7	19,067	3,132	7,306	3	36	525	654
Code	Niederreiter KeyGen [WSN18]	Str-V	–	–	39,122	–	827	230	75
	Niederreiter Encrypt [WSN18]	Str-V	–	6,977	4,276	–	0	448	50,000
	Niederreiter Decrypt [WSN18]	Str-V	–	48,050	20,815	–	88	290	12,500
Isogeny	SIKE 3-cores (Total) [KAK18]	Vir-7	27,713	38,489	11,277	288	61	205	27
	SIKE 6-cores (Total) [KAK18]	Vir-7	50,084	69,054	19,892	576	55	202	32
	SIKE 3-cores (Total) [RM19]	Vir-7	49,099	62,124	18,711	294	23	226	32
Lattice	NewHope KEX Server [KLC$^+$17]	Art-7	20,826	9,975	7,153	8	14	131	13,699
	NewHope KEX Client [KLC$^+$17]	Art-7	18,756	9,412	6,680	8	14	133	12,723
	NewHope KEX Server [OG17]	Art-7	5,142	4,452	1,708	2	4	125	731
	NewHope KEX Client [OG17]	Art-7	4,498	4,635	1,483	2	4	117	653
	Round5 (All) (SoC) [PQShield]	Art-7	7,168	3,337	2,344	0	–	100	–
	FrodoKEM-640 Encaps [HOKG18]	Art-7	6,745	3,528	1,855	1	11	167	51
	FrodoKEM-640 Decaps [HOKG18]	Art-7	7,220	3,549	1,992	1	16	162	49

PQC in Hardware to date

- → Code-based designs have large KeyGen / decryption, but fast encryption.
- → Isogeny-based also have large overall designs, but seem to be a lot slower.
- → Lattice-based designs nicely balance area/performance across all operations.

Table 2: PQC on FPGA, results taken from `pqczoo.com`.

	Cryptographic Implementation	Device	LUT	FF	Slice	DSP	BRAM	MHz	Thr-Put
H	SPHINCS-256 (Total) [ACZ18]	Kin-7	19,067	3,132	7,306	3	36	525	654
Code	Niederreiter KeyGen [WSN18]	Str-V	–	–	39,122	–	827	230	75
	Niederreiter Encrypt [WSN18]	Str-V	–	6,977	4,276	–	0	448	50,000
	Niederreiter Decrypt [WSN18]	Str-V	–	48,050	20,815	–	88	290	12,500
Isogeny	SIKE 3-cores (Total) [KAK18]	Vir-7	27,713	38,489	11,277	288	61	205	27
	SIKE 6-cores (Total) [KAK18]	Vir-7	50,084	69,054	19,892	576	55	202	32
	SIKE 3-cores (Total) [RM19]	Vir-7	49,099	62,124	18,711	294	23	226	32
Lattice	NewHope KEX Server [KLC$^+$17]	Art-7	20,826	9,975	7,153	8	14	131	13,699
	NewHope KEX Client [KLC$^+$17]	Art-7	18,756	9,412	6,680	8	14	133	12,723
	NewHope KEX Server [OG17]	Art-7	5,142	4,452	1,708	2	4	125	731
	NewHope KEX Client [OG17]	Art-7	4,498	4,635	1,483	2	4	117	653
	Round5 (All) (SoC) [PQShield]	Art-7	7,168	3,337	2,344	0	–	100	–
	FrodoKEM-640 Encaps [HOKG18]	Art-7	6,745	3,528	1,855	1	11	167	51
	FrodoKEM-640 Decaps [HOKG18]	Art-7	7,220	3,549	1,992	1	16	162	49

PQC in Hardware to date

- → Throughput per FPGA slice can tell us how performant designs are for the hardware resources they consume (1 Slice $\approx$ 4 LUTs $+$ 8 FFs).
- → However, this metric excludes BRAM/DSP usage $\nrightarrow$ not ASIC-friendly.
- → Not all use Artix-7 FPGAs, and require a v. expensive Virtex-7 ($50 vs $9k).

Table 3: PQC on FPGA, results taken from `pqczoo.com`.

	Cryptographic Implementation	Device	LUT	FF	Slice	DSP	BRAM	MHz	Thr-Put	Thr-Put / Slice
H	SPHINCS-256 (Total) [ACZ18]	Kin-7	19,067	3,132	7,306	3	36	525	654	0.088
Code	Niederreiter KeyGen [WSN18]	Str-V	–	–	39,122	–	827	230	75	0.002
	Niederreiter Encrypt [WSN18]	Str-V	–	6,977	4,276	–	0	448	50,000	11.693
	Niederreiter Decrypt [WSN18]	Str-V	–	48,050	20,815	–	88	290	12,500	0.601
Isogeny	SIKE 3-cores (Total) [KAK18]	Vir-7	27,713	38,489	11,277	288	61	205	27	0.002
	SIKE 6-cores (Total) [KAK18]	Vir-7	50,084	69,054	19,892	576	55	202	32	0.002
	SIKE 3-cores (Total) [RM19]	Vir-7	49,099	62,124	18,711	294	23	226	32	0.002
Lattice	NewHope KEX Server [KLC$^+$17]	Art-7	20,826	9,975	7,153	8	14	131	13,699	1.915
	NewHope KEX Client [KLC$^+$17]	Art-7	18,756	9,412	6,680	8	14	133	12,723	1.905
	NewHope KEX Server [OG17]	Art-7	5,142	4,452	1,708	2	4	125	731	0.428
	NewHope KEX Client [OG17]	Art-7	4,498	4,635	1,483	2	4	117	653	0.440
	Round5 (All) (SoC) [PQShield]	Art-7	7,168	3,337	2,344	0	–	100	–	–
	FrodoKEM-640 Encaps [HOKG18]	Art-7	6,745	3,528	1,855	1	11	167	51	0.028
	FrodoKEM-640 Decaps [HOKG18]	Art-7	7,220	3,549	1,992	1	16	162	49	0.025

Keccak as a seed expander

- For FrodoKEM [HOKG18], NewHope [OG17], and BLISS [PDG14] hardware designs, the Keccak mid-range core[1] is utilised, consuming ~750 slices.
- However, Keccak is a bottleneck in many of the PQC implementations.
- Keccak's high-speed core, increases area consumption by 3-8x [BDP$^+$12].
- This might make it more expensive than the PQC scheme itself $\not\rightarrow$ impractical.
- Recently, software implementations of PQC candidates have used alternatives:
  - FrodoKEM-640 is faster by 5x using xoshiro128** [BFM$^+$18][2].
  - Round5 is faster by 1.4x using LWC candidate SNEIK(HA) [Saa19].

---

[1] `https://keccak.team/hardware.html`

[2] This PRNG might not qualify for cryptographically secure randomness.

Keccak as a seed expander

- For FrodoKEM [HOKG18], NewHope [OG17], and BLISS [PDG14] hardware designs, the Keccak mid-range core[1] is utilised, consuming ~750 slices.
- However, Keccak is a bottleneck in many of the PQC implementations.
- Keccak's high-speed core, increases area consumption by 3-8x [BDP$^+$12].
- This might make it more expensive than the PQC scheme itself $\not\to$ impractical.
- Recently, software implementations of PQC candidates have used alternatives:
  - FrodoKEM-640 is faster by 5x using xoshiro128** [BFM$^+$18][2].
  - Round5 is faster by 1.4x using LWC candidate SNEIK(HA) [Saa19].

With parallelisation, this should also benefit hardware designs...

---

[1] `https://keccak.team/hardware.html`

[2] This PRNG might not qualify for cryptographically secure randomness.

What’s different?

- The proposed hardware designs follows FrodoKEM’s specifications, expect changing the use of SHAKE for PRNG / seed expanding.
- Instead, we propose using the more compact (unrolled) Trivium [DCP08].
- Trivium still qualifies for cryptographically secure randomness.
- Being more compact; we are able to stack more of them together to enable parallel multiplication of the (time consuming) matrix operations.

What’s different?

- The proposed hardware designs follows FrodoKEM’s specifications, expect changing the use of SHAKE for PRNG / seed expanding.
- Instead, we propose using the more compact (unrolled) Trivium [DCP08].
- Trivium still qualifies for cryptographically secure randomness.
- Being more compact; we are able to stack more of them together to enable parallel multiplication of the (time consuming) matrix operations.
- Additionally we estimate a first-order masking technique for decapsulation.

Efficient first-order masking

- The efficiency of Trivium also allows us to efficiently mask decapsulation.
- A random matrix $(\mathbf{R})$ is used to mask the operation $\mathbf{M} = \mathbf{C} - \mathbf{B}'\mathbf{S}$ as:

$$\mathbf{M}_1 = \mathbf{C} - \mathbf{B}'(\mathbf{S} + \mathbf{R}),$$

$$\mathbf{M}_2 = \mathbf{C} - \mathbf{B}'(\mathbf{S} - \mathbf{R}).$$

- Then, $\mathbf{M}$ is recovered by calculating $(\mathbf{M}_1 + \mathbf{M}_2)/2$.
- We parallelise these operations, as before, so that runtime is not affected.
- We also ensure no two operations of the same row/column are used in parallel, in case power traces can be combined to cancel out the masking.

Parallelising matrix multiplication

- We want to optimise are FrodoKEM's LWE calculations of the form:

$$\mathbf{C} \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'.$$

- $\mathbf{S}' \times \mathbf{A}$ is the real bottleneck, with at most ~7.5m 16-bit multiplications.
- Thus, we parallelise the matrix multiplication:

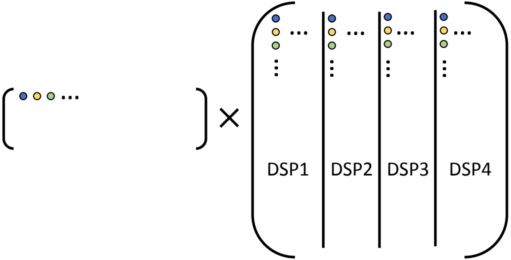

Figure 1: Parallelising matrix multiplication, for $\mathbf{S}' \times \mathbf{A}$, used within LWE computations for an example of $k = 4$ parallel multiplications.

Hardware design overview

- All designs require $k/2$ Triviums, outputing 32-bits of randomness per clock.
- Each 32-bit value is split into 16-bits and given to the DSP for MAC operations.
- Thus, we make a k-times improvement in the throughput / multiplication.

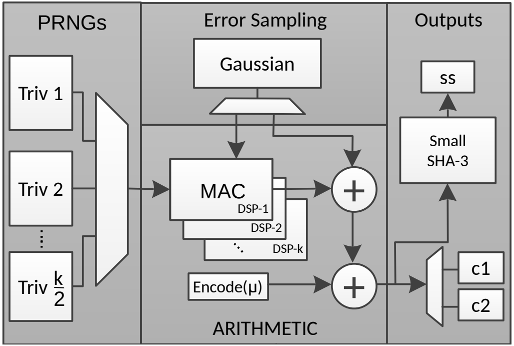

Figure 2: A high-level overview of the proposed hardware designs for FrodoKEM.

Hardware design overview

- All designs require $k/2$ Triviums, outputing 32-bits of randomness per clock.
- Each 32-bit value is split into 16-bits and given to the DSP for MAC operations.
- Thus, we make a k-times improvement in the throughput / multiplication.
- **But how does this affect the area consumption of the hardware designs?**

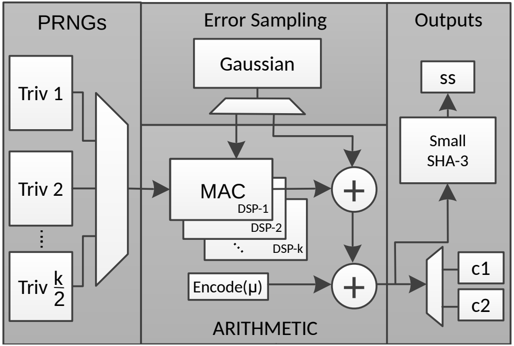

Figure 2: A high-level overview of the proposed hardware designs for FrodoKEM.

Table of results (encapsulation)

- We provide results for Encaps for two parameter sets.
- We reduce area consumption by ∼40% for the smallest Encaps design.
- We also increase the throughput by >16x and are still smaller than the state-of-the-art [HOKG18] without using BRAM.

Table 4: Artix-7 FPGA resource consumption of the proposed FrodoKEM Encaps hardware designs, using Trivium and *k* parallel multipliers. Results with BRAM usage have an asterisk (*).

FrodoKEM Protocol	LUT	FF	Slices	DSP	BRAM	MHz	Thr-Put
Encaps-640 1x	4,246	2,131	1,180	1	0	190	58
Encaps-640 4x	4,620	2,552	1,338	4	0	183	221
Encaps-640 8x	5,155	3,356	1,485	8	0	177	427
Encaps-640 16x	5,796	4,694	1,692	16	0	171	825
Encaps-640 [HOKG18]	6,745	3,528	1,855	1	11	167	51
Encaps-976 1x	4,650	2,118	1,272	1	0	187	25
Encaps-976 4x	4,996	2,611	1,455	4	0	180	94
Encaps-976 8x	5,562	3,349	1,608	8	0	175	183
Encaps-976 16x	6,188	4,678	1,782	16	0	168	350
Encaps-976 [HOKG18]	7,209	3,537	1,985	1	16	167	22

Table of results (decapsulation)

- We provide results for Decaps for two parameter sets.
- We reduce area consumption by $\sim$40% for the smallest Decaps design.
- We also increase the throughput by >14x and are still smaller than [HOKG18].

Table 5: Artix-7 FPGA resource consumption of the proposed FrodoKEM Decaps hardware designs, using Trivium and *k* parallel multipliers. Results with BRAM usage have an asterisk (*).

FrodoKEM Protocol	**LUT**	**FF**	**Slices**	**DSP**	**BRAM**	**MHz**	**Thr-Put**
*Decaps-640 1x	4,466	2,152	1,254	1	12.5	162	49
Decaps-640 1x	10,518	2,299	2,933	1	0	190	57
*Decaps-640 16x	6,881	5,081	1,947	16	12.5	149	710
Decaps-640 16x	14,528	5,335	4,020	16	0	160	763
*Decaps-640 [HOKG18]	7,220	3,549	1,992	1	16	162	49
*Decaps-976 1x	4,888	2,153	1,390	1	19	162	21
Decaps-976 1x	14,217	2,295	3,956	1	0	188	25
*Decaps-976 16x	7,213	5,087	2,042	16	19	148	306
Decaps-976 16x	18,960	5,285	5,274	16	0	157	325
*Decaps-976 [HOKG18]	7,773	3,559	2,158	1	24	162	21

Graphical representation of results

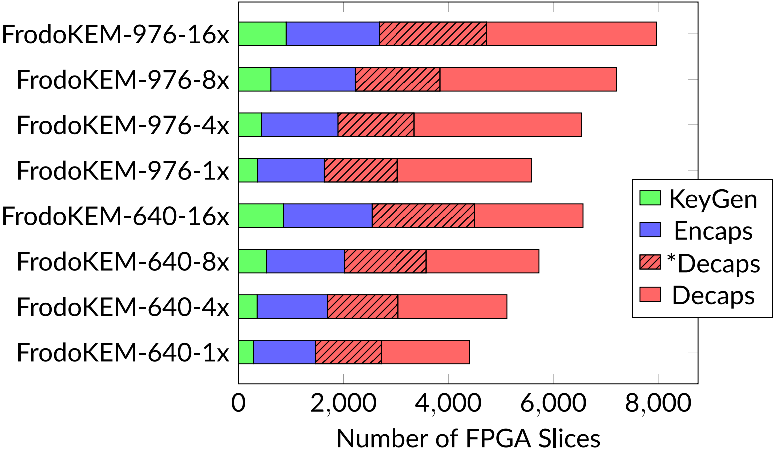

Figure 3: FPGA slice consumption of FrodoKEM protocols on a Xilinx Artix-7. Decaps values overlap to show results with (*) and without BRAM.

Graphical representation of performance

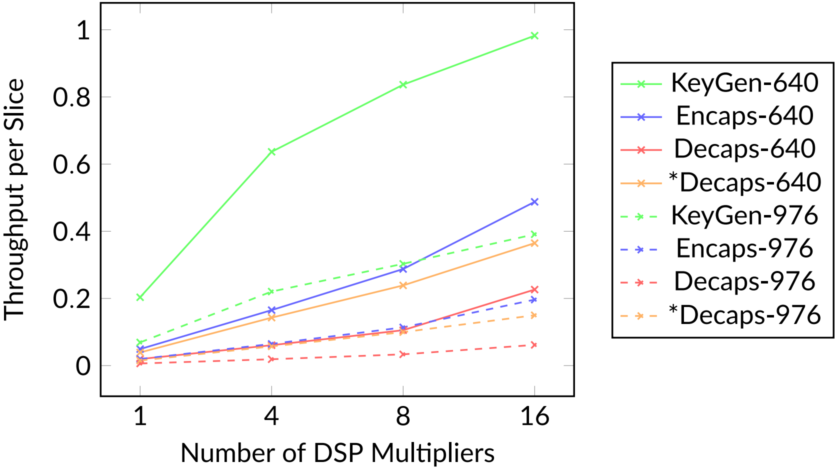

Figure 4: Comparison of the **throughput per slice** performance on Xilinx Artix-7 FPGA.

Conclusions

- We propose an alternative hardware design for FrodoKEM, using an unrolled Trivium as PRNG.
- We universally save $\sim$40% in hardware resources on the FPGA for the same throughput performance.
- Moreover, by using the same FPGA area we are able to increase the throughput, universally, by $\sim$16x.
- It would be interesting to see how other PQC schemes would benefit from this change, too.

Conclusions

- We propose an alternative hardware design for FrodoKEM, using an unrolled Trivium as PRNG.
- We universally save $\sim$40% in hardware resources on the FPGA for the same throughput performance.
- Moreover, by using the same FPGA area we are able to increase the throughput, universally, by $\sim$16x.
- It would be interesting to see how other PQC schemes would benefit from this change, too.
- **Thanks for listening! Any question?**

References I

Dorian Amiet, Andreas Curiger, and Paul Zbinden.
FPGA-based Accelerator for Post-Quantum Signature Scheme SPHINCS-256.
IACR Transactions on Cryptographic Hardware and Embedded Systems, pages 18–39, 2018.

Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche, and Ronny Van Keer.
Keccak implementation overview.
URL: http://keccak. neokeon. org/Keccak-implementation-3.2. pdf, 2012.

Joppe W. Bos, Simon Friedberger, Marco Martinoli, Elisabeth Oswald, and Martijn Stam.
Fly, you fool! faster frodo for the arm cortex-m4.
Cryptology ePrint Archive, Report 2018/1116, 2018.
`https://eprint.iacr.org/2018/1116`.

Christophe De Canniere and Bart Preneel.
Trivium.
In *New Stream Cipher Designs*, pages 244–266. Springer, 2008.

James Howe, Tobias Oder, Markus Krausz, and Tim Güneysu.
Standard lattice-based key encapsulation on embedded devices.
IACR Transactions on Cryptographic Hardware and Embedded Systems, pages 372–393, 2018.

Brian Koziel, Reza Azarderakhsh, and Mehran Mozaffari Kermani.
A high-performance and scalable hardware architecture for isogeny-based cryptography.
IEEE Transactions on Computers, 67(11):1594–1609, 2018.

Po-Chun Kuo, Wen-Ding Li, Yu-Wei Chen, Yuan-Che Hsu, Bo-Yuan Peng, Chen-Mou Cheng, and Bo-Yin Yang.
High performance post-quantum key exchange on FPGAs.
Cryptology ePrint Archive, Report 2017/690, 2017.
`https://eprint.iacr.org/2017/690`.

References II

Tobias Oder and Tim Güneysu.
Implementing the NewHope-simple key exchange on low-cost FPGAs.
Progress in Cryptology–LATINCRYPT, 2017, 2017.

Thomas Pöppelmann, Léo Ducas, and Tim Güneysu.
Enhanced lattice-based signatures on reconfigurable hardware.
In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 353–370. Springer, 2014.

Debapriya Basu Roy and Debdeep Mukhopadhyay.
Post Quantum ECC on FPGA Platform.
Cryptology ePrint Archive, Report 2019/568, 2019.
`https://eprint.iacr.org/2019/568`.

Markku-Juhani O. Saarinen.
Exploring nist lwc/pqc synergy with r5sneik: How sneik 1.1 algorithms were designed to support round5.
Cryptology ePrint Archive, Report 2019/685, 2019.
`https://eprint.iacr.org/2019/685`.

Wen Wang, Jakub Szefer, and Ruben Niederhagen.
FPGA-based Niederreiter cryptosystem using binary Goppa codes.
In *International Conference on Post-Quantum Cryptography*, pages 77–98. Springer, 2018.