11 September 2018

Standard Lattice-Based Key Encapsulation on Embedded Devices

James Howe[†], Tobias Oder[‡], Markus Krausz[‡], and Tim Güneysu[‡*].

[†]University of Bristol, UK; [‡]Ruhr-Universität Bochum, Germany; and [*]DFKI, Germany.

11 September 2018

Outline

- ~~Post-quantum cryptography and LWE~~
- Motivation
- Introduction to Frodo
- Microcontroller design
- Hardware design
- Results and performance analysis

Motivation

- NIST have started a post-quantum standardisation "competition".
- The call suggests future rounds will likely involve:
  - Evaluations on constrained devices, such as smart cards,
  - as well as comparisons of the schemes in hardware.
- Why focus on lattice-based / Frodo?
  - Extremely versatile and theoretically sound.
  - Probably the most secure lattice candidate.
  - Less implementations than ideal lattice schemes; has larger keys and no NTT.
  - Frodo is ideal for long-term security and constrained (hardware) platforms.

Frodo: Take off the ring!

The design philosophy of FrodoKEM [ABD+] combines:

- Conservative yet practical post-quantum constructions.
- Security derived from cautious parameterizations of the well-studied learning with errors problem.
- Thus, close connections to conjectured-hard problems on generic, “algebraically unstructured” lattices.
- Parameter selection is far less constrained than vs ideal lattice schemes.

Frodo: Why should we take off the ring?

These qualities are appealing for practitioners;

- Many IoT use cases require long-term, efficient cryptography.
- Post-quantum cryptography is becoming essential.
- Microcontrollers and FPGAs will play a role in future technologies.
- Suitable for use cases such as satellite communications and V2X.

Frodo: key encapsulation from standard lattices

Algorithm 1 The FrodoKEM encapsulation (shortened)

procedure ENCAPS($pk = \text{seed}_{\mathbf{A}} || \mathbf{b}$)

  Choose a uniformly random key $\mu \leftarrow U(\{0,1\}^{\text{len}_\mu})$

  Generate pseudo-random values $\text{seed}_{\mathbf{E}} || \mathbf{k} || \mathbf{d} \leftarrow G(pk||\mu)$

  Sample error matrix $\mathbf{S}', \mathbf{E}' \leftarrow \text{Frodo.SampleMatrix}(\text{seed}_{\mathbf{E}}, \bar{m}, n, T_\chi, \cdot)$

  Generate the matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ via $\mathbf{A} \leftarrow \text{Frodo.Gen}(\text{seed}_{\mathbf{A}})$

  Compute $\mathbf{C}_1 \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$

  Sample error matrix $\mathbf{E}'' \leftarrow \text{Frodo.SampleMatrix}(\text{seed}_{\mathbf{E}}, \bar{m}, \bar{n}, T_\chi, \cdot)$

  Compute $\mathbf{C}_2 \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}'' + \text{Frodo.Encode}(\mu)$

  Compute $\mathbf{ss} \leftarrow F(\mathbf{c}_1 || \mathbf{c}_2 || \mathbf{k} || \mathbf{d})$

  return ciphertext $\mathbf{c}_1 || \mathbf{c}_2 || \mathbf{d}$ and shared secret $\mathbf{ss}$

end procedure

Frodo: key encapsulation from standard lattices

FrodoKEM is comprised of a number of key modules:

- Matrix-matrix multiplication, up to sizes 976.
- Uniform and “Gaussian” error generation.
- Random oracles via cSHAKE for CCA security.

A massive design challenge was to balance **memory utilisation**, whilst not deteriorating the **performance** too much to not overexert the limited computing capabilities of the embedded devices.

FrodoKEM on constrained devices

FrodoKEM has a number of design options we cover:

- Both sets of parameters;
  - FrodoKEM-640 aims to match AES-128 security.
  - FrodoKEM-976 aims to match AES-192 security.
- PRNG from AES and cSHAKE modules.
- We focus on FrodoKEM, rather than the previous key exchange scheme FrodoCCS [BCD+16].

FrodoKEM on ARM

Contribution overview:

- Optimized memory allocation that makes the implementation small enough to fit on embedded microcontrollers.
- An assembly multiplication routine that speeds up our implementation, realizing a performance that fits the requirements of common use-cases.
- Utilises constant runtime to protect against simple side-channel analysis.
- FrodoKEM-640 has a total execution time of 836 ms, running at 168 MHz.

FrodoKEM on ARM

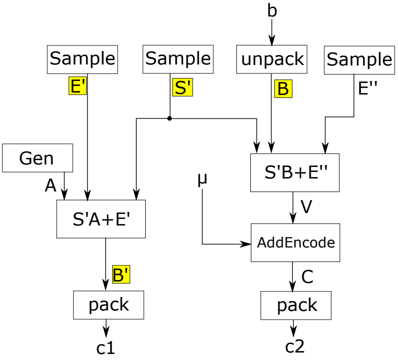


Figure: FrodoKEM encaps flowchart.

- We analysed the memory occupancy during each operation.
- Wherever possible, reusing already allocated memory.
- This minimised the memory usage for all designs.
- Memory usage for AES versions much simpler than for cSHAKE versions.

Results and Comparisons

- Clear difference between AES and cSHAKE implementations.
- Due to more efficent AES [SS16], cSHAKE needs load/save from RAM.
- Outperforms other Frodo design, but much slower than Kyber / NewHope.

Table: Cycle counts for our full microcontroller implementations (at 168 MHz).

Implementation	Platform	Security Level	Cycle counts
FrodoKEM-640-AES	Cortex-M4	128 bits	140,398,055
FrodoKEM-976-AES	Cortex-M4	192 bits	315,600,317
FrodoKEM-640-cSHAKE	Cortex-M4	128 bits	310,131,435
FrodoKEM-976-cSHAKE	Cortex-M4	192 bits	695,001,098
FrodoKEM-640-cSHAKE [pqm]	Cortex-M4	128 bits	318,037,129
KyberNIST-768 [pqm]	Cortex-M4	192 bits	4,224,704
NewHopeUSENIX-1024 [AJS16]	Cortex-M4	255 bits	2,561,438
ECDH scalar multiplication [DHH+15]	Cortex-M0	pre-quantum	3,589,850

Results and Comparisons

- Despite being slower, cSHAKE requires less memory than AES.
- Our memory optimisations save between 30-40% compared to PQM4.
- Versus the referenced designs we also save 66% in peak stack usage.

Table: Stack usage in bytes for our microcontroller implementations.

	FrodoKEM-AES		FrodoKEM-cSHAKE		FrodoKEM-cSHAKE [pqm]	
Operation	$n = 640$	$n = 976$	$n = 640$	$n = 976$	$n = 640$	% Savings
Keypair	23,396	35,484	22,376	33,800	36,536	39%
Encaps	41,292	63,484	37,792	57,968	58,328	35%
Decaps	51,684	63,628	48,184	58,112	68,680	30%

FrodoKEM on FPGA

Contribution overview:

- Proposes a generic LWE multiplication core which computes vector-matrix multiplication and error addition.
- Generates future random values in parallel, minimising delays between vector-matrix multiplications.
- Hybrid pre-calculated / on-the-fly memory management is used, which continuously updates previous values.
- Ensures constant runtime by parallelising other modules with multiplication.
- FrodoKEM-640 has a total execution time of 60 ms, running at 167MHz.

FrodoKEM on FPGA

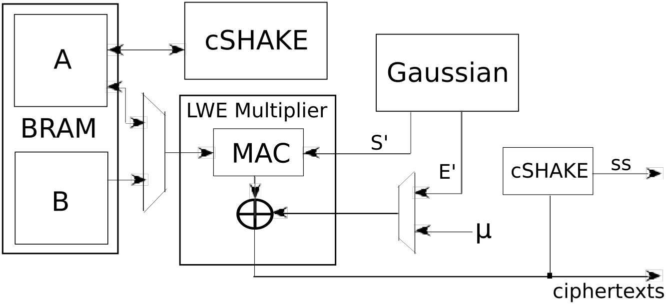


Figure: An overview of our FPGA design of FrodoKEM Encapsulation.

Results and Comparisons

- Competes with NewHope area consumption, but much slower performance.
- Huge savings in BRAM compared to LWE Encryption [HMO+16].

Table: FPGA consumption and performance of our proposed designs, benchmarked on Artix-7.

Cryptographic Operation	LUT/FF	Slice	DSP	BRAM	MHz	Ops/sec
FrodoKEM-640 Keypair	6621/3511	1845	1	6	167	51
FrodoKEM-640 Encaps	6745/3528	1855	1	11	167	51
FrodoKEM-640 Decaps	7220/3549	1992	1	16	162	49
FrodoKEM-976 Keypair	7155/3528	1981	1	8	167	22
FrodoKEM-976 Encaps	7209/3537	1985	1	16	167	22
FrodoKEM-976 Decaps	7773/3559	2158	1	24	162	21
cSHAKE*	2744/1685	766	0	0	172	1.2m
Error+AES Sampler*	1901/1140	756	0	0	184	184m
NewHopeUSENIX Server [OG17]	5142/4452	1708	2	4	125	731
NewHopeUSENIX Client [OG17]	4498/4635	1483	2	4	117	653
LWE Encryption [HMO+16]	6078/4676	1811	1	73	125	1272

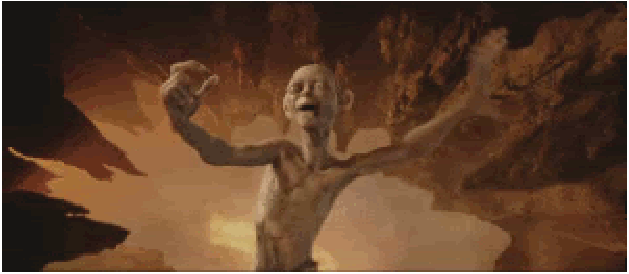
Conclusions

- We show that hardware significantly minimises the performance distance between standard and ideal lattice-based KEM, able to utilise less than 2000 slices and remain practical.
- Memory optimisations for microcontrollers show 66% savings vs reference design and 40% vs optimised PQM4 design.
- It would be interesting to see results for Frodo on FPGA with increased multipliers. As well as how it performs vs. other NIST PQC candidates.

Conclusions

- Our results show the efficiency of FrodoKEM and help to assess the practical performance of a possible future post-quantum standard.

Although rings are still good to use, unless you're Gollum...



Thank you for listening. Any questions?

References I

Erdem Alkim, Joppe W. Bos, Léo Ducas, Patrick Longa, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Chris Peikert, Ananth Raghunathan, Douglas Stebila, Karen Easterbrook, and Brian LaMacchia.
FrodoKEM Learning With Errors key encapsulation.
https://frodokem.org/files/FrodoKEM-specification-20171130.pdf.
Accessed: 2018-04-13.

Erdem Alkim, Philipp Jakubeit, and Peter Schwabe.
NewHope on ARM cortex-M.
In International Conference on Security, Privacy, and Applied Cryptography Engineering, pages 332–349. Springer, 2016.

Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila.
Frodo: Take off the ring! practical, quantum-secure key exchange from LWE.
In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016, pages 1006–1018. ACM, 2016.

References II

Michael Düll, Björn Haase, Gesine Hinterwälder, Michael Hutter, Christof Paar, Ana Helena Sánchez, and Peter Schwabe.
High-speed curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers.
Des. Codes Cryptography, 77(2-3):493–514, 2015.

James Howe, Ciara Moore, Máire O'Neill, Francesco Regazzoni, Tim Güneysu, and K. Beeden.
Lattice-based encryption over standard lattices in hardware.
In Proceedings of the 53rd Annual Design Automation Conference, DAC 2016, Austin, TX, USA, June 5-9, 2016, pages 162:1–162:6. ACM, 2016.

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

pqm4 - post-quantum crypto library for the ARM Cortex-M4.
`https://github.com/mupq/pqm4`.
Accessed: 2018-04-12.

References III

Peter Schwabe and Ko Stoffelen.
All the AES you need on Cortex-M3 and M4.
In Roberto Avanzi and Howard M. Heys, editors, Selected Areas in Cryptography - SAC 2016 - 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers, volume 10532 of Lecture Notes in Computer Science, pages 180–194. Springer, 2016.