University of BRISTOL

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.

Outline

- Introduction
  - Post-quantum cryptography
  - Lattice-based cryptography
  - Previous implementations
- Motivation
  - NIST PQC standardisation
  - Taking off the ring!
- Introduction to Frodo
- Microcontroller design
- Hardware design
- Results and performance analysis

Section 1

Introductions

Motivation

- What happens when quantum computers become a reality 10-15 years from now?
- Commonly used public-key cryptographic algorithms (based on integer factorization and discrete log problem) such as:

  RSA, DSA, Diffie-Hellman Key Exchange, ECC, ECDSA

  will be vulnerable to Shor's algorithm and will no longer be secure.

  - "Worse than Y2K: quantum computing and the end of privacy" – Forbes, 2018.
  - "The quantum clock is ticking on encryption - and your data is under threat" – Wired, 2016.
  - "Unbreakable: The race to protect our secrets from quantum hacks" – New Scientist, 2018.

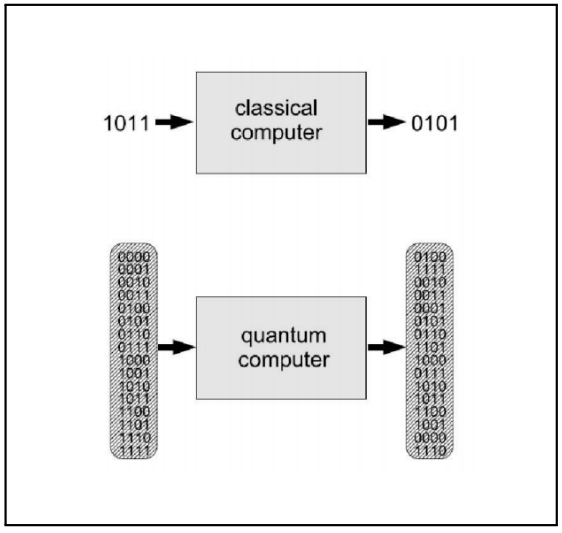
Motivation

The industry is starting to take this threat seriously.

- Microsoft Research and IBM Research.
- Infineon and NXP Semiconductors.
- PQShield and ISARA.
- National Cyber Security Centre (NCSC) and probably more...

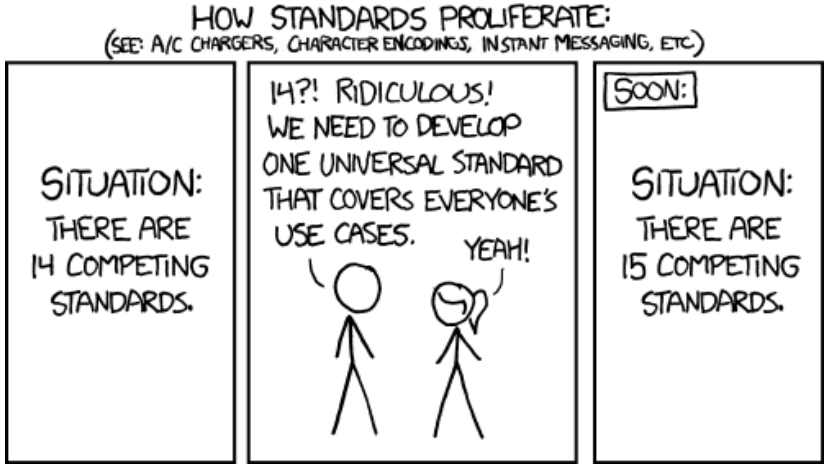
Post-quantum cryptography

- Quantum computers exploit the power of parallelism.
  - Some classically hard computational problems are now trivial.
- Shor's Algorithm (1994)
  - Can quickly factorise large numbers (exponential speed-up).
  - Significant implications for current public-key cryptography.
- Grover's Algorithm (1996)
  - Can search an unsorted database faster than a conventional computer, effects symmetric-key cryptography, so AES-128 now 64-bit secure.



Post-Quantum Cryptography

- NIST have started a post-quantum standardisation “competition”.
  - Similar to previous AES and SHA-3 standardisations.
- Submissions breakdown: 42% lattice-based, 25% code-based, 18% multivariate, 9% other, 3% hash-based, 2% SIDH.
- ETSI researching requirements for quantum-safe real-world deployments.



Related work (Microcontroller)

- Code-based relatively low memory consumption, slow performance.
- Lattices have good performance, isogenies significantly slower.

Crypto. Scheme	PQ Type	Device	Memory	Cycles
QC-MDPC Encrypt [HVMG13]	Code	ATxmega256	3705 Bytes	37,440,137
QC-MDPC Decrypt [HVMG13]	Code	ATxmega256	5496 Bytes	26,767,463
SIKE (Total) [SLLH18]	Isogenies	Cortex-A53	$\leq$35k Bytes	133,300,000
Saber Encaps [KMRV18]	Lattice	Cortex-M4	7k Bytes	1,530,000
Saber Decaps [KMRV18]	Lattice	Cortex-M4	8k Bytes	1,635,000
Kyber768 Encaps [pqm]	Lattice	Cortex-M4	13.5k Bytes	1,497,789
Kyber768 Decaps [pqm]	Lattice	Cortex-M4	14.5k Bytes	1,526,564
FrodoKEM-640-cSHAKE Encaps [pqm]	Lattice	Cortex-M4	58k Bytes	111,688,861
FrodoKEM-640-cSHAKE Decaps [pqm]	Lattice	Cortex-M4	68k Bytes	112,156,317
NewHope KEX [AJS16]	Lattice	Cortex-M4	23k Bytes	2,561,438
ECDH scalar multiplication [DHH$^+$15]	**ECC**	**Cortex-M0**	**8k Bytes**	**3,589,850**

Related work (FPGA)

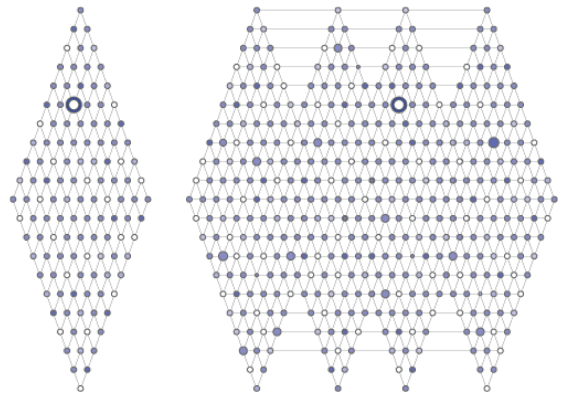
- Code-based systems have huge KeyGen / decryption, but fast encryption.
- Isogenies have fairly small designs but can be a lot slower.

Table: FPGA consumption and performance of related post-quantum schemes.

Crypto. Scheme	PQ Type	Device	LUT/FF	Slice	DSP	BRAM	MHz	Ops/sec
Niederreiter KeyGen [WSN18]	Code	Stratix-V	-/-	39122	-	827	230	75
Niederreiter Encrypt [WSN18]	Code	Stratix-V	-/6977	4276	-	0	448	83k
Niederreiter Decrypt [WSN18]	Code	Stratix-V	-/48050	20815	-	88	290	12k
SIDH (Total) [KAKJ17]	Isogenies	Virtex-7	13k/15k	5k	64	33	191	22
NewHope KEX Server [KLC+17]	Lattice	Artix-7	20826/9975	7153	8	14	131	19k
NewHope KEX Client [KLC+17]	Lattice	Artix-7	18756/9412	6680	8	14	133	12.7k
NewHope KEX Server [OG17]	Lattice	Artix-7	5142/4452	1708	2	4	125	731
NewHope KEX Client [OG17]	Lattice	Artix-7	4498/4635	1483	2	4	117	653
LWE Encryption [HMO+16]	Lattice	Spartan-6	6078/4676	1811	1	73	125	1272
ECDH [SG14]	**Curve25519**	**Zynq 7020**	**2783/3592**	**1029**	**20**	**2**	**200**	**2519**

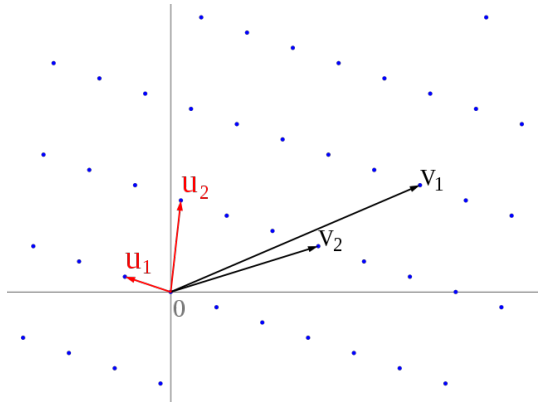
Why focus on lattice-based cryptography?

- In many cases, lattice-based cryptography outperforms RSA and ECC with competitive key / signature / ciphertext sizes [HPO+15].
- More versatile than code-based, isogeny-based, multivariate, and hash-based cryptography.
- Can be used for encryption, signatures, FHE, IBE, ABE, etc...
- Theoretical foundations are well-studied, no serious breaks (yet!).



Lattice-based cryptography in practice

- Lattice-based cryptography is important in its own right.
  - Benefits from simple mathematical operations such as integer multiplication, addition, and modular reduction.
- Lattice-based cryptography is flourishing:
  - 40% lattice-based NIST PQC submissions.
  - NewHope key exchange created.
  - Ring-LWE encryption and BLISS signatures outperform RSA and ECC in s/w and h/w.



- Lattice-based cryptography is already being considered:
  - VPN strongSwan supports post-quantum mode.
  - NewHope awarded Internet Defense Prize Winner 2016.
  - Google experimenting with NewHope key exchange.

The Learning With Errors Problem

- There is a secret vector $\mathbf{s} \leftarrow \mathbb{Z}_q^n$.
- An oracle (who knows $\mathbf{s}$) generates a uniform matrix $\mathbf{A}$ and noise vector $\mathbf{e}$ distributed normally with standard deviation αq.
- The oracle outputs:

$$(\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \mod q).$$

- The distribution of $\mathbf{A}$ is uniformly random, $\mathbf{b}$ is pseudo-random.

The Learning With Errors Problem

- There is a secret vector $\mathbf{s} \leftarrow \mathbb{Z}_q^n$.
- An oracle (who knows $\mathbf{s}$) generates a uniform matrix $\mathbf{A}$ and noise vector $\mathbf{e}$ distributed normally with standard deviation αq.
- The oracle outputs:

$$(\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \mod q).$$

- The distribution of $\mathbf{A}$ is uniformly random, $\mathbf{b}$ is pseudo-random.
- Can you find $\mathbf{s}$, given access to $(\mathbf{A}, \mathbf{b})$?

The Learning With Errors Problem

- There is a secret vector $\mathbf{s} \leftarrow \mathbb{Z}_q^n$.
- An oracle (who knows $\mathbf{s}$) generates a uniform matrix $\mathbf{A}$ and noise vector $\mathbf{e}$ distributed normally with standard deviation αq.
- The oracle outputs:

$$(\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \mod q).$$

- The distribution of $\mathbf{A}$ is uniformly random, $\mathbf{b}$ is pseudo-random.
- Can you find $\mathbf{s}$, given access to $(\mathbf{A}, \mathbf{b})$?
- Can you distinguish $(\mathbf{A}, \mathbf{b})$ from a uniformly random $(\mathbf{A}, \mathbf{b}')$?

Classes of lattices (simplified)

- Lattice-based cryptographic schemes generally fall under three classes.

$$\textbf{LWE} \longleftrightarrow \textbf{Module-LWE} \longleftrightarrow \textbf{Ring-LWE}$$

Classes of lattices (simplified)

- Lattice-based cryptographic schemes generally fall under three classes.

$$\textbf{LWE} \longleftrightarrow \textbf{Module-LWE} \longleftrightarrow \textbf{Ring-LWE}$$

- Added structures hinder security.

$$\textbf{LWE} \geq^{\text{sec.}} \textbf{Module-LWE} \geq^{\text{sec.}} \textbf{Ring-LWE}$$

Classes of lattices (simplified)

- Lattice-based cryptographic schemes generally fall under three classes.

$$\textbf{LWE} \longleftrightarrow \textbf{Module-LWE} \longleftrightarrow \textbf{Ring-LWE}$$

- Added structures hinder security.

$$\textbf{LWE} \geq^{\text{sec.}} \textbf{Module-LWE} \geq^{\text{sec.}} \textbf{Ring-LWE}$$

- However, it can also gain performance.

$$\textbf{LWE} \leq^{\text{per.}} \textbf{Module-LWE / Ring-LWE}$$

Classes of lattices (simplified)

- Lattice-based cryptographic schemes generally fall under three classes.

$$\textbf{LWE} \longleftrightarrow \textbf{Module-LWE} \longleftrightarrow \textbf{Ring-LWE}$$

- Added structures hinder security.

$$\textbf{LWE} \geq^{\text{sec.}} \textbf{Module-LWE} \geq^{\text{sec.}} \textbf{Ring-LWE}$$

- However, it can also gain performance.

$$\textbf{LWE} \leq^{\text{per.}} \textbf{Module-LWE} \,/\, \textbf{Ring-LWE}$$

- How does Ring-LWE compare with Module-LWE? What about NTRU?

Structured lattices in practice

Table: Microcontroller cycle counts of related lattice-based schemes.

Crypto. Scheme	Lattice Type	Device	Memory	Cycles
*Saber Encaps [KMRV18]	Module-LWE	Cortex-M4	7k Bytes	1,530,000
*Saber Decaps [KMRV18]	Module-LWE	Cortex-M4	8k Bytes	1,635,000
*Kyber768 Encaps [pqm]	Module-LWE	Cortex-M4	13.5k Bytes	1,497,789
*Kyber768 Decaps [pqm]	Module-LWE	Cortex-M4	14.5k Bytes	1,526,564
*NewHope KEM Encaps [pqm]	Ring-LWE	Cortex-M4	17.5k Bytes	1,966,358
*NewHope KEM Decaps [pqm]	Ring-LWE	Cortex-M4	19.5k Bytes	1,977,753
*FrodoKEM-640-cSHAKE Encaps [pqm]	LWE	Cortex-M4	58k Bytes	111,688,861
*FrodoKEM-640-cSHAKE Decaps [pqm]	LWE	Cortex-M4	68k Bytes	112,156,317
NTRU-HRSS-KEM KeyGen [pqm]	NTRU	Cortex-M4	10k Bytes	197,262,297
NTRU-HRSS-KEM Encaps [pqm]	NTRU	Cortex-M4	9k Bytes	5,166,153
NTRU-HRSS-KEM Decaps [pqm]	NTRU	Cortex-M4	10k Bytes	15,069,480
Str-NTRU-prime KEM KeyGen [pqm]	NTRU	Cortex-M4	14.5k Bytes	147,543,618
Str-NTRU-prime KEM Encaps [pqm]	NTRU	Cortex-M4	11k Bytes	10,631,675
Str-NTRU-prime KEM Decaps [pqm]	NTRU	Cortex-M4	16k Bytes	30,641,200

Frodo: Why should we 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;

- Probably the most secure lattice-based candidate.
  - Many IoT use cases require long-term, efficient cryptography.
- Frodo is ideal for long-term security and constrained (hardware) platforms.
  - Suitable for use cases such as satellite communications and V2X.
- Frodo is extremely versatile and theoretically sound.
- However, it has less implementations than ideal lattice schemes.
  - And how do we manage the larger keys and no NTT...

Frodo's inception as a key exchange

- Frodo was initially proposed as a key exchange scheme [BCD+16].
- Inspired by Regev [Reg05], Lindner and Peikert [LP11], Brakerski et al. [BLP+13], Peikert [Pei14], Bos et al. [BCNS15] and others...
- Uses a Diffie-Hellman-like protocol.
  - Key exchanges are hidden in LWE instances.
- NewHope's proposal very similar but uses Ring-LWE.
- Frodo trades some efficiency for higher security / trust vs NewHope.

Frodo vs NewHope key exchange

- Both offer strong classical and post-quantum secure parameters.
- NewHope requires polynomial multiplication, gains efficiency via NTT.
  - This makes parameters selection much more restrictive.
- Frodo is more simple, uses matrix multiplication and addition.
  - Parameters selection does not have any restrictions on structure.
- This allows Frodo to be more flexible and easier to scale.

Frodo vs NewHope key exchange

- Both offer strong classical and post-quantum secure parameters.
- NewHope requires polynomial multiplication, gains efficiency via NTT.
  - This makes parameters selection much more restrictive.
- Frodo is more simple, uses matrix multiplication and addition.
  - Parameters selection does not have any restrictions on structure.
- This allows Frodo to be more flexible and easier to scale.

But what about performance...

Frodo key exchange: Faster than a trip to Mordor

- Bandwidth for NewHope around 4 KB and Frodo around 22 KB.
- Integrated into TLS and combined with ECDHE:
  - Frodo is 1.5x slower than ECDHE for 1 Byte payload.
  - Frodo is 1.2x slower than ECDHE for 100 KByte payload.
- Integrated into TLS and combined with ECDHE:
  - Frodo is 1.6x slower than NewHope for 1 Byte payload.
  - Frodo is 1.4x slower than NewHope for 100 KByte payload.
- Frodo outperforms NTRU (EES743EP1) and SIDH.

Frodo key exchange: Faster than a trip to Mordor

- Bandwidth for NewHope around 4 KB and Frodo around 22 KB.
- Integrated into TLS and combined with ECDHE:
  - Frodo is 1.5x slower than ECDHE for 1 Byte payload.
  - Frodo is 1.2x slower than ECDHE for 100 KByte payload.
- Integrated into TLS and combined with ECDHE:
  - Frodo is 1.6x slower than NewHope for 1 Byte payload.
  - Frodo is 1.4x slower than NewHope for 100 KByte payload.
- Frodo outperforms NTRU (EES743EP1) and SIDH.

However, this might not be the ideal use case for Frodo...

Frodo use cases

"To this day it is not clear whether a future (quantum) attack might be able to exploit this additional structure, introduced in ideal lattices, in order to break the cryptosystem."

Frodo use cases

> *"To this day it is not clear whether a future (quantum) attack might be able to exploit this additional structure, introduced in ideal lattices, in order to break the cryptosystem."*

- Conservative, post-quantum use cases would benefit from Frodo.
- Satellite communications, critical infrastructure, banking, etc.
- Recommended for conservative use cases by PQCRYPTO [ABB+].
- SAFEcrypto / Thales investigate LWE for satellite communications[1].

[1] See `https://www.safecrypto.eu/more-information/casestudies/` and `https://youtu.be/ZME-ncE8nu0`

Frodo use cases

- Howe et al. [HMO+16] also recommend standard lattices.
- Satellite communications being their main focus.
- Research was undertaken at Thales during an internship.

Lattice-based Encryption Over Standard Lattices in Hardware

J. Howe[1], C. Moore[1], M. O'Neill[1], F. Regazzoni[2], T. Güneysu[3], and K. Beeden[4]

[1]Centre for Secure Information Technologies (CSIT), Queen's University Belfast, UK

[2]Advanced Learning and Research Institute, Università della Svizzera Italiana, Switzerland

[3]Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany

[4]Thales UK, Research and Technology, Reading, UK

ABSTRACT

Lattice-based cryptography has gained credence recently as a replacement for current public-key cryptosystems, due to its quantum-resilience, versatility, and relatively low key sizes. To date, encryption based on the learning with errors (LWE) problem has only been investigated from an ideal lattice standpoint, due to its computation and size efficiencies. However, a thorough investigation of standard lattices in practice has yet to be considered. Standard lattices may be preferred to ideal lattices due to their stronger security

protect almost all Internet communications become *completely insecure* by virtue of quantum reductions. In January 2014, it was revealed that the NSA is funding a $79.7 million research program to build "a cryptologically useful quantum computer" [30]. Cryptography resilient to quantum reductions (or post-quantum cryptography) is generally split into four types:

Code-based cryptography [23], such as the McEliece cryptosystem [21], is based on the hard problem of decoding a random linear code. That is, the secret code-generating ma-

Frodo use cases

- Howe et al. [HMO+16] also recommend standard lattices.
- Satellite communications being their main focus.
- Research was undertaken at Thales during an internship.

Lattice-based Encryption Over Standard Lattices in Hardware

J. Howe[1], C. Moore[1], M. O'Neill[1], F. Regazzoni[2], T. Güneysu[3], and K. Beeden[4]

[1]Centre for Secure Information Technologies (CSIT), Queen's University Belfast, UK

[2]Advanced Learning and Research Institute, Università della Svizzera Italiana, Switzerland

[3]Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany

[4]Thales UK, Research and Technology, Reading, UK

ABSTRACT

Lattice-based cryptography has gained credence recently as a replacement for current public-key cryptosystems, due to its quantum-resilience, versatility, and relatively low key sizes. To date, encryption based on the learning with errors (LWE) problem has only been investigated from an ideal lattice standpoint, due to its computation and size efficiencies. However, a thorough investigation of standard lattices in practice has yet to be considered. Standard lattices may be preferred to ideal lattices due to their stronger security

protect almost all Internet communications become *completely insecure* by virtue of quantum reductions. In January 2014, it was revealed that the NSA is funding a $79.7 million research program to build "a cryptologically useful quantum computer" [30]. Cryptography resilient to quantum reductions (or post-quantum cryptography) is generally split into four types:

Code-based cryptography [23], such as the McEliece cryptosystem [21], is based on the hard problem of decoding a random linear code. That is, the secret code-generating ma-

More on this later...

Summary

- Post-quantum cryptography is important, quantum threat is serious.
- NIST [NIS16] started standardisation of post-quantum cryptography.
  - Future rounds will likely involve evaluations on constrained devices, such as smart cards, as well as comparisons of the schemes in hardware.
- Frodo has its place in future cryptographic standards.
  - Frodo is ideal for long-term security and constrained (hardware) platforms.
- Unstructured lattices have far less implementations.

Summary

- Post-quantum cryptography is important, quantum threat is serious.
- NIST [NIS16] started standardisation of post-quantum cryptography.
  - Future rounds will likely involve evaluations on constrained devices, such as smart cards, as well as comparisons of the schemes in hardware.
- Frodo has its place in future cryptographic standards.
  - Frodo is ideal for long-term security and constrained (hardware) platforms.
- Unstructured lattices have far less implementations.
  - So, can we do better with unstructured lattices?

Summary

- Post-quantum cryptography is important, quantum threat is serious.
- NIST [NIS16] started standardisation of post-quantum cryptography.
  - Future rounds will likely involve evaluations on constrained devices, such as smart cards, as well as comparisons of the schemes in hardware.
- Frodo has its place in future cryptographic standards.
  - Frodo is ideal for long-term security and constrained (hardware) platforms.
- Unstructured lattices have far less implementations.
  - So, can we do better with unstructured lattices?
  - Are there related implementations we can use as a basis?

Section 2

FrodoKEM: Learning With Errors Key Encapsulation

Section 2

FrodoKEM: Learning With Errors Key Encapsulation

Frodo: key encapsulation from standard lattices

- Simple design:
  - Free modular arithmetic ($q = 2^{16}$).
  - Simple Gaussian sampling.
  - Parallelisable matrix-vector operations.
  - Key encapsulation without reconciliation.
  - Simple code, no complex use of NTT.
- CCA-secure with negligible error rate.
- Flexible, fine-grained choice of parameters.
- Dynamically generated $\mathbf{A}$ to defend against all-for-the-price-of-one attacks (AES and cSHAKE variants).

Frodo: key encapsulation from standard lattices

Table: Implemented FrodoKEM parameter sets.

Parameters	**FrodoKEM-640**	**FrodoKEM-976**
D	15	16
q	32768	65536
n	640	976
$\bar{m} = \bar{n}$	8	8
B	2	3
$\text{len}_\mu = l$	128	192
.	.	.
.	.	.
.	.	.
$\text{len}_{\mathbf{ss}}$	128	192
len_χ	16	16
Error (χ) std. dev.	2.75	2.3
H	cSHAKE128(·, 128, 0)	cSHAKE256(·, 128, 0)
G	cSHAKE128(·, 384, 3)	cSHAKE256(·, 576, 3)
F	cSHAKE128(·, 128, 7)	cSHAKE256(·, 192, 7)
Ciphertext size	9,736 Bytes	15,768 Bytes
Classical security	143-bit	103-bit
Quantum security	209-bit	150-bit

Frodo: key encapsulation from standard lattices

Table: Implemented FrodoKEM parameter sets.

Parameters	FrodoKEM-640	FrodoKEM-976
D	15	16
q	32768	65536
n	640	976
$\bar{m} = \bar{n}$	8	8
B	2	3
$\text{len}_\mu = l$	128	192
⋮	⋮	⋮
$\text{len}_{\mathbf{ss}}$	128	192
len_χ	16	16
Error (χ) std. dev.	2.75	2.3
H	cSHAKE128(·, 128, 0)	cSHAKE256(·, 128, 0)
G	cSHAKE128(·, 384, 3)	cSHAKE256(·, 576, 3)
F	cSHAKE128(·, 128, 7)	cSHAKE256(·, 192, 7)
Ciphertext size	9,736 Bytes	15,768 Bytes
Classical security	143-bit	103-bit
Quantum security	209-bit	150-bit

Parameters and security estimates are “paranoid”, satisfy NIST Level 1 and 3.

Frodo: key encapsulation from standard lattices

Algorithm 1 The FrodoKEM key pair generation

procedure KEYGEN(1^{ℓ})
  Choose uniformly random seeds $\mathbf{s}||\text{seed}_{\mathbf{E}}||\mathbf{z} \leftarrow_{\$} U(\{0,1\}^{\text{len}_{\mathbf{s}}+\text{len}_{\mathbf{E}}+\text{len}_{\mathbf{z}}})$
  Generate pseudo-random seed $\text{seed}_{\mathbf{A}} \leftarrow H(\mathbf{z})$
  Generate the matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ via $\mathbf{A} \leftarrow \text{Frodo.Gen}(\text{seed}_{\mathbf{A}})$
  $\mathbf{S}, \mathbf{E} \leftarrow \text{Frodo.SampleMatrix}(\text{seed}_{\mathbf{E}}, n, \bar{n}, T_{\chi}, \cdot)$
  Compute $\mathbf{B} \leftarrow \mathbf{AS} + \mathbf{E}$
  return public key $pk \leftarrow \text{seed}_{\mathbf{A}}||\mathbf{B}$ and secret key $sk' \leftarrow (\mathbf{s}||\text{seed}_{\mathbf{A}}||\mathbf{B}, \mathbf{S})$
end procedure

Frodo: key encapsulation from standard lattices

Algorithm 2 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

Algorithm 3 The FrodoKEM decapsulation (shortened)

procedure DECAPS($sk = (\mathbf{s}||\text{seed}_{\mathbf{A}}||\mathbf{B}, \mathbf{S}), \mathbf{c}_1||\mathbf{c}_2||\mathbf{d}$)
  Compute $\mathbf{M} \leftarrow \mathbf{C} - \mathbf{B}'\mathbf{S}$
  Compute $\mu' \leftarrow$ Frodo.Decode$(\mathbf{M})$
  Parse $pk \leftarrow \text{seed}_{\mathbf{A}}||\mathbf{b}$
  Generate pseudo-random values $\text{seed}'_{\mathbf{E}}||\mathbf{k}'||\mathbf{d}' \leftarrow G(pk||\mu')$
  Sample error matrix $\mathbf{S}', \mathbf{E}' \leftarrow$ 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 $\mathbf{A} \leftarrow$ Frodo.Gen$(\text{seed}_{\mathbf{A}})$
  Compute $\mathbf{B}'' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$
  Sample error matrix $\mathbf{E}'' \leftarrow$ Frodo.SampleMatrix$(\text{seed}'_{\mathbf{E}}, \bar{m}, \bar{n}, T_\chi, \cdot)$
  Compute $\mathbf{C}' \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}'' +$ Frodo.Encode(μ')
  if $\mathbf{B}'||\mathbf{C} = \mathbf{B}''||\mathbf{C}'$ and $\mathbf{d} = \mathbf{d}'$ **return** $\mathbf{ss} \leftarrow F(\mathbf{c}_1||\mathbf{c}_2||\mathbf{k}'||\mathbf{d})$
  else return $\mathbf{ss} \leftarrow F(\mathbf{c}_1||\mathbf{c}_2||\mathbf{s}||\mathbf{d})$
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.

As well as a number of subsidiary operations:

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

Frodo: key encapsulation from standard lattices

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 [ABD+], rather than the key exchange scheme FrodoCCS [BCD+16].
- Due to similarities in KeyGen, Encaps, and Decaps, discussions might be generic.

FrodoKEM on constrained devices

- Our implementations focus on commonly used devices.
- Our software design is implemented on the popular ARM Cortex-M4.
- Our hardware design is demonstrated on a Xilinx Artix-7 FPGA.
  - Using development environment Xilinx Vivado v2017.4.

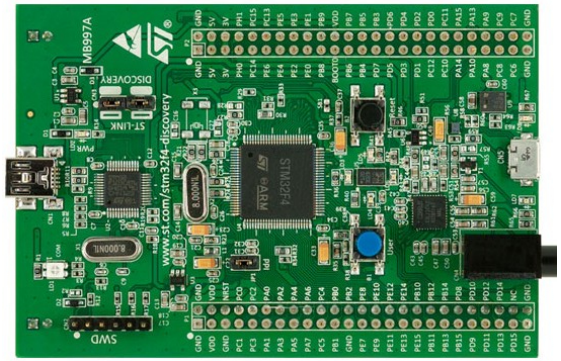


Figure: STM32F407 Discovery Board

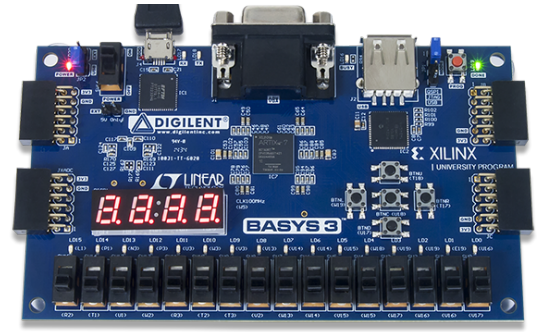
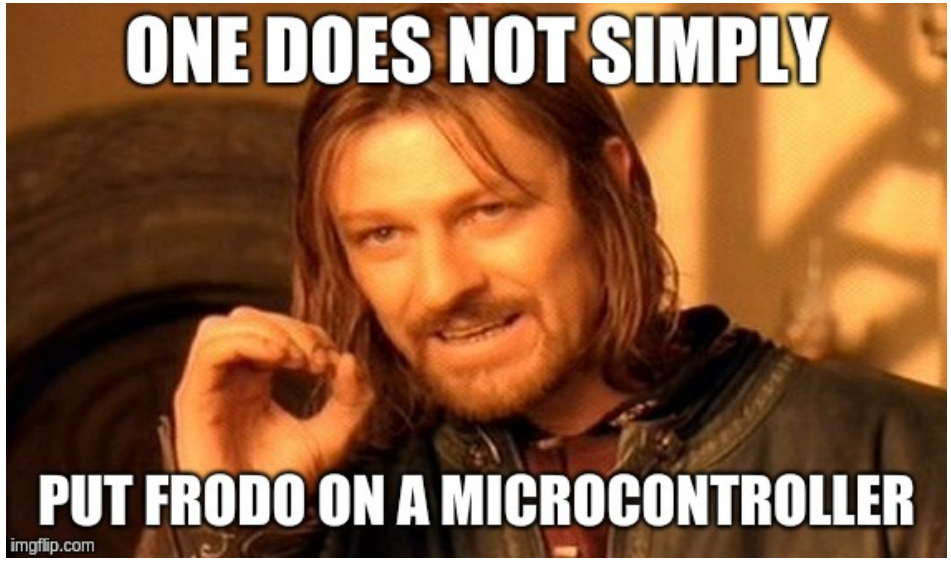


Figure: Basys 3 Artix-7 FPGA Board

Section 3

FrodoKEM on ARM Cortex M4



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.
- (Update) Will be added to the PQM4 [pqm] library soon!

FrodoKEM on ARM

Our first problem...

Table: Optimised reference implementation of FrodoKEM (AES from OpenSSL).

Parameter Set / Type	Peak Stack Memory Usage			Static Library Size
	KeyGen	Encaps	Decaps	
FrodoKEM-640-AES [ABD+]	72,192	103,072	123,968	81,836
FrodoKEM-976-AES [ABD+]	111,424	159,136	189,176	79,700

- Our target platform: STM32F4-Discovery board:
  - 192 KBytes RAM
  - Split into a 128 KByte module and a 64 KByte module.
  - Stack needs to fit into 128 KByte.

FrodoKEM on ARM

Our first problem...

Table: Optimised reference implementation of FrodoKEM (AES from OpenSSL).

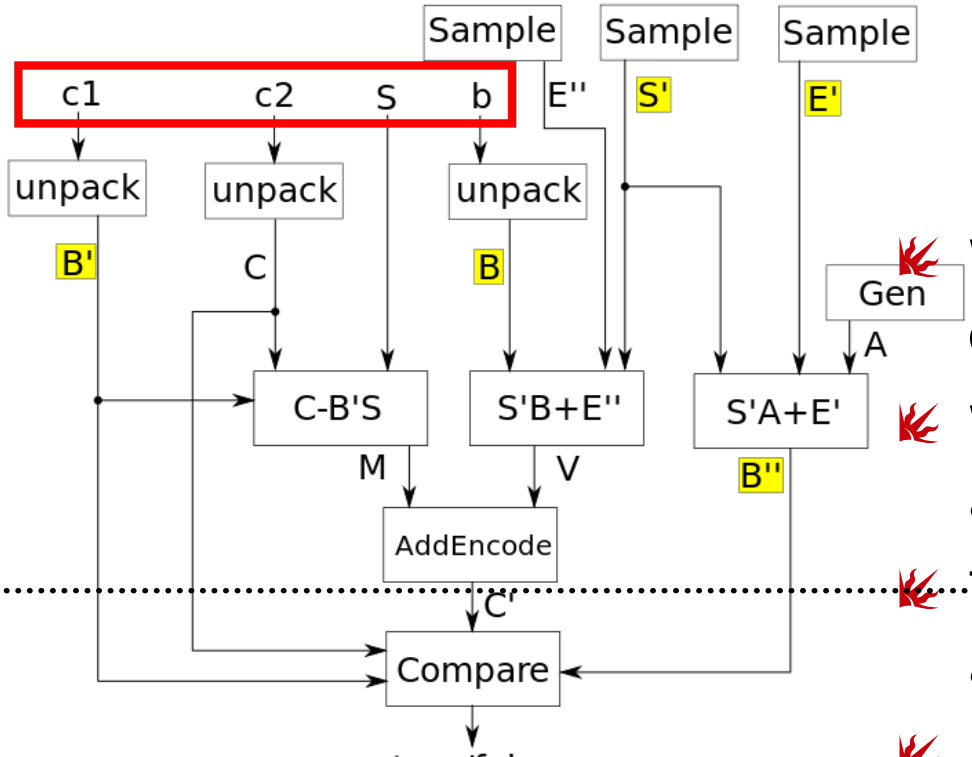
Parameter Set / Type	Peak Stack Memory Usage			Static Library Size
	KeyGen	Encaps	Decaps	
FrodoKEM-640-AES [ABD+]	72,192	103,072	123,968	81,836
FrodoKEM-976-AES [ABD+]	111,424	159,136	189,176	79,700

- Our target platform: STM32F4-Discovery board:
  - 192 KBytes RAM
  - Split into a 128 KByte module and a 64 KByte module.
  - Stack needs to fit into 128 KByte.

Problem: reference implementation needs too much memory!

FrodoKEM on ARM

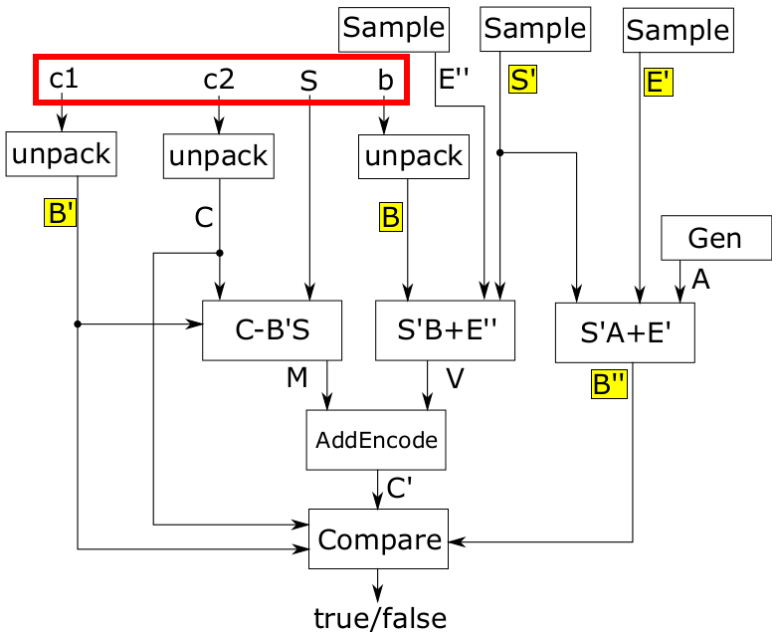
Memory analysis of decapsulation.



- 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

FrodoKEM on ARM

Memory analysis of decapsulation.



- 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.

FrodoKEM on ARM

Dynamic memory consumption of reference implementation.

- $\mathbf{E}'$ (Ep) can be generated on-the-fly.
- $\mathbf{S}'$ (Sp) is used multiple times.
  - Keep in memory first, later replace by $\mathbf{B}'$ (Bp).
- We need at least two large matrices.
  - Final operation is the comparison of $\mathbf{B}'$ (Bp) and $\mathbf{B}''$ (BBp).

```
int crypto_kem_dec(unsigned char *ss, const unsigned char *ct, const unsigned char *sk)
{ // Frodo-KEM's key decapsulation
    uint16_t B[PARAMS_N*PARAMS_NBAR] = {0}, Bp[PARAMS_N*PARAMS_NBAR] = {0}, W[PARAMS_NBAR*PARAMS_NBAR] = {0};
    uint16_t C[PARAMS_NBAR*PARAMS_NBAR] = {0}, CC[PARAMS_NBAR*PARAMS_NBAR] = {0};
    uint16_t *S = (uint16_t*)(sk+CRYPTO_BYTES+CRYPTO_PUBLICKEYBYTES);
    ALIGN_HEADER(32) uint16_t BBp[PARAMS_N*PARAMS_NBAR] ALIGN_FOOTER(32) = {0};
    ALIGN_HEADER(32) uint16_t Sp[(2*PARAMS_N+PARAMS_NBAR)*PARAMS_NBAR] ALIGN_FOOTER(32) = {0};
    uint16_t *Ep = (uint16_t *)&Sp[PARAMS_N*PARAMS_NBAR];
    uint16_t *Epp = (uint16_t *)&Sp[2*PARAMS_N*PARAMS_NBAR];
    uint8_t temp[CRYPTO_CIPHERTEXTBYTES+CRYPTO_BYTES], G[3*CRYPTO_BYTES], *seed_A = temp;
```

FrodoKEM on ARM

Dynamic memory consumption of reference implementation.

- $\mathbf{E}'$ (Ep) can be generated on-the-fly.
- $\mathbf{S}'$ (Sp) is used multiple times.
  - Keep in memory first, later replace by $\mathbf{B}'$ (Bp).
- We need at least two large matrices.
  - Final operation is the comparison of $\mathbf{B}'$ (Bp) and $\mathbf{B}''$ (BBp).

```
int crypto_kem_dec(unsigned char *ss, const unsigned char *ct, const unsigned char *sk)
{ // Frodo-KEM's key decapsulation
    uint16_t B[PARAMS_N*PARAMS_NBAR] = {0}, Bp[PARAMS_N*PARAMS_NBAR] = {0}, W[PARAMS_NBAR*PARAMS_NBAR] = {0};
    uint16_t C[PARAMS_NBAR*PARAMS_NBAR] = {0}, CC[PARAMS_NBAR*PARAMS_NBAR] = {0};
    uint16_t *S = (uint16_t*)(sk+CRYPTO_BYTES+CRYPTO_PUBLICKEYBYTES);
    ALIGN_HEADER(32) uint16_t BBp[PARAMS_N*PARAMS_NBAR] ALIGN_FOOTER(32) = {0};
    ALIGN_HEADER(32) uint16_t Sp[(2*PARAMS_N+PARAMS_NBAR)*PARAMS_NBAR] ALIGN_FOOTER(32) = {0};
    uint16_t *Ep = (uint16_t *)&Sp[PARAMS_N*PARAMS_NBAR];
    uint16_t *Epp = (uint16_t *)&Sp[2*PARAMS_N*PARAMS_NBAR];
    uint8_t temp[CRYPTO_CIPHERTEXTBYTES+CRYPTO_BYTES], G[3*CRYPTO_BYTES], *seed_A = temp;
```

FrodoKEM on ARM

Dynamic memory consumption of the proposed implementation.

- $\mathbf{S}'$ uses `temp` for storage.
- $\mathbf{E}'$ and $\mathbf{E}''$ both share `Epp`.

```
int crypto_kem_dec(unsigned char *ss, const unsigned char *ct, const unsigned char *sk)
{ // Frodo-KEM's key decapsulation
       uint16_t Bp[PARAMS_N*PARAMS_NBAR] = {0};
    uint16_t Epp[PARAMS_NBAR*PARAMS_NBAR] = {0}, W[PARAMS_NBAR*PARAMS_NBAR] = {0};
    uint16_t C[PARAMS_NBAR*PARAMS_NBAR] = {0}, CC[PARAMS_NBAR*PARAMS_NBAR] = {0};
    uint16_t *S = (uint16_t*)(sk+CRYPTO_BYTES+CRYPTO_PUBLICKEYBYTES);
    uint16_t BBp[PARAMS_N*PARAMS_NBAR] = {0};
    uint8_t temp[CRYPTO_CIPHERTEXTBYTES+CRYPTO_BYTES], G[3*CRYPTO_BYTES], *seed_A = sk+CRYPTO_BYTES;
```

FrodoKEM on ARM

Dynamic memory consumption of the proposed implementation.

- $\mathbf{S}'$ uses `temp` for storage.
- $\mathbf{E}'$ and $\mathbf{E}''$ both share `Epp`.
- We have to keep $\mathbf{B}'$ (Bp) and $\mathbf{B}''$ (BBp) for comparisons.

```
int crypto_kem_dec(unsigned char *ss, const unsigned char *ct, const unsigned char *sk)
{ // Frodo-KEM's key decapsulation
      uint16_t Bp[PARAMS_N*PARAMS_NBAR] = {0};
    uint16_t Epp[PARAMS_NBAR*PARAMS_NBAR] = {0}, W[PARAMS_NBAR*PARAMS_NBAR] = {0};
    uint16_t C[PARAMS_NBAR*PARAMS_NBAR] = {0}, CC[PARAMS_NBAR*PARAMS_NBAR] = {0};
    uint16_t *S = (uint16_t*)(sk+CRYPTO_BYTES+CRYPTO_PUBLICKEYBYTES);
    uint16_t BBp[PARAMS_N*PARAMS_NBAR] = {0};
    uint8_t temp[CRYPTO_CIPHERTEXTBYTES+CRYPTO_BYTES], G[3*CRYPTO_BYTES], *seed_A = sk+CRYPTO_BYTES;
```

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.

Operation	**FrodoKEM-AES**		**FrodoKEM-cSHAKE**		**FrodoKEM-cSHAKE [pqm]**	
	$n = 640$	$n = 976$	$n = 640$	$n = 976$	$n = 640$	% 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%

Section 4

FrodoKEM on Artix-7 FPGA

FrodoKEM on FPGA

Artix-7 FPGA – our target platform:

- High performance but low cost FPGA, used in hardware design of NewHope KEX by Oder and Güneysu [OG17].
- Includes DSPs with 25-by-18 two's complement multiplier/accumulator high-resolution (48-bit) signal processor.
- Plenty of memory (Block RAM) to store keys / matrices.
- Can potentially use space hardened FPGAs or IoT specialised FPGAs with dedicated crypto cores.

FrodoKEM on FPGA

Artix-7 FPGA – our target platform:

- High performance but low cost FPGA, used in hardware design of NewHope KEX by Oder and Güneysu [OG17].
- Includes DSPs with 25-by-18 two's complement multiplier/accumulator high-resolution (48-bit) signal processor.
- Plenty of memory (Block RAM) to store keys / matrices.
- Can potentially use space hardened FPGAs or IoT specialised FPGAs with dedicated crypto cores.
- Essentially zero restrictions on design, unlike Frodo on microcontrollers.

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

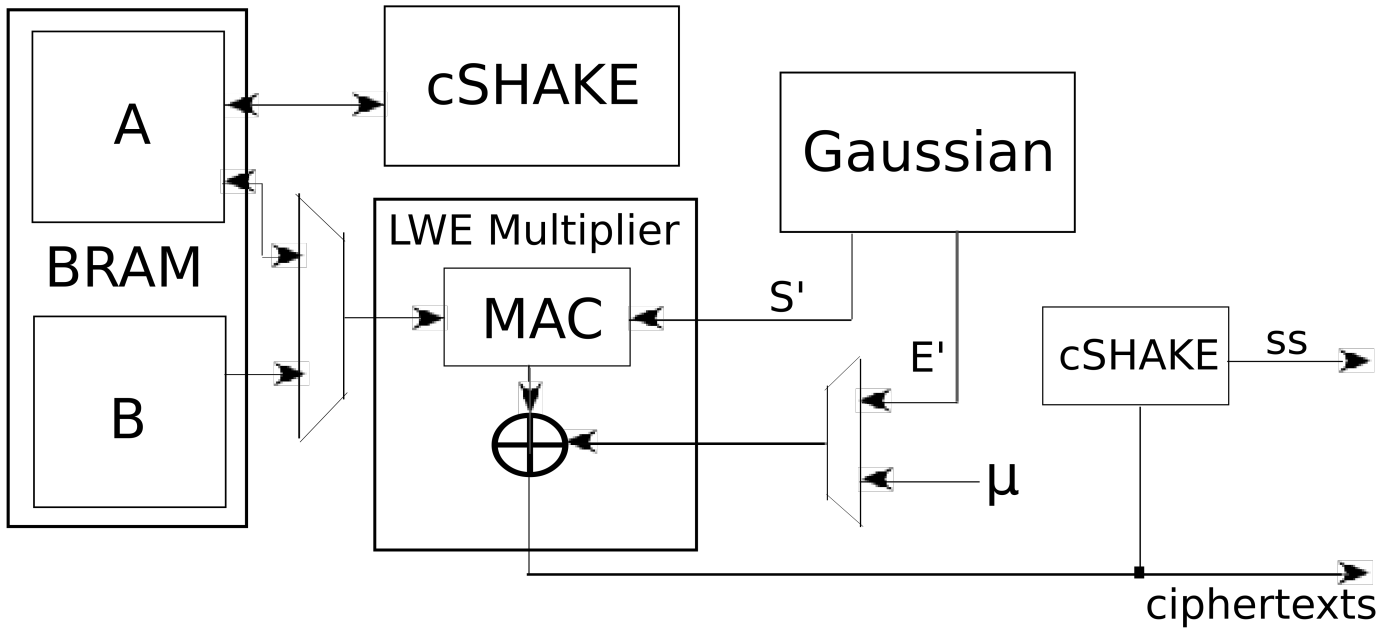


Figure: An overview of our FPGA design of FrodoKEM Encapsulation.

FrodoKEM on FPGA

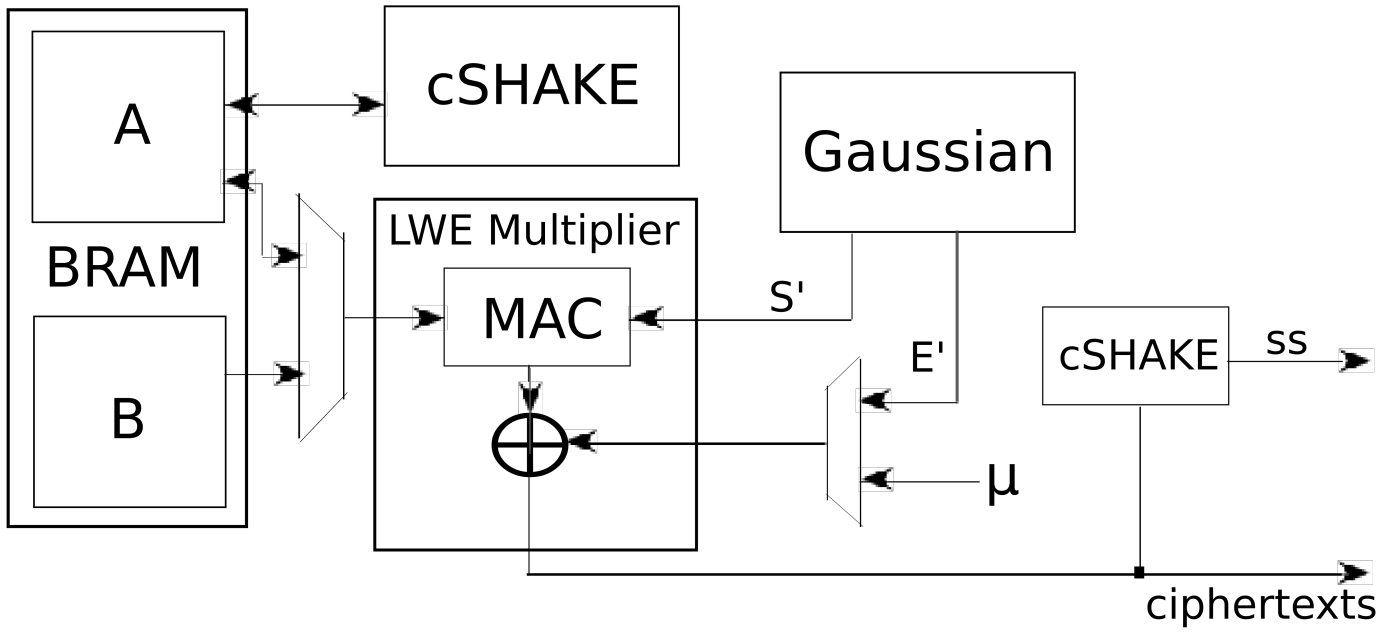


Figure: An overview of our FPGA design of FrodoKEM Encapsulation.

Similarities in KeyGen, Encaps, and Decaps mean much of this is reused.

FrodoKEM on FPGA

- Variable are dependent on the FSM and row/column (address) count.
- Especially, key_data (either matrix A or B) and flagkeccak conditionals.

Algorithm 4 The LWE multiplication core.

```
 1: if start_acc then
 2:     sum <= Resize(key_data * sp_data, SumWidth);
 3: else
 4:     sum <= sum + key_data * sp_data;
 5: end if
 6: if add_spm then
 7:     spm_data <= ep_data + m_data;
 8: end if;
 9: if mac_done then
10:     c_result <= Resize(unsigned(sum + spm_data) mod 2**CWidth, CWidth);
11: end if;
```

FrodoKEM on FPGA

Memory optimisations of the matrix A, a toy example.

$A_{(i,j)}$	col 1	col 2	col 3	col 4	col 5	$\cdots$	col n
row 1	53	-48	11	-63	-87	$\cdots$	-33
row 2	92	-14	-41	5	-6	$\cdots$	79
row 3	-85	43	52	42	69	$\cdots$	0
row 4	-15	-33	65	-33	89	$\cdots$	-86
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
row n	92	63	20	-68	100	$\cdots$	-5

FrodoKEM on FPGA

Memory optimisations of the matrix A, a toy example.

$A_{(i,j)}$	col 1	col 2	col 3	col 4	col 5	$\cdots$	col n
row 1	53	-48	11	-63	-87	$\cdots$	-33
row 2	92	-14	-41	5	-6	$\cdots$	79
row 3	-85	43	52	42	69	$\cdots$	0
row 4	-15	-33	65	-33	89	$\cdots$	-86
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
row n	92	63	20	-68	100	$\cdots$	-5

FrodoKEM on FPGA

Memory optimisations of the matrix A, a toy example.

$A_{(i,j)}$	col 1	col 2	col 3	col 4	col 5	$\cdots$	col n
row 1	53	-48	11	-63	-87	$\cdots$	-33
row 2	92	-14	-41	5	-6	$\cdots$	79
row 3	-85	43	52	42	69	$\cdots$	0
row 4	-15	-33	65	-33	89	$\cdots$	-86
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
row n	92	63	20	-68	100	$\cdots$	-5

FrodoKEM on FPGA

Memory optimisations of the matrix A, a toy example.

$A_{(i,j)}$	col 1	col 2	col 3	col 4	col 5	$\cdots$	col n
row 1	53	-48	11	-63	-87	$\cdots$	-33
row 2	92	-14	-41	5	-6	$\cdots$	79
row 3	-85	43	52	42	69	$\cdots$	0
row 4	-15	-33	65	-33	89	$\cdots$	-86
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
row n	92	63	20	-68	100	$\cdots$	-5

FrodoKEM on FPGA

Memory optimisations of the matrix A, a toy example.

$A_{(i,j)}$	col 1	col 2	col 3	col 4	col 5	$\cdots$	col n
row 1	53	-48	11	-63	-87	$\cdots$	-33
row 2	92	-14	-41	5	-6	$\cdots$	79
row 3	-85	43	52	42	69	$\cdots$	0
row 4	-15	-33	65	-33	89	$\cdots$	-86
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
row n	92	63	20	-68	100	$\cdots$	-5

FrodoKEM on FPGA

Memory optimisations of the matrix A, a toy example.

$A_{(i,j)}$	col 1	col 2	col 3	col 4	col 5	$\cdots$	col n
row 1	53	-48	11	-63	-87	$\cdots$	-33
row 2	92	-14	-41	5	-6	$\cdots$	79
row 3	-85	43	52	42	69	$\cdots$	0
row 4	-15	-33	65	-33	89	$\cdots$	-86
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
row n	92	63	20	-68	100	$\cdots$	-5

FrodoKEM on FPGA

Memory optimisations of the matrix A, a toy example.

$A_{(i,j)}$	col 1	col 2	col 3	col 4	col 5	$\cdots$	col n
row 1	53	-48	11	-63	-87	$\cdots$	-33
row 2	92	-14	-41	5	-6	$\cdots$	79
row 3	-85	43	52	42	69	$\cdots$	0
row 4	-15	-33	65	-33	89	$\cdots$	-86
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
row n	92	63	20	-68	100	$\cdots$	-5

FrodoKEM on FPGA

Memory optimisations of the matrix A, a toy example.

$A_{(i,j)}$	col 1	col 2	col 3	col 4	col 5	$\cdots$	col n
row 1	53	-48	11	-63	-87	$\cdots$	-33
row 2	92	-14	-41	5	-6	$\cdots$	79
row 3	-85	43	52	42	69	$\cdots$	0
row 4	-15	-33	65	-33	89	$\cdots$	-86
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
row n	92	63	20	-68	100	$\cdots$	-5

Ensures a fixed cost for memory, otherwise BRAMs overloaded.

FrodoKEM on FPGA

Fast “rounded continuous Gaussian” sampling.

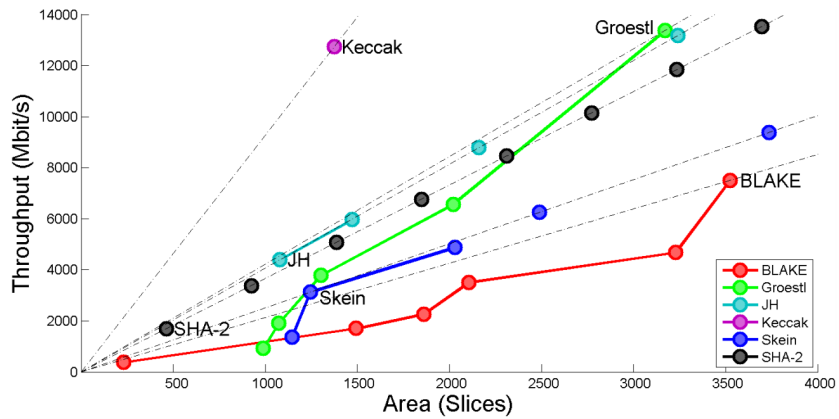
- FrodoKEM specifications define a constant-time error sampling method.
- We maintain this technique but increase throughput.
  - Instead of storing values in a table, a MUX is used, ensuring fast outputs.
- Ensures one output per clock cycle, hence constant-runtime.
- Area consumption essentially the same due to similar sigma values.
  - Essentially the same architecture, just replace the values.
- PRNG input to the sampler fulfilled by an AES core.

Parameters	σ	Probability of (in multiples of 2^{-15})											
		0	±1	±2	±3	±4	±5	±6	±7	±8	±9	±10	±11
$\chi_{\text{FrodoKEM-640}}$	2.75	9456	8857	7280	5249	3321	1844	898	384	144	47	13	3
$\chi_{\text{FrodoKEM-976}}$	2.3	11278	10277	7774	4882	2545	1101	396	118	29	6	1	-

FrodoKEM on FPGA

Design choices for cSHAKE.

- VHDL code taken from the KECCAK [BDP+12] team's SHA-3 implementation.
- Here, there are a number of design choices:
  - High-speed core; which over overexerts the FPGA's I/O pins.
  - *Mid-range core; used in other lattice-based hardware designs.
  - Low-area core; small but too slow for our requirements.



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

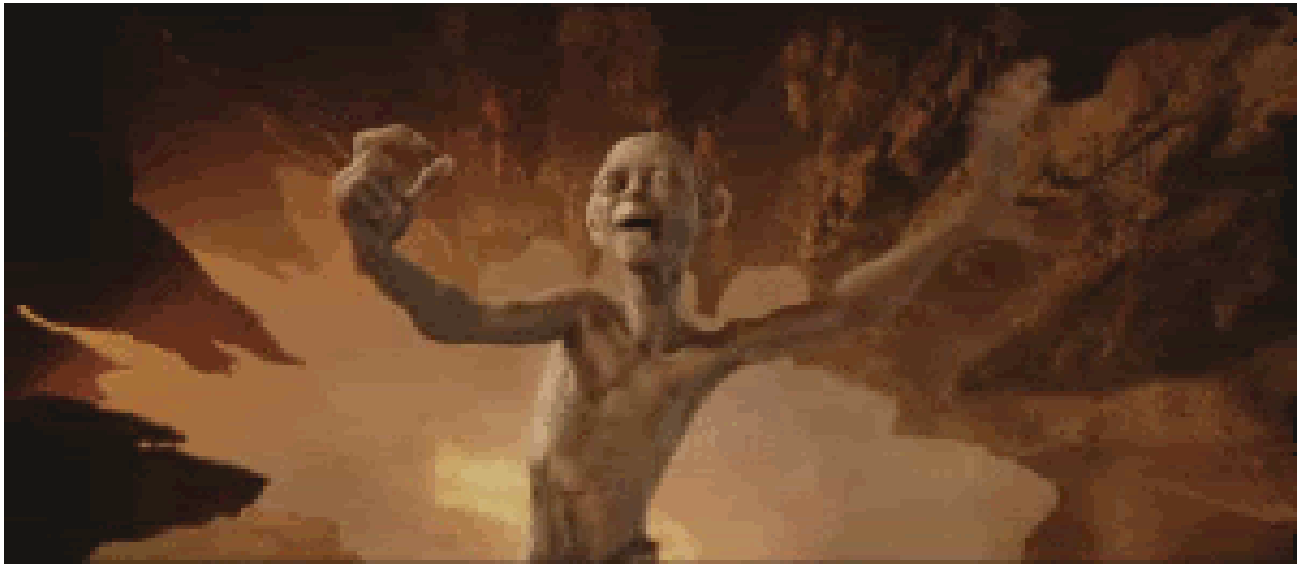
- We show that hardware significantly minimises the performance distance between standard and ideal lattice-based KEM, able to utilise less than 2000 slices and remain practical.
- Memory optimisations for microcontrollers show 66% savings vs reference design and 40% vs optimised PQM4 design.
- Memory optimisations in all software and hardware designs were critical, otherwise the platform would have been overexerted.

Conclusions

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

Conclusions

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



Future work

- (Software) We are looking at masking FrodoKEM, seeing cost analysis.
- (Hardware) How would FrodoKEM perform / scale with more multipliers.
  - Increase in multipliers would require faster cSHAKE / AES sampling.
- Comparison of FrodoKEM vs. other NIST post-quantum candidates,
  - Particularly in hardware and across post-quantum types.
  - We need more hardware designs! I'm interested in more collaborations.
  - See `PQCzoo.com` for collections of optimised designs and SCA results.
- How can we protect FrodoKEM from SCA? Bos et al. [BFM+18] show interesting results, higher black-box security == easier SCA.

References I

Daniel Augot, Lejla Batina, Daniel J Bernstein, Joppe Bos, Johannes Buchmann, Wouter Castryck, Orr Dunkelman, Tim Güneysu, Shay Gueron, Andreas Hülsing, et al.
Initial recommendations of long-term secure post-quantum systems (2015).
https://pqcrypto.eu.org/docs/initial-recommendations.pdf.

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.

References II

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.

Joppe W Bos, Craig Costello, Michael Naehrig, and Douglas Stebila.
Post-quantum key exchange for the tls protocol from the ring learning with errors problem.
In Security and Privacy (SP), 2015 IEEE Symposium on, pages 553–570. IEEE, 2015.

Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche, and Ronny Van Keer.
Keccak implementation overview.
`https://keccak.team/hardware.html`, 2012.

References III

Joppe W. Bos, Simon Friedberger, Marco Martinoli, Elisabeth Oswald, and Martijn Stam.
Assessing the Feasibility of Single Trace Power Analysis of Frodo.
In Selected Areas in Cryptography - SAC 2018, 2018.
https://eprint.iacr.org/2018/687.

Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé.
Classical hardness of learning with errors.
In Proceedings of the forty-fifth annual ACM symposium on Theory of computing, pages 575–584. ACM, 2013.

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.

References IV

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.

James Howe, Thomas Pöppelmann, Máire O'Neill, Elizabeth O'Sullivan, and Tim Güneysu.
Practical lattice-based digital signature schemes.
ACM Trans. Embedded Comput. Syst., 14(3):41:1–41:24, 2015.

Stefan Heyse, Ingo Von Maurich, and Tim Güneysu.
Smaller keys for code-based cryptography: Qc-mdpc mceliece implementations on embedded devices.
In International Workshop on Cryptographic Hardware and Embedded Systems, pages 273–292. Springer, 2013.

Brian Koziel, Reza Azarderakhsh, Mehran Mozaffari Kermani, and David Jao.
Post-quantum cryptography on fpga based on isogenies on elliptic curves.
IEEE Transactions on Circuits and Systems I: Regular Papers, 64(1):86–99, 2017.

References V

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.

Angshuman Karmakar, Jose Maria Bermudo Mera, Sujoy Sinha Roy, and Ingrid Verbauwhede.
Saber on ARM CCA-secure module lattice-based key encapsulation on ARM.
IACR Transactions on Cryptographic Hardware and Embedded Systems, 2018(3):243–266, Aug. 2018.

Richard Lindner and Chris Peikert.
Better key sizes (and attacks) for LWE-based encryption.
In Aggelos Kiayias, editor, Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings, volume 6558 of Lecture Notes in Computer Science, pages 319–339. Springer, 2011.

References VI

NIST.
Post-quantum crypto project.
http://csrc.nist.gov/groups/ST/post-quantum-crypto/, 2016.
Accessed: 17.05.2018.

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

Chris Peikert.
Lattice cryptography for the internet.
In International Workshop on Post-Quantum Cryptography, pages 197–219. Springer, 2014.

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

References VII

Oded Regev.
On lattices, learning with errors, random linear codes, and cryptography.
In Harold N. Gabow and Ronald Fagin, editors, Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005, pages 84–93. ACM, 2005.

Pascal Sasdrich and Tim Güneysu.
Efficient elliptic-curve cryptography using curve25519 on reconfigurable devices.
In International Symposium on Applied Reconfigurable Computing, pages 25–36. Springer, 2014.

Hwajeong Seo, Zhe Liu, Patrick Longa, and Zhi Hu.
SIDH on ARM: faster modular multiplications for faster post-quantum supersingular isogeny key exchange.
IACR Transactions on Cryptographic Hardware and Embedded Systems, 2018(3):1–20, Aug. 2018.

References VIII

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.

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.