PQCRYPTO
ICT-645622

Horizon 2020

PQCRYPTO
Post-Quantum Cryptography for Long-Term Security

Project number: Horizon 2020 ICT-645622

Small Devices: D1.3 Intermediate Report on Optimized Software

Due date of deliverable: 30. September 2016
Actual submission date: April 15, 2018

Start date of project: 1. March 2015 Duration: 3 years

Coordinator:
Technische Universiteit Eindhoven
Email: `coordinator@pqcrypto.eu.org`
`www.pqcrypto.eu.org`

Revision 1

Project co-funded by the European Commission within Horizon 2020		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission services)	
RE	Restricted to a group specified by the consortium (including the Commission services)	
CO	Confidential, only for members of the consortium (including the Commission services)	

Small Devices: D1.3 Intermediate Report on Optimized Software

Tim Güneysu, Peter Schwabe, Ko Stoffelen, Joost Rijneveld

April 15, 2018
Revision 1

The work described in this report has in part been supported by the Commission of the European Communities through the Horizon 2020 program under project number 645622 PQCRYPTO. The information in this document is provided as is, and no warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

Abstract

This document provides the PQCRYPTO project's intermediate report on optimized software. It provides the preliminary software implementation results of selected post-quantum schemes and corresponding parameters for embedded systems.

Keywords: Post-quantum cryptography, small devices, hardware implementations, public-key encryption, public-key signatures, secret-key encryption, secret-key authentication

Contents

1 Introduction

Modern asymmetric cryptosystems designed to face multiple threats and to maintain long-term security, require conservative parameter choices that typically pose a major implementation challenge for small embedded microcontrollers. In this report we survey modern post-quantum schemes in regard to their implementation on such small embedded microcontrollers. The primary target platform for this report is the widely deployed ARM Cortex-M4, a rather high-end 32-bit microcontroller. It reviews the most popular schemes in post-quantum cryptography that support confidentiality (by encryption) and authentication (by digital signatures). Note that the report is designed to focus on the implementation challenges and does not include any detailed discussion of the mathematical background required to understand the design rationales of the respective post-quantum constructions.

2 Target Platforms

The primary target platform considered in this report is the ARM Cortex-M4 family of microcontrollers. The primary testing platform for code targeting this family of microcontrollers is the *STM32F407 Discovery* development board, which features a Cortex-M4 processor with floating-point support running at a frequency of up to 168 MHz, 1 MB of flash storage, and 192 KB of RAM. The reasons for selecting this particular family of microcontrollers are primary target platform are the following:

- ARM Cortex-M 32-bit microcontrollers are increasingly becoming the de-facto standard for many applications; smaller microcontrollers of the same family like the Cortex M0 take over large parts of the market that has for a long time been dominated by 8-bit AVR microcontrollers. Optimizing for a Cortex-M processor thus ensures relevance of the results for many real-world applications.
- Selecting a rather high-end microcontroller from the Cortex-M family significantly extends the range of cryptographic primitives that we can fit into the available RAM and ROM and thus evaluate different trade-offs of primitives and parameter choices. The choice for a large microcontroller also reflects the fact that in most applications the deployment of post-quantum primitives is still going to take a few years (in many cases awaiting standardization by NIST and ETSI). Today's high-end microcontrollers are likely to reflect what low-end microcontrollers will look like by the time that applications migrate to post-quantum cryptography on a large scale.
- Selecting the STM32F407 Discovery board is motivated by the fact that it is widely available at low cost, which ensures easy reproducibility of benchmark results for other research groups.

In addition to the primary target platform we also report on optimization efforts of select post-quantum schemes for smaller microcontrollers, specifically the ARM Cortex-M0 32-bit microcontroller and the AVR ATmega family of 8-bit microcontrollers.

3 Optimization for embedded microcontrollers

Optimization of cryptographic software on large processors found, for example, in servers, desktop and laptop computers or smartphones typically aims at speed, i.e., minimizing the number of CPU cycles required for cryptographic operations. Optimization of cryptographic software on small embedded microcontrollers is a multidimensional task, typically aiming at maximizing speed, while minimizing ROM and RAM usage. Those optimization goals are often conflicting. For example, using tables of precomputed frequently used values speeds up the software (because the values do not need to be recomputed), but increases RAM or ROM usage (depending on where the precomputed tables are stored). This makes such optimization efforts highly non-trivial, but very rewarding, as a one-time effort can be deployed immediately in many different places.

Common good practice is to ensure that code does not leak secret information through timing. The reason is that timing attacks (i.e., attacks that exploit such timing leakage) are often feasible even remotely, so physical protection of devices does not thwart these attacks. Some deployment scenarios also need to take into account side-channel attacks with physical access to the device (such as power analysis or analysis of electromagnetic radiation). Implementations for such deployment scenarios require additional protection.

In the following we describe optimization techniques for our primary target platform, the ARM Cortex-M4 microcontroller. Most of these optimization techniques also apply to other embedded microcontrollers; some are specific to the ARMv7-M instruction set of the Cortex-M4.

Loop unrolling and inlining. When repeating a fragment of code (possibly with a slight variation for each repetition), one would typically encapsulate it in a loop or a function, depending on the specific repetition context. While this reduces the code size and makes the logic easier to understand and maintain, it increases the number of instructions that need to be performed. It also impacts register usage. In particular, keeping track of a loop iterator results in both administrative overhead and additional variables – either in registers, on the stack, or both.

To mitigate this, one can unroll loops and inline the content of functions: by literally specifying the repeated instructions, program flow code is avoided altogether. On embedded devices, however, an important consideration here is the additional storage requirement, as well as the overhead of having to load more code from the (often slow and uncached) memory.

Using compile-time macros or code generation, the program flow can still be specified using loops and function calls, but the resulting executable is free of such overhead. This provides flexibility that allows for careful code size and runtime trade-offs, depending on the available resources.

Memory alignment. When loading or storing instructions or data, or when performing branches, the Cortex-M4 processor prefers that the addresses are *word-aligned*; if they are not, penalty cycles may be introduced. A word is 4 bytes, so careful alignment ensures that all addresses are divisible by 4, which is not done automatically when programming in assembly, but can be enforced by placing the assembler directive `.align 2` at the start of all data blocks and branch targets, such as at the start of functions.

Starting code at a word boundary is not sufficient to ensure instruction alignment throughout the full computation. The reason is that some instructions can be encoded in either 2 bytes or 4 bytes. When a 4-byte instruction is spread over two separate words, a penalty cycle may be introduced when loading these instructions from flash memory. It can therefore be advantageous to use `add.w` instead of `add`, to force a 4-byte encoding for this instruction (see A6.7.4 of the Architecture Reference Manual [18]).

Pipelining loads. The ARM Cortex-M4 features a “load multiple” (`ldm`) instruction, which takes $N + 1$ cycles for loading N word-sized values from memory. A single load (`ldr`) takes two cycles, so grouping multiple loads together into an `ldm` improves both speed and code size.

Table lookups and caching. The performance difference between loading data from main memory and computing them is not as large on the Cortex-M4 as on a big Intel CPU. It can therefore quickly be favorable to precompute data and to use lookup tables. However, this should be avoided for secret data, as this could leak information about the secret in a context where side-channel attacks with physical access to the device are a concern.

When an assembly developer wants to store constant byte-sized values, the obvious way to proceed is to use the `.byte` directive. However, we have described that it can be faster to perform loads of word-sized values, especially when they can be pipelined. Depending on the desired trade-off between speed and ROM/RAM usage, sometimes using a full `.word` for every byte can be beneficial.

Data recomputation. Sometimes data is stored in memory, but an implementation can be made smaller by computing that data on the fly at the cost of only a small performance penalty. There are also intermediate alternatives where only some values are stored and the rest can somehow be recomputed from them.

4 Benchmarking on embedded microcontrollers

In order to gain insight in the cost of deploying post-quantum cryptography in many different contexts, reliable benchmarks (measurements of speed and memory usage) of the various cryptographic systems on a range of platforms are essential. Especially in the context of embedded microcontrollers, a few kilobytes less memory usage or a few milliseconds faster implementation can make the difference in earlier widespread adoption of post-quantum cryptography. We should therefore be able to benchmark at a high accuracy.

The highest level of accuracy is gained when measuring speed at the level of CPU clock cycles, and by measuring ROM and ROM usage in the exact number of bytes that are used. While measuring ROM and RAM usage is fairly straightforward, getting reliable speed measurements of an implementation can be notoriously difficult.

The main reason is that modern CPUs are so complex, that there is a large amount of external effects that can have significant impact on cycle counts. While this problem might be smaller on an embedded microcontroller compared to a big Intel CPU, it is still something that should be considered. A problem that is larger on embedded microcontrollers is the larger range of different memory blocks and peripherals that all have different timing characteristics. Other reasons for a large deviation in cycle counts could be that the cryptography algorithms are simply designed that way (for example, the use of rejection sampling in lattice-based cryptoschemes) or the use of randomness that has to be supplied by an external peripheral.

We now describe a few considerations that are required to get reliable benchmarks on our ARM Cortex-M4 target platform.

Choosing a cycle counter. The Cortex-M4 comes with two methods of measuring CPU cycles. First of all, the Data Watchpoint and Trace component is responsible for providing several debugging features. It also comes with a 32-bit incrementing cycle-accurate counter that can be read by an application using the special `DWT_CYCCNT` register. However, 2^{32} cycles is not enough to benchmark all post-quantum crypto schemes, so this register might overflow many times. A debug event is emitted on overflow, but capturing this requires additional debugging hardware.

The alternative is the SysTick system timer, which is a 24-bit decrementing counter that is decremented at a set frequency, which should be the same as the main clock frequency for accurate benchmarks. Its current value can also be read from a special register, `SYST_CVR`. On underflow, an interrupt is fired. One can keep track of how many times this interrupt was fired and combine that with the values before and after a cryptographic operation to get a complete picture of the number of cycles that were required.

CPU wait states and clock frequency. On the STM32F407, loading instructions and data from flash memory can be a serious bottleneck when the main CPU clock frequency is set to the maximum 168 MHz. The memory is not fast enough to keep up with the main CPU, which then has to idle until the instructions or data are retrieved. Moreover, how long the CPU has to idle depends on the specific chip. A reliable benchmark of post-quantum cryptographic algorithms should be meaningful across all Cortex-M4 chips, which means that this waiting behavior should be avoided to exclude the effect of the specific chip. On the STM32F407, this can be achieved by selecting a lower clock frequency and configuring a zero wait state latency in the special `FLASH_ACR` register.

Randomness. Specifications of post-quantum cryptographic systems typically require a source of fresh uniform randomness. The implementer then has to decide where to get this high-quality randomness from, which is far from trivial. The STM32F407 development board comes with a hardware random number generator that passes the FIPS PUB 140-2 tests with a success ratio of 99%. It delivers a fresh 32-bit value at at most every 40 periods of a special RNG clock that is derived from the PLL clock. However, in practice this introduces some deviation in speed benchmarks, which should therefore be repeated multiple times.

Measurement setup. Benchmarks should always happen multiple times, preferably as often as is feasible. One could average the results, but a few outliers (for example, due to some faulty hardware) can then have a large effect on the average. Using the median values is more robust.

ïż£

5 Software Implementations

This section gives an overview of implementations of post-quantum cryptographic schemes optimized for embedded microcontrollers that resulted from the PQCRYPTO project. We give an overview of implementations of public-key encryption (and key encapsulation) in Section 5.1 and of implementations of digital-signature schemes in Section 5.2.

5.1 Implementations of public-key encryption and key encapsulation

Table 5.1 lists implementations of post-quantum digital-signature schemes optimized for embedded microcontrollers. All implementations listed are results of the PQCRYPTO project that are described in scientific publications.

Scheme	**Platform**	**Cycles**	**RAM Bytes**	**ROM Bytes**
NewHope [2]	ARM Cortex-M4	**gen:** 964 440	**gen:** ?	22 828
		enc: 1 418 124	**enc:** ?	22 828
		dec: 178 874	**dec:** ?	22 828
NewHope [2]	ARM Cortex-M0	**gen:** 1 168 224	**gen:** ?	30 178
		enc: 1 738 922	**enc:** ?	30 178
		dec: 298 877	**dec:** ?	30 178
QcBits [6]	ARM Cortex-M4	**gen:** 140 372 822	**gen:** ?	62 KiB
		enc: 2 244 489	**enc:** ?	62 KiB
		dec: 14 679 937	**dec:** ?	62 KiB
RLWE encryption [5]	ARM Cortex-M0	**gen:** ?	**gen:** ?	?
		enc: 1 573 x10^3	**enc:** ?	1.6 KiB
		dec: 740 x10^3	**dec:** ?	1.1 KiB
RLWE encryption [5]	AVR ATxmega	**gen:** ?	**gen:** ?	?
		enc: 999 x10^3	**enc:** ?	3.5 KiB
		dec: 437 x10^3	**dec:** ?	2.1 KiB
QC-MDPC encryption [20]	ARM Cortex-M4	**gen:** ?	**gen:** ?	?
		enc: 7 018 493	**enc:** 2.7 KiB	5.7 KiB
		dec: 42 129 589	**dec:** 2.7 KiB	5.7 KiB
QC-MDPC hybrid encryption [25]	ARM Cortex-M4	**gen:** 63 185 108	**gen:** 3 136	8 784
		enc: 2 623 432	**enc:** 2 048	8 621
		dec: 18 416 012	**dec:** 2 048	3 064

Table 5.1: Microcontroller implementation results of post-quantum public-key encryption schemes. All implementations listed are results of the PQCRYPTO project.

Traditionally, the two main directions for post-quantum public-key encryption and key agreement are lattice-based schemes and code-based schemes. A potentially interesting additional candidate is supersingular-isogeny based key encapsulation [17, 7], but as this approach only fairly recently received increased attention and as it is rather inefficient in terms of speed, there are no optimized microcontroller implementations, yet.

5.1.1 Lattice-based public-key encryption and key encapsulation

Lattice-based cryptography has probably received most attention among the different areas of post-quantum cryptography over the last decade. Unsurprisingly, also implementations optimized for microcontrollers have been an active area of research. Most of the efforts of optimizing lattice-based encryption and key encapsulation focused on ideal lattices. For example, both [24] and [19] present optimized Ring-LWE encryption targeting the AVR 8-bit microcontrollers; [5] describe implementations on the ARM Cortex M0 and AVR ATXmega128; [8] presents results for the ARM Cortex-M4F; and [2] describe implementations of the NewHope lattice-based key agreement on ARM Cortex-M0 and ARM Cortex-M4, The general pattern of these papers is to optimize two main building blocks: arithmetic in the underlying polynomial ring and noise sampling. The typical approach to perform multiplications in the polynomial ring is to use the number-theoretic transform (NTT), which is not only very efficient in terms of speed, but can also be performed in place, i.e., without requiring any additional memory. For quite some time, noise sampling focused on different approaches for discrete-Gaussian sampling, investigating multiple different algorithms with different tradeoffs between speed, memory requirements and side-channel characteristics. In [1] established that a much simpler centered-binomial distribution is sufficient for lattice-based encryption and key agreement. Consequently, [2] use this simpler distribution in their implementation of NewHope.

5.1.2 Code-based public-key encryption and key encapsulation

The McEliece cryptosystems [21] with binary Goppa codes is widely considered one of the most conservative choices of post-quantum public-key encryption. Its implementation on small embedded microcontrollers is hampered by the large public key, but this does not mean that nobody tried to implement it. For example, [11] optimizes for the AVR ATXmega192 and concludes that *"the large public-key matrix K_{pub} does not fit into the 192 kByte internal Flash memory. Hence, at least 512 kByte external memory are required for storing the public key"*. More recent works focus on optimizing McEliece with quasi-cyclic medium-density parity-check (QC-MDPC) codes, that have considerably smaller public keys, but do not have the same long-term security track record as McElice with binary Goppa codes. For example, [14] targets AVR 8-bit microcontrollers; [20] optimizes McEliece with QC-MDPC codes on ARM Cortex-M4 microcontrollers; the same architecture targeted by the "QcBits" key-encapsulation mechanism described in [6] and in the recent work [9]. Both AVR 8-bit microcontrollers and 32-bit ARM microcontrollers (specifically, the Cortex-M4) are the target of optimization in [25].

5.2 Digital Signatures

Table 5.2 lists implementations of post-quantum digital-signature schemes optimized for embedded microcontrollers. All implementations listed are results of the PQCRYPTO project that are described in scientific publications.

Scheme	Platform	Cycles		RAM Bytes		ROM Bytes
SPHINCS-256 [16]	ARM Cortex-M3	**gen:**	28 205 671	**gen:**	?	47 948
		sign:	589 018 151	**sign:**	8 755	26 944
		verify:	16 414 251	**verify:**	?	26 976
XMSS^{MT} [16]	ARM Cortex-M3	**gen:**	8 857 708 189	**gen:**	?	?
		sign:	19 441 021	**sign:**	?	?
		verify:	4 961 447	**verify:**	?	?

Table 5.2: Microcontroller implementation results of post-quantum digital-signature schemes. All implementations listed are results of the PQCRYPTO project.

The two most promising approaches for post-quantum signatures, in particular on embedded devices, are hash-based signatures and lattice-based signatures. Multivariate-based signatures are also a potentially interesting candidate because of their small signature size. However, they suffer from large public keys that prohibits verification of multivariate signatures on many embedded platforms.

5.2.1 Hash-based Signatures

Hash-based signatures are without much doubt the most conservative choice for post-quantum cryptography, or possibly more generally for public-key cryptography. The reason is that they can be built from *only* a cryptographic hash function, a building block that is also required for all other signature schemes. This promise of high security come at a cost: the so-called 'stateful' XMSS^{MT} scheme has a non-standard API that requires updating the secret key, while the 'stateless' SPHINCS scheme produces large signatures and is considerably slower than its competitors. On embedded devices, the consequences of having to maintain a state (i.e. synchronization, complex backups, *etc.*) are not as severe. Indeed, [15] presents a smart card implementation of XMSS, and its authors demonstrate its practicality by efficiently generating signatures as well as key pairs on-card. Implementing SPHINCS on constrained devices is less straight-forward. In addition to the large runtime, its memory requirements present a potentially unsurmountable hurdle: on devices with limited storage space available, it is not possible to store the full signature. The implementation described in [16] shows that this is not a hard limit, streaming out the SPHINCS signature part by part as it is generated. Still, its poor performance may prove prohibitive in many typical use cases for small devices.

5.2.2 Lattice-based Signatures

Optimization of lattice-based signatures on embedded microcontrollers so far focused mainly on two schemes: the GLP signature scheme presented in [13] and the BLISS signature scheme presented in [10]. For example, [24] presents optimizations of BLISS for AVR ATXmega microcontrollers. Also [22] presents optimized implementations of BLISS, but on the ARM Cortex-M4F. In [3], the authors describe a conversion of the signature schemes from [13] and [10] to authentication protocols and implementations of those protocols on AVR ATmega and on a smart card equipped with an ARM7TDMI 32-bit processor. Optimization of lattice-based signatures—like with lattice-based public-key encryption and key encapsulation—focuses on fast arithmetic in polynomial rings (typically via the NTT) and efficient sampling of noise. One might think that lattice-based signatures and encryption would naturally share large parts of optimized code (like is the case with today's elliptic-curve cryptography), however, the situation is more complex: lattice-based signatures need larger parameters and are much more sensitive to the selection of the noise distribution. As a result, carefully optimized implementations of signatures and encryption schemes cannot share code without sacrificing performance, mainly for encryption. The BLISS signature schemes, like several earlier proposals for lattice-based signatures, critically relies on Gaussian noise, which is hard to sample efficiently without leaking information through timing. Consequently, implementations of BLISS have been successfully attacked via timing attacks [4, 23, 12].

6 Conclusions

A lot of post-quantum schemes have been implemented already, especially lattice-based and code-based schemes. So far, ideal lattice-based schemes appear to be the most efficient. However, the additional structure in the underlying lattice is still considered to be a potential threat to its security even though no attacks have been found yet that could exploit this structure. It would be interesting to also compare these implementations to implementations of hash-based schemes, like XMSS or SPHINCS.

References

[1] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange – a new hope. In *Proceedings of the 25th USENIX Security Symposium*. USENIX, (to appear). Document ID: 0462d84a3d34b12b75e8f5e4ca032869, http://cryptojedi.org/papers/#newhope.

[2] Erdem Alkim, Philipp Jakubeit, and Peter Schwabe. A new hope on ARM Cortex-M. In Claude Carlet, Anwar Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Advanced Cryptography Engineering*, Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg, 2016 (to appear). Document ID: c7a82d41d39c535fd09ca1b032ebca1b, http://cryptojedi.org/papers/#newhopearm.

[3] Ahmad Boorghany, Siavash Bayat Sarmadi, and Rasool Jalili. On constrained implementation of lattice-based cryptographic primitives and schemes on smart cards. *ACM Trans. Embed. Comput. Syst.*, 14(3):42:1–42:25, 2015. https://eprint.iacr.org/2014/514.pdf.

[4] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. Flush, Gauss, and Reload – a cache attack on the BLISS lattice-based signature scheme. In Benedikt Gierlichs and Axel Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, volume 9813 of *LNCS*, pages 323–345. Springer, 2016. https://eprint.iacr.org/2016/300/.

[5] Johannes Buchmann, Florian Göpfert, Tim Güneysu, Tobias Oder, and Thomas Pöppelmann. High-performance and lightweight lattice-based public-key encryption. In *Proceedings of the 2nd ACM International Workshop on IoT Privacy, Trust, and Security*, pages 2–9. ACM, 2016. http://sha.rub.de/media/sh/veroeffentlichungen/2016/11/22/buchmann-iotpts.pdf.

[6] Tung Chou. QcBits: Constant-time small-key code-based cryptography. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 280–300. Springer, 2016. http://www.win.tue.nl/~tchou/papers/qcbits.pdf.

[7] Craig Costello, Patrick Longa, and Michael Naehrig. Efficient algorithms for supersingular isogeny diffie-hellman. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016*, volume 9814 of *LNCS*, page 572–601. Springer, 2016. https://eprint.iacr.org/2016/413/.

[8] Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Efficient software implementation of ring-LWE encryption. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2015*, pages 339–344. EDA Consortium, 2015. http://eprint.iacr.org/2014/725.

[9] Nir Drucker and Shay Gueron. A toolbox for software optimization of QC-MDPC code-based cryptosystems. IACR ePrint report 2017/1251, 2017. https://eprint.iacr.org/2017/1251.pdf.

[10] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, volume 8042 of *LNCS*, page 40–56. Springer, 2013.

[11] Thomas Eisenbarth, Tim Güneysu, Stefan Heyse, and Christof Paar. MicroEliece: McEliece for embedded devices. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2009.

[12] Thomas Espitau, Pierre-Alain Fouque, Benoït Gérard, and Mehdi Tibouchi. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS'17*. ACM, 2017 (to appear). https://eprint.iacr.org/2017/505/.

[13] Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 530–547. Springer, 2012.

[14] Stefan Heyse, Ingo von Maurich, and Tim Güneysu. Smaller keys for code-based cryptography: QC-MDPC McEliece implementations on embedded devices. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, volume 8086 of *Lecture Notes in Computer Science*, pages 273–292. Springer, 2013.

[15] Andreas Hülsing, Christoph Busold, and Johannes A. Buchmann. Forward secure signatures on smart cards. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, volume 7707 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2012.

[16] Andreas Hülsing, Joost Rijneveld, and Peter Schwabe. Armed SPHINCS - computing a 41 KB signature in 16 KB of RAM. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I*, volume 9614 of *Lecture Notes in Computer Science*, pages 446–470. Springer, 2016.

[17] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography*, volume 7071 of *LNCS*, page 19–34. Springer, 2011. https://eprint.iacr.org/2011/506/.

[18] ARM Limited. ARMv7-M architecture reference manual, issue C_errata_v3, 2010.

[19] Zhe Liu, Hwajeong Seo, Sujoy Sinha Roy, Johann GroÃ§schädl, Howon Kim, and Ingrid Verbauwhede. Efficient Ring-LWE encryption on 8-bit AVR processors. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems – CHES 2015*, volume 9293 of *LNCS*, pages 663–682. Springer, 2015. https://eprint.iacr.org/2015/410.pdf.

[20] Ingo Von Maurich, Tobias Oder, and Tim Güneysu. Implementing QC-MDPC McEliece encryption. *ACM Transactions on Embedded Computing Systems (TECS)*, 14(3):44, 2015. https://pqcrypto.eu/docs/1-MDPCforACMTECS_preprint.pdf.

[21] R. J. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*, 44:114–116, January 1978.

[22] Tobias Oder, Thomas Pöppelmann, and Tim Güneysu. Beyond ECDSA and RSA: Lattice-based digital signatures on constrained devices. In *The 51st Annual Design Automation Conference 2014, DAC '14, San Francisco, CA, USA, June 1-5, 2014*, pages 110:1–110:6. ACM, 2014.

[23] Peter Pessl, Leon Groot Bruinderink, and Yuval Yarom. To BLISS-B or not to be – attacking strongSwanâĂŹs implementation of post-quantum signatures. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS'17*. ACM, 2017 (to appear). https://eprint.iacr.org/2017/490/.

[24] Thomas PÃűppelmann, Tobias Oder, and Tim GÃijneysu. High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers. In Kristin Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology – LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 346–365. Springer, 2015. http://eprint.iacr.org/2015/382/.

[25] Ingo von Maurich, Lukas Heberle, and Tim Güneysu. IND-CCA secure hybrid encryption from QC-MDPC Niederreiter. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, volume 9606 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2016.