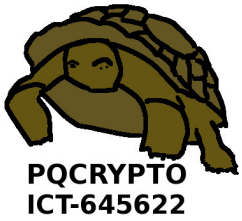


PQCRYPTO

Post-Quantum Cryptography for Long-Term Security

Project number: Horizon 2020 ICT-645622

Small Devices: D1.1 Intermediate Report on Algorithms

Due date of deliverable: 30. September 2016
Actual submission date: November 9, 2016

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: D1.1 Intermediate Report on Algorithms

Tim Güneysu, Tobias Oder, Peter Schwabe

Contributors: Daniel J. Bernstein, Joppe W. Bos, Tung Chou, Andreas Hülsing, Tanja Lange, Joost Rijneveld, Ko Stoffelen

November 9, 2016
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.

Abstract

This document provides the PQCRYPTO project's intermediate report for post-quantum cryptographic algorithms that focus small devices. Algorithms are selected based on a level on confidence and their suitability for the constraints of small embedded devices.

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

Contents

1 Introduction **1**

2 Target Definition and Requirements **1**

3 Symmetric Cryptography **2**
3.1 Encryption . . . 2
3.1.1 AES-256 Block Cipher . . . 2
3.1.2 Salsa20 Stream Cipher . . . 3
3.2 Authentication . . . 4
3.2.1 Authenticated Encryption: AES-GCM . . . 4
3.2.2 Message Authentication Codes: Poly1305 . . . 4

4 Asymmetric Cryptography **5**
4.1 Encryption . . . 5
4.1.1 Goppa-based McEliece/Niederreiter . . . 5
4.1.2 QC-MDPC McEliece . . . 5
4.1.3 Ring-LWE Encryption . . . 6
4.1.4 Implementation Results . . . 7
4.2 Digital Signatures . . . 7
4.2.1 XMSS . . . 8
4.2.2 SPHINCS . . . 9
4.2.3 BLISS . . . 9
4.2.4 Implementation Results . . . 10
4.3 Key Exchange . . . 12
4.3.1 QC-MDPC KEM . . . 12
4.3.2 NewHope . . . 12
4.3.3 Frodo . . . 13
4.3.4 Implementation Results . . . 14

5 Directions & Outlook **15**

1 Introduction

The EU and governments around the world are investing heavily in building quantum computers. Society needs to be prepared for the consequences, including cryptanalytic attacks accelerated by these computers. In particular, Shor's algorithm [50] shatters the foundations for deployed public-key cryptography: RSA and the discrete-logarithm problem in finite fields and elliptic curves. Long-term confidential documents such as patient health-care records and state secrets have to guarantee security for many years, but information encrypted today using RSA or elliptic curves and stored until quantum computers are available will then be as easy to decipher as Enigma-encrypted messages are today.

The PQCRYPTO project's mission is to allow users to switch to post-quantum cryptography: cryptographic systems that are not merely secure for today but that will also remain secure long-term against attacks by quantum computers. During the project, PQCRYPTO will design a portfolio of high-security post-quantum public-key systems, and will improve the speed of these systems, adapting to the different performance challenges of mobile devices, the cloud, and the Internet of Things.

This document provides PQCRYPTO's intermediate report and recommendations for post-quantum algorithms suitable for small devices. These systems were selected for confidence in their security against cryptanalytic attacks and given constraints by common hardware and software systems.

Beyond these recommendations, this document also lists some further examples of systems that are currently under evaluation, but this document does not mention any new systems under construction inside or outside PQCRYPTO. This document focuses on cryptographic primitives to be used inside higher-level cryptographic protocols and security protocols; it does not give specific recommendations for those protocols.

2 Target Definition and Requirements

This report discusses algorithms that are suitable for an implementation on embedded devices. As embedded device we consider small and constrained computing platforms that play an important part in the internet of things. On the one hand we consider the suitability of the algorithms for hardware implementations, but on the other hand we also consider microcontroller implementations. While ASIC implementations of the algorithms are possible, we focus on FPGA implementations as they are much cheaper (for low quantities) and much faster to develop. The microcontroller implementations in this report have a broad range of target architectures. While some target low-cost microcontrollers like 8-bit AVR ATxmega or 32-bit ARM Cortex-M0 we also consider implementations for more powerful microcontrollers like ARM Cortex-M4.

Implementing post-quantum cryptography on embedded devices needs to adopt several requirements, depending if the implementation targets software, hardware or both. The requirements can be summarized as follows:

1. *Minimum Memory (HW+SW).* It is important to keep the memory consumption of the implementation as low as possible as embedded devices only have access to a limited amount of memory. For software implementation this typically corresponds to the amount of required registers, on-chip RAM and Flash/EEPROM memory that determines the cost of an implementation. In hardware this determines the need for flip-flops

or other storage elements such as ROM or RAM.

2. *Low Combinatorial Logic Complexity (HW).* Hardware implementations also aim for a low area consumption as this directly influences the cost. Performance is also important, especially for real-time applications where there is only a narrow time frame in which a response is expected.

3. *Minimum Energy consumption (HW+SW).* As many embedded products are battery-powered, it is often required to reduce the energy consumption below a specific limit.

4. *Minimum Power Consumption (HW).* Likewise, in particular for cryptographically supported RFID units, it is important not to exceed the instantaneous power consumption beyond a specific threshold at any point in time.

5. *Physical Protection (SW+HW).* Finally, for devices that store secret keys it is important that the cryptographic operations executed by this device are secured against physical attacks, such as side-channel attacks, fault-injection attacks, reverse-engineering or active tampering.

3 Symmetric Cryptography

3.1 Encryption

Symmetric systems are usually not affected by Shor's algorithm, but they are affected by Grover's algorithm. Under Grover's attack, the best security a key of length n can offer is $2^{n/2}$, so AES-128 offers only 2^{64} post-quantum security. PQCRYPTO recommends thoroughly analyzed ciphers with 256-bit keys to achieve 2^{128} post-quantum security that are (a) AES-256 [17] for block encryption and (b) Salsa20 [7] with a 256-bit key as stream cipher.

3.1.1 AES-256 Block Cipher

AES was published as Rijndael in 1998 and standardized in FIPS PUB 197 in 2001. Highly optimized implementations have been written for most common architectures, ranging from 8-bit AVR microcontrollers to x86-64 and NVIDIA GPUs. See, for example, [10, 30, 41]. Implementing optimized AES on any of these architectures essentially requires to start from scratch to find out which implementation approach is going to be the most efficient. Sometimes an embedded device contains a coprocessor that can perform AES encryption in hardware, but such a coprocessor is not always available. It makes a device more expensive and it can increase the power consumption of a device. Simply compiling an existing implementation written in, for example, the C programming language, is unlikely to produce optimal performance. Even worse, embedded systems are typical targets for timing attacks, power analysis attacks, and other forms of side-channel attacks, so software for those devices typically needs to include adequate protection against such attacks.

Schwabe et al. [49] fill these gaps by providing highly optimized AES software implementations for two of the most popular modern microprocessors for constrained embedded devices, the ARM Cortex-M3 and the Cortex-M4. Their implementations of AES-$\{128, 192, 256\}$-CTR are more than twice as fast as existing implementations. They also provide a single-block

Algorithm	**Speed (cycles)**		**ROM (bytes)**		**RAM (bytes)**	
	M3	**M4**	**Code**	**Data**	**I/O**	**Stack**
AES-128 key expansion encryption	289.8	294.8	902	1,024	176	32
AES-128 key expansion decryption	1,180.0	1,174.6	3,714	2,048	176	176
AES-128 single block encryption	659.4	661.7	2,034	1,024	$176 + 2m$	44
AES-128 single block decryption	642.5	648.3	1,974	2,048	$176 + 2m$	44
AES-128-CTR	546.3	554.4	2,192	1,024	$192 + 2m$	72
AES-192 key expansion	264.9	272.2	810	1,024	240	32
AES-192-CTR	663.2	673.0	2,576	1,024	$224 + 2m$	72
AES-256 key expansion	364.8	371.8	1,166	1,024	240	32
AES-256-CTR	786.9	791.7	2,960	1,024	$256 + 2m$	72
AES-128 bitsliced key expansion	1,027.8	1,033.8	3,434	1,036	368	188
AES-128-CTR bitsliced constant-time	1,616.6	1,617.6	12,120	12	$368 + 2m$	108
AES-128-CTR masked constant-time	N/A	7,422.6	3,9916	12	$368 + 2m$	1,588

Table 3.1: Performance of different AES implementations [49]

AES-128 implementation, a constant-time AES-128-CTR implementation and a masked implementation that is secure against first-order power analysis attacks. All of them are the fastest of their kind. The performance is summarized in Table 3.1.1.

3.1.2 Salsa20 Stream Cipher

Salsa20 is a stream cipher which has been proposed in 2005 [7]. It has been included in the final portfolio of the eSTREAM project initiated by the European Network of Excellence for Cryptology (ECRYPT) in 2004. The cipher consists of 20 rounds where an internal state is modified by various (logical and arithmetic) transformations. It supports various key-lengths; in the PQCRYPTO context a 32-byte (256-bit) key should be used.

Hutter and Schwabe proposed a high-speed and low-area implementation for the AVR family [29]. In their work they identify how to efficiently map the initialization and round function to the 8-bit architectures. Their initialization design consists of 7 loop iterations where the state x (and a copy of the state j which is later added to the cipher output) gets initialized with the 32-byte key, the 64-byte input, and a 16-byte nonce. The initialization takes 718 clock cycles in total. The round-calculation function provides the most promising potential to increase the speed of Salsa20. It consists of ten loop iterations that include 8 quarterround function calls (thus 80 function calls in total). Within one quarterround function, three different 32-bit operations (addition, bitwise addition, and rotations) are performed on either the rows or the columns of the state x. Their quarterround function call requires 174 clock cycles in total. The entire round calculation needs 15,623 clock cycles. The entire crypto stream function needs 18 166 clock cycles for a 64-byte message. The code size of their high-speed implementation of Salsa20 is 1,750 bytes. The performance of their low-area implementation is slightly reduced by 697 clock cycles, resulting in 18,863 clock cycles for crypto stream; the code size is reduced by 658 bytes to only 1,092 bytes, i.e., by 37.6 % of the

former code size.

3.2 Authentication

Some message-authentication codes provide "information-theoretic security", guaranteeing that they are as secure as the underlying cipher (within a negligible mathematically guaranteed forgery probability), even against an adversary with unlimited computing power. These authentication mechanisms are not affected by quantum computing. Particular instances that can be used and are recommend by PQCRYPTO in this context are GCM [37] using a 96-bit nonce and a 128-bit authenticator. As an alternative, Poly1305 [6] seems to be a suitable choice for small devices as well.

3.2.1 Authenticated Encryption: AES-GCM

Galois/Counter mode is a NIST-standardized block cipher mode of operation for authenticated encryption [36]. The 128-bit authentication key H is derived from the master encryption key K during key setup as the encryption of an all-zero input block. The computation of the authentication tag then requires, for each 16-byte data block, a 128-bit multiplication by H in the finite field $\mathbb{F}_{2^{128}} = \mathbb{F}_2[X]/(X^{128} + X^7 + X^2 + X + 1)$. The full details can be found in the specification [36].

The feasibility of AES-GCM on embedded devices has been shown in several publications. We exemplarily show the results of a microcontroller implementation [23] and an FPGA implementation [1]. In [23] the target architecture is an MSP430X microcontroller that has an AES accelerator. Without this accelerator the authors report a runtime of 696 cycles/byte for a 16-bytes message and 314 cycles/byte for a 4-kbytes message. Using the AES accelerator, it is possible to improve theses results such that processing a 16-bytes message takes 426 cycles/byte and 180 cycles/byte for a 4-kbytes message. The authors of [1] implemented AES-GCM on a Virtex5 FPGA. They achieve a throughput of 32.46 Gbit/s for a BRAM-based implementation, 31.36 Gbit/s for an implementation using composite fields, and 36.92 Gbit/s for a LUT-based implementation.

3.2.2 Message Authentication Codes: Poly1305

Poly1305 is a cryptographic message authentication code as proposed by Bernstein in [6]. The name is related to the underlying polynomial $2^{130} - 5$. A message m with variable size n is authenticated using a (random) 32-byte one-time secret key s typically computed from a nonce. The secret key s consists of two parts, each 16-bytes in length, i.e., $s = (k, r)$. First, the message m is split into 16-byte blocks where each block is padded with a 1. The resulting 17-byte chunks c_i, where $i \in [1, q]$ and $q = \lceil n/16 \rceil$, are then represented as unsigned little-endian integers. After that, one addition and one modular multiplication is performed for each chunk c resulting in the 16-byte authenticator h, i.e.,

$$h = (((c_1 \cdot r^q + c_2 \cdot r^{q-1} + \ldots + c_q \cdot r^1) \bmod 2^{130} - 5) + k \bmod 2^{128}$$

Poly1305 has been implemented in 8-bit AVR microcontrollers by Hutter and Schwabe in [29]. They provide a high-speed and a low-area implementation and evaluate the performance for different message sizes. The results can be found in Table 3.2.

Implementation	Message bytes	Cycles	Stack bytes
High-speed	8	4,411	148
	64	12,525	
	576	98,477	
	1024	173,685	
	2048	345,588	
Low-area	8	4,773	148
	64	13,270	
	576	103,286	
	1024	182,050	
	2048	362,081	

Table 3.2: Poly1305 cycle counts on the AVR ATmega2560 microcontroller.

4 Asymmetric Cryptography

For public-key encryption the currently used algorithms based on RSA and ECC are easily broken by quantum computers. Code-based cryptography has been studied since 1978 and has withstood attacks very well, including attacks using quantum computers.

4.1 Encryption

PQCRYPTO recommends the following parameters as included in McBits [8] to achieve 2^{128} post-quantum security, defined as McEliece with binary Goppa codes using length $n = 6960$, dimension $k = 5413$ and adding $t = 119$ errors. PQCRYPTO also evaluates quasi-cyclic MDPC codes [38] for McEliece with parameters at least $n = 2^{16} + 6, k = 2^{15} + 3, d = 274$ and adding $t = 264$ errors as well as ring-LWE encryption [34] as representative of the family of lattice-based encryption schemes.

4.1.1 Goppa-based McEliece/Niederreiter

The public-key cryptosystem discussed here is a code-based cryptosystem with a long history and a well-established security track record: namely, Niederreiter's dual form [39] of McEliece's hidden-Goppa-code cryptosystem [35]. This cryptosystem is well known to provide extremely fast encryption and reasonably fast decryption. Goppa codes are a conservative choice of code for the cryptosystem as they are well studied and understood, but they result in rather large public keys. For instance, the implementation of Bernstein et al. [8] targeting an Intel Ivy Bridge processor reports a public key size of 221 kbytes at a pre-quantum security level of 128 bits. As the public key usually has to be transferred at some point, such dimensions are problematic for embedded applications. Hence, up to now only small instances of Goppa-based McEliece encryption (up to a security level of 80-bit) have been implemented on embedded systems [19, 21].

4.1.2 QC-MDPC McEliece

To avoid the huge keys that come with the Goppa code-based instatiation of the McEliece cryptosystem, a variant using QC-MDPC codes instead has been proposed by Misoczki et al.[38]. While McEliece with Goppa codes usually has key sizes of 50-100 kbytes or more, the

public key in the QC-MDPC McEliece scheme is only 0.6 kbytes for 80-bit of pre-quantum security. The difference stems from the additional structure in the parity check matrix. In the Goppa code-based scheme, the complete matrix has to be stored, but in the in QC-MDPC code-based scheme, only the first row of the matrix has to be stored and the remaining ones are generated by cyclic shifts of that row. This additional structure might reduce the security of the scheme. However, as far as we know, QC-MDPC McEliece has not been broken so far and is thus a valid candidate for post-quantum public-key encryption. Most implementations focus on a security level of 80 bits of pre-quantum security [25, 51, 53]. For long-term security this is obviously not enough and thus we encourage further performance evaluation of parameter sets with a higher security level.

The performance and resource consumption of QC-MDPC McEliece has been evaluated on reconfigurable hardware in [25] and [51]. While the work of [25] aims for a high-speed implementation for Virtex-6 FPGAs, [51] focuses more on developing a lightweight implementation that even fits on a low-cost Spartan 6-FPGA. Thus the results are very different. While the high-speed implementation of [25] takes 13.7 microseconds for encryption and 125.4 microseconds for decryption, the lightweight implementation of [51] is two orders of magnitude slower as it takes 3.4 milliseconds for encryption and 23 milliseconds for decryption. On the other hand, the lightweight implementation takes much less resources. The encryption takes only 119 FFs, 226 LUTs, and 64 slices while the high-speed encryption needs 14,429 FFs, 9,201 LUTs, and 2,924 Slices. However, the lightweight implementation requires 1 resp. 3 BRAMs for encryption resp. decryption while the high-speed implementation does not require any.

4.1.3 Ring-LWE Encryption

Beside post-quantum schemes based on codes, lattice-based cryptography also offers encryption schemes. One candidate is the ring-LWE encryption scheme [34]. Its security is based on the ring variant of the learning with errors problem. It consists of three algorithms:

- **Gen**(a): Choose $r_1, r_2 \leftarrow D_\sigma$ and let $p = r_1 - a \cdot r_2 \in R$. The public key is p and the secret key is r_2 while r_1 is just noise and not needed anymore after key generation. The value $a \in R$ can be defined as a global constant or chosen uniformly random during key generation.
- **Enc**$(a, p, m \in \{0,1\}^n)$: Choose the noise terms $e_1, e_2, e_3 \leftarrow D_\sigma$. Let $m' = \texttt{encode}(m)$ $\in R$, and compute the ciphertext $[c_1 = a \cdot e_1 + e_2, c_2 = p \cdot e_1 + e_3 + m' \in R^2$.
- **Dec**(c_1, c_2, r_2): Output $\texttt{decode}(c_1 \cdot r_2 + c_2) \in \{0,1\}^n$.

The first research addressing the feasibility of ring-LWE encryption on reconfigurable hardware was proposed by Göttert et al. [22] in 2012. The authors presented a hardware implementation of the ring-LWE encryption scheme on a Virtex 7 FPGA. To achieve an acceptable level of performance, the authors tweaked the parameters of the scheme to be able to use the Number-theoretic transform (NTT) for lowering the complexity of polynomial multiplication from $O(n^2)$ to $O(n \log n)$. In contrast to matrix-vector multiplication, polynomial multiplication in the frequency domain, computed using NTT, can be optimized in several ways. During the transformation, it is necessary to compute the so called twiddle factors, which are powers of a root of unity. Those twiddle factors can be precomputed or calculated on-the-fly. Designers can choose the preferred implementation depending on the design goals, namely

whether the implementation should be optimized for memory consumption or performance. The core operation of the NTT is the so called butterfly, which operates on two coefficients of the polynomial and performs one multiplication, one addition, and one subtraction. Multiple butterfly operations can be executed in parallel. Furthermore, the Gaussian sampler, a fundamental operation in lattice-based schemes, can also be optimized. Three implementation approaches have been proposed in the past [22]: rejection sampling, which does not need any precomputations but is usually very slow, a table-based sampler, that requires a large amount of memory but is very fast, and a rounding-based approach, which however differs from an actual Gaussian distribution.

Pöppelmann and Güneysu [44] presented an optimized NTT multiplier. Their work was further extended to implement a complete ring-LWE encryption scheme in 2013 [45]. While previous designs [22] were area inefficient and needed to be implemented on large FPGAs, such as the Virtex 7 device, Pöppelmann and Güneysu [45] proposed an architecture suitable for smaller reconfigurable devices, such as a Spartan 6 device. Furthermore, since their implementation relies on a generic microcode engine, it can also be used for other lattice-based implementations. Since then, several further optimizations have been proposed. Aysu et al. reduced the area consumption of the NTT [4]. Roy et al. enhanced the performance of NTT [48] by optimizing the memory access and simplifying the structure of the algorithm. That design was further optimized by using a more efficient Knuth-Yao sampler [31] which requires less FPGA area. The smallest FPGA implementation, to the best of our knowledge, has been presented by Pöppelmann and Güneysu [46], the overall resource occupation is 32 slices, 1 BRAM, and 1 DSP. To achieve such a low area design, the authors chose a parameter set for which the modulus is a power of two. As a result, there was no need for a modular reduction step. The drawback of the proposed set of parameters is that the NTT is no longer applicable. As a result, the computation time is increased by one order of magnitude.

Furthermore there are implementations targeting 8-bit AVR microcontrollers. The work of Liu et al. [32] focuses on the modular reduction. The authors provide a highly optimized implementation of the Barrett reduction [5] in assembly language and thus achieve a performance of 21 milliseconds for encryption and 8.6 milliseconds for decryption for $n = 256$. In comparison, the work of Pöppelmann et al. [47] focuses on the application and implementation of the NTT. As their modular reduction is less optimized than the one from [32], they report a performance of 27 milliseconds for encryption and 6.7 milliseconds for decryption. Another difference is that the program code of the implementation from [47] has a size of 6,668 bytes and the one from [32] is significantly bigger code size of 13,604 bytes.

4.1.4 Implementation Results

An overview of selected implementations results is given by Table 4.1 representing the performance of aforementioned schemes on microcontrollers while Table 4.2 shows hardware implementations.

4.2 Digital Signatures

Similar to encryption, currently used signatures are based on problems that become easy to solve with a quantum computer. Signatures use cryptographic hash functions in order to hash the message and then sign the hash. Hash-based signatures use nothing but such a hash function and thus assume the minimum requirement necessary to build signatures.

Scheme	Security	Platform	Encryption	Decryption
QC-MDPC McE[25]	80 bits	ATxmega256	26,767,463	86,874,388
QC-MDPC McE[52]	80 bits	STM32F4	2,623,432	18,416,012
QC-MDPC McE[52]	128 bits	STM32F4	13,725,688	80,260,696
Goppa McE [19]	80 bits	ATxmega256	14,406,080	19,751,094
Ring-LWE[47]	105 bits	ATxmega128A1	874,347	215,863

Table 4.1: Microcontroller implementation cycle counts of post-quantum encryption schemes. The given security is the pre-quantum security.

Scheme	Security	Platform	FFs	LUTs	Slices	BRAMs	Time
QC-MDPC McE enc[25]	80 bits	XC6VLX240T	14,429	9,201	2,924	0	13.7 μs
QC-MDPC McE dec[25]	80 bits	XC6VLX240T	32,974	36,554	10,271	0	125.4 μs
QC-MDPC McE enc[51]	80 bits	XC6SLX4	119	226	64	1	3.4 ms
QC-MDPC McE dec[51]	80 bits	XC6SLX4	413	605	159	3	23.0 ms
Goppa McE enc[19]	80 bits	XC3S1400AN	804	1,044	668	3	2.2 ms
Goppa McE dec[19]	80 bits	XC3S1400AN	8,977	22,034	11,218	20	21.6 ms
Ring-LWE enc[47]	105 bits	XC6SLX9	238	317	95	2	0.9 ms
Ring-LWE dec[47]	105 bits	XC6SLX9	87	112	32	1	0.4 ms

Table 4.2: FPGA implementation results of post-quantum encryption schemes. Note that the given security levels are considering the pre-quantum setting.

PQCRYPTO recommends the following two hash-based systems to achieve 2^{128} post-quantum security:

- XMSS [16] with any of the parameters specified in [27]. XMSS requires maintaining a state.
- SPHINCS-256 [9]. SPHINCS is stateless.

We furthermore also evaluate the lattice-based signature scheme BLISS [18].

4.2.1 XMSS

XMSS [16] is short for extended Merkle signature scheme. It belongs to the family of hash-based signature schemes as it consists of a tree of hash values. Its security is only based on the assumption that the underlying hash function is secure and it provides forward secrecy. The leaves of the hash tree are hashes of one-time public keys, i.e. public keys that can only be used once. Thus the number of signatures the can be generated from a key pair is limited and the signer has to maintain a state that keeps track of which one-time public keys have been used already.

An optimized variant of this scheme has been implemented by Hülsing et al. in [26]. Their target platform is the Infineon SLE78 16-bit microcontroller that runs at 33 MHz. For a security level of 128 considering the best known attacks it is possible to generate one signature in 97 milliseconds (worst case) and verify it in 83 milliseconds (average case). The key generation is noticeably slower and takes 6.7 seconds and has to be repeated for every 2^{16} signatures. The secret key has a size of 3,232 byte.

4.2.2 SPHINCS

Another hash-based signature is SPHINCS [9]. Unlike most hash-based designs, this signature scheme is stateless, allowing it to be a drop-in replacement for current signature schemes as standard APIs cannot deal with stateful signatures that require to update the secret key after each signing. SPHINCS is carefully designed so that its security can be based on weak standard-model assumptions, avoiding collision resistance and the random-oracle model. Hash-based signature schemes are usually organized as Merkle trees. SPHINCS introduces two new ideas that together drastically reduce signature size. First, to increase the security level of randomized index selection, SPHINCS replaces the leaf one-time signature (OTS) with a hash-based *few-time* signature scheme (FTS). An FTS is, as the name suggests, a signature scheme designed to sign a few messages; in the context of SPHINCS this allows a few index collisions, which in turn allows a smaller tree height for the same security level.

Second, SPHINCS views Goldreich's construction as a hyper-tree construction with h layers of trees of height 1, and generalizes to a hyper-tree with d layers of trees of height h/d. This introduces a tradeoff between signature size and time controlled by the number of layers d. The signature size is $|\sigma| \approx d|\sigma_{OTS}| + hn$ assuming a hash function with n-bit outputs. Recall that the size of a one-time signature $|\sigma_{OTS}|$ is roughly $\mathcal{O}(n^2)$, so by decreasing the number of layers we get smaller full signatures. The tradeoff is that signing time increases exponentially in the decrease of layers: signing takes $d2^{h/d}$ OTS key generations and $d2^{h/d} - d$ hash computations.

The selected parameters provide 128 bits of security against quantum attackers and keep a balance between signature size and time. Using this parameter set results in a signature size of 41 KB that exceeds the memory capabilities of most low-cost microcontrollers. To get an implementation of SPHINCS running on a Cortex-M3, Hülsing et al. [28] implemented a streaming interface and divide the signature into portions that get streamed out of the board over the serial port before the next portion is computed. At a clock frequency of 32 MHz the key generation takes 0.88 ms. The signing takes 18.41 seconds and the verification 513 milliseconds. The slow speed of the signing process is mainly due to the fact that SPHINCS is a stateless hash-based signature scheme (in comparison to XMSS that is stateful). The communication overhead for the streaming of the signature is not significant.

4.2.3 BLISS

Another family of post-quantum signature schemes is based on hard lattice problems. The Bimodal Lattice Signature scheme (BLISS) as presented in [18] is one famous example and offers a high efficiency and terms of speed and signature size. The algorithms for key generation, signing, and verification are given in Alg. 4.1-4.3. The security of BLISS is based on NTRU assumption and the ring variant of the short integer solution problem. Lattice-based signature schemes that were generated using the Fiat-Shamir transform ([20], [18], [24], [33]) have in common that they feature a rejection step that prevents the leakage of secret information through the signature. The main improvement of BLISS in comparison to the previous signature schemes is that the authors make use of a bimodal Gaussian distribution to reduce the rejection rate and thus improve the performance of the scheme.

The performance of BLISS on embedded devices has been studied well. There are implementations on microcontrollers (AVR [47, 12, 4] and ARM[40]) as well as FPGA implementations [43]. In the following we briefly summarize the results of these implementations. Note

that the discussed implementations target a security level of 128 bits against classical attackers. Thus the actual post-quantum security is expected to be lower than 128 bits. We are not aware of any BLISS implementation that explicitly implements a post-quantum parameter set. In [40] BLISS has been implemented on a (in comparison to other embedded devices) rather powerful ARM Cortex-M4F. The sampling is performed by using the Bernoulli approach as presented in [18]. This first implementation of BLISS on an embedded device achieved a performance of 35.3 and 6 ms for signing and verification. The downside is that the key generation is quite slow and takes 2.19 seconds. The reason is that the NTRU-like approach of the key generation of BLISS requires the inversion of a polynomial, there is a restart condition in the key generation, and the computation of the $\mathrm{N}_\kappa(\mathbf{S})$ bound is quite expensive. However, an embedded device typically stores long-term keys for signing and verification and thus the key generation is only executed rarely or even never if there are factory-set keys. The implementation also takes a lot of memory, as signing requires 18.5 kbytes of RAM and 24.6 kbytes of Flash memory. In return, the signature size is only 5,600 bits what is the theoretically smallest possible signature size for the given parameter set. It has been achieved by applying a Huffman encoding to the signature.

The FPGA implementation in [43] uses a different sampling approach that is based in table-look ups and improve this approach even further by exploiting the Kullback-Leibler divergence. As a result, the authors are able to reduce the size of the precomputed table and make the implementation fit into a low-cost Xilinx Spartan-6 FPGA. As expected a hardware implementation performs much better than a software implementation and thus the authors report only 114.1 microseconds per signing operation and 61.2 microseconds per verification. Their BLISS-I core uses 2,291 slices, 5.5 BRAMs, and 5 DSPs. This implementation applies Huffman encoding as well. Boorghany et al. implemented an authentication protocol based on BLISS for AVR microcontrollers [12, 13]. The main difference is that the sparse polynomial $\mathbf{c}$ is not obtained from a random oracle but randomly generated by the other protocol party. As the target devices (ATmega64 and ATxmega64A3) belong to an older generation of microcontrollers and are less powerful than current ones, the reported performance is only 5.3 seconds ([12]) and 0.6 seconds ([13]) per protocol run.

Algorithm 4.1: BLISS Key Generation

Result: Key Pair $(\mathbf{A}, \mathbf{S}) = q \bmod 2q$

begin
- Choose **f, g** as uniform polynomials with exactly d_1 entries in $\{\pm 1\}$ and d_2 entries in $\{\pm 2\}$
- $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2) \leftarrow (\mathbf{f}, 2\mathbf{g} + 1)^t$
- **if** $N_\kappa(\boldsymbol{S}) \geq C^2 \cdot 5 \cdot (\lceil \delta_1 n + 4\delta_2 n) \cdot \kappa$ **then**
  - restart
- $\mathbf{a}_q = (2\mathbf{g} + 1)/f \bmod q$ restart if $\mathbf{f}$ is not invertible)
- $\mathbf{A} \leftarrow (2\mathbf{a}_q, q - 2) \bmod 2q$

4.2.4 Implementation Results

Selected implementation results of lattice-based signature schemes for microcontrollers are given in Table 4.3.

Algorithm 4.2: BLISS Signature Algorithm

Data: Message μ, public key $\mathbf{A} = (\mathbf{a}_1, q-2) \in \mathcal{R}_{2q}^{1\times 2}$, secret key $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2)^t \in \mathcal{R}_{2q}^{2\times 1}$
Result: Signature $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$

begin
- $\mathbf{y}_1, \mathbf{y}_2 \leftarrow D_{\mathbb{Z}^n,\sigma}$
- $\mathbf{u} = \zeta \cdot \mathbf{a}_1 \cdot \mathbf{y}_1 + \mathbf{y}_2 \bmod 2q$
- $\mathbf{c} = H(\lfloor \mathbf{u} \rceil_d \bmod p, \mu)$
- Choose a random bit b
- $\mathbf{z}_1 \leftarrow \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \mathbf{c}$
- $\mathbf{z}_2 \leftarrow \mathbf{y}_2 + (-1)^b \mathbf{s}_2 \mathbf{c}$
- Continue with a probability $1/(M\exp(-\frac{||\mathbf{Sc}||^2}{2\sigma^2})\cosh(\frac{\langle \mathbf{z}, \mathbf{Sc}\rangle}{\sigma^2}))$ otherwise restart
- $\mathbf{z}_2^\dagger = (\lfloor \mathbf{u} \rceil_d - \lfloor \mathbf{u} - \mathbf{z}_2 \rceil_d) \bmod p$

Algorithm 4.3: BLISS Verification Algorithm

Data: Message μ, public key $\mathbf{A} = (\mathbf{a}_1, q-2) \in \mathcal{R}_{2q}^{1\times 2}$, Signature $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$
Result: Accept or Reject

begin
- **if** $||(\boldsymbol{z}_1 | 2^d \cdot \boldsymbol{z}_2^\dagger)||_2 > B_2$ **then**
  - Reject
- **if** $||(\boldsymbol{z}_1 | 2^d \cdot \boldsymbol{z}_2^\dagger)||_\infty > B_\infty$ **then**
  - Reject
- Accept if and only if $\mathbf{c} = H(\lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} \rceil_d + \mathbf{z}_2^\dagger \bmod p, \mu)$

Scheme	Security	Platform	Signing	Verification
SPHINCS[28]	256 bits	STM32L100C	584,384,791	5,991,643
XMSS^{MT}[28]	256 bits	STM32L100C	19,441,021	4,961,447
BLISS [40]	128 bits	STM32F4	5,927,441	1,002,299

Table 4.3: Microcontroller implementation cycle counts of post-quantum signature schemes. The given security is the pre-quantum security.

4.3 Key Exchange

The third important family of algorithms in asymmetric cryptography consists of key exchange schemes. We evaluate a key encapsulation mechanism based on the McEliece cryptosystem [42] as well as the "A new hope" lattice-based key exchange scheme [2].

4.3.1 QC-MDPC KEM

QC-MDPC codes can also be used to construct a hybrid encryption scheme. Hybrid encryption schemes are divided into two independent components: (1) a key encapsulation mechanism (KEM) and (2) a data encapsulation mechanism (DEM). The KEM is a public-key encryption scheme that encrypts a randomly generated symmetric session key under the public key of the intended receiver. The DEM then encrypts the plaintext under the randomly generated session key using a symmetric encryption scheme. Hybrid encryption is usually beneficial in practice because symmetric encryption is orders of magnitude more efficient than pure asymmetric encryption, especially for large plaintexts. On the other hand sole usage of symmetric schemes is not practical due to the symmetric key distribution problem. Hybrid encryption takes the best of two worlds, efficient symmetric data encryption combined with asymmetric key distribution. The idea behind the code-based KEM from [42] is to derive a key from an error vector and send the syndrome of the error vector to the other party. The scheme has been implemented on ARM Cortex-M4F by Maurich et al. [52]. The key generation takes 386.4 ms for 80 bits of security, the hybrid encryption takes 16.5 ms and the hybrid decryption takes 111.0 ms. For 128 bits of security the performance is 1511.8 ms for key generation, 83.2 ms for encryption, 477.5 ms for decryption.

4.3.2 NewHope

In this section we cover the lattice-based key exchange by Alkim et al. [2] that is called "a new hope" and is an extension of previous lattice-based key exchange schemes [15, 54]. The algorithm is given in Protocol 1 and all polynomials except for $\mathbf{r} \in \mathcal{R}_4$ are defined in the ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ with $n = 1024$ and $q = 12289$. The authors decided to keep the dimension $n = 1024$ the same as in [15] to be able to achieve appropriate long-term security. As polynomial arithmetic is fast and also scales better (doubling n roughly doubles the time required for a polynomial multiplication), the choice of n appears to be acceptable from a performance point of view. The modulus $q = 12289$ is chosen as it is the smallest prime for which it holds that $q \equiv 1 \mod 2n$ so that the number-theoretic transform (NTT) can be realized efficiently. The main improvements of "a new hope" in comparison to [15] stem from a more detailed security analysis and an improved analysis of the failure probability of the protocol that allows an instantiation of the protocol with smaller parameters. Furthermore, the

authors replaced the almost-perfect discrete Gaussian distribution by a binomial distribution that is relatively close, but much easier to sample from.

Parameters: $q = 12289 < 2^{14},\ n = 1024$ Error distribution: ψ_{16}		
Alice (server)		**Bob (client)**
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$\mathbf{a} \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$		
$\mathbf{s}, \mathbf{e} \xleftarrow{\$} \psi_{16}^n$		$\mathbf{s}', \mathbf{e}', \mathbf{e}'' \xleftarrow{\$} \psi_{16}^n$
$\mathbf{b} \leftarrow \mathbf{as} + \mathbf{e}$	$\xrightarrow{(\mathbf{b}, seed)}$	$\mathbf{a} \leftarrow \mathsf{Parse}(\mathsf{SHAKE\text{-}128}(seed))$
		$\mathbf{u} \leftarrow \mathbf{as}' + \mathbf{e}'$
		$\mathbf{v} \leftarrow \mathbf{bs}' + \mathbf{e}''$
$\mathbf{v}' \leftarrow \mathbf{us}$	$\xleftarrow{(\mathbf{u}, \mathbf{r})}$	$\mathbf{r} \xleftarrow{\$} \mathsf{HelpRec}(\mathbf{v})$
$\nu \leftarrow \mathsf{Rec}(\mathbf{v}', \mathbf{r})$		$\nu \leftarrow \mathsf{Rec}(\mathbf{v}, \mathbf{r})$
$\mu \leftarrow \mathsf{SHA3\text{-}256}(\nu)$		$\mu \leftarrow \mathsf{SHA3\text{-}256}(\nu)$

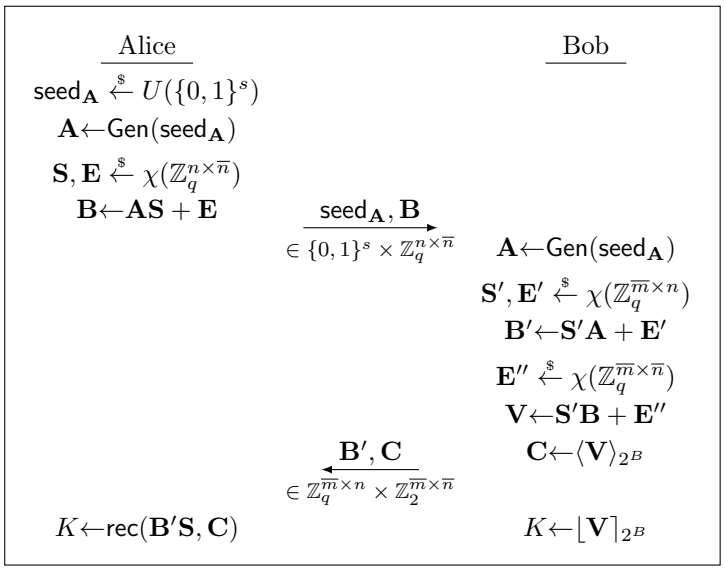
Protocol 1: NewHope key exchange.

Alkim et al. also present a microcontroller implementation of the "a new hope" key exchange scheme in [3]. The target architectures are the Cortex-M0 and Cortex-M4 and the implemented parameter set achieves 128 bits of post-quantum security (with a comfortable margin). The implementation makes heavy use of optimization techniques on assembly level like merging multiple stages of the NTT and pipelining of load and store instructions. As far as we know, they provide the fastest implementation of the NTT on Cortex-M microcontrollers and thus other ARM Cortex implementations of ideal lattice-based schemes will benefit from this NTT implementation if they adapt it. The overall performance of the scheme on the Cortex-M0 is 1.5 million cycles on the server side and 1.7 million cycles and the client side what translates to 31/36 milliseconds at a clock frequency of 48 MHz. On the more powerful Cortex-M4 the runtime is 860,388 cycles at the server side and 984,761 cycles at the client side. When operated at 168 MHz, this means that the server needs 5.1 milliseconds for execution and the client 5.9.

4.3.3 Frodo

While the security of "a new hope" is based on ideal lattice problems, the Frodo proposal by Bos et al. [14] is a key exchange scheme based on standard lattices. While ideal lattices facilitate major efficiency and storage benefits over their non-ideal counterparts, the additional ring structure that enables these advantages also raises concerns about the assumed difficulty of the underlying problems. Protocol 2 summarizes the key exchange scheme. Both Alice and Bob generate the same large matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ that is combined with the LWE secrets to compute their public keys as instances of the LWE problem. Alice's $\overline{n}$ LWE instances and Bob's $\overline{m}$ LWE instances are combined to compute a secret matrix in $\mathbb{Z}_q^{\overline{m} \times \overline{n}}$, where B uniform bits are extracted from each entry to form the session key K. Thus, the dimensions $\overline{n}$ and $\overline{m}$ should be chosen such that K has (at least) the number of required bits for the target security level. For example, in targeting 128 bits of post-quantum security, it should be the case that $\overline{n} \cdot \overline{m} \cdot B \geq 256$. This condition ensures that we obtain a uniform 256-bit

secret for the session key and even an exhaustive key search via Grover's quantum algorithm would take 2^{128} operations. Protocol 2 allows for the ratio between $\overline{n}$ and $\overline{m}$ to be changed, in order to trade-off between Bob's amount of uploaded data for Alice's computational load. The major challenge for the implementation of Frodo on embedded devices is to deal with the large parameters. To avoid storing complete matrices it would be possible to heavily exploit on-the-fly computation and use a streaming interface to transmit the matrix $\mathbf{B}$. On-the-fly computation usually comes with a performance penalty. A designer of embedded systems thus has to carefully find a good trade-off between the memory requirement and the performance of the scheme.



Protocol 2: The LWE-based key exchange protocol with LWE parameters (n, q, χ), and protocol specific parameters $\overline{n}, \overline{m}, B \in \mathbb{Z}$. The matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ is generated from $\mathsf{seed}_\mathbf{A}$ via a pseudo-random function Gen.

4.3.4 Implementation Results

Table 4.4 shows selected microcontroller implementations of key exchange/key encapsulation schemes.

Scheme	Security	Platform	Server	Client
NewHope [3]	281 bits	STM32F0	1,467,101	1,738,922
NewHope [3]	281 bits	STM32F4	1,143,314	1,418,124
McEliece/Niederreiter KEM[52]	128 bits	STM32F4	80,260,696	13,725,688

Table 4.4: Microcontroller implementation cycle counts of post-quantum key exchange schemes. The given security margin is considering the pre-quantum setting.

5 Directions & Outlook

As discussed in the previous sections, a range of implementations of post-quantum cryptosystem for embedded devices have been reported to date. However, still very few implementation already meet a security level that is conjectured to provide medium-term or long-term resistance against quantum computers. We briefly review our findings and identify directions for future work.

Symmetric Cryptography. For encryption and authentication efficient solutions based on AES-256 or Salsa20 seem already available and ready for deployment. Still additional options for post-quantum secure authenticated encryption is currently under investigation as part of the CAESAR competition.

Asymmetric Cryptography – Code-based Cryptography. Encryption systems over binary Goppa codes with conservative post-quantum-secure parameters ($n = 6960$, dimension $k = 5413$, $t = 119$ errors) seem to be impossible with available versions of 8-bit and 32-bit microcontrollers or (moderate-cost) FPGAs. Experimental QC-MDPC McEliece encryption exhibit a significantly smaller memory footprint and reasonable performance on 32-bit ARM Cortex-M4 microcontrollers and low-cost FPGAs. However, reported embedded implementations have been investigated for a short-term pre-quantum security level only. Further investigation is required to identify how QC-MDPC-based designs can still be efficiently implemented on embedded devices in the post-quantum setting with larger security parameters.

Asymmetric Cryptography – Hash-based Cryptography. XMSS and SPHINCS are both promising hash-based digital signatures schemes that come with solid security guarantees in the post-quantum settings. Due to their maturity, recent standardization processes are particularly focussing on these cryptosystems. In the context of embedded systems, successful implementations were reported. Yet SPHINCS suffers from large signatures and comparably low performance, XMSS has the disadvantage of maintaining a state. For deployment, the appropriate and secure realization of maintaining this state for XMSS pose an additional challenge for security applications and protocols.

Asymmetric Cryptography – Lattice-based Cryptography. Considering lattice-based cryptography a number of proposals have emerged, including encryption, digital signature schemes and key exchange schemes. In the context of embedded systems it has been shown that particularly schemes based on ideal lattices are extremely efficient, yet cryptanalytically experimental. Assuming thorough cryptanalysis and given maturity, however, lattice-based cryptography to date provide the best fit with respect to efficiency and versatility for the embedded context.

Asymmetric Cryptography – MQ-based Cryptography. Although it seems that there is less activity on MQ-based cryptosystems, Unbalanced Oil-and-Vinegar or HFEv- signature schemes might be viable further alternatives. Due to the absence of recent results considering their implementation on embedded systems, further investigation is required.

References

[1] Karim M Abdellatif, Roselyne Chotin-Avot, and Habib Mehrez. AES-GCM and AEGIS: Efficient and high speed hardware implementations. *Journal of Signal Processing Systems*, pages 1–12, 2016.

[2] 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.

[3] 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.

[4] Aydin Aysu, Cameron Patterson, and Patrick Schaumont. Low-cost and area-efficient FPGA implementations of lattice-based cryptography. In *HOST*, pages 81–86. IEEE Computer Society, 2013.

[5] Paul Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Proceedings on Advances in cryptology—CRYPTO '86*, pages 311–323, London, UK, 1987. Springer-Verlag.

[6] Daniel J. Bernstein. The Poly1305-AES message-authentication code. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*, pages 32–49. Springer, 2005.

[7] Daniel J. Bernstein. The Salsa20 Family of Stream Ciphers. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2008.

[8] Daniel J. Bernstein, Tung Chou, and Peter Schwabe. McBits: Fast constant-time code-based cryptography. In Bertoni and Coron [11], pages 250–272.

[9] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: Practical Stateless Hash-Based Signatures. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 368–397. Springer, 2015.

[10] Daniel J. Bernstein and Peter Schwabe. New AES software speed records. In Dipanwita Roy Chowdhury and Vincent Rijmen, editors, *Progress in Cryptology – INDOCRYPT 2008*, volume 5365 of *LNCS*, pages 322–336. Springer, 2008. http://cryptojedi.org/users/peter/#aesspeed.

[11] 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*. Springer, 2013.

[12] Ahmad Boorghany and Rasool Jalili. Implementation and comparison of lattice-based identification protocols on smart cards and microcontrollers. *IACR Cryptology ePrint Archive*, 2014:78, 2014.

[13] Ahmad Boorghany, Siavash Bayat Sarmadi, and Rasool Jalili. On constrained implementation of lattice-based cryptographic primitives and schemes on smart cards. *IACR Cryptology ePrint Archive*, 2014:514, 2014.

[14] 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. *IACR Cryptology ePrint Archive*, 2016:659, 2016.

[15] 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 *2015 IEEE Symposium on Security and Privacy*, pages 553–570, 2015. http://eprint.iacr.org/2014/599.

[16] Johannes A. Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - A practical forward secure signature scheme based on minimal security assumptions. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*, volume 7071 of *Lecture Notes in Computer Science*, pages 117–129. Springer, 2011.

[17] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[18] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In *Advances in Cryptology–CRYPTO 2013*, pages 40–56. Springer, 2013.

[19] 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.

[20] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 186–194. Springer, 1986.

[21] Santosh Ghosh, Jeroen Delvaux, Leif Uhsadel, and Ingrid Verbauwhede. A speed area optimized embedded co-processor for McEliece cryptosystem. In *23rd IEEE International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2012, Delft, The Netherlands, July 9-11, 2012*, pages 102–108. IEEE Computer Society, 2012.

[22] Norman Göttert, Thomas Feller, Michael Schneider, Johannes Buchmann, and Sorin Huss. On the design of hardware building blocks for modern lattice-based encryption schemes. In *Cryptographic Hardware and Embedded Systems–CHES 2012*, pages 512–529. Springer, 2012.

[23] Conrado Porto Lopes Gouvêa and Julio López. High speed implementation of authenticated encryption for the MSP430X microcontroller. In Alejandro Hevia and Gregory Neven, editors, *Progress in Cryptology - LATINCRYPT 2012 - 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, October 7-10, 2012. Proceedings*, volume 7533 of *Lecture Notes in Computer Science*, pages 288–304. Springer, 2012.

[24] 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 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*, pages 530–547. Springer, 2012.

[25] Stefan Heyse, Ingo von Maurich, and Tim Güneysu. Smaller keys for code-based cryptography: QC-MDPC McEliece implementations on embedded devices. In Bertoni and Coron [11], pages 273–292.

[26] 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.

[27] Andreas Hülsing, Denis Butin, Stefan Gazdag, and Aziz Mohaisen. XMSS: Extended Hash-Based Signatures. Crypto Forum Research Group Internet-Draft, 2015. https://datatracker.ietf.org/doc/draft-irtf-cfrg-xmss-hash-based-signatures/.

[28] 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.

[29] Michael Hutter and Peter Schwabe. NaCl on 8-bit AVR microcontrollers. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *Progress in Cryptology - AFRICACRYPT 2013, 6th International Conference on Cryptology in Africa, Cairo, Egypt, June 22-24, 2013. Proceedings*, volume 7918 of *Lecture Notes in Computer Science*, pages 156–172. Springer, 2013.

[30] Emilia Käsper and Peter Schwabe. Faster and timing-attack resistant AES-GCM. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *LNCS*, pages 1–17. Springer, 2009. https://cryptojedi.org/papers/#aesbs.

[31] Donald E Knuth and Andrew C Yao. The complexity of nonuniform random number generation. *Algorithms and complexity: new directions and recent results*, pages 357–428, 1976.

[32] Zhe Liu, Hwajeong Seo, Sujoy Sinha Roy, Johann Großschädl, 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 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 663–682. Springer, 2015.

[33] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 738–755. Springer, 2012.

[34] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer, 2010.

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

[36] David A. McGrew and John Viega. The Galois/Counter Mode of operation (GCM). http://www.cryptobarn.com/papers/gcm-spec.pdf.

[37] David A. McGrew and John Viega. The security and performance of the Galois/Counter Mode (GCM) of operation. In *Progress in Cryptology – INDOCRYPT 2004*, volume 3348 of *LNCS*, pages 343–355. Springer, 2005.

[38] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo S. L. M. Barreto. MDPC-McEliece: New McEliece variants from moderate density parity-check codes. In *Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7-12, 2013*, pages 2069–2073. IEEE, 2013.

[39] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems Control Inform. Theory/Problemy Upravlen. Teor. Inform.*, 15(2):159–166, 1986.

[40] 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.

[41] Dag Arne Osvik, Joppe W. Bos, Deian Stefan, and David Canright. Fast software AES encryption. In Seokhie Hong and Tetsu Iwata, editors, *Fast Software Encryption*, volume 6147 of *LNCS*, pages 75–93. Springer, 2010.

[42] Edoardo Persichetti. Secure and anonymous hybrid encryption from coding theory. In Philippe Gaborit, editor, *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings*, volume 7932 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 2013.

[43] Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 353–370. Springer, 2014.

[44] Thomas Pöppelmann and Tim Güneysu. Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware. In *Progress in Cryptology–LATINCRYPT 2012*, pages 139–158. Springer, 2012.

[45] Thomas Pöppelmann and Tim Güneysu. Towards practical lattice-based public-key encryption on reconfigurable hardware. In *Selected Areas in Cryptography–SAC 2013*, pages 68–85. Springer, 2013.

[46] Thomas Pöppelmann and Tim Güneysu. Area optimization of lightweight lattice-based encryption on reconfigurable hardware. In *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, pages 2796–2799. IEEE, 2014.

[47] Thomas Pöppelmann, Tobias Oder, and Tim Güneysu. High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, volume 9230 of *Lecture Notes in Computer Science*, pages 346–365. Springer, 2015.

[48] Sujoy Sinha Roy, Frederik Vercauteren, Nele Mentens, Donald Donglong Chen, and Ingrid Verbauwhede. Compact ring-LWE cryptoprocessor. In *Cryptographic Hardware and Embedded Systems–CHES 2014*, pages 371–391. Springer, 2014.

[49] Peter Schwabe and Ko Stoffelen. All the AES you need on Cortex-M3 and M4. *IACR Cryptology ePrint Archive*, 2016:714, 2016.

[50] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134. IEEE Computer Society, 1994.

[51] Ingo von Maurich and Tim Güneysu. Lightweight code-based cryptography: QC-MDPC McEliece encryption on reconfigurable devices. In Gerhard Fettweis and Wolfgang Nebel, editors, *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*, pages 1–6. European Design and Automation Association, 2014.

[52] 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.

[53] Ingo von Maurich, Tobias Oder, and Tim Güneysu. Implementing QC-MDPC McEliece encryption. *ACM Trans. Embedded Comput. Syst.*, 14(3):44, 2015.

[54] Jiang Zhang, Zhenfeng Zhang, Jintai Ding, Michael Snook, and Özgür Dagdelen. Authenticated key exchange from ideal lattices. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9057 of *LNCS*, pages 719–751. Springer, 2015. https://eprint.iacr.org/2014/589/.