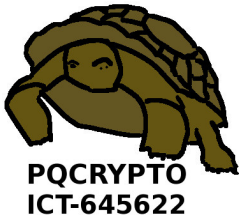

PQCRYPTO

Post-Quantum Cryptography for Long-Term Security

Project number: Horizon 2020 ICT-645622

Small Devices: D1.7 Final Report

Due date of deliverable: 28. February 2018
Actual submission date: April 16, 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: D1.7 Final Report

Tim Güneysu, Peter Schwabe, Ko Stoffelen, Joost Rijneveld, Tobias Oder

April 16, 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.

Abstract

This document provides the PQCRYPTO project's final report on all results achieved by the partners in work package 1. It summarizes recommendations considering the efficient software and hardware implementation of post-quantum algorithms on embedded low-cost devices. In particular, it also includes results and countermeasures obtained from mounting physical attacks against the implementations.

Keywords: Post-quantum cryptography, small devices, software implementations, hardware implementations, physical security

Contents

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 [77] 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.

Modern asymmetric cryptosystems that are designed to face multiple threats and maintain long-term security require conservative parameter choices that typically pose a major implementation challenge for constrained devices. We developed a number of new implementation techniques (and also adapted already known techniques) to make post-quantum cryptography feasible on constrained devices despite the limited computing resources available to these devices.

In this report we summarize the work that has been carried out by WP1. To tackle the challenging task of quantum-safe cryptography on constrained devices, PQCRYPTO developed a number of software and hardware implementations for embedded devices, like microcontrollers and FPGAs. As in many applications an attacker has physical access to these devices, they also need to be secured against side-channel attacks. To improve the side-channel security of post-quantum implementations the partners also examined possible attacks and proposed countermeasures against those. Below we summarize the software and hardware implementations developed by WP1 and the side-channel attacks resp. countermeasures that have been found resp. developed by WP1.

1.1 Implementations developed by PQCRYPTO

Table 1.1 gives an overview of all WP1 microcontroller implementations. Our hardware implementations are shown in Table 1.2. We go into more detail about the software implementations in Section 3 and about the hardware implementations in Section 4.

1.2 Side-channel attacks and countermeasures by PQCRYPTO

In Table 1.3 we summarize the side-channel attacks and countermeasures that have been carried out by PQCRYPTO. A more detailed discussion can be found in Section 5.

2 Target Platforms

The goal of this work package is to develop implementations for small devices that have constrained computing capabilities. The software implementations discussed in this report therefore target microcontroller architectures. The challenging part about developing microcontroller implementations is to make to implementation fit into the devices constrained memory and still achieve a practical performance despite the low clock frequency the microcontroller is running out. Furthermore, the developer has to deal with a reduced instruction set, for instance some microcontroller do not even have access to a hardware multiplier.

Scheme	Platform	Cycles		RAM Bytes		ROM Bytes
NewHope [3]	ARM Cortex-M4	**gen:**	964 440	**gen:**	-	22 828
		enc:	1 418 124	**enc:**	-	22 828
		dec:	178 874	**dec:**	-	22 828
NewHope [3]	ARM Cortex-M0	**gen:**	1 168 224	**gen:**	-	30 178
		enc:	1 738 922	**enc:**	-	30 178
		dec:	298 877	**dec:**	-	30 178
QcBits [21]	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 [15]	ARM Cortex-M0	**gen:**	-	**gen:**	-	-
		enc:	1 573 x10^3	**enc:**	-	1.6 KiB
		dec:	740 x10^3	**dec:**	-	1.1 KiB
RLWE encryption [15]	AVR ATxmega	**gen:**	-	**gen:**	-	-
		enc:	999 x10^3	**enc:**	-	3.5 KiB
		dec:	437 x10^3	**dec:**	-	2.1 KiB
QC-MDPC encryption [57]	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 [83]	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
SPHINCS-256 [44]	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} [44]	ARM Cortex-M3	**gen:**	8 857 708 189	**gen:**	-	-
		sign:	19 441 021	**sign:**	-	-
		verify:	4 961 447	**verify:**	-	-

Table 1.1: Microcontroller implementation results of post-quantum public-key encryption schemes. All implementations listed are results of the PQCRYPTO project.

Scheme	Security	Platform	Pub.
NewHope	281 bits	XC7A35T	[62]
BLISS	128 Bits	XC6SLX25	[67]

Table 1.2: Hardware implementations developed by PQCRYPTO. For a better comparison the stated security levels assume a classical attacker.

Name	Scheme	Pub.
CCA2 Masking	R-LWE	[64]
Cache Attack	BLISS	[13]
Cache Attack	BLISS	[66]

Table 1.3: Side-channel attacks and countermeasures.

The primary testing platform for code targeting 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.

Regarding the hardware implementations, there are two classes of target architectures for efficient cryptographic implementations, field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs). An FPGA is a reconfigurable integrated circuit. In contrast to an ASIC, an FPGA is not bounded to one specific purpose since the configuration can be changed at any time after manufacturing and even after distribution in the field. This is especially useful when the security measures of a production system in the field turn out to be insecure. To achieve a high level of flexibility, the FPGA largely consists of a regular grid of configurable logic blocks (CLBs). The CLBs themselves consist of a number of slices and a switching matrix. While the slices contain the reconfigurable logic, the switch matrices connect adjacent CLBs and realize the routing of the signals. Besides CLBs, there are other resources available such as block RAM memory and digital signal processors (DSPs). FPGAs are used increasingly often in commercial products since their reconfigurability allows fast prototyping and therefore reduces the time to market. Also the regular structure leads to a simpler design cycle since routing, placement, and timing can mostly be automated.

On the other side, ASICs are designed to serve one particular purpose for which it comprises a specifically tailored hardware circuit. Once manufactured, the design is static and adjustments can only be made by physically exchanging the hardware. But this allows for more compact designs and speed optimizations since only components that are actually necessary are included in the design while FPGA designs most likely will not reach 100% utilization. More compact designs mean cheaper unit costs for high volume designs. Additionally, ASIC

designs are usually less power-consuming than FPGA designs. But on the other hand, designing an ASIC comes with upfront non-recurring costs for development tools and expensive respins.

The choice of the target architecture mainly depends on the use case. If a highly optimized implementation is expected to meet the design requirements and high quantities of the product should be produced, it is probably best to choose an ASIC as the target platform as they are cheaper in production. For lower quantities and greater flexibility after distribution, FPGA implementations should be considered as they offer faster prototyping and are cheaper in development. In particular, the aspect of flexibility by FPGAs is often regarded as highly beneficial for security applications since their cryptographic cores can be easily replaced in the field in case they turn out to be insecure. This particularly holds true for post-quantum cryptography: since many of the proposed constructions are rather new and thus have not undergone a thorough and long-lasting cryptanalytic review yet. FPGA implementations are preferable when quick upgrades are necessary and expected over the lifetime of the service. In this sense, we will put particular emphasis on FPGA implementations in this report.

3 Software Implementations

3.1 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.

3.1.1 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.

3.1.2 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 [53]).

3.1.3 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.

3.1.4 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.

3.1.5 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.

3.2 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.

3.2.1 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.

3.2.2 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.

3.2.3 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.

3.2.4 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.

3.3 Implementations of public-key encryption and key encapsulation

Table 1.1 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. Note that this table only summarizes "standalone" implementations. For benchmarking results of the final software portfolio packaged in the **pqm4** library, see Deliverable 1.6.

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 [45, 24], 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.

3.3.1 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 [72] and [55] present optimized Ring-LWE encryption targeting the AVR 8-bit microcontrollers; [15] describe implementations on the ARM Cortex M0 and AVR ATXmega128; [25] presents results for the ARM Cortex-M4F; and [3] 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 [2] established that a much simpler centered-binomial distribution is sufficient for lattice-based encryption and key agreement. Consequently, [3] use this simpler distribution in their implementation of NewHope.

3.3.2 Code-based public-key encryption and key encapsulation

The McEliece cryptosystems [58] 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, [29] 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, [39] targets AVR 8-bit microcontrollers; [57] optimizes McEliece with QC-MDPC codes on ARM Cortex-M4 microcontrollers; the same architecture targeted by the "QcBits" key-encapsulation mechanism described in [21] and in the recent work [26]. Both AVR 8-bit microcontrollers and 32-bit ARM microcontrollers (specifically, the Cortex-M4) are the target of optimization in [83].

3.4 Digital Signatures

Table 1.1 also 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.

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.

3.4.1 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, [43] 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 insurmountable hurdle: on devices with limited storage space available, it is not possible to store the full signature. The implementation described in [44] 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.

3.4.2 Lattice-based Signatures

Optimization of lattice-based signatures on embedded microcontrollers so far focused mainly on two schemes: the GLP signature scheme presented in [35] and the BLISS signature scheme presented in [27]. For example, [72] presents optimizations of BLISS for AVR ATXmega microcontrollers. Also [63] presents optimized implementations of BLISS, but on the ARM Cortex-M4F. In [12], the authors describe a conversion of the signature schemes from [35] and [27] 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.

4 Hardware Implementations

In this section we give an overview of existing implementations of post-quantum schemes. We are not aware of any hardware implementations of multivariate quadratic schemes. For hash-based scheme, there are implementations of one-time signature schemes [6], but as those are

only able to generate a single signature for a given key pair, we did not consider those. Thus we mainly focus on implementations of lattice-based and code-based schemes. We also briefly review some hardware implementations of hash-functions as those are the building blocks for the hash-based signature schemes XMSS [16] and SPHINCS [8].

4.1 Encryption and Key Exchange

There are several hardware implementations of post-quantum encryption and key exchange schemes. The most relevant ones are reviewed here and summarized in Table 4.1.

Scheme	Security	Platform	FFs	LUTs	Slices	BRAMs	Time
QC-MDPC McE enc[39]	80 bits	XC6VLX240T	14,429	9,201	2,924	0	13.7 μs
QC-MDPC McE dec[39]	80 bits	XC6VLX240T	32,974	36,554	10,271	0	125.4 μs
QC-MDPC McE enc[81]	80 bits	XC6SLX4	119	226	64	1	3.4 ms
QC-MDPC McE dec[81]	80 bits	XC6SLX4	413	605	159	3	23.0 ms
Goppa McE enc[29]	80 bits	XC3S1400AN	804	1,044	668	3	2.2 ms
Goppa McE dec[29]	80 bits	XC3S1400AN	8,977	22,034	11,218	20	21.6 ms
Ring-LWE enc[70]	105 bits	XC6SLX9	238	317	95	2	0.9 ms
Ring-LWE dec[70]	105 bits	XC6SLX9	87	112	32	1	0.4 ms
Ring-LWE enc/dec[69]	105 bits	V6LX75T	3624	4549	1506	12	26 μs
Standard-LWE enc[42]	128 bits	S6LX45	4,676	6,078	1,866	73	0.8 ms
Standard-LWE dec[42]	128 bits	S6LX45	58	63	32	13	0.2 ms
Lattice-based IBE [80]	80 bits	S6LX25	6,067	7,023	-	16	80 μs
Lattice-based IBE [80]	192 bits	S6LX25	8,686	8,882	-	27	164 μs
NewHope (server)[62]	281 bits	XC7A35T	4,452	5,142	-	4	1.4 ms
NewHope (client)[62]	281 bits	XC7A35T	4,635	4,498	-	4	1.5 ms

Table 4.1: FPGA implementation results of post-quantum encryption schemes. Note that the given security levels are considering the pre-quantum setting.

4.1.1 Lattice-based Cryptography

Lattice-based cryptography can be divided into two groups, one for the schemes that base their security on standard lattices and one for schemes that base their security on ideal lattices. While the latter offers a higher performance and smaller key sizes they also introduce an additional structure in the underlying lattice and that is why standard lattice schemes are usually considered to be a more conservative choice.

The (standard) learning with errors (LWE) encryption scheme [65] has been implemented by Howe et al [42]. The two main operations in this scheme are matrix-vector multiplication and Gaussian sampling. The matrix-vector multiplications are performed serially with the help of a single DSP. Gaussian samples are generated using a Bernoulli sampler [27] as it does not require large precomputed tables. The uniformly distributed random numbers that the Gaussian sampler requires as input are generated using the stream cipher Trivium [17]. Their implementation of the LWE encryption requires 6,078 LUTs, 4,676 FFs, and 1,811 slices on a Spartan-6 FPGA. Due to the size of the keys the implementation also requires 73 BRAM modules. This number can be reduces by generating parts of the public key on-the-fly instead of storing it precomputed in BRAMs.

The counterpart to LWE in ideal lattices is called ring learning with errors (R-LWE) [56]. This scheme has been implemented several times. The first implementation of R-LWE has been published by Göttert et al. in 2012 [34]. he authors presented a hardware implementation of the RLWE 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 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 butterfly operation that takes two coefficients of the polynomial and performs one multiplication, one addition, and one subtraction. Multiple butterfly operations can be executed in parallel.

Pöppelmann and Güneysu [68] presented an optimized NTT multiplier. Their work was further extended to implement a complete RLWE encryption scheme in 2013 [69]. While the design by Göttert et al. was large and could only be placed on large Virtex-7 FPGAs, Pöppelmann and Güneysu 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 [5]. Roy et al. enhanced the performance of NTT [76] 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 [50] which requires less FPGA resources. The smallest FPGA implementation, to the best of our knowledge, has been presented by Pöppelmann and Güneysu [70], the overall resource occupation of which 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. New Hope by Alkim et al. [2] is an efficient key exchange scheme based on R-LWE. The implementation by Oder and Güneysu [62] shows that the scheme is practical on FPGAs. Due to the higher security level (and therefore larger parameters) the implementation is significantly slower than implementations of plain R-LWE encryption.

There are also a number of FPGA implementations of the NTRU encryption scheme [41], like [47] or [54]. However as NTRU is still protected by patents [40] we do not further consider NTRU implementations.

Lattice-based cryptography also allows practical identity-based encryption. The identity-based encryption proposed by Ducas et al. [28] has been implemented for an FPGA as well [80]. However, the implementation only covers the encryption and the decryption that are very similar the the R-LWE encryption scheme. The master key generation and the user key generation has not been implemented as the constrained resources of an FPGA are not sufficient to perform these operations.

4.1.2 Code-based Cryptography

The code-based encryption scheme McEliece [59] and its variant by Niederreiter [61] have also been implemented on hardware devices. McEliece can be instantiated with different codes. While the original scheme uses Goppa codes, quasi-cyclic modest density parity-check (QC-MDPC) codes provide a better efficiency. However, QC-MDPC codes provide an additional structure that might be exploitable. Choosing QC-MDPC codes over Goppa codes is therefore similar to choosing ideal lattices over standard lattice.

McEliece with Goppa codes has been implemented on a Xilinx Spartan-3AN FPGA by Eisenbarth et al [29]. As the implementation relies on Goppa codes, the public and the secret key are huge matrices. The McEliece decryption is much more complex than the encryption due to the decoding algorithm that is used to recover the message. Thus, the implementation of [29] needs 668 slices, 1,044 LUTs, 804 FFs, and 3 BRAMs for the encryption, but 11,218 slices, 22,034 LUTs, 8,977 FFs, 20 BRAMs for the decryption. Both modules (encryption and decryption) also need 4,644 Kbits of Flash memory. The design was improved by Ghosh et al. [32] by reducing the number of required slices during the decryption to 2,979 and the number BRAMs to 5. Furthermore the decryption latency was reduced to 1 ms instead of 10.8 ms.

The performance and resource consumption of QC-MDPC McEliece has been evaluated on reconfigurable hardware in [39] and [81]. While the work of [39] aims for a high-speed implementation for Virtex-6 FPGAs, [81] 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 [39] takes 13.7 microseconds for encryption and 125.4 microseconds for decryption, the lightweight implementation of [81] 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.

4.2 Digital Signatures

For post-quantum digital signatures, less hardware implementations exist. The implementations we are aware of are summarized in Table 4.2.

Scheme	Security	Platform	FFs	LUTs	Slices	BRAMs	Time
GLP-Sign[35]	80 bits	XC6SLX16	8,993	7,465	2,273	29.5	1 ms
GLP-Verify[35]	80 bits	XC6SLX16	6,663	6,225	2,263	15	1 ms
BLISS-Sign[67]	128 bits	XC6SLX16	6,420	7,193	2,291	5.5	114 μs
BLISS-Verify[67]	128 bits	XC6SLX16	4,312	5,065	1,687	4	58 μs
SPHINCS[4]	256 bits	Kintex 7	38,132	19,067	-	36	1.53 ms

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

4.2.1 Lattice-based Cryptography

Implementing lattice-based signature schemes is a more challenging task, since the standard deviation of the Gaussian sampler is usually much higher compared to encryption schemes.

Furthermore, additional components, such as hash functions and a rejection step, are required. So far, only ideal lattice-based signature schemes have been implemented on FPGAs. The main reason is that standard lattice signature schemes like TESLA [1] need to be instantiated with very large parameters.

The GLP signature scheme [35] was presented by G$\ddot{\text{u}}$neysu et al. in 2012 and implemented on a Spartan 6 FPGA. The BLISS digital signature scheme [27] was also implemented in hardware [67]. The major difference between both schemes is that the error polynomials in BLISS have Gaussian distributed coefficients and ternary coefficients in GLP. The BLISS FPGA design uses a table-based Gaussian sampler which, thanks to the Kullback-Leibler divergence, can be implemented using little memory without affecting the performance. Implementing Gaussian samplers in hardware is a challenging task: Many samplers have an non-constant running time; e.g., when using rejection sampling, the sampler could theoretically require an infinite number of iterations. Other samplers need large precomputed tables that contribute to a significant share of the implementations overall memory consumption. In general, the Knuth-Yao sampler is considered a good trade-off between runtime and memory consumption. For small standard deviations, the binomial sampler [2] is a good choice since it does not require any precomputed tables and has a constant running time. But as the standard deviation grows, other techniques, like the already mentioned Knuth-Yao, become more interesting since they feature a lower average running time and entropy consumption. The entropy consumption of a sampler is important because it is also necessary to implement a pseudo-random number generator that produces uniformly random bits.

4.2.2 Hash-based Cryptography

The only hardware implementation of hash-based signature schemes is the SPHINCS implementation by Amiet et al. [4]. The main building block in hash-based cryptography are hash functions and hardware implementations of those are plentiful. The Third SHA-3 Candidate Conference brought up lots of implementations results, like [37, 31, 52, 49, 48, 46]. It outperforms SHA-2 by an order of magnitude and is therefore an interesting candidate to instantiate hash-based signature schemes with.

5 Physical Attacks on Cryptographic Implementations

In this section, we discuss attacks exploiting physical properties of an implementation to gain knowledge of the secret key used in the executed algorithm. One distinguishes between passive attacks in which the attacker only monitors information, like execution time, power consumption, or electromagnetic radiation, and active attacks in which the attacker is allowed to interfere in the execution of the cipher.

When dealing with active attacks, one distinguishes between different levels of invasiveness. A non-invasive attacker is only allowed to modify the environment like the temperature, the voltage of the power supply, or the duration of clock cycles. These attacks usually aim to generate a faulty result that can be used to reveal the secret key. A semi-invasive attacker removes the package material of the device and introduces faults by shooting at the a specific location at the device with light or electromagnetic radiation. Invasive attackers aim to even alters the device itself and reverse-engineer the implementation.

Implementations are vulnerable to timing attacks if their execution time depends on secret data. Power analysis exploits the fact that in CMOS technology the dynamic power

consumption is dominating in comparison to the static power consumption. An attacker executes the algorithm and measures the power consumption during the execution. The most important types of attacks on the power consumption leakage are simple power analysis (SPA) and differential power analysis (DPA).

5.1 Fault attacks

The idea of fault attacks is to induce a fault into a circuit and use the faulty output to get information about the secret key. This can be achieved by high temperature, unsupported supply voltage or current, excessively high overclocking, strong electric or magnetic fields, or even ionizing radiation. Fault attacks are usually non-invasive as the induced fault is only temporary and the device is not permanently damaged. The most prominent fault attack in cryptography was carried out by Boneh et al. [11]. Their attack on RSA-CRT signatures requires only one (arbitrary) faulty output and one correct output to break the scheme.

5.2 Timing attacks

When implementing cryptographic algorithms, the developer has to make sure that the execution time is independent of the secret data that is processed. Otherwise an attacker might be able to exploit the information about the execution time. Such attacks should not only be considered for embedded devices for which the attacker has physical access to, but also remote timing attacks are a threat that must be considered as shown by Brumley and Boneh [14]. Timing information can be leaked by conditional branches, instructions with non-constant execution time, and memory accesses that trigger cache hits or misses [7].

5.3 Simple power analysis

Simple power analysis [51] works similar to timing attacks. However, while timing attacks exploit the timing information of one or many executions of the algorithms, one or a few power traces of the executed algorithms are used to perform a simple power analysis. An attacker uses visual examination to identify leaking instructions whose execution depends on secret data. Thus, this attack is especially effective when the order of the executed instructions differs from run to run. For instance, an RSA implementation with a naive implementation of the square-and-multiply algorithm can easily be broken by SPA as the square operations and multiply operations are usually easily distinguishable in the power trace. Signal-processing techniques, like frequency filters, might improve the result and make the visual inspection easier.

5.4 Differential power analysis

While SPA targets the operation-dependency of the power consumption, DPA exploits its data-dependency. Introduced in 1998 by Kocher et al. [51], DPA (in contrast to SPA) needs many power traces and one analyzes the set of traces with statistical methods. When performing DPA an attacker does not attack the whole key at once, but only a part, e.g. one byte. A DPA is divided in an online phase and an offline phase. During the online phase, the attacker runs a vast amount of executions of the algorithm to be attacked with different inputs and measures the power consumption of the target device during each run. DPA requires a leakage model that is a prediction of the power consumption. Some leakage models are rather simple. For

example, the Hamming weight model is based on the observation that the Hamming weight of a value that is stored in a register, influences the power consumption. During the offline phase, the attacker guesses the key byte and computes the intermediate value that he considers suitable to apply the power model to. Depending on the power model and the intermediate value, she assigns the corresponding power trace to one of two sets where one contains power traces with high predicted power consumption and one set contains traces with low prediction power consumption. For all power traces, the attacker stores the difference of the means of the sets. If the attack worked, the correct key guess has a much higher difference of means than the other guesses.

The Hamming weight leakage model is often applied when attacking software implementations. For hardware implementations a promising model is the Hamming distance model. The Hamming distance model assumes that the more bit positions of the input and output values of a circuit differ, the more switching operations happened within a circuit, and the higher is the power consumption. Other models that are more accurate require less traces to successfully attack a target, but also need a deep knowledge of the implementation that is attacked.

A commonly used methodology for side-channel analysis is the t-test leakage detection method initially proposed in [33, 22]. For the non-specific *fixed vs. random* t-test one takes two types of measurements, one with fixed input and one with random input. The t-statistic t is computed as

$$t = \frac{\mu_F - \mu_R}{\sqrt{\frac{\sigma_F^2}{n_F} + \frac{\sigma_R^2}{n_R}}}$$

where μ_F, σ_F^2, and n_F (resp. μ_R, σ_R^2, and n_R) denote the mean, variance, and number of measurements set with fixed input (resp. random input). If the value exceeds the threshold $|t| > 4.5$, the test has detected leakage. As this test does not perform an actual attack and does not consider a certain power model it is called *non-specific*. Apart from the *fixed vs. random* t-test it is also possible to perform a *semi-fixed vs. random* t-test. Such a test does not fix the input but some intermediate values, e.g. part of the state of a block cipher to get a more accurate result.

6 Countermeasures against Physical Attacks

In this section we discuss different approach to prevent physical attacks such as timing and side-channel analysis as well as fault-injection attacks. Note that there is not a single countermeasure that can be applied to fix all vulnerabilities, in practice usually a combination of countermeasures is applied.

6.1 Side-Channel Countermeasures

Hiding countermeasures are applied to raise the difficulty for an attacker to detect sensitive information in a set of power traces. This can be achieved by introducing additional noise or by trying to equalize the power consumption of all operations.

The first approach can be achieved by other computations that are executed in parallel or by shuffling the order of operations. For hardware implementations one can even instantiate

dedicated noise generators to randomize the power consumption. If shuffling is applied an attacker needs to perform an extra alignment step before analyzing the power traces. Otherwise the number of required power traces drastically increases.

The second approach is more suitable for hardware implementations as in microcontrollers the developer has only limited influence on the power consumption of an instruction and only one instruction can be executed in parallel (except the microcontroller features SIMD instructions).

The idea behind masking is to split a secret value into several shares. The secret value can only be reconstructed with the knowledge of all shares. The splitting of the secret value can be performed in a Boolean way or in an arithmetic way. Boolean masking means that the XOR-sum of all shares results in the secret value and arithmetic masking means that the arithmetic sum or difference of the shares results in the secret value. There are conversion approaches to switch between arithmetic and Boolean masking [23]. The major advantage of masking schemes is that they allow to prove the side-channel security of an algorithm. Nevertheless, there are still implementation challenges that have to be taken care of. Otherwise, a provably secure algorithm might still have a side-channel leakage. To achieve higher-order security, it is necessary to split the secret value into more shares.

6.2 Constant-time implementation

To prevent timing attacks and simple power analysis it is crucial to develop an implementation that has a constant (or at least secret-independent) execution time. Some pitfalls that should be avoided are:

- **Comparison of secret strings:** Such a comparison must not stop at the first unequal character.
- **Branches:** Branches must not be dependent on secret data. Ideally the same branches are taken for every run of the implementation.
- **Table look-ups:** On platforms with a cache, table look-ups can have varying access times. Thus the index must not depend on secret data for such platforms.
- **Compiler optimization:** A developer must take care that the compiler does not remove instructions that are critical for the security of the implementation but irrelevant for its functionality.

6.3 Countermeasures against fault attacks

The most intuitive way to detect a fault is to utilize redundant computations that are used to check the correctness of the result. Spatial redundancy is a possible countermeasure for hardware implementations and means the same operation is executed twice in parallel. This countermeasures has only a small performance overhead but the area consumption doubles. In contrast to that, temporal redundancy means executing another operation after the original operations has been finished. This can either be an additional decryption after an encryption operation to check whether the result matches the original plaintext or simply another encryption to compare both ciphertexts.

For fault attacks that must induce the fault at a specific point in time, it is also possible to randomize the order of the instructions to make an attack harder. Partial reconfiguration on

FPGAs can also be used to randomize the location of the circuit that compute the operation. For linear operations error correcting codes can be used to detect faults.

7 Physical security of quantum-secure schemes

In this section, we review the state-of-the-art of research targeting the physical security of post-quantum cryptography. We are not aware of any work related physical attacks on multivariate quadratics and thus we do refrain from discussing these schemes. Similarly, there has not been done much research on side-channel analysis of hash-based primitives yet. However the underlying hash functions have been analyzed thoroughly in works like [9, 78, 84].

7.1 Code-based Cryptography

The McEliece encryption scheme [59] has been proposed in 1978 and belongs to the family of code-based cryptography. Much effort has been spent on analyzing the side-channel security of this scheme and developing suitable countermeasures. Simple power analysis of the scheme has been performed for FPGA implementations [60] and microcontroller implementations [38, 82]. These works also show that the attack can be made much harder by providing a timing- and instruction-invariant implementation.

Further more Chen et al. attacked McEliece FPGA implementations with DPA [18, 20]. The authors analyze the syndrome computation and are able to recover the complete secret key with an additional algebraic step that exploits the relation between the public and private key. As counteremasures to the presented attacks, Chen et al. also propose a masking scheme for McEliece in [19] and evaluate the side-channel security of their masked FPGA implementation. Their masking scheme is a hybrid of Boolean and arithmetic masking. The masks are generated on-the-fly using a pseudo-random number generator.

7.2 Lattice-based Cryptography

The lattice-based ring learning with errors (R-LWE) encryption scheme [56] has also been analyzed for its resistance against side-channel attacks in several works mainly focusing on DPA. The first approach to secure R-LWE against DPA has been proposed by Reparaz et al. in 2015 [75, 74]. The authors attempt to protect the secret key by splitting it into two shares and perform all operations separately on both shares. However, the last step of the algorithm is a decoding function that is not a linear operation and thus requires the knowledge of both shares. To solve this problem [75] proposed a masked decoder. As this decoder has a number of drawbacks, like being non-deterministic and increasing the failure rate of the scheme, Reparaz et al. [73] proposed another approach in 2016. In [73] not the secret key, but the ciphertext is split into two shares. This approach introduces a heavy computational overhead as it requires another run of the decryption during the encryption. Oder et al. [64] combined the ideas of [75] and [73] to avoid the aforementioned problems and also applied a CCA2-conversion to R-LWE to make it secure against adaptive chosen-ciphertext attackers. Furthermore the masking scheme from [64] has a proof to support its claim. Additionally, [75, 73, 64] all provide results of practical measurements to demonstrate that the masking schemes indeed prevent a leakage. Oder et al. also discuss the fault sensitvity of R-LWE in [64].

Lattice-based signatures schemes, like BLISS [27] and GLP [35], have also been analyzed for their vulnerability to fault attacks in [10] and [30]. Both papers consider instruction-skipping resulting in potential loop aborts and how to exploit such a fault. The work of Bindel et al. [10] furthermore examines the impact of zeroing or randomization of critical values. The proposed countermeasures mainly boil down to redundant computations that are used for correctness checks. Another proposed countermeasure is to prevent instruction-skipping is to deliberately induce a segmentation fault by allocating new memory for every intermediate result.

Bruinderink et al. [13] also found a cache-timing attack on the signature scheme BLISS. More specifically, they attacked the Gaussian sampler that is used to generate noise polynomials in BLISS and are able to extract the secret key with only 3,500 signatures. To prevent timing attacks many implementations of lattice-based schemes provide a constant or secret-independent execution time, like vectorized implementations of the GLP signature scheme and the New Hope key exchange for Intel CPUs [2, 36]. Furthermore there are also microcontroller implementations of R-LWE that are protected against timing attacks [71, 64].

8 Conclusions

In this report we presented the WP1 contributions to the PQCRYPTO project. We presented our hardware and software implementations as well as side-channel attacks and countermeasures. We furthermore review the state-of-the-art of research of applied cryptography in the field of embedded security to put our results into context.

References

[1] Erdem Alkim, Nina Bindel, Johannes A. Buchmann, and Özgür Dagdelen. TESLA: tightly-secure efficient signatures from standard lattices. *IACR Cryptology ePrint Archive*, 2015:755, 2015.

[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] Dorian Amiet, Andreas Curiger, and Paul Zbinden. FPGA-based accelerator for post-quantum signature scheme SPHINCS-256. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):18–39, 2018.

[5] 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.

[6] Aydin Aysu and Patrick Schaumont. Precomputation methods for hash-based signatures on energy-harvesting platforms. *IEEE Trans. Computers*, 65(9):2925–2931, 2016.

[7] Daniel J Bernstein. Cache-timing attacks on aes, 2005.

[8] 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.

[9] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Building power analysis resistant implementations of keccak. In *Second SHA-3 candidate conference*, volume 142. Citeseer, 2010.

[10] Nina Bindel, Johannes Buchmann, and Juliane Krämer. Lattice-based signature schemes and their sensitivity to fault attacks. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2016 Workshop on*, pages 63–77. IEEE, 2016.

[11] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[12] 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.

[13] 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 *International Conference on Cryptographic Hardware and Embedded Systems*, pages 323–345. Springer, 2016.

[14] David Brumley and Dan Boneh. Remote timing attacks are practical. In *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003*. USENIX Association, 2003.

[15] 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.

[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] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *Information Security, 9th International Conference, ISC 2006, Samos Island, Greece, August 30 - September 2, 2006, Proceedings*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2006.

[18] Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. Differential power analysis of a mceliece cryptosystem. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*, volume 9092 of *Lecture Notes in Computer Science*, pages 538–556. Springer, 2015.

[19] Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. Masking large keys in hardware: A masked implementation of mceliece. In *International Conference on Selected Areas in Cryptography*, pages 293–309. Springer, 2015.

[20] Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. Horizontal and vertical side channel analysis of a mceliece cryptosystem. *IEEE Trans. Information Forensics and Security*, 11(6):1093–1105, 2016.

[21] 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.

[22] Jeremy Cooper, Elke Demulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, and Pankaj Rohatgi. Test Vector Leakage Assessment (TVLA) Methodology in Practice. International Cryptographic Module Conference, 2013.

[23] Jean-Sébastien Coron, Johann Großschädl, Mehdi Tibouchi, and Praveen Kumar Vadnala. Conversion from arithmetic to boolean masking with logarithmic complexity. In *International Workshop on Fast Software Encryption*, pages 130–149. Springer, 2015.

[24] 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/.

[25] 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.

[26] 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.

[27] 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.

[28] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient identity-based encryption over NTRU lattices. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 22–41. Springer, 2014.

[29] 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.

[30] Thomas Espitau, Pierre-Alain Fouque, BenoÃőt GÃľrard, and Mehdi Tibouchi. Loop-abort faults on lattice-based fiatâĂŞshamir and hash-and-sign signatures. Cryptology ePrint Archive, Report 2016/449, 2016. http://eprint.iacr.org/2016/449.

[31] Kris Gaj, Ekawat Homsirikamol, Marcin Rogawski, Rabia Shahid, and Malik Umar Sharif. Comprehensive evaluation of high-speed and medium-speed implementations of five SHA-3 finalists using xilinx and altera fpgas. *IACR Cryptology ePrint Archive*, 2012:368, 2012.

[32] 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.

[33] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. A testing methodology for side channel resistance validation. In *NIST non-invasive attack testing workshop*, 2011.

[34] 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.

[35] 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.

[36] Tim Güneysu, Tobias Oder, Thomas Pöppelmann, and Peter Schwabe. Software speed records for lattice-based signatures. 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 67–82. Springer, 2013.

[37] Frank K Gürkaynak, Kris Gaj, Beat Muheim, Ekawat Homsirikamol, Christoph Keller, Marcin Rogawski, Hubert Kaeslin, and Jens-Peter Kaps. Lessons learned from designing a 65 nm asic for third round sha-3 candidates. 2012.

[38] Stefan Heyse, Amir Moradi, and Christof Paar. Practical power analysis attacks on software implementations of mceliece. In *International Workshop on Post-Quantum Cryptography*, pages 108–125. Springer, 2010.

[39] 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.

[40] J. Hoffstein, J. Pipher, and J.H. Silverman. Public key cryptosystem method and apparatus, June 27 2000. US Patent 6,081,597.

[41] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe Buhler, editor, *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.

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

[43] 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.

[44] 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.

[45] 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/.

[46] Bernhard Jungk. Evaluation of compact fpga implementations for all sha-3 finalists. In *The Third SHA-3 Candidate Conference*, 2012.

[47] Abdel Alim Kamal and Amr M Youssef. An fpga implementation of the ntruencrypt cryptosystem. In *Microelectronics (ICM), 2009 International Conference on*, pages 209–212. IEEE, 2009.

[48] Jens-Peter Kaps, Panasayya Yalla, Kishore Kumar Surapathi, Bilal Habib, Susheel Vadlamudi, and Smriti Gurung. Lightweight implementations of sha-3 finalists on fpgas. In *The Third SHA-3 Candidate Conference*, 2012.

[49] Elif Bilge Kavun and Tolga Yalcin. On the suitability of sha-3 finalists for lightweight applications. 2012.

[50] 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.

[51] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011.

[52] Kashif Latif, M Muzaffar Rao, Arshad Aziz, and Athar Mahboob. Efficient hardware implementations and hardware performance evaluation of sha-3 finalists. 2012.

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

[54] Bingxin Liu and Huapeng Wu. Efficient architecture and implementation for ntruencrypt system. In *Circuits and Systems (MWSCAS), 2015 IEEE 58th International Midwest Symposium on*, pages 1–4. IEEE, 2015.

[55] 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.

[56] 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.

[57] 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.

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

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

[60] H. Gregor Molter, Marc Stöttinger, Abdulhadi Shoufan, and Falko Strenzke. A simple power analysis attack on a mceliece cryptoprocessor. *Journal of Cryptographic Engineering*, 1(1):29–36, 2011.

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

[62] Tobias Oder and Tim Güneysu. Implementing the newhope-simple key exchange on low-cost fpgas. *Progress in Cryptology–LATINCRYPT*, 2017, 2017.

[63] 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.

[64] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical cca2-secure and masked ring-lwe implementation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):142–174, 2018.

[65] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: Extended abstract. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 333–342, New York, NY, USA, 2009. ACM.

[66] Peter Pessl, Leon Groot Bruinderink, and Yuval Yarom. To BLISS-B or not to be: Attacking strongswan's implementation of post-quantum signatures. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1843–1855. ACM, 2017.

[67] 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.

[68] 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.

[69] 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.

[70] 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.

[71] 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.

[72] 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/.

[73] Oscar Reparaz, Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Additively homomorphic ring-lwe masking. In Takagi [79], pages 233–244.

[74] Oscar Reparaz, Sujoy Sinha Roy, Ruan de Clercq, Frederik Vercauteren, and Ingrid Verbauwhede. Masking ring-lwe. *J. Cryptographic Engineering*, 6(2):139–153, 2016.

[75] Oscar Reparaz, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. A masked ring-lwe implementation. 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 683–702. Springer, 2015.

[76] 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.

[77] 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.

[78] Mostafa Taha and Patrick Schaumont. Side-channel analysis of mac-keccak. In *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, pages 125–130. IEEE, 2013.

[79] 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*. Springer, 2016.

[80] Tobias Oder Tim Güneysu. Towards lightweight identity-based encryption for the post-quantum-secure internet of things. In *18th International Symposium on Quality Electronic Design (ISQED 2017)*. IEEE, 2017.

[81] 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.

[82] Ingo Von Maurich and Tim Güneysu. Towards side-channel resistant implementations of qc-mdpc mceliece encryption on constrained devices. *PQCrypto*, 2014:266–282, 2014.

[83] Ingo von Maurich, Lukas Heberle, and Tim Güneysu. IND-CCA secure hybrid encryption from QC-MDPC Niederreiter. In Takagi [79], pages 1–17.

[84] Michael Zohner, Michael Kasper, Marc Stöttinger, and Sorin A Huss. Side channel analysis of the sha-3 finalists. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1012–1017. EDA Consortium, 2012.