Post-Quantum Implementations

Matthias J. KANNWISCHER[1], **Ruben NIEDERHAGEN**[2,3], **Francisco RODRÍGUEZ-HENRÍQUEZ**[4], **Peter SCHWABE**[5,6]

[1] *Chelpis Quantum Corp., Taipei, Taiwan*
[2] *Academia Sinica, Taipei, Taiwan*
[3] *University of Southern Denmark, Odense, Denmark*
[4] *Technology Innovation Institute, Abu Dhabi, United Arab Emirates*
[5] *Max Planck Institute for Security and Privacy, Bochum, Germany*
[6] *Radboud University, Nijmegen, The Netherlands*

1.1. Introduction

In this chapter we give an overview of techniques for secure and efficient implementation of so-called post-quantum cryptography, the anticipated next generation of asymmetric cryptography. Today most cryptographic systems deployed in the real world use asymmetric primitives that rely on the hardness of factoring (most notably RSA public-key encryption and signatures), or the (elliptic-curve) discrete-logarithm problem. While those systems, with suitably chosen parameters, are believed to resist attacks by classical computers, it is known since Shor's seminal 1994 paper, that a large universal quantum computer will be able to solve both factoring and discrete logarithms in polynomial time.

Luckily, even if sufficiently large quantum computer become a reality, this does not mean the end of efficient public-key cryptography. There exist various approaches for constructing public-key encryption or key-encapsulation mechanisms (KEMs) and

Pre-submission version of Chapter 11 of *Embedded Cryptography 2*, ISBN: 9781789452143.
coordinated by Emmanuel Prouff, Guénaël Renault, Matthieu Rivain, and Colin O'Flynn.
© ISTE SCIENCE PUBLISHING Ltd 2025.

signatures that — as far as we know — resist attacks even by large universal quantum computers. There are five main approaches to construct such post-quantum asymmetric schemes:

– Problems related to finding short vectors in high-dimensional lattices can be used to construct both efficient key-encapsulation schemes and efficient signatures.

– Hard problems in coding theory are mostly used to construct efficient public-key encryption or key-encapsulation schemes.

– The problem of solving large systems of multivariate quadratic equations is mostly useful to construct signature schemes.

– Cryptographic signatures can be built just from cryptographic hash functions.

– Finally, there are schemes based on the hardness of finding high-degree isogenies between supersingular elliptic curves.

First schemes of these families reach as far back as the 1970s and the research field of post-quantum cryptography has explicitly been established in the early 2000s. However, research into post-quantum cryptography, in particular concrete instantiations of schemes and implementations, received a huge boost in early 2016, when the US-American National Institute of Standards and Technology (NIST) announced that they would launch an effort of evaluating and eventually standardizing such schemes. NIST issued a public call for proposals in late 2016 with a deadline for submissions in November 2017.

At the time of writing this chapter, in early 2023, NIST's post-quantum project has just reached a major milestone with the announcement of the first batch of algorithms NIST are planning to standardize: the lattice-based signature schemes CRYSTALS-Dilithium and Falcon, the hash-based signature scheme SPHINCS^{+}, and the lattice-based key-encapsulation mechanism CRYSTALS-Kyber. However, post-quantum schemes are still far less well understood than cryptographic schemes considered in the other chapters of this book. The NIST standardization effort continues with multiple KEMs forwarded to another round of evaluation and a renewed call for signature proposals with a submission deadline in June 2023. Consequently, the state of the art in scheme design, cryptanalysis, and techniques for secure implementation is still very actively researched and thus likely to change over the next years.

In order to provide a snapshot of the current state of the art, we take the following approach: we pick one example scheme for lattice-based, one for code-based, and one for isogeny-based key agreement and one example scheme for lattice-based signatures, for multivariate signatures, and for hash-based signatures. Where possible, we focus on schemes that are likely to be relevant for real-world deployments, i.e., schemes that have been selected by NIST for standardization, are still considered for standardization in the NIST competition, or are going to be submitted to the upcoming on-ramp for signatures. In our description of these schemes, we focus on the essentials of the construction and aspects that are particularly relevant for efficient and secure implementation. We omit details that are less relevant here, for example, data serialization or compression routines. We provide references to the full specifications at the end of this chapter.

In the sections explaining secure implementation techniques, we will typically start by discussing "constant-time" implementations of the respective primitive. We use this term to refer to (software) implementations that systematically avoid data flow from secret inputs into branch conditions, memory addresses, and variable-time arithmetic instructions. This together ensures protection against classical timing attacks, i.e., timing attacks against sequential execution of the program at a fixed clock frequency. Systematic protection against more advanced microarchitectural attacks that exploit, for example, transient execution or data-dependent dynamic frequency scaling is still mostly unresolved, not just for post-quantum cryptography.

1.2. Post-quantum encryption and key encapsulation

Most current systems are using some variant of the Diffie-Hellman (DH) protocol for key exchange and in some cases, with static keys, also for authentication. The most common instantiation is elliptic-curve Diffie-Hellman. We do not know any post-quantum scheme that we could use as a drop-in replacement for DH, at least not without massively impacting system performance. The abstract primitive that comes closest to DH are key-encapsulation mechanisms (KEMs). In many contexts, KEMs *can* be used to straight-forwardly replace DH; and they have been used as a very flexible building block in the construction of multiple post-quantum cryptographic protocols.

Consequently, most proposals aiming at post-quantum confidentiality build a KEM; for scenarios where a public-key encryption (PKE) scheme is needed, we can build such a PKE from a KEM efficiently and generically. Most post-quantum KEMs are constructed by first building a passively-secure public-key encryption scheme and then using a generic transform on top to obtain a KEM that is secure against active, i.e., chosen-ciphertext, attackers. We thus first consider the construction of passively secure lattice-based PKEs (§1.2.1), code-based PKEs (§1.2.2), and isogeny-based PKEs (§1.2.3); and then look into transforms to actively secure KEMs in §1.2.4.

1.2.1. *Lattice-based KEMs – Kyber*

Most lattice-based public-key encryption schemes and key-encapsulation mechanisms are based on variants of the Learning-with-Errors (LWE) or Learning-with-Rounding (LWR) problems. The scheme that we are considering as an example in this section is the NIST PQC finalist Kyber. As outlined above, in this section we focus on a simplified version of the passively secure PKE underlying Kyber.

Kyber is relying on the hardness of the Module-Learning-with-Errors (MLWE) problem. Consider the ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$, where $\mathbb{Z}_q$ is the ring of integers modulo q and n is a positive integer; in the case of Kyber we have $q = 3329$ and $n = 256$. Consider further a positive integer k; in Kyber we have $k \in \{2, 3, 4\}$, depending on the security level. The search version of the MLWE problem is to find $\mathbf{s} \in \mathcal{R}_q^k$, given samples $(\mathbf{A}, \mathbf{As} + \mathbf{e})$, where $\mathbf{A} \in \mathcal{R}_q^{k\times k}$ is sampled uniformly at random and $\mathbf{e} \in \mathcal{R}_q^k$ is sampled from a so-called *noise distribution* ψ. The decisional version of the problem is to distinguish such samples from values $(\mathbf{U}, \mathbf{v})$ sampled uniformly at random from $\mathcal{R}_q^{k\times k} \times \mathcal{R}_q^k$. In the original definition of these problems, also $\mathbf{s}$ was assumed to be sampled uniformly at random, but later results showed that the problem remains hard if $\mathbf{s}$ is sampled from other distributions, most importantly, if $\mathbf{s}$ is sampled from the noise distribution ψ.

The noise distribution outputs "small" values in $\mathbb{Z}_q$, centered around zero. For example, in Kyber the noise distribution is generating polynomial coefficients in $\mathbb{Z}_q$ from the set $\{-\eta, \ldots, \eta\}$. More concretely each coefficient c is obtained from a centered binomial distribution as $c = \sum_{i=1}^{\eta}(a_i - b_i)$ for bits a_i, b_i that are the output of a PRF, i.e., that are assumed to be indistinguishable from uniformly random bits. Most parameter sets of Kyber use the parameter $\eta = 2$.

1.2.1.1. *Scheme definition*

Let us take a closer look at a simplified version of the public-key encryption scheme underlying Kyber. We will omit details about encoding and decoding of keys and the ciphertext. Note however, that the complete picture involves lossy compression of the ciphertext.

Key generation of the PKE underlying Kyber consists of the following steps:

1) Sample a public 32-byte string ρ uniformly at random.

2) Sample a secret 32-byte string σ uniformly at random.

3) Expand ρ through an extendable output function (XOF) and run rejection sampling on the output to obtain a matrix $\mathbf{A} \in \mathcal{R}_q^{k\times k}$ that "looks uniformly random".

4) Use the output of a PRF with key σ to obtain $\mathbf{s} \in \mathcal{R}_q^k$ and $\mathbf{e} \in \mathcal{R}_q^k$ with coefficients from the noise distribution (see above).

5) Compute $\mathbf{t} = \mathbf{As} + \mathbf{e}$.

6) Return secret key $\mathsf{sk} = \mathbf{s}$ and public key $(\mathbf{t}, \rho)$.

Encryption takes as input a public key $\mathsf{pk} = (\mathbf{t}, \rho)$, a 32-byte message m, and a 32-byte string r of random coins. It then proceeds as follows:

1) Expand ρ to obtain the same matrix $\mathbf{A}$ that was generated in key generation.

2) Use the output of a PRF with key r to obtain $\mathbf{r} \in \mathcal{R}_q^k$, $\mathbf{e}_1 \in \mathcal{R}_q^k$, and $e_2 \in \mathcal{R}_q$ with coefficients from the noise distribution (see above).

3) Compute $\mathbf{u} = \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$.

4) Compute $v = \mathbf{t}^T\mathbf{r} + e_2 + \mathsf{MapToPoly}(m)$.

5) Return ciphertext $c = (\mathbf{u}, v)$.

In this encryption routine, the MapToPoly function maps each bit of the message m to one coefficient of a polynomial in $\mathcal{R}_q$. A zero bit is mapped to 0; a one bit is mapped to $\lceil q/2 \rfloor$.

Decryption takes as input a ciphertext $c = (\mathbf{u}, v)$ and a secret key $\mathbf{s}$ and recovers a message m' as $m' = \mathsf{MapFromPoly}(v - \mathbf{s}^T\mathbf{u})$. Here, the MapFromPoly function maps each coefficient of a polynomial in $\mathcal{R}_q$ to a single bit; a full polynomial is thus mapped to a string of 256 bits or 32 bytes. A coefficient in $\mathbb{Z}_q$ is mapped to 1, iff it is in $\{\lceil q/4 \rceil, \ldots, \lfloor 3q/4 \rfloor\}$, otherwise it is mapped to 0.

With very high probability, decryption will succeed and recover the original message, i.e., $m' = m$. Let us understand why this is the case: During encryption we add $\mathbf{t}^T\mathbf{r} + e_2$ to $\mathsf{MapToPoly}(m)$ to obtain v. In decryption we subtract $\mathbf{s}^T\mathbf{u}$ from v and use MapFromPoly to obtain m'. The difference of the two terms is

$$
\begin{aligned}
&\mathbf{t}^T\mathbf{r} - \mathbf{s}^T\mathbf{u} \\
=&(\mathbf{A}\mathbf{s} + \mathbf{e})^T\mathbf{r} + e_2 - \mathbf{s}^T(\mathbf{A}^T\mathbf{r} + \mathbf{e}_1) \\
=&\mathbf{s}^T\mathbf{A}^T\mathbf{r} + \mathbf{e}^T\mathbf{r} + e_2 - \mathbf{s}^T\mathbf{A}^T\mathbf{r} + \mathbf{s}^T\mathbf{e}_1 \\
=&\mathbf{e}^T\mathbf{r} + e_2 + \mathbf{s}^T\mathbf{e}_1
\end{aligned}
$$

As all the coefficients in $\mathbf{s}$, $\mathbf{r}$, $\mathbf{e}$, $\mathbf{e}_1$, and e_2 are small and centered around zero, also the coefficients of the difference are expected to be relatively small. This means that MapFromPoly in decryption receives as input a noisy version of the output of MapToPoly in encryption; as long as this noise has all coefficients smaller than $\lfloor q/4 \rfloor$, decryption will succeed.

1.2.1.2. *Techniques for efficient implementation*

The core operation of Kyber as well as for most other lattice-based KEMs is polynomial multiplication. Specifically, Kyber requires to multiply polynomials in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$. Kyber's ring is chosen in a way that allows fast polynomial multiplication using the number-theoretic transform (NTT). The idea of NTT-based polynomial multiplication is to transform the inputs into a different domain (called "NTT domain" or "frequency domain") in which multiplication is cheap (i.e., has linear complexity). The result is then transformed back to "normal domain" (or "time domain"). To obtain the product ab with $a, b \in \mathcal{R}_q$, we compute

$$ab = \mathtt{NTT}^{-1}\left(\mathtt{NTT}(a) \circ \mathtt{NTT}(b)\right)$$

where $\texttt{NTT}$ and $\texttt{NTT}^{-1}$ are the transformations to and from NTT domain, and $\circ$ is the multiplication in NTT domain. These transformations are particularly fast for power-of-two n. At the core of any (radix-2) NTT is the following ring isomorphism:

$$\mathbb{Z}_q[x]/(X^{2n} - c^2) \cong \mathbb{Z}_q[X]/(X^n - c) \times \mathbb{Z}_q[x]/(X^n + c);$$
$$\sum_{i=0}^{2n-1} f_i X^i \leftrightarrow \left(\sum_{i=0}^{n-1} (f_i + c f_{n+i}) X^i, \sum_{i=0}^{n-1} (f_i - c f_{n+i}) X^i \right).$$

It allows to split a polynomial $f \in \mathbb{Z}_q[x]/(X^{2n} - c^2)$ into two half-sized polynomials modulo $X^n + c$ and modulo $X^n - c$. The inverse can be computed by applying the Chinese remainder theorem on the two elements in $\mathbb{Z}_q[X]/(X^n - c)$ and $\mathbb{Z}_q[x]/(X^n + c)$ to obtain the element in $\mathbb{Z}_q[x]/(X^{2n} - c^2)$. In the case of $\mathcal{R}_q$, this only works if a c exists, such that $1 \equiv -c^2 \mod q$, i.e., we require a 4-th primitive root of unity. Note that both the splitting and re-combining only requires to work on two coefficients of the inputs at a time which is commonly referred to as a butterfly operation. Both the forward and the inverse transformations require $n/2$ multiplications by c (or $c^{-1}/2$), $n/2$ additions, and $n/2$ subtractions. Note that since straightforward polynomial multiplications requires n^2 multiplications, the multiplication of half-sized polynomials is about 4 times faster than multiplication of full-sized polynomials. Hence, the splitting does result in a net speed-up for larger n. For smaller n (e.g., $n = 2$), the additions and subtractions outweigh the performance improvement.

This trick can be applied recursively to further split the resulting polynomials into halves. However, it requires the existence of the corresponding roots of unity. The splitting is usually referred to as an NTT layer. For a k-layer NTT starting from $\mathcal{R}_q$, one requires a 2^{k+1}-th primitive root of unity. In case one splits all the way down to degree-0 polynomials, it is called a "complete" NTT, otherwise it is called an "incomplete" NTT. In case of a complete NTT, $\circ$ becomes a coefficient-wise (or point-wise) multiplication of the inputs. In case of an incomplete NTT, $\circ$ multiplies 2^k polynomials with $n/2^k$ coefficients. $\circ$ is usually referred to as base multiplication in this case. For the Kyber $\mathcal{R}_q$, a complete NTT would require a 512-th primitive root of unity which does not exist modulo $q = 3329$. Hence, Kyber uses an incomplete 7-layer NTT.

Due to the use of NTTs, Kyber is particularly suitable for vectorized implementations (e.g., using AVX2 or Arm Neon). In each NTT layer, each coefficient is used in exactly one butterfly with one other coefficient. By loading multiple coefficients into one vector register, one can compute multiple butterflies in parallel. This works straightforwardly while polynomials have more coefficients than fit into one register, i.e., in the first NTT layers. Once polynomials are split into polynomials of smaller size, one has to shuffle the registers, such that the coefficients involved in one butterfly are in separate registers.

For implementing the modular coefficient multiplications within the NTT, a commonly used technique is Montgomery multiplication. Montgomery's reduction approach is to replace expensive division by odd q with a cheap division by a power of two $R = 2^l$. Assume we have computed the product $c = ab$ of two coefficients. When reducing a value c ($< qR$) which coincidentally is a multiple of $R > q$, we can simply store c/R which reduces the size of c by l bits and is cheaply computed. Montgomery's idea is to make sure that c is a multiple of R by introducing a correction step, i.e., we want to find a value t, such that, $c - tq$ is divisible by R. Montgomery computes t as $cq^{-1} \mod R$, such that, $c - aq^{-1}q \mod R = 0$. Note that the result of the reduction is then $abR^{-1} \mod q$. To obtain the correct result, one has two multiply by R again (or by $R^2 \mod q$ when using Montgomery multiplication). If one of the multiplicands is a constant, one may pre-compute $aR \mod q$. The result of the Montgomery multiplication is then $abRR^{-1} \equiv ab \mod q$. This is commonly used in the NTT.

It is important to note that while polynomial arithmetic accounts for a large fraction of the Kyber run-time, another building block is often even more dominating: hashing based on the Keccak permutation (including both SHA-3 and SHAKE). A hardware-accelerated Keccak implementation or, alternatively, a heavily-optimized Keccak software implementation vastly benefits Kyber implementations. Due to the module-structure, Kyber implementations can make good use of vectorized Keccak implementations running multiple permutations simultaneously (e.g., 4 for AVX2, or 2 for Arm Neon).

1.2.1.3. *Techniques for secure implementation*

Kyber is very easy to implement following the constant-time paradigm. In fact, the passively-secure PKE underlying Kyber is designed in such a way that even a straight-forward implementation that simply follows the specification will be free of secret-dependent branches and memory addresses, as long as reductions modulo q do not employ any conditional branches.

There are some common sub-routines in other lattice-based PKE schemes and KEMs that require a bit more care for constant-time implementations. Most notably, some schemes use noise from a fixed-weight distribution, for example, polynomials with a fixed number of coefficients that are 1, a fixed number of coefficients that are -1, and all other entries equal to zero. The standard way to sample a random polynomial satisfying this property is to start with a fixed polynomial with the prescribed number of 1 and -1 coefficients and then randomly shuffle the coefficients. Performing this random shuffling in constant time is not straight-forward, standard algorithms like Fisher–Yates shuffle are not constant time. Secure implementations typically use permutation networks like the Benes network or sorting networks like Batcher sort. Furthermore, some schemes involve multiplication of a random polynomial by a polynomial $\mathbf{s}$ with coefficients in $\{-1, 0, 1\}$. It is tempting to implement such a multiplication through a sequence of conditional additions and subtractions, but straight-forward implementations of this approach will leak information about $\mathbf{s}$.

To protect against side-channel attacks beyond timing attacks, implementations of lattice-based KEMs typically use arithmetic masking, i.e., sharing each secret element s of $\mathbb{Z}_q$ in d shares $s_0, \dots, s_{d-1}$, such that $s = \sum_{i=0}^{d-1} s_i \mod q$. The core arithmetic operations are, from a masking perspective, linear and thus very cheap to mask. Clearly this is the case for addition of noise polynomials. It also holds for matrix-vector multiplications of the form $\mathbf{A}\mathbf{s}$, because $\mathbf{A}$ is public and therefore unmasked.

What makes masking of lattice-based KEMs costly is switching between boolean masking used inside the PRF and arithmetic masking used for arithmetic on polynomials over $\mathbb{Z}_q$, and operations in the CCA transform (see §1.2.4).

Another generic side-channel countermeasure that has been implemented to protect lattice-based PKE and KEMs is shuffling, i.e., executing operations in a randomized order. This is fairly easy and efficiently to do for polynomial addition, subtraction, compression, and decompression, by processing coefficients in a randomized order. Also the NTT can be efficiently shuffled by randomizing the order of butterflies in each layer.

1.2.2. *Code-based KEMs – Classic McEliece*

Code-based cryptography is almost as old as RSA: It was introduced by McEliece in 1978. The basic idea of code-based public key encryption (PKE) schemes is to use a secret code $\mathcal{G}$ with a code length n and a code rank k that can correct up to t errors as private key and its garbled generator matrix $G \in \mathbb{F}_2^{k \times n}$ as public key. The sender encodes a message into a binary vector $v \in \mathbb{F}_2^k$ and encrypts it to the ciphertext $c \in \mathbb{F}_2^n$ by computing $c = vG + e$ using an error vector $e \in \mathbb{F}_2^n$ of weight t. The receiver uses the secret code $\mathcal{G}$ to obtain $vG = c - e$ by correcting the t bit errors and decodes vG back to v.

A dual-variant to the McEliece cryptosystem was proposed by Niederreiter in 1986. Instead of a generator matrix G, he uses a parity-check matrix $H \in \mathbb{F}_2^{(n-k) \times n}$ as public key. The sender then encodes the message as error vector $e \in \mathbb{F}_2^n$ and encrypts it to a ciphertext $c \in \mathbb{F}_2^{n-k}$ as syndrome $c = He$. As in the McEliece cryptosystem, the receiver uses the secret code $\mathcal{G}$ to find the error positions and hence recovers the message. The dual-variant by Niederreiter is equivalent to the McEliece cryptosystem in terms of security.

A common tweak to reduce the size of the public key in the McEliece and Niederreiter cryptosystems is to compute the systematic form of H, i.e., apply row operations to transform H into $(I_{n-k}|T)$ where I_{n-k} is an identity matrix of $n - k$ rows and columns. Thus, only T needs to be communicated and stored and the front identity matrix can be regenerated when needed. (This, however, requires a semantically secure conversion for the McEliece cryptosystem, which nowadays is state-of-the-art, to ensure that the first $n-k$ bits of the plaintext are not revealed.)

The choice of the code family is crucial for the security of code-based cryptosystems. McEliece proposed to use binary Goppa codes, which is still considered secure. Binary Goppa codes are defined by a support $\alpha \in \mathbb{F}_{2^m}^n$ for a small $m \in \mathbb{N}$ and a Goppa polynomial g of degree t. Niederreiter proposed to use a different code family in his dual-variant, which later was broken. However, also Niederreiter's dual-variant remains secure when using, e.g., binary Goppa codes. Recent proposals for code-based systems are using either McEliece's or Niederreiter's variant and, e.g, quasi-cyclic codes or rank metric to further reduce the size of the public key matrices.

1.2.2.1. *Scheme definition*

Classic McEliece is the only code-based third-round finalist in the NIST standardization process. While honoring the founder of code-based cryptography in its name, Classic McEliece is using the Niederreiter variant with binary Goppa codes in its construction. Its parameter sets aiming at NIST security levels 1, 3, and 5 are using $m \in \{12, 13\}$, $n \in \{3488, 4608, 6688, 6960, 8192\}$, $k \in \{2720, 3360, 5024, 5283, 6528\}$, and $t \in \{64, 96, 119, 128\}$.

The secret key is a random seed δ, an irreducible Goppa polynomial g of degree t, the support $\alpha \in \mathbb{F}_{2^m}^n$, and a random bit string s. The public key is computed by generating the $t \times n$ matrix $\tilde{H} = \{h_{i,j}\}$, where $h_{i,j} = \alpha_j^{i-1}/g(\alpha_j)$ for $i = 1, \dots, t$ and $j = 1, \dots, n$. Then $\tilde{H}$ is converted to the $(n-k) \times n$ matrix $\hat{H}$ by expanding each $\mathbb{F}_{2^m}$ entry of $\tilde{H}$ into a column of m bits. Finally, $\hat{H}$ is reduced to systematic form $(I_{n-k}|T)$ and T is used as public key.

For encapsulation, a random vector $e \in \mathbb{F}_2^n$ of weight t is encoded to $C_0 = (I_{n-k}|T)e \in \mathbb{F}_2^{n-k}$. The error vector e is hashed to obtain C_1 to obtain the ciphertext $C = (C_0, C_1)$. Finally, C is hashed together with e to obtain the session key K.

For decapsulation, the ciphertext C is split into C_0 and C_1 and C_0 is decoded to e' using the private key. If the weight of e' is t and if $C_0 = (I_{n-k}|T)e$, C_1' is computed by hashing e'. The check of $C_0 = (I_{n-k}|T)e$ is required to achieve CCA security. If $C_1' = C_1$, then $e' = e$ and the session key is computed by hashing e and C. Otherwise, decoding has failed and a false session key is computed by hashing s and C.

1.2.2.2. *Techniques for efficient implementation*

The most time consuming operation in the key generation of Classic McEliece is the "systemization" of $\hat{H}$, i.e., the computation of the systematic form of $\hat{H}$, which usually is done using Gaussian elimination. With a relatively high probability, $\hat{H}$ does not have a systematic form. Therefore, Classic McEliece specifies two variants of the key-generation algorithm: For the systematic variant, the systemization is aborted in case of failure and a new private key is computed. For the semi-systematic variant, an alternative algorithm is specified that allows to swap columns (in constant-time) such that a systematic public-key can be obtained without repeating key generation.

Efficient implementations of the systematic variant split $\hat{H}$ into a $(n-k)\times(n-k)$ matrix M and a $(n-k)\times k$ matrix $\hat{T}$ such that $\hat{H} = (M|\hat{T})$ and $(I_{n-k}|T) = M^{-1}\hat{H}$ and first operate on M to check if $\hat{H}$ has full rank. During this computation, an LUP-decomposition of M with $PM = LU$ is computed such that T can be efficiently obtained as $T = (U^{-1}L^{-1}P)\hat{T}$ if $\hat{H}$ has full rank. This reduces the cost of repeatedly computing on a $(n-k)\times n$ matrix to only computing on a $(n-k)\times(n-k)$ matrix plus a final matrix multiplication once a public key with systematic form has been found.

There are several decoding algorithms for binary Goppa codes. Here we introduce the use of the Berlekamp-Massey algorithm as implemented in the Classic McEliece reference implantation. Due to the error-correction capabilities of this decoding algorithm, first a double-size $2t \times n$ parity-check matrix $\tilde{H}^{(2)} = \{h^{(2)}_{i,j}\}$, where $h^{(2)}_{i,j} = \alpha_j^{i-1}/g^2(\alpha_j)$ for $i = 1,\ldots,2t$ and $j = 1,\ldots,n$ and a double syndrome $s^{(2)} = \tilde{H}^{(2)}(C_0|0)$ is computed. Then the Berlekamp-Massey algorithm is used to compute an error-locator polynomial σ from $s^{(2)}$. The error positions are then determined by evaluating σ at $\alpha = \{\alpha_1,\ldots,\alpha_n\}$. Instead of re-encrypting the obtained error vector e' using H, the double syndrome $s'^{(2)}$ of the error vector can be computed as $s'^{(2)} = \tilde{H}^{(2)}e'$ and compared with $s^{(2)}$: If $s^{(2)} = s'^{(2)}$ then also $C_0 = (I_{n-k}|T)e'$ and hence $e = e'$. Therefore, the large public key is not required for re-encryption during decapsulation and hence the public key does not need to be stored alongside the private key.

For computing $\hat{H}$ during key generation as well as for computing $\tilde{H}^{(2)}$ and for obtaining the error positions from the error-locator polynomial during decapsulation, a polynomial needs to be evaluated for all $\alpha = \{\alpha_1,\ldots,\alpha_n\}$. Since α contains most of the elements in $\mathbb{F}_{2^m}$ for most parameter sets of Classic McEliece, this can efficiently be accomplished using the additive FFT algorithm by Gao and Mateer. The syndrome computation can efficiently performed using a transposed FFT.

Classic McEliece requires efficient sorting algorithms for the computation of a random permutation of field elements in $\mathbb{F}_{2^m}$ for obtaining α as well as for computing a random error vector of weight t. Furthermore, the specification of Classic McEliece requires that the support $\alpha = \{\alpha_1,\ldots,\alpha_n\}$ is not stored as a list of elements in $\mathbb{F}_2$ but as $(2m-1)2^{m-4}$ control bits for a Beneš network, which as well can be obtained using an efficient sorting algorithm. Arithmetic in $\mathbb{F}_{2^m}$ typically can be parallelized and hence is suitable for bitslicing.

1.2.2.3. *Techniques for secure implementation*

The specification of Classic McEliece is designed to support constant-time implementations as much as possible. The rationale for storing the control bits of a Beneš network as part of the secret key is that on the one hand the storage requirements for the private key are significantly reduced and that on the other hand the correct values for α can be obtained after the evaluation of a polynomial using an additive FFT (which operates on the natural order of field elements) securely in constant time by applying the Beneš network with the stored control bits to the output of the additive FFT. The computation of a (transposed) additive FFT, the Berlekamp-Massey algorithm, as well as sorting algorithms lend themselves for a straight-forward constant-time implementation.

Special care needs to be taken for the encoding routine: Both the generation of a weight-t error vector e as well as the computation of $C_0 = He$ might leak timing information. A constant-time implementation of the decoding operation can be achieved quite easily, as long as a transposed FFT is used for the re-encryption based on the double syndrome. Otherwise, caution is required when multiplying the parity-check matrix with e'.

During key generation, the Gaussian elimination on $\hat{H}$ for computing the systematic parity-check matrix $(I_{n-k}|T)$ must be performed in constant time. Here, an early abort in case $\hat{H}$ cannot be systemized does not leak timing information since then key generation is re-started with a fresh seed. The irreducible Goppa polynomial can be computed using Gaussian elimination on a matrix over $\mathbb{F}_{2^m}$ or using the Berlekamp-Massey algorithm, which must be implemented in constant time as well.

There is not much work on exploiting other side channels than timing for code-based systems. A general recommendation is to mask the use of the private key. There are message-recovery attacks against (Classic) McEliece, e.g., single-trace attacks on encapsulation and differential attacks on decapsulation that exploit power side channels. More research on effective side-channel attacks and countermeasures for such attacks is required.

1.2.3. *Isogeny-based KEMs*

Isogeny-based cryptography was proposed by Couveignes in his “Hard Homogeneous Spaces” manuscript, whose material was first presented during a seminar held in the mid-nineties of the last century. One decade after, Rostovtsev and Stolbunov independently proposed a public-key cryptosystem based on isogenies of ordinary elliptic curves. That same year, Charles, Lauter and Goren, proposed a hash function whose collision resistance was provably guaranteed by the isogeny problem defined over supersingular elliptic curves.

Let E and E' be two supersingular isogenous elliptic curves defined over a finite field $\mathbb{F}_q$, with q a power of a large prime p. Computing an isogeny map between E and E' is widely believed to be a hard computational problem in the classical an the quantum settings. It is known as the Supersingular Isogeny Path problem. In many scenarios, this isogeny is of known degree ℓ^e for some small prime ℓ and we refer to this variant as the Supersingular Fixed-Degree Isogeny Path (*SIPFD*) problem. An isogeny graph is a graph that represents the relationships between different elliptic curves that are related to each other by isogenies. The vertices of the graph represent elliptic curves which are determined by their j-invariant, which classify elliptic curves in equivalence classes defined up to endomorphisms. The edges of the graph represent isogenies between the curves.

Variants of the *SIPFD* problem form the basis of several isogeny-based signatures, including: the Short Quaternion and Isogeny Signature (SQISign), which was proposed in 2020 by De Feo, Kohel, Leroux, Petit and Wesolowski. This problem also provides the security guarantees of the Commutative Supersingular Isogeny-based Diffie-Hellman (CSIDH) key exchange scheme, which was introduced in 2018 by Castryck, Lange, Martindale, Panny, and Renes.

In 2011, Jao and De Feo proposed the Supersingular Isogeny-based Diffie-Hellman key exchange protocol (SIDH). Apart from disclosing the degree of its secret isogeny, SIDH also reveals the evaluation of its secret isogenies at a large torsion subgroup. This weaker variant of *SIPFD* was dubbed by the authors as the *Computational Supersingular Isogeny* (CSSI) problem. SIKE, which is a variant of SIDH equipped with a key encapsulation mechanism, was one of the few schemes that made it to the fourth round of the NIST PQC standardization effort as an alternate KEM candidate.

For over a decade, the best-known algorithms for breaking SIDH or SIKE had an exponential time complexity in both, classical and quantum settings. However, a recent polynomial-time attack by Castryck and Decru that was quickly followed by two other variants, ingeniously exploited the auxiliary information leaked in SIDH and SIKE to completely break their security, and in the process, easily breaking and claiming a bounty for an isogeny challenge proposed by Microsoft Research.

1.2.3.1. *Schemes definitions*

SIKE was the solely isogeny-based fourth-round scheme in the NIST standardization process. The official SIKE proposal presented four parameter sets, namely, SIKEp434, SIKEp503, SIKEp610 and SIKEp751, achieving NIST security levels 1, 2, 3 and 5, respectively. The notation used for these instantiations alludes to the bitlength of the underlying prime field characteristic.

SIKE operates on Montgomery supersingular elliptic curves defined over $\mathbb{F}_{p^2}$, where p is a large prime number of the form $p = 2^{e_2}3^{e_3} - 1$. The exponents e_2

and e_3 are chosen such that $2^{e_2} \approx 3^{e_3}$. The public parameters of SIKE are given by a supersingular starting curve $E_0/\mathbb{F}_{p^2} : y^2 = x^3 + 6x^2 + x$, and a set of six x-coordinates corresponding to the basis points $P_2, Q_2, P_3, Q_3 \in E_0$ of order 2^{e_2} and 3^{e_3}, respectively. Additionally, the protocol also uses the auxiliary public points $D_2 = P_2 - Q_2$ and $D_3 = P_3 - Q_3$. SIKE comprises three main algorithms: Key generation, Encapsulation and Decapsulation (which are usually called KeyGen, Encaps and Decaps).

In KeyGen, Bob's public and private key $\{pk_3, (s, sk_3)\})$ are computed by performing the following steps. Bob's private key sk_3, is a uniformly chosen integer in the range $[\![1 \mathrel{..} 2^{e_3-1} - 1]\!]$. Then, Bob computes the 3^{e_3}-isogeny ϕ_3 generated by the kernel point $R_3 = P_3 + [sk_3]Q_3$, obtaining his public key as the tuple of image points, $pk_3 = (\phi_3(P_2), \phi_3(Q_2), \phi_3(D_2))$, along with a randomly selected n-bit string s.

SIKE's Encapsulation algorithm computes Alice's public and private key (pk_2, sk_2) using an analogous procedure as the one described above. Alice then uses a key derivation procedure to compute a secret j, which corresponds to the j-invariant of the image supersingular elliptic curve $\phi_3(\phi_2(E_0))$. This algorithm outputs Alice's public key along with the encryption of an n-bit random message m XOR'ed with $F(j)$, where F is a hash function that maps j to bitstrings.

SIKE's Decapsulation algorithm recovers the secret j' corresponding to the j-invariant of the image supersingular elliptic curve $\phi_2(\phi_3(E_0))$. The value j' is then used for finding Alice's private key and from it her public key. If the received public key is identical to the one recovered, this procedure returns as the shared secret the hash of the secret key and the encapsulation output. Otherwise, it returns a random string.

The main computation of CSIDH consists of the evaluation of its class group action, which takes as input an elliptic curve E_0, represented by its A-coefficient, and an ideal class $\mathfrak{a} = \prod_{i=1}^{n} \mathfrak{l}_i^{e_i}$, represented by its list of exponents $(e_i, \ldots, e_n) \in [\![-m \mathrel{..} m]\!]^n$. This list of exponents is the CSIDH secret key. The output of the class group action is the A-coefficient of the elliptic curve E_A defined as,

$$E_A = a{*}E_0 = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_0.$$

One remarkable feature of the CSIDH group action is its commutative property. This allows one to apply the group action directly to the key exchange between two parties by mimicking the Diffie-Hellman protocol. Having agreed upon using a starting elliptic curve E_0, Alice and Bob choose a secret key a and b, respectively. Then they can produce their corresponding public keys by computing the group actions $E_A = a * E_0$ and $E_B = b * E_0$. After exchanging these public keys, Alice and Bob can obtain a common secret by computing,

$$a * E_B = (a \cdot b) * E_0 = (b \cdot a) * E_0 = b * E_A.$$

1.2.3.2. *Techniques for efficient implementation*

The two most expensive computational tasks of SIKE are the computation of large smooth-degree isogenies of supersingular elliptic curves along with the evaluation of the image of elliptic curve points in those isogenies and; elliptic curve scalar multiplication computations via three-point Montgomery ladder procedures. In their 2014 SIDH paper, De Feo, Jao, and Plüt presented optimal computation of large degree isogenies. The cost of optimal strategies for computing a degree-ℓ^e isogeny is of about $\frac{e}{2}\log_2 e$ point multiplications by ℓ, $\frac{e}{2}\log_2 e$ degree-ℓ isogeny evaluations, and e constructions of degree-ℓ isogenous curves. Optimal strategies have also been applied to achieve faster implementations of CSIDH.

Montgomery curves is the preferred curve model for SIKE and CSIDH, since they enable the usage of x-only point arithmetic. This can be advantageously used for performing scalar multiplications with x-coordinates that always lie in the base prime field $\mathbb{F}_p$. The Montgomery model is also useful for computing close to optimal isogeny evaluations.

For practical implementations of CSIDH, constructing and evaluating n degree-ℓ_i isogenies, plus $O(n^2)$ scalar multiplications by the prime factors ℓ_i, overwhelmingly dominate its computational cost. CSIDH uses a prime p such that $p+1 = 4\prod_{i=1}^{n}\ell_i$, where $\ell_1, \ldots, \ell_n$ are small odd primes. This choice permits to compute efficiently the degree-ℓ_i isogenies corresponding to the group action of the ideal $\mathfrak{l}_i$ of norm ℓ_i.

Vélu's formulas have been profusely used in the last fifty years for constructing and evaluating degree-ℓ isogenies by performing three main algorithms known as `KPS`, `xISOG` and `xEVAL`. `KPS` computes the first ℓ multiples of the kernel point P, namely, the set $\{P, [2]P, \ldots, [\ell]P = \infty\}$. The calculations done in `KPS` are then used as precomputation database for `xISOG` and `xEVAL`. `xISOG` and `xEVAL` find the constants $A' \in \mathbb{F}_p$ that determine the codomain curve E', and the x-coordinate $\phi_x(\alpha)$ of the image point Q, respectively. The computational cost associated to Vélu's formulas is of $O(\ell)$ operations. In 2020, Bernstein, De Feo, Leroux, and Smith presented a breakthrough approach for constructing and evaluating degree-ℓ isogenies at a combined cost of just $O(\sqrt{\ell})$ operations. This approach has been referred as square-root Vélu.

1.2.3.3. *Techniques for secure implementation*

In July 2022, Castryck and Decru presented a devastating attack against SIKE. This attack can heuristically solve the CSSI problem in polynomial time, as it was convincingly shown by an accompanying Magma script, which breaks in a matter of hours, random SIKE instances that were once aimed for achieving NIST levels 1, 2, 3, and 5. This original attack was soon improved to break all SIKE instances in a matter of minutes.

The Castryck and Decru attack relies on the knowledge of three crucial pieces of information, namely, (i) The degree of the isogeny ϕ_2; (ii) the ring endomorphism of

SIKE's starting curve E_0; and (iii) the images $\phi_3(P_2), \phi_3(Q_2)$ of Alice's generator points $\langle P_2, Q_2 \rangle = E[2^{e_2}]$, where the prime $p = 2^{e_2}3^{e_3} - 1$ is the underlying prime used by SIKE instantiations. We stress here that (ii) and (iii) are only known in the specific case of the CSSI problem, but not in the more general case of the *SIPFD* problem. Furthermore, another attack by Maino and Martindale and yet another one by Robert quickly followed. Maino and Martindale's attack does not require knowledge of the endomorphism ring associated to the base curve. Robert's attack can also break SIDH in polynomial time for any random initial supersingular elliptic curve E_0.

Since the publication of the Castryck and Decru attack, several countermeasures have already been proposed by trying to hide the degree of the isogeny or the images of the torsion points. These solutions are considerably less efficient and practical than the original construction.

On the other hand, CSIDH does not leak any auxiliary information and, for a sensitive choice of parameters, remains secure against all known classical and quantum attacks. Despite its low performance, CSIDH has the smallest public keys of all post-quantum key exchange schemes and is the only one that is commutative. Furthermore, CSIDH admits an efficient public key validation procedure, a feature that permits its usage in the static-dynamic and static-static key exchange settings.

The security guarantees of CSIDH (and its variant CTIDH) rest on an analogue of the discrete logarithm problem: given the base elliptic curve E_0 and the public-key elliptic curve $E_A = \mathfrak{a} * E_0$, deduce the ideal class $\mathfrak{a}$. Hence, the classical security of CSIDH lies in the problem of finding an isogeny path from the isogenous supersingular elliptic curves E_0 and E_A. From a quantum setting perspective, the best-known attack is Kuperberg's procedure, which has a quantum time and space sub-exponential complexity of $\exp\left(O(\sqrt{\log p})\right)$.

1.2.4. *IND-CCA2 Security*

Most post-quantum key-encapsulation mechanisms have in common that the core constructions only achieve chosen-plaintext attack (CPA) security, i.e., they are only secure in the presence of a passive adversary that cannot perform chosen-ciphertext attacks (CCA). While in some cases CPA security may be sufficient (e.g., when secret keys are only used once), many protocols do require CCA security, i.e., assuming an active adversary that can manipulate ciphertexts. As it is preferable to standardize and deploy as few cryptographic systems as possible we commonly focus on CCA-secure schemes. We distinguish between IND-CCA1 and IND-CCA2 security. For the former, the adversary can query arbitrary ciphertexts from a decryption oracle only before receiving the ciphertext he wants to attack, while for IND-CCA2 the attacker can perform oracle queries before and after receiving the target ciphertext. In practice, we want to achieve IND-CCA2 security.

Fortunately, generic transformations from CPA security to IND-CCA2 security exist, such as the Fujisaki–Okamoto transform for probabilistic PKEs, which is used for example in Kyber, and Dent's variant for deterministic cases, which is used for example in Classic McEliece (see decapsulation in §1.2.2.1).

1.2.4.1. *The Fujisaki–Okamoto transform*

The general idea of CCA transforms is simple: We have to ensure that a given ciphertext was honestly generated before revealing the decryption. The Fujisaki–Okamoto achieves this by re-encrypting the decrypted plaintext and then comparing the re-encryption to the ciphertext. In case the re-encryption matches the ciphertext, the ciphertext was generated honestly. If there is a mismatch this indicates that the input ciphertext was faulty and possibly malicious. In that case, the plaintext must not be revealed, but instead either a random value (implicit rejection) or an error message (explicit rejection) needs to be returned.

For allowing the re-encryption to produce the same ciphertext, the encryption procedure requires to be fully deterministic, which is achieved by generating all required randomness pseudo-randomly from the (random) message. In its simplest form, the FO transform constructs an IND-CCA2-secure KEM from a CPA public-key encryption system consisting of the algorithms `CPA.KeyGen()`, `CPA.Encrypt(pk, m, coins)`, and `CPA.Decrypt(sk, c)` (with `coins` corresponding to the required randomness) using a hash-function $\mathcal{H}$ in the following way:

```
– CCA.KeyGen():
    - pk,sk ← CPA.KeyGen()
– CCA.Encaps(pk):
    - x ← $\{0,1\}^{256}$
    - k, coins ← $\mathcal{H}$(x)
    - c ← CPA.Encrypt(pk, x, coins)
– CCA.Decaps(sk, c):
    - x' ← CPA.Decrypt(sk, c)
    - k, coins' ← $\mathcal{H}$(x')
    - c' ← CPA.Encrypt(pk, x', coins')
    - verify that c' = c
```

Variants of the FO transform (as for example used by Kyber) additionally apply two tweaks to this construction: Firstly, the shared secret is derived from the so-called pre-key `k` by hashing it together with the ciphertext `c`. Secondly, the public key gets added as input to the hash deriving `coins` for protecting against multi-target attacks and turning the construction into a contributory KEM.

1.2.4.2. *Secure implementation*

Implementing the FO transform securely is an astonishingly hard problem that has not been entirely solved in the literature so far. For passive side-channel attacks this is primarily caused by the much larger attack surface. The processed plaintext has to be considered secret requiring the entire FO re-encryption to be protected against side-channel attacks. Furthermore, it often sufficient for an adversary to distinguish if two ciphertexts decrypt to the same plaintext or not to mount so-called plaintext-checking oracle (PC oracle) attacks. PC-oracle attacks have been demonstrated for code-based KEMs exploiting tiny timing differences in the re-encryption depending on the plaintext. While these timing-based PC-oracle attacks can be thwarted by ensuring that no timing leakage occurs in re-encryption revealing information about the processed plaintext, protection is much harder when considering more powerful side-channel attacks. When using masking as a countermeasure against power-consumption-based side-channel attacks, this primarily requires to use a much higher masking order than commonly used on other cryptography. For example, while cryptography without the FO transformation may be sufficiently secure at masking order 2 or 3, post-quantum schemes require much higher masking order with some literature showing that even 5th order masking is insufficient.

The FO transform can also be attacked using fault injection attacks. In the most-straightforward attack, the attacker simply skips the comparison of the ciphertext and the re-encryption, hence completely disabling the purpose of the FO. Another attack avenue is to transmit a faulty ciphertext and to use fault injection to fault the ciphertext back to the original ciphertext causing the FO to accept the faulty ciphertext in case it decrypt to the same message, i.e., providing a PC oracle to the attacker. Countermeasures against fault attacks on the FO are thus far limited to generic countermeasures like fault detection using redundant computation or shuffling computations to increase the attack complexity. The efficacy of such countermeasures is not yet well understood.

1.3. Post-quantum signatures

Just like for KEMs, also post-quantum signature schemes are constructed for a variety of underlying assumptions. The first batch of signatures that will be standardized by NIST includes two lattice-based schemes and one hash-based scheme. In addition, schemes based on the hardness of solving large multivariate systems of equations were prominently represented in the first 3 rounds of the NIST competition, and we expect that more schemes following this approach will be submitted to the upcoming NIST call for additional post-quantum signatures. Also multiple signature schemes based on isogenies of elliptic curves (see §1.2.3) have been proposed in the last couple of years and are likely to enter the NIST competition for additional signatures.

Unlike KEMs, post-quantum signatures are diverse also from a different point of view: While all CCA-secure KEMs follow the approach of first constructing a passively secure PKE and then using some generic transform to construct a CCA-secure KEM, signature schemes are built using very different “templates”: Dilithium is using Fiat-Shamir, Falcon is using a hash-and-sign approach, hash-based signatures like

SPHINCS$^+$ are typically constructed using Merkle trees, yet other schemes (like the round-3 alternate scheme Picnic) use multiparty-computation in the head.

1.3.1. *Lattice-based signatures – Dilithium*

Lattice-based signatures are divided into two major families either following the hash-and-sign paradigm or the Fiat–Shamir-with-aborts approach. Hash-and-sign schemes are represented in the NIST competition by Falcon and date back to 2003 when Hoffstein, Howgrave-Graham, Pipher, Silverman, and Whyte introduced NTRUSign. Fiat–Shamir-with-aborts signatures date back to work by Lyubashevsky in 2009. The corresponding finalist in the NIST competition is Dilithium. In the following we focus on the Fiat–Shamir-with-aborts signatures and Dilithium.

Dilithium relies on the hardness of the MLWE problem (as introduced in §1.2.1) and the Module-short-integer-solutions (MSIS) problem. The MSIS problem is to find a "short" $\mathbf{v} \in \mathcal{R}_q$, given a matrix $\mathbf{A} \in \mathcal{R}_q^{k\times\ell}$, such that $\mathbf{A}\mathbf{v} \equiv 0 \mod q$. In particular, Dilithium requires a variant of MSIS called SelfTargetMSIS which asks to find a vector $\begin{bmatrix}\mathbf{z} & c & \mathbf{v}\end{bmatrix}^{\mathsf{T}}$ with small coefficients and a hash of a message μ, such that

$$H\left(\mu || [\mathbf{A}|\mathbf{t}|\mathbf{I}] \cdot \begin{bmatrix}\mathbf{z}\\ c\\ \mathbf{v}\end{bmatrix}\right) = c$$

with H a random oracle, $\mathbf{A} \in \mathcal{R}_q^{k\times\ell}$ and $\mathbf{t} \in \mathcal{R}_q^k$ uniformly random, and $\mathbf{I}$ the identity matrix.

1.3.1.1. *Scheme definition*

Dilithium is one of the two lattice-based signature finalists in the NIST standardization process. It consists of three parameter sets aiming at the security levels 2, 3, and 5. Across all parameter sets, the same polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ with $n = 256$ and $q = 2^{23} - 2^{13} + 1$ is used, allowing efficient polynomial multiplication using NTTs. The core differences for each security level are the dimensions of the matrix $(k \times \ell)$, the range of the masking vector (γ_1), and the range of the secret key (η). The parameters $(k, \ell, \gamma_1, \eta)$ are $(4, 4, 2^{17}, 2)$ for security level 2, $(6, 5, 2^{19}, 4)$ for security level 3, and $(8, 7, 2^{19}, 2)$ for security level 5. The remaining parameters $(d, \tau, \gamma_2, \omega)$ are chosen accordingly.

We describe a simplified form of Dilithium without public key compression and deterministic signing. The public key is $(\rho, \mathbf{t})$ consisting of the seed ρ used to pseudorandomly sample the uniform matrix $\mathbf{A} \in \mathcal{R}_q^{k\times\ell}$ and $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$. The private key is $(\rho, \mathbf{t}, \mathbf{s}_1, \mathbf{s}_2)$ consisting of the public key, and the two secret vectors $\mathbf{s}_1$ and $\mathbf{s}_2$.

To sign a hash M of a message, a uniformly random masking vector $\mathbf{y} \in \mathcal{R}_q^\ell$ with coefficients in $[-\gamma_1, \gamma_1]$ is sampled. The signer then computes $\mathbf{w} = \mathbf{A}\mathbf{y}$ and computes the high bits of $\mathbf{w}$ (referred to as $\mathbf{w}_1$), such that $\mathbf{w} = 2\gamma_2\mathbf{w}_1 + \mathbf{w}_0$. The challenge polynomial c is derived from the hash $H(M||\mathbf{w}_1)$. Here $H()$ is a special hash function that outputs a "short" polynomial which has a small number of coefficients (τ) set to -1 or 1. A potential signature is computed as $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$. Outputting this signature directly would potentially leak information about the secret $\mathbf{s}_1$ and, hence, rejection sampling has to be used. If a signature is rejected, the signer starts with a fresh masking vector $\mathbf{y}$ and tries again. The signer rejects $\mathbf{z}$ checks if any coefficient of $\mathbf{z}$ is larger than $\gamma_1 - \tau\eta$. Additionally, the signer rejects if the lower bits of $\mathbf{A}\mathbf{y} - c\mathbf{s}_2$ are larger than $\gamma_2 - \tau\eta$ as this would result in a carry from the lower bits into the higher bits of $\mathbf{w}$ during verification and, consequently, an invalid signature. The signature then consists of $(\mathbf{z}, c)$.

To check a signature $(\mathbf{z}, c)$, the verifier re-computes the high bits of $\mathbf{w}$ as $\mathbf{w}_1' = \mathbf{A}\mathbf{z} - c\mathbf{t}$. They can then check that $c = H(M||\mathbf{w}_1')$. One also needs to verify that $\mathbf{z}$ does not contain any coefficients larger than $\gamma_1 - \tau\eta$.

There are two main differences between the simplified version of Dilithium above compared to the specification submitted to NIST. A major difference is that Dilithium uses compressed public keys, i.e., only the high bits $\mathbf{t}_1$ of $\mathbf{t} = 2^d\mathbf{t}_1 + \mathbf{t}_0$ are stored in the public key. While this reduces public key size, it also slightly complicates the scheme as the verifier is no longer able to reliably compute $\mathbf{w}_1$ from $\mathbf{z}$, c, and $\mathbf{t}_1$. This is due to the carries introduced by $c\mathbf{t}_0$ from the lower part to the higher part of $\mathbf{w}_1$. This is resolved by adding a hint vector $\mathbf{h}$ containing the position of said carries to the signature. Another difference is that Dilithium is deterministic rather than the probabilistic scheme presented above. Instead of sampling $\mathbf{y}$ at random, Dilithium derives it pseudo-randomly from a seed K (part of the secret key), the hash of the message M, a hash of the public key tr (precomputed as a part of the secret key), and a counter that is incremented in case of rejections. This results in a deterministic signature scheme which is needed by some of the security proofs and is also preferable in case no good randomness source is available.

1.3.1.2. *Techniques for efficient implementation*

Similar to Kyber, the core arithmetic operation is polynomial multiplication and $\mathcal{R}_q$ is chosen, such that fast NTTs can be used. Dilithium uses a modulus q for which 512-th roots of unity exist and, hence, a complete NTT is used. The implementation techniques for NTTs described in §1.2.1 carry over to Dilithium. The core difference to the Kyber NTTs is the much larger q. As the modulus is 23 bits, the natural size of coefficients in software implementations is 32 bits.

The large modulus, however, presents a challenge for some architectures as a long 32×32-bit constant-time multiplier is not always available. For example, the Arm Cortex-M3 only has a variable-time long-multiply instruction which cannot be safely used for computing 32-bit NTTs on secret inputs. Note that not all NTTs in Dilithium are operating on secret inputs. For example, all NTTs in verifications as well as the

NTT of c and $\mathbf{t}_0$ may be performed using variable-time instructions. Another challenge is that, for example, AVX2 does not have a high-multiply for 32-bit inputs, but only for 16-bit inputs. However, such an instruction is needed for fast Montgomery multiplications. This is results in a much slower implementation as one has to fall back to widening multiplications.

1.3.1.3. *Techniques for secure implementation*

For Dilithium implementations protected from timing side-channel attacks one has to slightly deviate from the standard definition of constant-time implementations. Usually, a constant-time implementation is defined as not having any data flow from secret data into branching conditions, memory addresses, or into instructions that have data-dependent timing. However, Dilithium does have data flow from the secret key to the signature which is used in a branching condition in the rejection sampling. However, since the (candidate) signature is public according to the security proof, this does not present any problem. When using standard tools for finding timing leaks in Dilithium implementations, this branch will likely be found as a false positive.

Protecting Dilithium from other side-channel attacks proves to be significantly harder than for lattice-based encryption. The polynomial arithmetic itself is easily masked using arithmetic masking. However, the remainder of the scheme is much harder to mask. In particular, the sampling of $\mathbf{y}$ and the rejection sampling are best implemented using Boolean masking requiring a conversion between arithmetic and Boolean masking. As of today, there is no fully masked implementation of Dilithium available. The available masked implementations either implement the predecessor GLP or modify Dilithium to use a power-of-two modulus allowing for more efficient masking.

Differential fault attacks present a serious threat to Dilithium. When using the deterministic variant of Dilithium an adversary can inject a fault into the computation of c. The faulty c' is then used to compute a faulty signature $\mathbf{z}' = \mathbf{y} + c'\mathbf{s}_1$. By subtracting the faulty signature from a valid signature $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$, it is easy to compute $\mathbf{s}_1$. To prevent this attack, one should use the randomized variant of Dilithium whenever fault attacks present a viable attack vector. For protecting against weak randomness, Dilithium derives the masking vector from a secret seed K, the hash of the message M, the hash of the public key tr, and a random value ρ'. If ρ' is weak, the resulting scheme is still secure.

1.3.2. *Multivariate-quadratic-based signatures – UOV*

Multivariate cryptography is based on the NP-hardness of solving a system of multivariate polynomials. The public key is a multivariate polynomial system $\mathcal{P} : \mathbb{F}^n \mapsto \mathbb{F}^m$ over some finite field $\mathbb{F}$, while the private key is a trapdoor secret that allows the owner of the key to invert the multivariate system and to compute $\mathcal{P}^{-1}$. For a multivariate encryption scheme, the ciphertext c is obtained by evaluating the system for the plain text m as $c = \mathcal{P}(m)$. In order to decrypt, the receiver computes $m = \mathcal{P}^{-1}(c)$ using the trapdoor secret. A signature scheme is constructed correspondingly by computing the signature s as $s = \mathcal{P}^{-1}(m)$. This signature can be verified by evaluating the public map $m' = \mathcal{P}(s)$ and by verifying that $m = m'$.

The first multivariate-quadratic (MQ) signature scheme called C* was proposed by Matsumoto and Imai in 1988. However, C* was later found to insecure by Patarin in 1995. Shortly after, in 1996, Patarin introduced an improved cryptosystem following the idea of Matsumoto and Imai called Hidden Field Equations (HFE). While HFE in general remains unbroken up until today, all efficient instantiation have been shown to be insecure. Besides the HFE proposals there have been a number of proposals based on the “Oil and Vinegar” scheme proposed by Patarin in 1997. Soon after, in 1998 the original OV was shown to be insecure by Kipnis and Shamir. However, another year later, in 1999, Kipnis, Patarin, and Goubin proposed an improved cryptosystem called “Unbalanced oil and vinegar” (UOV). In the third round of the NIST PQC competition, there were two schemes based on MQ: the finalist Rainbow and the alternate GeMSS. Rainbow is an extension of UOV, while GeMSS is based on HFE. In addition, there was a third MQ-based scheme in the second round of the NIST PQC competition called MQDSS neither based on UOV nor HFE. Unfortunately, all three have suffered severe attacks in the process of the NIST PQC competition. We restrict this chapter to the original UOV which remains unbroken up until today.

1.3.2.1. *Scheme definition*

The core construction of all MQ schemes is similar: Let $\mathbb{F}_q$ be a finite field. A common choice is $\mathbb{F}_{2^k}$, e.g., $\mathbb{F}_{16}$ or $\mathbb{F}_{256}$. A MQ scheme has a public map $\mathcal{P} = \mathcal{S} \circ \mathcal{Q} \circ \mathcal{T} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ where $\mathcal{S} : \mathbb{F}_q^m \to \mathbb{F}_q^m$ and $\mathcal{T} : \mathbb{F}_q^n \to \mathbb{F}_q^n$ are randomly chosen affine and easy to invert maps, i.e., $\mathcal{S} : w \mapsto x = M_S w + v_S$ and $\mathcal{T} : y \mapsto z = M_T y + v_T$. The central map $\mathcal{Q} : x \mapsto y$, however, is quadratic and easy to invert. For UOV, the map $\mathcal{S}$ may be omitted, and hence, $\mathcal{P} = \mathcal{Q} \circ \mathcal{T}$. This is, however, not the case for more advanced schemes schemes like Rainbow and HFE.

Key generation consists of choosing $\mathcal{S}$, $\mathcal{Q}$, and $\mathcal{T}$, and computing $\mathcal{P}$. To sign (a hash of) a message $w \in \mathbb{F}_q^m$, one applies the inverse of $\mathcal{S}$, $\mathcal{Q}$, and $\mathcal{T}$ to obtain the signature $z \in \mathbb{F}_q^n$. To verify a signature $z \in \mathbb{F}_q^n$, one checks if (the hash of) the message w' is equal to $\mathcal{P}(z)$. For an MQ scheme to be secure $\mathcal{P}$ must be hard to invert without the knowledge of $\mathcal{S}$, $\mathcal{Q}$, or $\mathcal{T}$.

The security of any MQ scheme stems from the construction of the central map $\mathcal{Q}$. For UOV, $\mathcal{Q}$ consists of m quadratic equations of the form

$$y_k = \sum_{i=1}^{v} \sum_{j=1}^{v} \alpha_{ij}^{(k)} x_i x_j + \sum_{i=1}^{v} \sum_{j=v+1}^{n} \beta_{ij}^{(k)} x_i x_j + \sum_{i=1}^{n} \gamma_i^{(k)} x_i + \delta^{(k)}$$

with v being the number of vinegar variables (x_i for $1 \leq i \leq v$) and n being g the total number of variables. It is easy to see that the $o = n - v$ oil variables are not combined with other oil variables in the quadratic terms. Without a loss of security, the linear and constant terms can be left out, i.e., $\gamma_i^{(k)} = 0$ and $\delta^{(k)} = 0$ To compute the inverse of $\mathcal{Q}$, one choses the vinegar variables at random which turns the quadratic equations into a linear system of equations which can be solved for the oil variables (x_i for $v < i \leq n$). In case no solution exists one restarts with another set of randomly chosen vinegar variables.

A common optimization that does not affect the security is to restrict the map $\mathcal{T}$ to linear maps, i.e., $v_T = 0$. Another optimization is to chose $\mathcal{T}$ to have a special structure

$$\mathcal{T} = \begin{bmatrix} I_{v\times v} T^{(1)}_{v\times o} \\ 0_{o\times v} I_{o\times o} \end{bmatrix}.$$

This optimization does not impact the security of the schemes as for any public key $\mathcal{P}$ there exists a $\mathcal{T}$ and $\mathcal{Q}$ and finding any pair $\mathcal{T}$, $\mathcal{Q}$, such that $\mathcal{P} = \mathcal{Q} \circ \mathcal{T}$ breaks the security of the scheme.

The public key consists of the m polynomials in $\mathcal{P}(z)$. Each polynomial has $\frac{n\cdot(n+1)}{2}$ coefficients corresponding to $z_i z_j$ for $1 \leq i \leq j \leq n$. For efficient implementation, we represent these polynomials as a Macauley matrix with m rows and $\frac{n\cdot(n+1)}{2}$ columns. The columns are ordered, such that they correspond to the monomials $z_i z_j$ in lexicographic order. The matrix is stored in a column-major form.

1.3.2.2. *Techniques for efficient implementation*

Throughout key generation, signing, and verification, one needs efficient $\mathbb{F}_q$ arithmetic. We restrict the following to the most common case where $q = 2^k$. For other choices (e.g., the popular choice $q = 31$), other techniques are used. While addition is trivial for $\mathbb{F}_{2^k}$, multiplication and inversion require more attention. Multiplication is much more dominant than inversion. It benefits well from vectorization as one always operated on many field elements in parallel. If available, specialized instructions for $\mathbb{F}_{2^k}$ should be used. For example, Intel's Galois Field New Instructions (GFNI) and Arm's binary polynomial multiplication instructions can be beneficial with the former being much more powerful than the latter. Alternatively, in-register table lookups (e.g., Arm Neon's VTBL or Intel AVX2's VPSHUFB) should be used if supported by the platform. On platforms without either of the above, bitslicing is the preferred approach for implementing $\mathbb{F}_{2^k}$ multiplication. On platforms without a cache, one may want to consider using table look-ups from memory.

During signing, the most time-consuming operation is the inversion of the central map $\mathcal{Q}$ which requires to solve a linear system of equation. This is achieved using constant-time Gaussian elimination. It proceeds as normal Gauss elimination, except when making sure that the pivot element is non-zero, one has to conditionally add all the subsequent rows instead of just one. In case the system of linear equations does not have a solution, one may abort early as one has to start with a fresh set of vinegar variables.

For verification, the only operation besides hashing of the message, is the evaluation of the public map. This consists of the computation of all monomials $z_i z_j$ for $1 \leq i \leq j \leq n$ followed by a multiplication of the product with the corresponding column in the Macauley matrix. The products are then accumulated. Since multiplication is much more expensive than accumulation, it is beneficial to minimize the number of field multiplications. This can be done by using one accumulator for each possible value of $z_i z_j \neq 0$, i.e., 15 accumulators for $\mathbb{F}_{16}$. For $\mathbb{F}_{256}$, one can use two sets of accumulators for the lower and higher nibble of $z_i z_j$ separately, i.e., one requires 2×15 accumulators. Note that this results in a signature-dependent memory access pattern, which can only be used if the signature is considered public or memory access is constant-time.

1.3.2.3. *Techniques for secure implementation*

Work on protecting UOV or Rainbow from other side-channel attacks other than timing attacks is very limited. For Rainbow, the straightforward target for DPA during the signing operation is to attack the $\mathcal{T}^{-1}$ computation which processes the secret key $\mathcal{T}^{-1}$ and the known input w. In UOV, the knowledge of $\mathcal{T}^{-1}$ is sufficient to forge signatures. In Rainbow, one additionally needs to recover $\mathcal{S}^{-1}$ which can either be done using algebraic attacks or by another side-channel attacks. The side-channel attacks is possible since the signature z is known and $\mathcal{S}^{-1}$ has a special sparse structure allowing for a full recovery using CPA. Protection against CPA is not sufficiently studied yet and up until today no masked implementations of either UOV or Rainbow are available.

1.3.3. *Hash-based signatures – XMSS and SPHINCS$^+$*

Hash-based signature schemes go back to the work by Lamport in 1979 on one-time signature schemes constructed from a one-way function. The basic idea is to use a secret bit string as private key and its hash value as public key. The most basic construction is a one-time one-bit signature scheme: Alice generates two secret strings s_0 and s_1 and publishes their hash values $h_0 = \mathrm{h}(s_0)$ and $h_1 = \mathrm{h}(s_1)$ as public verification key. Since Alice is using a cryptographically secure hash function $\mathrm{h}()$, nobody can obtain s_0 and s_1 from h_0 and h_1. If later Alice wants to sign the one-bit message m, she releases the secret string s_m alongside m. Anyone that earlier obtained the public key (h_0, h_1) from Alice can now verify that $h_m = \mathrm{h}(s_m)$. Since previously only Alice knew s_0 and s_1, the one-bit message m must be from Alice. However, since Alice released part of the secret information, she cannot use this private-public key pair again. Hence, this simple scheme is a one-bit one-time signature scheme.

To construct signature schemes for longer messages, more hash values can be published as public key: To sign a 256-bit message, we could generate 2×256 hash values, using one pair to sign each bit of a 256-bit message. More efficiently, hash chains can be used instead: Winternitz proposed to split a message of n bit into $l_0 = n/w$ words of w bits and to concatenate the message with its check sum computed over the message words, also split into l_1 words of w bits. The Winternitz one-time signature scheme (WOTS) then uses $l = l_0 + l_1$ hash chains, each of length 2^w steps. To generate a public key, Alice starts out with l secret bit strings $r_{i,0}$ with $0 \leq i < l$, she hashes these strings 2^w times such that $r_{i,j+1} = \mathrm{h}(r_{i,j})$ for $0 < i \leq 2^w$, and publishes $r_{i,2^w}$ as public key. To sign, she takes the ith w-bit word of the message for $0 \leq i < l$, treats the w-bit word as an integer M_i and recomputes the M_ith chain step r_{i,M_i} of the ith hash chain. The corresponding elements of all chains are then the signature. For verification, Bob also for all $0 \leq i < l$ takes the ith w-bit word of the message and treats it as an integer M_i. He then hashes the signature element r_{i,M_i} another $2^w - M_i$ times and verifies that he reaches the public key element $r_{i,2^w}$. If that is the case for all l w-bit words of the message, the signature is successfully verified. This gives us a fairly efficient arbitrary-length one-time signature scheme.

To get from arbitrary-length one-time signature schemes to arbitrary-length many-time signature schemes, we can use binary Merkle trees as proposed by Merkle in 1989. The leave nodes of the Merkle tree are (a hash of) the public keys of one-time signature schemes. The leave nodes are pairwise hashed over several layers of a binary tree until a single root node is computed. The value of this root node is the public key. For generating a signature in a Merkle tree, a verification path needs to be provided, i.e., the pair-nodes on the layers of the Merkle tree that are required to reach the root node from a given leave node such that for verification, the verifier can re-compute the root node from a leave node and compare it to the public key. Using many arbitrary-length one-time signature schemes as leave nodes of a binary Merkle hash-tree, we obtain a arbitrary-length many-time signature scheme. Observe that the number of possible signatures per signature key pair is limited by the height of the Merkle tree and hence needs to be decided and fixed at key-generation time.

1.3.3.1. *Scheme definition*

Common hash-based signature schemes are the stateful signature schemes XMSS and LMS as well as the stateless signature scheme SPHINCS$^+$. In stateful signature schemes, information needs to be stored at the signers side which one-time signature schemes at the leaves of the Merkle tree already have been used; in stateless signature scheme, such a state does not need to be maintained.

Both XMSS and LMS work as described above: They are using variants of WOTS at the leaves of a binary Merkle tree. There are multi-tree versions of XMSS and LMS were a leave node on one tree is used to sign a root node of another tree, which provides a trade-off between computation time (key generation and signing) as well as the signature length.

As mentioned above, SPHINCS$^+$ is a stateless signature scheme. This is achieved by on the one hand using many layers in a multi-tree setting and on the other hand few-time signature schemes at the highest tree level such that the probability is negligible that a randomly selected leave is used more often than the few-time signature scheme tolerates.

Both XMSS and SPHINCS$^+$ are using masks as well as keyed hash-functions to ward of multi-target attacks and provide strong security proofs while the construction of LMS is slightly simpler and cheaper at the cost of reduced security guarantees.

1.3.3.2. *Techniques for efficient implementation*

The main cost factor of hash-based signature schemes is the computation of a hash function with small inputs and small outputs many times. Hence, any optimization that is applied to the hash function directly translates into a speed-up of key-generation, signing, and verification. Common techniques for accelerating hash functions are to provide hardware accelerators or instruction set extensions. Since many of the hash operations are independent (e.g., hashing in the different WOTS chains), vectorization and SIMD parallelization can be used.

Depending on the hash function and the parameter set, some prefixes in the hash inputs may be repeated frequently during the hash computations. If these common prefixes fit into one hash block, the resulting intermediate state can be stored and restored instead of repeating corresponding hash computations.

For signing of stateful hash-based signature schemes, the Merkle trees are traversed in order. Hence, some of the computations and some parts of the signature can be shared between consecutive signing operations. Such information can be cached and elaborate tree-traversal algorithms can be used to balance computational cost and storage requirements for efficient signing operations. Such approaches typically are less helpful for stateless signature schemes since the Merkle trees here are not traversed in-order but the starting leaves are selected randomly between consecutive signature operations.

1.3.3.3. *Techniques for secure implementation*

The attack surface for side-channel attacks on hash-based signature schemes is small. The majority of the computation occurs in the Merkle trees which only consist of public data and is, hence, not relevant for passive side-channel attacks. The only place where secret data is processed is the hash chains of the WOTS signatures and, for SPHINCS$^+$, the few-time signatures. However, these are only processing the secret key and no other variable data known to the attacker. Hence, differential attacks (e.g., CPA) are not possible. The only viable attack type for these schemes appears to be single-trace attacks. Another target is the generation of the secret keys. Those are expanded from a seed together with the address of the corresponding secret key in the tree using a pseudo-random function. As the seed is global for the entire (hyper-)tree, it is called many times with different addresses within a single signature generation. As the address is known to the attacker, it allows attacking the seed in multiple invocations of the pseudo-random function in a differential attack. Hence, the pseudo-random function needs to be protected against side-channel attacks.

Fault attacks present a much larger threat for hash-based signatures, in particular if they are stateless. The multi-tree structure of SPHINCS+ works and is secure despite the use of one-time signatures at the leaves of the inner trees, because the entire structure is deterministically defined: Whenever the same inner leave node is visited during a signing operation, then the same three node of the next higher tree is being signed — hence, the same chain nodes of the one-time signature scheme are computed and released as part of the signature. If, however, a fault is introduced randomly at any time during the signing process (e.g., during the pair-wise hashing in one of the Merkle trees), then with high probability, the root node of the corresponding tree is going to be different than in a non-faulted signature generation. Therefore, a different root node is signed by the next lower one-time signature scheme. Signing different message with the same one-time signature scheme breaks its security. Hence, if an attacker can inject different faults during the computation of a specific sub-tree, an alternative private key for the following one-time signature scheme can be derived, allowing the attacker to sign arbitrary messages. A signature that has been faulted as described above still verifies correctly (as long as the lowest tree is not affected by the fault). Therefore, such a faulty signature cannot simply be detected by verification. This puts SPHINCS+ (and other hash-based signature schemes using a multi-tree structure such as XMSS-MT) into significant risk in scenarios were an attacker can inject faults during signature generation. For stateful schemes this can be prevented by caching intermediate signatures, which is usually done to improve performance anyway.

Notes and Further References

§1.2.1 Lattice-based KEMs – Kyber: Lattice-based public-key encryption and key encapsulation goes back to the NTRU scheme by Hoffstein, Pipher, and Silverman Hoffstein *et al*. (1998). The public-key encryption scheme used in Kyber was introduced by Lyubashevsky, Peikert, and Regev in Lyubashevsky *et al*. (2010, 2013) and goes back to the work by Regev in Regev (2005, 2009). Kyber was introduced in Bos *et al*. (2018); the latest version of the specification is in Schwabe *et al*. (2022). It also draws many ideas, in particular with regards to a design enabling efficient and secure implementations from the NewHope scheme from Alkım, Ducas, Pöppelmann, and Schwabe Alkim *et al*. (2016). The other two lattice-based KEM finalist schemes in the NIST PQC project were NTRU Chen *et al*. (2020) and Saber D'Anvers *et al*. (2020).

Over the last decade, many works improved the performance of multiplication in polynomial rings that are used by lattice-based encryption schemes and KEMs. Some of these works employ Karatsuba or Toom techniques Bernstein *et al*. (2017); Karmakar *et al*. (2018); Kannwischer *et al*. (2019), but most recent works focus on NTT-based multiplication in NTT-friendly rings Alkim *et al*. (2020); Becker *et al*. (2021); Abdulrahman *et al*. (2022) and also in NTT-unfriendly rings Alkim *et al*. (2021); Chung *et al*. (2021). Another recent direction of work on implementations of lattice-based crypto is formally verifying correctness of arithmetic Hwang *et al.* (2022) or full schemes Almeida *et al.* (2023).

Various papers present masked implementations of lattice-based public-key encryption schemes and KEMs Oder *et al.* (2018); Bos *et al.* (2021); Heinz *et al.* (2022); Kamucheka *et al.* (2022); Beirendonck *et al.* (2021); Kundu *et al.* (2022) and building blocks D'Anvers *et al.* (2023). Combined countermeasures against side-channel and fault analysis are presented in Heinz and Pöppelmann (2022); these are based on a redundant representation of coefficients in $\mathbb{Z}_q$ and adding a sort of "checksum computations" through a CRT-based technique. A survey of side-channel protections of lattice-based schemes is given in Ravi *et al.* (2022).

Even more papers present side-channel and fault attacks against lattice-based KEMs, some also against protected implementations; see, e.g., Hamburg *et al.* (2021); Ngo *et al.* (2021); Delvaux and Merino Del Pozo (2021); Ngo *et al.* (2022,?); Backlund *et al.* (2022); Dubrova *et al.* (2022). Another class of attack papers considers chosen-ciphertext attacks against the passively secure PKE schemes that are typically underlying CCA-secure KEMs; see e.g., Fluhrer (2016); Bauer *et al.* (2019); Qin *et al.* (2021). Although these attacks do not violate any security claims of the schemes, they become relevant in a scenario where the information hidden by the FO transform (i.e., the information if decryption succeeded) can be recovered from side-channel information.

§1.2.2 Code-based KEMs – Classic McEliece: The McEliece cryptosystem has been introduced in McEliece (1978) and its dual variant by Niederreiter in Niederreiter (1986). The third-round specification of Classic McEliece can be found in Albrecht *et al.* (2020). Implementation tricks using additive FFT and transpose FFT as well as a Beneš network are expained in Chou (2017). Further implementation tweaks can be found in Chen and Chou (2021). The hardware implementations accompanying the Classic McEliece submission is described in Wang *et al.* (2017, 2018).

Code-based alternative candidates in the third round of the NIST standardizaton process are BIKE (see Aragon *et al.* (2020)) and HQC (see Aguilar Melchor *et al.* (2020)). Since both BIKE and HQC are using quasi-cyclic codes, their implementation requires efficient polynomial arithmetic. BIKE is using a bit-flipping decoder while HQC uses the Berlekamp-Massey algorithm.

Another interesting code-family for the use in code-based cryptography are rank codes. Low Rank Parity Check (LRPC) cods are used, e.g., for the round-1 schemes Ouroboros-R (see Aguilar Melchor *et al.* (2017)), LAKE (see Aragon *et al.* (2017*a*)) and LOCKER (see Aragon *et al.* (2017*b*)). These schemes havee been joined to ROLLO in round 2 (see Aragon *et al.* (2019)) and strongly depend on efficient Gaussian reduction for rank computations during decapsulation.

Besides cryptanalytic attacks that attempt to exploit the code structure of a code-based scheme, Information Set Decoding (ISD) is the most relevant generic attack. ISD goes back to Prange (1962) and has been improved over time, e.g., by Lee and Brickell (1988); Stern (1988); Finiasz and Sendrier (2009) and Becker *et al.* (2012). Security analyses of code-based schemes can be found in, e.g., Baldi *et al.* (2019) and Esser and Bellini (2021).

§1.2.3 Isogeny-based KEMs: In a seminar held in 1997, Couveignes proposed an isogeny-based scheme for mimicking the Diffie-Hellman key exchange protocol. Couveignes notes were later posted in Couveignes (2006). The first concrete isogeny-based cryptographic primitive was presented by Charles, Lauter and Goren in Charles *et al.* (2009), where the authors proposed a hash function whose collision resistance was extracted from the problem of path-finding in supersingular isogeny graphs. As early as 2006, Rostovtsev and Stulbunov introduced in Rostovtsev and Stolbunov (2006) isogeny-based cryptographic schemes, proposing a Diffie-Hellman-like protocol whose security guarantees were based on the difficulty of finding smooth-degree isogenies between ordinary elliptic curves. While the best classical algorithm by Galbraith and Stolbunov, solves this problem in full exponential time, Childs, Jao and Soukharev devised in 2014 a quantum algorithm computing such isogenies in subexponential time. Jao and De Feo Jao and De Feo (2011) proposed in late 2011 a Diffie-Hellman key-exchange scheme, this time relying on the difficulty of constructing isogenies between supersingular elliptic curves. Since the endomorphism ring of a supersingular elliptic curve is no longer commutative, the underlying isogeny problem was, for more than a decade, supposed to be much more difficult to solve. Within the context of the NIST standardization process, it was proposed in Jao *et al.* (2020) a SIDH variant equipped with a key encapsulation mechanism called SIKE. SIKE was selected as one of the fourth-round alternate KEM candidates of the NIST contest. Castryck and Decru Castryck and Decru (2022) presented in July 2022 a devastating attack against SIKE that was quickly followed by Maino and Martindale (2022); Robert (2022). Quickly after that, the authors of ISKE officially withdrawn their NIST submission.

§1.2.4 IND-CCA2 Security: The Fujisaki–Okamoto transform was introduced in 1999 in Fujisaki and Okamoto (1999). The security as part of post-quantum constructions was extensively studied in Hofheinz *et al*. (2017). Higher-order masking of the FO transform within lattice-based cryptography is covered in Bos *et al*. (2021). PC oracle attacks were introduced and analyzed in D'Anvers *et al*. (2019) and Guo *et al*. (2022). Fault attacks on CCA-secure lattice schemes was first studied in Pessl and Prokop (2021) and Hermelink *et al*. (2021).

§1.3.1 Lattice-based signatures – Dilithium: Lattice-based signature have a rather young history and date back to NTRUSign Hoffstein *et al*. (2003). The NIST finalist Falcon Prest *et al*. (2020) is based on NTRUSign. However, Dilithium Ducas *et al*. (2018); Lyubashevsky *et al*. (2020) is unrelated to NTRUSign and instead is based on Fiat–Shamir-with-aborts Lyubashevsky (2009). Efficient implementations of Dilithium are described in Seiler (2018) (AVX2) and Greconici *et al*. (2021) (Cortex-M4). Fault attacks on Dilithium have been studied in Bruinderink and Pessl (2018). In Barthe *et al*. (2018) masking GLP Güneysu *et al*. (2012) is studied. Masking Dilithium was studied in Migliore *et al*. (2019).

§1.3.2 Multivariate-quadratic-based signatures – UOV: Multivariate-quadratic signature date back to Matsumoto and Imai (1988). However, their original proposal C* got broken in Patarin (1995). Later proposals include HFE Patarin (1996) and OV Patarin (1997). While the original OV parameters got broken in Kipnis and Shamir (1998), the variation UOV described in Kipnis *et al*. (1999) remains unbroken up until today. Rainbow was proposed in Ding and Schmidt (2005) and uses multiple layers of the UOV scheme. It was shown in Ding *et al*. (2008), that it can only be secure with at most two layers. Rainbow is also a third-round finalist in the NIST PQC project. The specification can be found in Ding *et al*. (2020). Other proposals in the NIST PQC competition include GeMSS (specified in Casanova *et al*. (2020)), LUOV (specified in Beullens *et al*. (2019)), and MQDSS (specified in Samardjiska *et al*. (2019)). However, all three have been shown to be not secure in Tao *et al*. (2021), Ding *et al*. (2021), and Kales and Zaverucha (2020), respectively. Rainbow also suffered numerous attacks Beullens (2021); Baena *et al*. (2022); Beullens (2022). The most serious attack by Beullens in 2022 vastly reduces the security of Rainbow, essentially breaking the level 1 parameter sets. The use of GFNI for Rainbow was studied in Drucker and Gueron (2020). Rainbow implementations on x86 were studied in Chen *et al*. (2009). Bitsliced implementations of Rainbow on the Cortex-M4 were studied in Chou *et al*. (2021). Constant-time Gauss elimination was first described in Bernstein *et al*. (2013). The only work studying side-channel attacks on Rainbow is Park *et al*. (2018). Fault attacks on Rainbow and UOV were studied in Krämer and Loiero (2019).

§1.3.3 Hash-based signatures – XMSS and SPHINCS$^+$: The initial work on hash-based signatures by Lamport is described in Lamport (1979) and the following constructions by Winternitz and Merkle in Merkle (1990). The stateful signature schemes XMSS and LMS have been introduced in in Buchmann *et al*. (2011) and Leighton and Micali (1995) and specified in IETF RFC 8391 Huelsing *et al*. (2018) and RFC 8554 McGrew *et al*. (2019). The NIST PQC submission SPHINCS+ is specified in Hulsing *et al*. (2020). Parallelized vector implementations are described in, e.g., Kölbl (2018); Becker and Kannwischer (2022). Embedded and hardware implementations of hash-based schemes are provided in, e.g., Wang *et al*. (2019); Campos *et al*. (2020). Side-channel analysis of hash-based schemes is described in, e.g., Kannwischer *et al*. (2018). Fault-attacks are described in, e.g., Castelnovi *et al*. (2018); Genêt *et al*. (2018).

1.4. Bibliography

Abdulrahman, A., Hwang, V., Kannwischer, M. J., Sprenkels, D. (2022), Faster kyber and dilithium on the cortex-M4, *in* G. Ateniese, D. Venturi, (eds), ACNS 22: 20th International Conference on Applied Cryptography and Network Security, vol. 13269 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Rome, Italy, pp. 853–871.

Aguilar Melchor, C., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.-C., Gaborit, P., Hauteville, A., Zémor, G. (2017), Ouroboros-R, Technical report, National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.

Aguilar Melchor, C., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.-C., Gaborit, P., Persichetti, E., Zémor, G., Bos, J. (2020), HQC, Technical report, National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

Albrecht, M. R., Bernstein, D. J., Chou, T., Cid, C., Gilcher, J., Lange, T., Maram, V., von Maurich, I., Misoczki, R., Niederhagen, R., Paterson, K. G., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Tjhai, C. J., Tomlinson, M., Wang, W. (2020), Classic McEliece, Technical report, National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

Alkim, E., Bilgin, Y. A., Cenk, M., Gérard, F. (2020), Cortex-M4 optimizations for {R,M}LWE schemes, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3), 336–357. https://tches.iacr.org/index.php/TCHES/article/view/8593.

Alkim, E., Cheng, D. Y.-L., Chung, C.-M. M., Evkan, H., Huang, L. W.-L., Hwang, V., Li, C.-L. T., Niederhagen, R., Shih, C.-J., Wälde, J., Yang, B.-Y. (2021), Polynomial multiplication in NTRU prime, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1), 217–238. https://tches.iacr.org/index.php/TCHES/article/view/8733.

Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P. (2016), Post-quantum key exchange - A new hope, *in* T. Holz, S. Savage, (eds), USENIX Security 2016: 25th USENIX Security Symposium, USENIX Association, Austin, TX, USA, pp. 327–343.

Almeida, J. B., Barbosa, M., Barthe, G., Grégoire, B., Laporte, V., Léchenet, J.-C., Oliveira, T., Pacheco, H., Quaresma, M., Schwabe, P., Séré, A., Strub, P.-Y. (2023), 'Formally verifying kyber part i: Implementation correctness', Cryptology ePrint Archive, Paper 2023/215. https://eprint.iacr.org/2023/215.
URL: *https://eprint.iacr.org/2023/215*

Aragon, N., Barreto, P., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.-C., Gaborit, P., Gueron, S., Guneysu, T., Aguilar Melchor, C., Misoczki, R., Persichetti, E., Sendrier, N., Tillich, J.-P., Zémor, G., Vasseur, V., Ghosh, S. (2020), BIKE, Technical report, National Institute of Standards and Technology. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.

Aragon, N., Blazy, O., Deneuville, J.-C., Gaborit, P., Hauteville, A., Ruatta, O., Tillich, J.-P., Zémor, G. (2017*a*), LAKE, Technical report, National Institute of Standards and Technology. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

Aragon, N., Blazy, O., Deneuville, J.-C., Gaborit, P., Hauteville, A., Ruatta, O., Tillich, J.-P., Zémor, G. (2017*b*), LOCKER, Technical report, National Institute of Standards and Technology. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

Aragon, N., Blazy, O., Deneuville, J.-C., Gaborit, P., Hauteville, A., Ruatta, O., Tillich, J.-P., Zémor, G., Aguilar Melchor, C., Bettaieb, S., Bidoux, L., Bardet, M., Otmani, A. (2019), ROLLO, Technical report, National Institute of Standards and Technology. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

Backlund, L., Ngo, K., Gärtner, J., Dubrova, E. (2022), 'Secret key recovery attacks on masked and shuffled implementations of crystals-kyber and saber', Cryptology ePrint Archive, Paper 2022/1692. `https://eprint.iacr.org/2022/1692`.
URL: *https://eprint.iacr.org/2022/1692*

Baena, J., Briaud, P., Cabarcas, D., Perlner, R. A., Smith-Tone, D., Verbel, J. A. (2022), Improving support-minors rank attacks: Applications to GeMSS and rainbow, *in* Y. Dodis, T. Shrimpton, (eds), Advances in Cryptology – CRYPTO 2022, Part III, vol. 13509 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Santa Barbara, CA, USA, pp. 376–405.

Baldi, M., Barenghi, A., Chiaraluce, F., Pelosi, G., Santini, P. (2019), A finite regime analysis of information set decoding algorithms, *Algorithms*, 12(10).

Barthe, G., Belaïd, S., Espitau, T., Fouque, P.-A., Grégoire, B., Rossi, M., Tibouchi, M. (2018), Masking the GLP lattice-based signature scheme at any order, *in* J. B. Nielsen, V. Rijmen, (eds), Advances in Cryptology – EUROCRYPT 2018, Part II, vol. 10821 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Tel Aviv, Israel, pp. 354–384.

Bauer, A., Gilbert, H., Renault, G., Rossi, M. (2019), Assessment of the key-reuse resilience of NewHope, *in* M. Matsui, (ed.), Topics in Cryptology – CT-RSA 2019, vol. 11405 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, San Francisco, CA, USA, pp. 272–292.

Becker, A., Joux, A., May, A., Meurer, A. (2012), Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding, *in*

D. Pointcheval, T. Johansson, (eds), Advances in Cryptology – EUROCRYPT 2012, vol. 7237 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Cambridge, UK, pp. 520–536.

Becker, H., Hwang, V., Kannwischer, M. J., Yang, B.-Y., Yang, S.-Y. (2021), Neon ntt: Faster dilithium, kyber, and saber on cortex-a72 and apple m1, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1), 221–244.
URL: *https://tches.iacr.org/index.php/TCHES/article/view/9295*

Becker, H., Kannwischer, M. J. (2022), Hybrid scalar/vector implementations of Keccak and SPHINCS$^+$ on AArch64, *in* T. Isobe, S. Sarkar, (eds), Progress in Cryptology - INDOCRYPT 2022 - 23rd International Conference on Cryptology in India, Kolkata, India, December 11-14, 2022, Proceedings, vol. 13774 of *Lecture Notes in Computer Science*, Springer, pp. 272–293.

Beirendonck, M. V., D'Anvers, J., Karmakar, A., Balasch, J., Verbauwhede, I. (2021), A side-channel-resistant implementation of SABER, *ACM J. Emerg. Technol. Comput. Syst.*, 17(2), 10:1–10:26.
URL: *https://doi.org/10.1145/3429983*

Bernstein, D. J., Chou, T., Schwabe, P. (2013), McBits: Fast constant-time code-based cryptography, *in* G. Bertoni, J.-S. Coron, (eds), Cryptographic Hardware and Embedded Systems – CHES 2013, vol. 8086 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Santa Barbara, CA, USA, pp. 250–272.

Bernstein, D. J., Chuengsatiansup, C., Lange, T., van Vredendaal, C. (2017), NTRU prime: Reducing attack surface at low cost, *in* C. Adams, J. Camenisch, (eds), SAC 2017: 24th Annual International Workshop on Selected Areas in Cryptography, vol. 10719 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Ottawa, ON, Canada, pp. 235–260.

Beullens, W. (2021), Improved cryptanalysis of UOV and Rainbow, *in* A. Canteaut, F.-X. Standaert, (eds), Advances in Cryptology – EUROCRYPT 2021, Part I, vol. 12696 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Zagreb, Croatia, pp. 348–373.

Beullens, W. (2022), Breaking rainbow takes a weekend on a laptop, *in* Y. Dodis, T. Shrimpton, (eds), Advances in Cryptology – CRYPTO 2022, Part II, vol. 13508 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Santa Barbara, CA, USA, pp. 464–479.

Beullens, W., Preneel, B., Szepieniec, A., Vercauteren, F. (2019), LUOV, Technical report, National Institute of Standards and Technology. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Stehlé, D. (2018), CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM, *in* 2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, IEEE. To appear. `https://eprint.iacr.org/2017/634`.

Bos, J. W., Gourjon, M., Renes, J., Schneider, T., van Vredendaal, C. (2021), Masking kyber: First- and higher-order implementations, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4), 173–214. https://tches.iacr.org/index.php/TCHES/article/view/9064.

Bruinderink, L. G., Pessl, P. (2018), Differential fault attacks on deterministic lattice signatures, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3), 21–43. https://tches.iacr.org/index.php/TCHES/article/view/7267.

Buchmann, J. A., Dahmen, E., Hülsing, A. (2011), XMSS - A practical forward secure signature scheme based on minimal security assumptions, *in* B.-Y. Yang, (ed.), Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Springer, Heidelberg, Germany, Tapei, Taiwan, pp. 117–129.

Campos, F., Kohlstadt, T., Reith, S., Stöttinger, M. (2020), LMS vs XMSS: Comparison of stateful hash-based signature schemes on ARM cortex-M4, *in* A. Nitaj, A. M. Youssef, (eds), AFRICACRYPT 20: 12th International Conference on Cryptology in Africa, vol. 12174 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Cairo, Egypt, pp. 258–277.

Casanova, A., Faugère, J.-C., Macario-Rat, G., Patarin, J., Perret, L., Ryckeghem, J. (2020), GeMSS, Technical report, National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

Castelnovi, L., Martinelli, A., Prest, T. (2018), Grafting trees: A fault attack against the SPHINCS framework, *in* T. Lange, R. Steinwandt, (eds), Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Springer, Heidelberg, Germany, Fort Lauderdale, Florida, United States, pp. 165–184.

Castryck, W., Decru, T. (2022), 'An efficient key recovery attack on SIDH (preliminary version)', Cryptology ePrint Archive, Report 2022/975. https://eprint.iacr.org/2022/975.

Charles, D. X., Lauter, K. E., Goren, E. Z. (2009), Cryptographic hash functions from expander graphs, *Journal of Cryptology*, 22(1), 93–113.

Chen, A. I.-T., Chen, M.-S., Chen, T.-R., Cheng, C.-M., Ding, J., Kuo, E. L.-H., Lee, F. Y.-S., Yang, B.-Y. (2009), SSE implementation of multivariate PKCs on modern x86 CPUs, *in* C. Clavier, K. Gaj, (eds), Cryptographic Hardware and Embedded Systems – CHES 2009, vol. 5747 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Lausanne, Switzerland, pp. 33–48.

Chen, C., Danba, O., Hoffstein, J., Hulsing, A., Rijneveld, J., Schanck, J. M., Schwabe, P., Whyte, W., Zhang, Z., Saito, T., Yamakawa, T., Xagawa, K. (2020), NTRU, Technical report, National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

Chen, M.-S., Chou, T. (2021), ‘Classic McEliece on the ARM cortex-M4’, Cryptology ePrint Archive, Report 2021/492. https://eprint.iacr.org/2021/492.

Chou, T. (2017), McBits revisited, *in* W. Fischer, N. Homma, (eds), Cryptographic Hardware and Embedded Systems – CHES 2017, vol. 10529 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Taipei, Taiwan, pp. 213–231.

Chou, T., Kannwischer, M. J., Yang, B.-Y. (2021), Rainbow on cortex-M4, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4), 650–675. https://tches.iacr.org/index.php/TCHES/article/view/9078.

Chung, C.-M. M., Hwang, V., Kannwischer, M. J., Seiler, G., Shih, C.-J., Yang, B.-Y. (2021), NTT multiplication for NTT-unfriendly rings, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2), 159–188. https://tches.iacr.org/index.php/TCHES/article/view/8791.

Couveignes, J.-M. (2006), ‘Hard homogeneous spaces’, Cryptology ePrint Archive, Report 2006/291. https://eprint.iacr.org/2006/291.

D’Anvers, J.-P., Karmakar, A., Roy, S. S., Vercauteren, F., Mera, J. M. B., Beirendonck, M. V., Basso, A. (2020), SABER, Technical report, National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

D’Anvers, J., Tiepelt, M., Vercauteren, F., Verbauwhede, I. (2019), Timing attacks on error correcting codes in post-quantum schemes, *in* B. Bilgin, S. Petkova-Nikova, V. Rijmen, (eds), Proceedings of ACM Workshop on Theory of Implementation Security, TIS@CCS 2019, London, UK, November 11, 2019, ACM, pp. 2–9.

Delvaux, J., Merino Del Pozo, S. (2021), ‘Roulette: Breaking kyber with diverse fault injection setups’, Cryptology ePrint Archive, Report 2021/1622. https://eprint.iacr.org/2021/1622.

Ding, J., Chen, M.-S., Petzoldt, A., Schmidt, D., Yang, B.-Y., Kannwischer, M., Patarin, J. (2020), Rainbow, Technical report, National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

Ding, J., Deaton, J., Vishakha, Yang, B.-Y. (2021), The nested subset differential attack - A practical direct attack against LUOV which forges a signature within 210 minutes, *in* A. Canteaut, F.-X. Standaert, (eds), Advances in Cryptology – EUROCRYPT 2021, Part I, vol. 12696 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Zagreb, Croatia, pp. 329–347.

Ding, J., Schmidt, D. (2005), Rainbow, a new multivariable polynomial signature scheme, *in* J. Ioannidis, A. Keromytis, M. Yung, (eds), ACNS 05: 3rd International Conference on Applied Cryptography and Network Security, vol. 3531 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, New York, NY, USA, pp. 164–175.

Ding, J., Yang, B.-Y., Chen, C.-H. O., Chen, M.-S., Cheng, C.-M. (2008), New differential-algebraic attacks and reparametrization of Rainbow, *in* S. M. Bellovin, R. Gennaro, A. D. Keromytis, M. Yung, (eds), ACNS 08: 6th International Conference on Applied Cryptography and Network Security, vol. 5037 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, New York, NY, USA, pp. 242–257.

Drucker, N., Gueron, S. (2020), 'Speed up over the rainbow', Cryptology ePrint Archive, Report 2020/408. https://eprint.iacr.org/2020/408.

Dubrova, E., Ngo, K., Gärtner, J. (2022), Breaking a fifth-order masked implementation of crystals-kyber by copy-paste, *IACR Cryptol. ePrint Arch.*, . https://eprint.iacr.org/2022/1713.

Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D. (2018), CRYSTALS-Dilithium: A lattice-based digital signature scheme, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1), 238–268. https://tches.iacr.org/index.php/TCHES/article/view/839.

D'Anvers, J.-P., Van Beirendonck, M., Verbauwhede, I. (2023), Revisiting higher-order masked comparison for lattice-based cryptography: Algorithms and bit-sliced implementations, *IEEE Transactions on Computers*, 72(2), 321–332.

Esser, A., Bellini, E. (2021), 'Syndrome decoding estimator', Cryptology ePrint Archive, Report 2021/1243. https://eprint.iacr.org/2021/1243.

Finiasz, M., Sendrier, N. (2009), Security bounds for the design of code-based cryptosystems, *in* M. Matsui, (ed.), Advances in Cryptology – ASIACRYPT 2009, vol. 5912 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Tokyo, Japan, pp. 88–105.

Fluhrer, S. (2016), 'Cryptanalysis of ring-LWE based key exchange with key share reuse', Cryptology ePrint Archive, Report 2016/085. https://eprint.iacr.org/2016/085.

Fujisaki, E., Okamoto, T. (1999), Secure integration of asymmetric and symmetric encryption schemes, *in* M. J. Wiener, (ed.), Advances in Cryptology – CRYPTO'99, vol. 1666 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Santa Barbara, CA, USA, pp. 537–554.

Genêt, A., Kannwischer, M. J., Pelletier, H., McLauchlan, A. (2018), 'Practical fault injection attacks on SPHINCS', Cryptology ePrint Archive, Report 2018/674. https://eprint.iacr.org/2018/674.

Greconici, D. O. C., Kannwischer, M. J., Sprenkels, D. (2021), Compact dilithium implementations on cortex-M3 and cortex-M4, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1), 1–24. https://tches.iacr.org/index.php/TCHES/article/view/8725.

Güneysu, T., Lyubashevsky, V., Pöppelmann, T. (2012), Practical lattice-based cryptography: A signature scheme for embedded systems, *in* E. Prouff, P. Schaumont, (eds), Cryptographic Hardware and Embedded Systems – CHES 2012, vol. 7428 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Leuven, Belgium, pp. 530–547.

Guo, Q., Hlauschek, C., Johansson, T., Lahr, N., Nilsson, A., Schröder, R. L. (2022), Don't reject this: Key-recovery timing attacks due to rejection-sampling in HQC and BIKE, *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(3), 223–263.

Hamburg, M., Hermelink, J., Primas, R., Samardjiska, S., Schamberger, T., Streit, S., Strieder, E., van Vredendaal, C. (2021), Chosen ciphertext k-trace attacks on masked CCA2 secure kyber, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4), 88–113. https://tches.iacr.org/index.php/TCHES/article/view/9061.

Heinz, D., Kannwischer, M. J., Land, G., Pöppelmann, T., Schwabe, P., Sprenkels, D. (2022), 'First-order masked kyber on ARM cortex-M4', Cryptology ePrint Archive, Report 2022/058. https://eprint.iacr.org/2022/058.

Heinz, D., Pöppelmann, T. (2022), Combined fault and dpa protection for lattice-based cryptography, *IEEE Transactions on Computers*, pp. 1–12.

Hermelink, J., Pessl, P., Pöppelmann, T. (2021), Fault-enabled chosen-ciphertext attacks on kyber, *in* A. Adhikari, R. Küsters, B. Preneel, (eds), Progress in Cryptology - INDOCRYPT 2021 - 22nd International Conference on Cryptology in India, Jaipur, India, December 12-15, 2021, Proceedings, vol. 13143 of *Lecture Notes in Computer Science*, Springer, pp. 311–334.

Hoffstein, J., Howgrave-Graham, N., Pipher, J., Silverman, J. H., Whyte, W. (2003), NTRUSIGN: Digital signatures using the NTRU lattice, *in* M. Joye, (ed.), Topics in Cryptology – CT-RSA 2003, vol. 2612 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, San Francisco, CA, USA, pp. 122–140.

Hoffstein, J., Pipher, J., Silverman, J. H. (1998), NTRU: A ring-based public key cryptosystem, *in* Third Algorithmic Number Theory Symposium (ANTS), vol. 1423 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 267–288.

Hofheinz, D., Hövelmanns, K., Kiltz, E. (2017), A modular analysis of the Fujisaki-Okamoto transformation, *in* Y. Kalai, L. Reyzin, (eds), TCC 2017: 15th Theory of Cryptography Conference, Part I, vol. 10677 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Baltimore, MD, USA, pp. 341–371.

Huelsing, A., Butin, D., Gazdag, S.-L., Rijneveld, J., Mohaisen, A. (2018), 'XMSS: eXtended Merkle Signature Scheme', RFC 8391.
URL: *https://www.rfc-editor.org/info/rfc8391*

Hulsing, A., Bernstein, D. J., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.-L., Kampanakis, P., Kolbl, S., Lange, T., Lauridsen, M. M., Mendel, F., Niederhagen, R., Rechberger, C., Rijneveld, J., Schwabe, P., Aumasson, J.-P., Westerbaan, B., Beullens, W. (2020), SPHINCS+, Technical report, National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

Hwang, V., Liu, J., Seiler, G., Shi, X., Tsai, M.-H., Wang, B.-Y., Yang, B.-Y. (2022), Verified ntt multiplications for nistpqc kem lattice finalists: Kyber, saber, and ntru, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4), 718–750.
URL: *https://tches.iacr.org/index.php/TCHES/article/view/9838*

Jao, D., Azarderakhsh, R., Campagna, M., Costello, C., De Feo, L., Hess, B., Jalali, A., Koziel, B., LaMacchia, B., Longa, P., Naehrig, M., Renes, J., Soukharev, V., Urbanik, D., Pereira, G., Karabina, K., Hutchinson, A. (2020), SIKE, Technical report, National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

Jao, D., De Feo, L. (2011), Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies, *in* B.-Y. Yang, (ed.), Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Springer, Heidelberg, Germany, Tapei, Taiwan, pp. 19–34.

Kales, D., Zaverucha, G. (2020), An attack on some signature schemes constructed from five-pass identification schemes, *in* S. Krenn, H. Shulman, S. Vaudenay, (eds), CANS 20: 19th International Conference on Cryptology and Network Security, vol. 12579 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Vienna, Austria, pp. 3–22.

Kamucheka, T., Nelson, A., Andrews, D., Huang, M. (2022), 'A masked pure-hardware implementation of kyber cryptographic algorithm', Cryptology ePrint Archive, Paper 2022/1547. https://eprint.iacr.org/2022/1547.
URL: *https://eprint.iacr.org/2022/1547*

Kannwischer, M. J., Genêt, A., Butin, D., Krämer, J., Buchmann, J. (2018), Differential power analysis of XMSS and SPHINCS, *in* J. Fan, B. Gierlichs, (eds), COSADE 2018: 9th International Workshop on Constructive Side-Channel Analysis and Secure Design, vol. 10815 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Singapore, pp. 168–188.

Kannwischer, M. J., Rijneveld, J., Schwabe, P. (2019), Faster multiplication in $\mathbb{Z}_{2^m}[x]$ on cortex-M4 to speed up NIST PQC candidates, *in* R. H. Deng, V. Gauthier-Umaña, M. Ochoa, M. Yung, (eds), ACNS 19: 17th International Conference on Applied Cryptography and Network Security, vol. 11464 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Bogota, Colombia, pp. 281–301.

Karmakar, A., Mera, J. M. B., Roy, S. S., Verbauwhede, I. (2018), Saber on ARM, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3), 243–266. https://tches.iacr.org/index.php/TCHES/article/view/7275.

Kipnis, A., Patarin, J., Goubin, L. (1999), Unbalanced Oil and Vinegar signature schemes, *in* J. Stern, (ed.), Advances in Cryptology – EUROCRYPT'99, vol. 1592 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Prague, Czech Republic, pp. 206–222.

Kipnis, A., Shamir, A. (1998), Cryptanalysis of the Oil & Vinegar signature scheme, *in* H. Krawczyk, (ed.), Advances in Cryptology – CRYPTO'98, vol. 1462 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Santa Barbara, CA, USA, pp. 257–266.

Kölbl, S. (2018), Putting wings on SPHINCS, *in* T. Lange, R. Steinwandt, (eds), Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Springer, Heidelberg, Germany, Fort Lauderdale, Florida, United States, pp. 205–226.

Krämer, J., Loiero, M. (2019), Fault attacks on UOV and Rainbow, *in* I. Polian, M. Stöttinger, (eds), COSADE 2019: 10th International Workshop on Constructive Side-Channel Analysis and Secure Design, vol. 11421 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Darmstadt, Germany, pp. 193–214.

Kundu, S., D'Anvers, J., Beirendonck, M. V., Karmakar, A., Verbauwhede, I. (2022), Higher-order masked saber, *in* C. Galdi, S. Jarecki, (eds), Security and Cryptography for Networks - 13th International Conference, SCN 2022, Amalfi, Italy, September 12-14, 2022, Proceedings, vol. 13409 of *Lecture Notes in Computer Science*, Springer, pp. 93–116.
URL: *https://doi.org/10.1007/978-3-031-14791-3_5*

Lamport, L. (1979), Constructing digital signatures from a one-way function, Technical Report SRI-CSL-98, SRI International Computer Science Laboratory.

Lee, P. J., Brickell, E. F. (1988), An observation on the security of McEliece's public-key cryptosystem, *in* C. G. Günther, (ed.), Advances in Cryptology – EUROCRYPT'88, vol. 330 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Davos, Switzerland, pp. 275–280.

Leighton, F. T., Micali, S. (1995), 'Large provably fast and secure digital signature schemes based on secure hash functions', U.S. Patent 5,432,852.

Lyubashevsky, V. (2009), Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures, *in* M. Matsui, (ed.), Advances in Cryptology – ASIACRYPT 2009, vol. 5912 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Tokyo, Japan, pp. 598–616.

Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., Seiler, G., Stehlé, D., Bai, S. (2020), CRYSTALS-DILITHIUM, Technical report, National Institute of Standards and Technology. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.

Lyubashevsky, V., Peikert, C., Regev, O. (2010), On ideal lattices and learning with errors over rings, *in* H. Gilbert, (ed.), Advances in Cryptology – EUROCRYPT 2010, vol. 6110 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, French Riviera, pp. 1–23.

Lyubashevsky, V., Peikert, C., Regev, O. (2013), On ideal lattices and learning with errors over rings, *Journal of the ACM*, 60(6), 43:1–43:35. `http://www.cims.nyu.edu/~regev/papers/ideal-lwe.pdf`.

Maino, L., Martindale, C. (2022), 'An attack on SIDH with arbitrary starting curve', Cryptology ePrint Archive, Report 2022/1026. https://eprint.iacr.org/2022/1026.

Matsumoto, T., Imai, H. (1988), Public quadratic polynominal-tuples for efficient signature-verification and message-encryption, *in* C. G. Günther, (ed.), Advances in Cryptology – EUROCRYPT'88, vol. 330 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Davos, Switzerland, pp. 419–453.

McEliece, R. J. (1978), A public-key cryptosystem based on algebraic coding theory, The deep space network progress report 42-44, Jet Propulsion Laboratory, California Institute of Technology. https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF.

McGrew, D., Curcio, M., Fluhrer, S. (2019), 'Leighton-Micali Hash-Based Signatures', RFC 8554.
URL: *https://www.rfc-editor.org/info/rfc8554*

Merkle, R. C. (1990), A certified digital signature, *in* G. Brassard, (ed.), Advances in Cryptology – CRYPTO'89, vol. 435 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Santa Barbara, CA, USA, pp. 218–238.

Migliore, V., Gérard, B., Tibouchi, M., Fouque, P.-A. (2019), 'Masking Dilithium: Efficient implementation and side-channel evaluation', Cryptology ePrint Archive, Report 2019/394. https://eprint.iacr.org/2019/394.

Ngo, K., Dubrova, E., Guo, Q., Johansson, T. (2021), A side-channel attack on a masked IND-CCA secure saber KEM implementation, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4), 676–707. https://tches.iacr.org/index.php/TCHES/article/view/9079.

Ngo, K., Wang, R., Dubrova, E., Paulsrud, N. (2022), 'Side-channel attacks on lattice-based KEMs are not prevented by higher-order masking', Cryptology ePrint Archive, Report 2022/919. https://eprint.iacr.org/2022/919.

Niederreiter, H. (1986), Knapsack-type cryptosystems and algebraic coding theory, *Problems of Control and Information Theory*, 15(2), 159–166.

Oder, T., Schneider, T., Pöppelmann, T., Güneysu, T. (2018), Practical CCA2-secure masked Ring-LWE implementations, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1), 142–174. https://tches.iacr.org/index.php/TCHES/article/view/836.

Park, A., Shim, K.-A., Koo, N., Han, D.-G. (2018), Side-channel attacks on post-quantum signature schemes based on multivariate quadratic equations, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3), 500–523. https://tches.iacr.org/index.php/TCHES/article/view/7284.

Patarin, J. (1995), Cryptanalysis of the Matsumoto and Imai public key scheme of eurocrypt'88, *in* D. Coppersmith, (ed.), Advances in Cryptology – CRYPTO'95, vol. 963 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Santa Barbara, CA, USA, pp. 248–261.

Patarin, J. (1996), Hidden fields equations (HFE) and isomorphisms of polynomials (IP): Two new families of asymmetric algorithms, *in* U. M. Maurer, (ed.), Advances in Cryptology – EUROCRYPT'96, vol. 1070 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Saragossa, Spain, pp. 33–48.

Patarin, J. (1997), 'The oil and vinegar algorithm for signatures', Dagstuhl Workshop on Cryptography.

Pessl, P., Prokop, L. (2021), Fault attacks on CCA-secure lattice KEMs, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2), 37–60. https://tches.iacr.org/index.php/TCHES/article/view/8787.

Prange, E. (1962), The use of information sets in decoding cyclic codes, *IRE Transactions on Information Theory*, 8(5), 5–9.

Prest, T., Fouque, P.-A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z. (2020), FALCON, Technical report, National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

Qin, Y., Cheng, C., Zhang, X., Pan, Y., Hu, L., Ding, J. (2021), A systematic approach and analysis of key mismatch attacks on lattice-based NIST candidate KEMs, *in* M. Tibouchi, H. Wang, (eds), Advances in Cryptology – ASIACRYPT 2021, Part IV, vol. 13093 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Singapore, pp. 92–121.

Ravi, P., Chattopadhyay, A., Baksi, A. (2022), 'Side-channel and fault-injection attacks over lattice-based post-quantum schemes (kyber, dilithium): Survey and new results', Cryptology ePrint Archive, Report 2022/737. https://eprint.iacr.org/2022/737.

Regev, O. (2005), On lattices, learning with errors, random linear codes, and cryptography, *in* H. N. Gabow, R. Fagin, (eds), 37th Annual ACM Symposium on Theory of Computing, ACM Press, Baltimore, MA, USA, pp. 84–93.

Regev, O. (2009), On lattices, learning with errors, random linear codes, and cryptography, *Journal of the ACM*, 56(6), 34. http://www.cims.nyu.edu/~regev/papers/qcrypto.pdf.

Robert, D. (2022), 'Breaking SIDH in polynomial time', Cryptology ePrint Archive, Report 2022/1038. https://eprint.iacr.org/2022/1038.

Rostovtsev, A., Stolbunov, A. (2006), 'Public-Key Cryptosystem Based On Isogenies', Cryptology ePrint Archive, Report 2006/145. https://eprint.iacr.org/2006/145.

Samardjiska, S., Chen, M.-S., Hulsing, A., Rijneveld, J., Schwabe, P. (2019), MQDSS, Technical report, National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Seiler, G., Stehlé, D., Ding, J. (2022), CRYSTALS-KYBER, Technical report, National Institute of Standards and Technology. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`.

Seiler, G. (2018), 'Faster AVX2 optimized NTT multiplication for ring-LWE lattice cryptography', Cryptology ePrint Archive, Report 2018/039. `https://eprint.iacr.org/2018/039`.

Stern, J. (1988), A method for finding codewords of small weight, *in* G. D. Cohen, J. Wolfmann, (eds), Coding Theory and Applications, 3rd International Colloquium, Toulon, France, November 2-4, 1988, Proceedings, vol. 388 of *Lecture Notes in Computer Science*, Springer, pp. 106–113.

Tao, C., Petzoldt, A., Ding, J. (2021), Efficient key recovery for all HFE signature variants, *in* T. Malkin, C. Peikert, (eds), Advances in Cryptology – CRYPTO 2021, Part I, vol. 12825 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Virtual Event, pp. 70–93.

Wang, W., Jungk, B., Wälde, J., Deng, S., Gupta, N., Szefer, J., Niederhagen, R. (2019), XMSS and embedded systems, *in* K. G. Paterson, D. Stebila, (eds), SAC 2019: 26th Annual International Workshop on Selected Areas in Cryptography, vol. 11959 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Waterloo, ON, Canada, pp. 523–550.

Wang, W., Szefer, J., Niederhagen, R. (2017), FPGA-based key generator for the niederreiter cryptosystem using binary goppa codes, *in* W. Fischer, N. Homma, (eds), Cryptographic Hardware and Embedded Systems – CHES 2017, vol. 10529 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, Taipei, Taiwan, pp. 253–274.

Wang, W., Szefer, J., Niederhagen, R. (2018), FPGA-based niederreiter cryptosystem using binary goppa codes, *in* T. Lange, R. Steinwandt, (eds), Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Springer, Heidelberg, Germany, Fort Lauderdale, Florida, United States, pp. 77–98.