QuantumRISC

Work Package 2, Deliverable 2.1:
Analysis and Optimization of PQC Schemes

Version	1.0
Project Coordination	Fraunhofer Institute for Secure Information Technology
Date of preparation	May 4, 2023

SPONSORED BY THE

Federal Ministry
of Education
and Research

Authors

- Continental AG:
  - Maurice Heymann
- Elektrobit Automotive GmbH:
  - Hannes Hennig
- Fraunhofer SIT:
  - Norman Lahr
  - Richard Petri
  - Julian Wälde
- Hochschule RheinMain:
  - Thorsten Knoll
- MTG AG:
  - Evangelos Karatsiolis
  - Johannes Roth
- Ruhr-Universität Bochum:
  - Georg Land
  - Jan Philipp Thoma
- Universität Regensburg:
  - Juliane Krämer
  - Michael Meyer
  - Marcel Müller

Project Coordination

Norman Lahr
Fraunhofer Institute for Secure Information Technology
Advanced Cryptographic Engineering
Rheinstr. 75
D-64295 Darmstadt
Germany

Phone +49 6151 869100
Fax +49 6151 869224
Mail norman.lahr@sit.fraunhofer.de

Contents

1 Executive summary

QuantumRISC is a project targeted at the secure and efficient use of quantum-resistant cryptographic algorithms (post-quantum cryptography; PQC) for embedded devices. This work reports on work package 2 (WP2) of the QuantumRISC project, focusing on the analysis and improvements of quantum-resistant schemes, along with a study of their suitability for the project.

First we develop several general criteria such as the (non-)existence of security concerns or patent issues. We pick five relevant and representative use cases from WP1, and present corresponding criteria for the suitability of schemes, e.g., concerning key sizes or running times. Following this, we go through relevant PQC algorithms, mainly participants of the second round of the PQC standardization effort by the National Institute of Standards and Technology (NIST), and analyze their suitability for our use cases. We conclude this part by an explicit choice of schemes that are considered in the following work packages of QuantumRISC.

Furthermore, we discuss various optimization avenues for improving the applicability of PQC schemes in our context of embedded devices. This includes performance optimizations, in particular, we explore the choice of parameters for isogeny-based signature schemes, which allow for a more efficient instantiation.

For several PQC schemes, we discuss memory-constrained implementations. This includes many promising signature schemes, as well as the Classic McEliece key encapsulation. E.g., we show how PQC signatures can be verified in a memory-constrained environment by streaming-in signatures.

In terms of robustness against physical attacks, we explore several attacks and their countermeasures for isogeny-based schemes. This includes both active attacks, such as fault injection attacks, and passive side-channel attacks. For embedded devices, the security against physical attacks is an important prerequisite for the secure application of cryptographic schemes.

2 Introduction

2.1 Post-quantum cryptography

Public-key cryptography is one of the main building blocks required for cybersecurity. Its basic protocols, public-key encryption, key agreement schemes, and digital signatures, cover many usual cryptographic applications, and serve as a basis for more advanced protocols. Widespread public-key schemes are given by RSA [82] for encryption and digital signatures, and elliptic curve cryptography (ECC) [64, 73] for key agreement and digital signatures. Since especially ECC allows for efficient implementations and uses very small key sizes, it can be applied in many use cases, even if computations have to be run on resource-constrained devices.

However, public-key schemes based on the discrete logarithm problem, such as RSA and ECC, can be efficiently broken on a large-scale quantum computer via Shor's algorithm [87]. Although such a quantum computer is not available yet, it is inevitable to analyze how a transition to quantum-resistant alternatives for these schemes can be realized. In particular, we aim for deploying *post-quantum cryptography* (PQC), that is, schemes that cannot be broken using both classical and quantum computers. In order to fuel more research and initiate the transition to PQC, the US-based National Institute of Standards and Technology (NIST) started a PQC standardization process in 2016 [75]. Over three rounds, it led to an announcement of schemes to be standardized in July 2022. The first PQC schemes to be standardized are the Key Encapsulation Mechanism (KEM) CRYSTALS-Kyber [21], and the signature schemes CRYSTALS-Dilithium [53], Falcon [81], and SPHINCS$^+$ [59]. More KEMs are expected to be standardized after a fourth round of evaluation. In particular, the remaining candidates in the fourth round of KEM evaluations are BIKE[11], Classic McEliece[6], HQC [2], and SIKE [62].[1] Furthermore, a new round for submissions of signature schemes will open in 2023. In addition to the NIST standardization effort, the Internet Engineering Task Force (IETF) standardized the stateful hash-based schemes LMS and XMSS (see Section 3.7), which are out of scope of the NIST process.

In contrast to classical schemes like RSA and ECC, PQC does not offer a single solution that is applicable to all use cases. In particular, choosing a specific PQC scheme always corresponds to a certain tradeoff between running times, key sizes, signature sizes, and memory requirements. Thus, for each use case with its individual requirements, e.g., regarding resource-constrained devices, it is not trivial to select a suitable PQC scheme, but a careful analysis for the best such tradeoff is necessary. Therefore, more research and analysis of the available PQC schemes is necessary, in order to provide suitable schemes for many use cases, and simplify the transition to PQC in practice.

[1] Note that recently the SIKE submission was broken by a series of attack papers, see Section 3.4.1.

2.2 Document goals and structure

Implementing PQC schemes for resource-constrained devices such as microcontrollers, like small instances of a RISC-V processor, is at the heart of the QuantumRISC research project. This report contains the following deliverable from work package WP2:

- D2.1 - Documentation on implemented PQC schemes and protocols.

The report focuses on analyzing existing PQC schemes for specific use cases discussed in the report of QuantumRISC WP1 [78], mainly including participants of the NIST standardization process, and presents optimizations and physical attacks. It therefore lays the theoretical foundations for further QuantumRISC work packages, in particular, for PQC software libraries (WP3), hardware implementations and accelerators (WP4), hardware-software co-design (WP5), and practical evaluation through a demonstrator for QuantumRISC use cases (WP6).

Chapter 3 analyzes the relevant PQC schemes with respect to their performance profiles. It picks a set of widespread use cases from [78], and analyzes the suitability of each scheme for these specific applications. Furthermore, we discuss our choice of schemes we consider for the usage in the following work packages of QuantumRISC.

The remainder of this report presents the contributions by the QuantumRISC consortium. Their aim is to improve the applicability of PQC schemes to our use cases, which further applies to implementing PQC primitives for embedded systems in general. This includes performance and memory requirement optimizations, and an analysis of physical attacks such as side-channel or fault injection attacks.

Chapter 4 presents work on performance optimizations of PQC schemes. In particular, we analyze the choice of parameters for the isogeny-based signature scheme SQISign. In order to be efficient, SQISign requires a prime number of a special shape as a system parameter. Finding such primes translates to a number theoretical problem, and we give a new approach to find suitable parameters.

Chapter 5 discusses reductions of the memory requirements of PQC schemes. We show how PQC signatures can be verified on memory-constrained devices by streaming in signatures. Similarly, we describe how signing and verifying in SPHINCS$^+$, and running the key encapsulation scheme Classic McEliece can be achieved with lower memory requirements, using similar techniques. Furthermore, we show how the on-the-fly computation of twiddle factors in the Number Theoretic Transform (NTT) can be applied to reduce memory requirements for RISC-V implementations.

Chapter 6 presents physical attacks on PQC schemes, such as side-channel and fault attacks, and presents countermeasures to mitigate these. In particular, we focus on fault attacks, zero-value and correlation attacks on the isogeny-based schemes SIKE and CSIDH.

3 Analysis of PQC schemes

In this section we analyze the most relevant quantum-resistant schemes, that is, most practical participants of the NIST standardization process, and a limited number of potentially suitable schemes outside of the scope of NIST. For each scheme, we briefly describe the main characteristics, and evaluate the most important features, such as key sizes, signature sizes, and performance numbers.

In order to simplify comparisons between schemes, we assign ratings for most of these features of each scheme by the means of colored symbols. A green triangle pointing upwards (▲) reflects that the corresponding feature appears to be suitable for practical applications of the scheme. A yellow square (■) indicates worse suitability than a green mark, while practical application may still be possible. A red triangle pointing downwards (▼) shows that the corresponding feature may prevent the scheme from being applicable in many situations. In some instances, we will also use a neutral rating, which is represented by a gray circle (●).

Furthermore, we chose five use cases from WP1 that cover the most important cryptographic applications in our context. For each scheme, we evaluate the suitability for these five use cases and their requirements by assigning the respective symbols.

We emphasize that all of these ratings are not solely based on objective data, but are, to a certain extent, also based on subjective assessment. In particular, red marks do not indicate that the corresponding scheme is not usable for our specific use cases, but rather that other schemes may be more suitable with respect to this feature.

3.1 Overall criteria

Common criteria. We present the following data and ratings for all schemes:

1. Year of publication of the scheme – In general, older schemes that did not suffer a serious attack are considered to be conservative choices with respect to security. However, the non-existence of a serious attack may also be due to limited cryptanalytic research interest. Thus, we only present the year of publication, but refrain from assigning ratings for this data.

2. Standardization – We indicate if schemes are already standardized or recommended, e.g., by BSI or IETF, or to which round of the NIST standardization process they advanced. Standardization and advancing to further rounds improves the rating:

   ▲ : Standardized/recommended or participant of NIST Round 3

   ■ : Participant of NIST Round 2

   ▼ : Participant of NIST Round 1

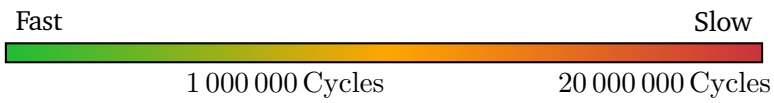
3. Security – We indicate if the security assumptions of the corresponding scheme are conservative and well-understood, if they are immature, or if there is doubt about the security of the scheme or its proposed parameters. The ratings are rather subjective and are based on discussions in the NIST PQC forum [77] and the NIST status reports [4, 5]. We note that the highest rating is only assigned to schemes whose security features are especially well-understood. Thus, a lower rating does not indicate a priori that the respective scheme is not secure enough for practical application.
   - ▲ : Conservative/well-understood security assumptions
   - □ : Immature security assumptions or debated parameterization
   - ▼ : Broken
4. Patent issues – Patents or potential patent infringements of schemes may significantly hamper their practical application. Thus, we rate the schemes based on patents or potential patent applicability. The ratings are rather subjective and are based on discussions in the NIST PQC forum [77]. We note that even if no patents are known for a scheme, there may still be applicable patents that the research community is not aware of.
   - ▲ : No patents known
   - □ : Debates on patent applicability
   - ▼ : Known patents

The rating of these and the following features allows for a simple comparison between different schemes. However, they cannot directly be used to qualify or disqualify schemes for certain use cases, since they only represent the speed and sizes relative to other schemes. We detail the suitability for specific use cases in Section 3.2.

Signature scheme criteria. For digital signature schemes we present running times of key generation, signature generation, and signature verification in clock cycles, and sizes of signatures, public keys and private keys in bytes. As for all categories, most of the data was taken from official submissions to the NIST standardization process. Furthermore, we indicate if there is a limited number of signatures that can be created for a single key pair. For signature schemes, we use the following ratings, which are translated to the symbols introduced above (▲, □, ▼) in the subsequent sections.

1. Signature and Key Sizes in bytes.

2. Key Generation, Signature, and Verification Time in cycles.



3. Maximal Number of Signatures per Key Pair.
   - ▲ : Unlimited
   - ■ : Limited

Key Encapsulation Mechanism (KEM) criteria. For KEMs we present running times of key generation, encapsulation, and decapsulation, and sizes for ciphertexts, public and private keys as above. As before, ratings are only intended for a simple comparison between different schemes. For KEMs, we use the following ratings.

1. Key and Ciphertext Sizes in bytes.

2. Key Generation, Encapsulation, Decapsulation Time in cycles.

Key Exchange (KEX) criteria. KEXs have the same criteria as KEMs, only substituting encapsulation and decapsulation timings by the key exchange time. For KEXs, we use the following ratings.

1. Secret Key, Public Key, in bytes.

2. Key Generation and Exchange Time in cycles.

Public Key Encryption (PKE) criteria. Most of the KEM schemes that participate in the NIST PQC standardization process also offer a PKE variant. However, the key size and performance profile is very similar to the respective KEM schemes. Thus, we refrain from explicitly detailing their features in the following analysis.

Parameters. In order to ensure comparability of performance data and ratings, we only consider parameters from the official NIST submissions of the schemes, where applicable, for the following analysis in Sections 3.3 to 3.7. In particular, whenever available, we detail the performance profile of each scheme for NIST security levels 1 and 3, which correspond to the security of AES-128 and AES-192, respectively, and are most likely to be used in our context of implementations on microcontrollers.

We note that some PQC schemes, for instance lattice-based proposals, offer a large number of parameters, which could potentially be adapted for special use cases. However, NIST will standardize schemes along with concrete parameters. Hence, it is highly recommended not to deviate from these parameters, in order not to introduce potential vulnerabilities. Due to this, we refrain from adapting parameters to special use cases.

3.2 Use case criteria

For this section, we chose five specific use cases from the QuantumRISC WP1 report [78]. For each of the following PQC schemes, we rate the suitability for these use cases. That is, we evaluate if the schemes potentially satisfy the use case requirements from [78]. We note that the timing requirements from [78] are given in wall clock time instead of clock cycles. Thus, we use the given cycle counts per scheme, and assume a 100 MHz processor for obtaining an estimated wall clock timing. This also means that the rating presented here only serves as a normalization, while concrete evaluations of the use case suitability heavily depend on the respective hardware.

For each scheme, we present a table that rates the size and timing features with respect to the respective use case requirements, where the same color and symbol encoding as above is used. That is, a green triangle pointing upwards (▲) means that the corresponding requirement is met; a yellow square (■) indicates that the requirement is not satisfied, but rather close to the stated goal; a red triangle pointing downwards (▼) shows that the corresponding feature may prevent the scheme from being applicable for this use case. Again we note that these ratings may differ heavily if optimizations for schemes are found, or different hardware is used. Thus, the ratings should only be seen as an indication as to which schemes might be the best choice for the specific use cases. Nevertheless, we consider the results for our final choice of PQC schemes for the QuantumRISC project.

The following use cases from [78] were selected:

- 3.2.1 – Secure Download
- 3.2.2 – Feature Activation
- 3.5.1 – Secure Access Control
- 3.7.1 – Key Generation on Device
- 3.8.1 – Session-based Secure Channel

We briefly recall the details and requirements for each of these use cases.

Use case 3.2.1 – Secure Download (Sec. DL). The secure download mechanism downloads software, and enforces that it is only flashed to an ECU if correctly authenticated. That is, its aim is to prevents attackers from flashing manipulated software to an ECU. To this end, secure download uses a digital signature scheme, attaches a signature to the respective software, and only flashes it to the ECU if the signature is valid, and therefore grants authentication and non-repudiation. This use case usually features a backend

server that generates signatures, and a deeply embedded ECU that verifies signatures. The requirements for signature schemes are detailed in Table 3.1.

Properties	**Backend**	**ECU**
Latency	few minutes	<1 s
# Executions over product lifetime	limited	limited
Size of processed data	$\leq$1MB: $>$1GB	$\leq$1MB: $>$1GB
Physical accessibility	restricted	accessible
Computational power	Server	Deeply embedded
RAM availability	Server	Deeply embedded
Storage availability	Server	Deeply embedded
# Key pairs	$\leq$10	$\leq$10
Data dissipation	one-to-many	
Life time	$>$5 years	
Current security level	$\leq$192 bits : $\leq$256 bits	

Table 3.1: Requirements of the use case Secure Download [78, Table 3.4].

Use case 3.2.2 – Feature Activation (Feat. Ac.). The feature activation mechanism is deployed for business models such as pay-on-demand services. In this scenario, a set of functionalities is initially deployed in software on a device/ECU, but only a certain subset of these functionalities is activated by default due to a policy set. A user can then request the activation of additional features for a limited time period via the pay-on-demand service.

This means that in contrast to Use Case 3.2.1, the software for the requested functionalities is already available on the device, and hence a software update is not required. Instead, a certificate that is signed by an authorized backend entity usually enables the requested policy change. That is, from a cryptographic point of view this use case involves a backend server that generates signatures, and an embedded device that verifies signatures. The requirements for signature schemes are detailed in Table 3.2.

Use case 3.5.1 – Secure Access Control (Sec. AC). This use case involves a sender, usually a server backend, and a receiver, an automotive ECU. The aim is for the sender to gain access to certain services of the ECU, such as access to data, or for flashing. In order to ensure security, the sender must authenticate to the receiver before access can be granted. This is done via a challenge response scheme, based on signatures or public key encryption. In all these cases, the sender possesses a key pair, while the corresponding public key is deployed to the receivers. If a sender requests to unlock a receiver, the receiver replies with a challenge to the sender, which is either a message that has to be signed, or an encrypted message that has to be decrypted. In both cases, the sender can only solve the challenge if in possession of the corresponding private key. Thus, the sender replies to the challenge with a signature or plaintext. In the case of successful verification, the receiver then grants access to the sender.

In our context this means that we can use signature schemes, KEMs, or KEXs. The requirements are detailed in Table 3.3.

Properties	Backend	ECU
Latency	<1 s	<1 s
# Executions over product lifetime	unlimited	unlimited
Size of processed data	≤1MB	≤1MB
Physical accessibility	restricted	accessible
Computational power	Server	Embedded
RAM availability	Server	Embedded
Storage availability	Server	Embedded
# Key Pairs	≤10	≤10
Data dissipation	one-to-many	
Life time	<1 year	
Current security level	≤128 bits	

Table 3.2: Requirements of the use case Feature Activation [78, Table 3.5].

Use case 3.7.1 – Key Generation on Device (KeyGen oD). In this use case the embedded device creates a fresh key pair, consisting of a private and public key. It then sends the public key to the certification authority (CA), which has to integrate it into the chain of trust of the corresponding public key infrastructure. Prior to this, the CA checks if the public key is well-formed, and potentially issues a proof-of-possession request to the generating device. This means that it has to answer a signing or decryption challenge that proofs its possession of the corresponding private key. In the case of success, the CA accepts the new public key, signs it and integrates it to the chain of trust. The generating device completes the protocol by verifying the signature issued by the CA.

In our context this means that all sorts of schemes, namely KEMs, KEXs, and signature schemes, can be considered for this use case. The generation of a new key pair is the main operation performed by the embedded device, and therefore the most important criterion for this use case. The requirements are detailed in Table 3.4.

Use case 3.8.1 – Session-based Secure Channel (Ses. SC). The session-based secure channel use case involves a client, usually represented by an embedded device, and a server backend. They want to set up a secure channel to ensure confidential and authentic communication. This channel has to be established over an untrusted communication medium. Depending on the specific security requirements, this can either be achieved through the Transport Layer Security (TLS) protocol, or a Virtual Private Network (VPN).

From a cryptographic point of view, this use case uses symmetric encryption, while the respective symmetric key has to be established through a KEM or KEX scheme. The requirements for KEMs and KEXs are detailed in Table 3.5.

Properties	Tool	ECU
Latency	<1 s	<1 s
# Executions over product lifetime	unlimited	unlimited
Size of processed data	≤32B: ≤1MB	≤32B: ≤1MB
Physical accessibility	restricted	accessible
Computational power	Server	Deep. Embedded : Embedded
RAM availability	Server	Deep. Embedded : Embedded
Storage availability	Server	Deep. Embedded : Embedded
# Key pairs	≤10 : ≤1.000.000	≤10 : ≤1.000.000
Data dissipation	many-to-many	
Life time	>5 years	
Current security level	≤80 bits : ≤128 bits	

Table 3.3: Requirements of the use case Secure Access Control [78, Table 3.16].

3.3 Code-based schemes

3.3.1 KEM

BIKE

Publication: 2017 ●
Standardization: NIST Round 4 ▲
Security: immature □
Patents: none known ▲

Meta criteria

Public Key Size:	1541 B ▲
Secret Key Size:	281 B ▲
Ciphertext Size:	1573 B ▲
Key Generation Time:	600 000 Cycles ▲
Encapsulation Time:	220 000 Cycles ▲
Decapsulation Time:	2 220 000 Cycles □

(a) Level 1

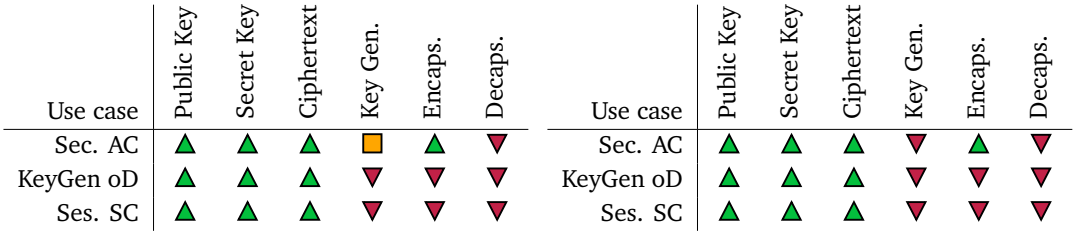
Public Key Size:	3083 B ▲
Secret Key Size:	419 B ▲
Ciphertext Size:	3115 B ▲
Key Generation Time:	1 780 000 Cycles □
Encapsulation Time:	465 000 Cycles ▲
Decapsulation Time:	6 610 000 Cycles □

(b) Level 3

Figure 3.1: Criteria for BIKE from [10].

Properties	**Device**	**CA**
Latency	<1 s	<1 s
# Executions over product lifetime	limited	limited
Size of processed data	≤16KB	≤16KB
Physical accessibility	untrusted	trusted
Computational power	Embedded	Server
RAM availability	Embedded	Server
Storage availability	Embedded	Server
# Key pairs	≤10	>1.000.000
Data dissipation	one-to-one	
Life time	>5 years	
Current security level	≤128 bits	

Table 3.4: Requirements of the use case Key Generation on Device [78, Table 3.24].



(a) Level 1 (b) Level 3

Figure 3.2: Use case suitability for BIKE.

BIKE is a structured code-based Key Encapsulation Mechanism. It offers a balanced performance and key sizes for general use, although it has a slightly slower decapsulation and key generation than structured lattice-based schemes. NIST regards BIKE as a useful fallback scheme in case of cryptanalytic progress on structured lattice based schemes, due to BIKE's well-understood and stable attack complexity [4].

However, NIST reports on doubts about side-channel protection and the CCA security of BIKE, which thus requires further research.

Classic McEliece

Publication:	1978 ●
Standardization:	BSI Recommendation, NIST Round 4 ▲
Security:	well-understood ▲
Patents:	none known ▲

Meta criteria

Public Key Size:	261 120 B □
Secret Key Size:	6452 B ▲
Ciphertext Size:	128 B ▲
Key Generation Time:	58 034 411 Cycles ▼
Encapsulation Time:	44 350 Cycles ▲
Decapsulation Time:	134 745 Cycles ▲

(a) McEliece348864 (L. 1)

Public Key Size:	524 160 B ▼
Secret Key Size:	13 608 B ▲
Ciphertext Size:	188 B ▲
Key Generation Time:	215 785 433 Cycles ▼
Encapsulation Time:	117 782 Cycles ▲
Decapsulation Time:	271 694 Cycles ▲

(b) McEliece460896 (L. 3)

Figure 3.3: Criteria for Classical McEliece from [6].

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▼	▲	▲
KeyGen oD	▲	▲	▲	▼	▲	▼
Ses. SC	▲	▲	▲	▼	▲	▼

(a) McEliece348864 (L. 1)

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▼	▲	▲	▼	▲	▲
KeyGen oD	▼	▲	▲	▼	□	▼
Ses. SC	▼	▲	▲	▼	□	▼

(b) McEliece460896 (L. 3)

Figure 3.4: Use case suitability for Classical McEliece.

Classic McEliece is a code-based Key Encapsulation Mechanism that is based on the McEliece scheme from 1978 using Goppa codes [71]. It is the oldest among the proposed PQC schemes and its security has not been highly impacted through research since its publication. Due to this, NIST considers Classic McEliece to be a stable and conservative choice for suitable applications. Similarly to FrodoKEM (Section 3.5.2), Classic McEliece has also been recommended by the BSI [65].

However, Classic McEliece features very large public keys, which may prevent its usage for many applications. In contrast, its ciphertext sizes and encapsulation and decapsulation performance are competitive with other PQC schemes.

We further note that the NIST Round 2 candidate NTS-KEM merged with the Classic McEliece submission for NIST Round 3, and is thus not listed separately in this document.

HQC

Publication:	2017 ●
Standardization:	NIST Round 4 ▲
Security:	well-understood ▲
Patents:	none known ▲

Meta criteria

Public Key Size:	2249 B ▲
Secret Key Size:	40 B ▲
Ciphertext Size:	4481 B ▲
Key Generation Time:	83 000 Cycles ▲
Encapsulation Time:	197 000 Cycles ▲
Decapsulation Time:	349 000 Cycles ▲

(a) HQC-128 (L. 1)

Public Key Size:	4522 B ▲
Secret Key Size:	40 B ▲
Ciphertext Size:	9026 B ▲
Key Generation Time:	200 000 Cycles ▲
Encapsulation Time:	456 000 Cycles ▲
Decapsulation Time:	740 000 Cycles ▲

(b) HQC-192 (L. 3)

Figure 3.5: Criteria for HQC from [1].

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▲	▲	▲
KeyGen oD	▲	▲	▲	▲	▼	▼
Ses. SC	▲	▲	▲	▲	▼	▼

(a) HQC-128 (L. 1)

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▲	▲	▼
KeyGen oD	▲	▲	▲	▼	▼	▼
Ses. SC	▲	▲	▲	▼	▼	▼

(b) HQC-192 (L. 3)

Figure 3.6: Use case suitability for HQC.

HQC is a code-based Key Encapsulation Mechanism based on the decisional quasi-cyclic syndrome decoding (QCSD) with parity problem. In comparison to the code-based competitor BIKE, HQC features larger public key and ciphertext sizes, but has faster key generation and decapsulation. Nevertheless, as for BIKE, HQC is outperformed by structured lattice-based schemes.

NIST notes that the HQC submission features a thorough security analysis, which favored its consideration for NIST Round 4 [4, 5].

LEDAcrypt

Publication:	2017 ●
Standardization:	NIST Round 2 ■
Security:	immature ■
Patents:	none known ▲

Meta criteria

Public Key Size:	2928 B ▲
Secret Key Size:	50 B ▲
Ciphertext Size:	2952 B ▲
Key Generation Time:	4 457 600 Cycles ■
Encapsulation Time:	243 200 Cycles ▲
Decapsulation Time:	1 795 200 Cycles ■

(a) IND-CCA2 LEDAcrypt-KEM (L. 1)

Public Key Size:	5104 B ▲
Secret Key Size:	66 B ▲
Ciphertext Size:	5136 B ▲
Key Generation Time:	11 132 800 Cycles ■
Encapsulation Time:	620 800 Cycles ▲
Decapsulation Time:	5 411 200 Cycles ■

(b) IND-CCA2 LEDAcrypt-KEM (L. 3)

Figure 3.7: Criteria for LEDAcrypt KEM from [14].

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▼	▲	▼
KeyGen oD	▲	▲	▲	▼	▼	▼
Ses. SC	▲	▲	▲	▼	▼	▼

(a) IND-CCA2 LEDAcrypt-KEM (L. 1)

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▼	■	▼
KeyGen oD	▲	▲	▲	▼	▼	▼
Ses. SC	▲	▲	▲	▼	▼	▼

(b) IND-CCA2 LEDAcrypt-KEM (L. 3)

Figure 3.8: Use case suitability for LEDAcrypt KEM.

LEDAcrypt [15] is a structured code-based Key Encapsulation Mechanism that uses a similar construction as BIKE. However, its differing design choices allowed for an attack on the Round 2 submission, which identified a large class of weak keys (see [4]).

As a countermeasure, the LEDAcrypt team proposed several changes that made the scheme very similar to BIKE. This, combined with the large differences to the initial proposal of LEDAcrypt, led NIST to not considering LEDAcrypt for Round 3 [4].

ROLLO

Publication:	2018 ●
Standardization:	NIST Round 2 ■
Security:	immature ■
Patents:	none known ▲

Meta criteria

Public Key Size:	1941 B ▲	**Public Key Size**:	2341 B ▲
Secret Key Size:	40 B ▲	**Secret Key Size**:	40 B ▲
Ciphertext Size:	2089 B ▲	**Ciphertext Size**:	2469 B ▲
Key Generation Time:	3 690 000 Cycles ■	**Key Generation Time**:	3 689 000 Cycles ■
Encapsulation Time:	331 000 Cycles ▲	**Encapsulation Time**:	334 000 Cycles ▲
Decapsulation Time:	1 195 000 Cycles ■	**Decapsulation Time**:	1 258 000 Cycles ■

(a) ROLLO-II-128 (L. 1) (b) ROLLO-II-192 (L. 3)

Figure 3.9: Criteria for ROLLO-II from [12].

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▼	▲	▼
KeyGen oD	▲	▲	▲	▼	▼	▼
Ses. SC	▲	▲	▲	▼	▼	▼

(a) ROLLO-II-128 (L. 1)

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▼	▲	▼
KeyGen oD	▲	▲	▲	▼	▼	▼
Ses. SC	▲	▲	▲	▼	▼	▼

(b) ROLLO-II-192 (L. 3)

Figure 3.10: Criteria for ROLLO-I.

ROLLO is a code-based Key Encapsulation Mechanism that resulted from the merge of the three NIST Round 1 candidates LAKE, LOCKER, and Ouroboros-R. Its security is based on the rank syndrome decoding (RSD) problem. ROLLO was subject to an algebraic attack that significantly reduced the security of the proposed parameter sets. In order to mitigate this attack, larger parameter sets have been proposed [12].

However, NIST states that the security analysis of ROLLO requires more research and time to mature [4]. Thus, ROLLO did not advance to NIST Round 3.

Publication:	2017 ●
Standardization:	NIST Round 2 ■
Security:	immature ■
Patents:	none known ▲

Meta criteria

Public Key Size:	1834 B ▲	**Public Key Size**:	2853 B ▲
Secret Key Size:	40 B ▲	**Secret Key Size**:	40 B ▲
Ciphertext Size:	3652 B ▲	**Ciphertext Size**:	5690 B ▲
Key Generation Time:	370 000 Cycles ▲	**Key Generation Time**:	760 000 Cycles ▲
Encapsulation Time:	530 000 Cycles ▲	**Encapsulation Time**:	1 160 000 Cycles ■
Decapsulation Time:	2 580 000 Cycles ■	**Decapsulation Time**:	5 650 000 Cycles ■

(a) RQC-128 (L. 1) (b) RQC-192 (L. 3)

Figure 3.11: Criteria for RQC from [3].

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▲	■	▼
KeyGen oD	▲	▲	▲	▼	▼	▼
Ses. SC	▲	▲	▲	▼	▼	▼

(a) RQC-128 (L. 1)

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▼	▼	▼
KeyGen oD	▲	▲	▲	▼	▼	▼
Ses. SC	▲	▲	▲	▼	▼	▼

(b) RQC-192 (L. 3)

Figure 3.12: Use case suitability for RQC.

RQC is a code-based Key Encapsulation Mechanism based on the rank syndrome decoding (RSD) problem. Similar to ROLLO, RQC was subject to an algebraic attack that broke all proposed parameter sets (see [4]).

Although larger parameter sets were proposed, and performance and key sizes remain competitive, RQC did not advance to NIST Round 3. Analogously to ROLLO, more research on the security of rank-based cryptography is required for potential practical applications of RQC.

3.3.2 Suitability for QuantumRISC

Code-based KEMs can be summarized as being relatively efficient, but requiring rather large public keys or ciphertexts. This is especially true for Classic McEliece, which is considered to be a conservative option in terms of security due to its long standing history without being majorly affected by new attacks. However, the large public keys of Classic McEliece may be problematic for its usage on memory-constrained devices. Alternatives like BIKE or HQC require smaller public keys, but feature slower running times. The fourth round of the NIST PQC standardization process furthermore includes these three schemes, i.e., BIKE, Classic McEliece, and HQC.

For QuantumRISC, we mainly focus on Classic McEliece, due to its high likeliness of being standardized and the existing BSI recommendation for its usage.

3.4 Isogeny-based schemes

3.4.1 KEM

SIKE

Publication:	2011 ●
Standardization:	NIST Round 4 ▲
Security:	broken ▼
Patents:	none known ▲

Meta criteria

Public Key Size:	330 B ▲
Secret Key Size:	374 B ▲
Ciphertext Size:	346 B ▲
Key Generation Time:	5 927 000 Cycles ■
Encapsulation Time:	9 681 000 Cycles ■
Decapsulation Time:	10 343 000 Cycles ■

(a) SIKEp434 (L. 1)

Public Key Size:	462 B ▲
Secret Key Size:	524 B ▲
Ciphertext Size:	486 B ▲
Key Generation Time:	14 890 000 Cycles ■
Encapsulation Time:	27 254 000 Cycles ▼
Decapsulation Time:	27 445 000 Cycles ▼

(b) SIKEp610 (L. 3)

Figure 3.13: Criteria for SIKE from [61].

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▼	▼	▼
KeyGen oD	▲	▲	▲	▼	▼	▼
Ses. SC	▲	▲	▲	▼	▼	▼

(a) SIKEp434 (L. 1)

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▼	▼	▼
KeyGen oD	▲	▲	▲	▼	▼	▼
Ses. SC	▲	▲	▲	▼	▼	▼

(b) SIKEp610 (L. 3)

Figure 3.14: Use case suitability for SIKE.

SIKE is an isogeny-based Key Encapsulation Mechanism, based on the key exchange scheme SIDH [60]. Its security is based on the problem of finding an isogeny between two supersingular elliptic curves, which is a relatively young security assumption. SIKE has the smallest key and ciphertext sizes among all NIST candidates, but is more than an order of magnitude slower than most other KEMs.

SIKE further offers an optional key compression, where public key and ciphertext sizes can be compressed by roughly 40% at the cost of a performance overhead of factor 1.5-2, see [61].

As a promising scheme for applications that require small keys and ciphertexts, SIKE advanced to NIST Round 4 [5]. However, shortly after SIKE moved to NIST Round 4, a series of attack papers led to a polynomial time attack against SIKE [31, 70, 83]. Thus, SIKE as specified in [61] is broken.

3.4.2 KEX

CSIDH

Publication:	2018 ●
Standardization:	no data ●
Security:	debated ■
Patents:	none known ▲

Meta criteria

Public Key Size:	64 B ▲
Secret Key Size:	32 B ▲
Key Pair Size:	96 B ▲
Key Generation Time:	89 110 000 Cycles ▼
Key Exchange Time:	93 230 000 Cycles ▼

(a) CSIDH-512 (L. 1)

Public Key Size:	128 B ▲
Secret Key Size:	64 B ▲
Key Pair Size:	192 B ▲
Key Generation Time:	469 520 000 Cycles ▼
Key Exchange Time:	511 190 000 Cycles ▼

(b) CSIDH-1024 (L. 3)

Figure 3.15: Criteria for CSIDH from [16].

Use case	Public Key	Secret Key	Key Pair	Key Gen.	Key Ex.	Ephemeral
Sec. AC	▲	▲	▲	▼	▼	▼
KeyGen oD	▲	▲	▲	▼	▼	▼
Ses. SC	▲	▲	▲	▼	▼	▼

(a) CSIDH-512 (L. 1)

Use case	Public Key	Secret Key	Key Pair	Key Gen.	Key Ex.	Ephemeral
Sec. AC	▲	▲	▲	▼	▼	▼
KeyGen oD	▲	▲	▲	▼	▼	▼
Ses. SC	▲	▲	▲	▼	▼	▼

(b) CSIDH-1024 (L. 3)

Figure 3.16: Use case suitability for CSIDH.

CSIDH [32] is an isogeny-based Key Exchange. It was published in 2018, and thus does not participate in the NIST standardization process, which started in 2017. CSIDH is a non-interactive scheme, which makes it unique among the PQC schemes, and allows for using it as a potential drop-in replacement for Diffie-Hellman key exchange schemes. The public key sizes are smaller than in SIKE, while the performance is an order of magnitude slower.

Similar to SIKE, the security of CSIDH is based on an isogeny finding problem. However, unlike SIKE, CSIDH possesses a commutative structure, which allows for the application of Kuperberg's subexponential quantum algorithm [66]. The exact implications of this on the parameter choices are heavily debated. The parameter sets presented in Figure 3.15 are aggressive choices, which may need to be scaled up in the future. Note that the mentioned attacks against SIKE do not apply to CSIDH.

3.4.3 Suitability for QuantumRISC

All isogeny-based schemes show a clear trend: While their key and ciphertext sizes are very small and thus a perfect fit for all of our use cases, the running times seem to be too slow for our applications. However, isogeny-based cryptography is the youngest among the known PQC families, and thus may receive further attention with regards to optimizations in its performance. Furthermore, in many situations larger keys may lead to bandwidth latency, which additionally favors isogeny-based schemes.

While recently attacks that run in polynomial time against SIKE and variants like SIDH [60] or B-SIDH [40] were found, they do not affect the security of CSIDH or the very recent isogeny-based signature scheme SQISign [48]. Thus, isogeny-based schemes remain an interesting option for practical applications of PQC. In particular, SQISign might be considered in future rounds of the NIST standardization process. It features the smallest combined public key and signature sizes among PQC signatures, yet suffers from relatively slow running times, partly due to the involved choice of parameters [46, 41, 24].

3.5 Lattice-based schemes

3.5.1 Signature

CRYSTALS-Dilithium

Publication:	2017 ●
Standardization:	NIST standardization choice ▲
Security:	immature ■
Patents:	debated ■

Meta criteria

Number of Signatures:	∞ # ▲	**Number of Signatures**:	∞ # ▲
Public Key Size:	1312 B ▲	**Public Key Size**:	1952 B ▲
Secret Key Size:	96 B ▲	**Secret Key Size**:	96 B ▲
Signature Size:	2420 B ▲	**Signature Size**:	3293 B ▲
Key Generation Time:	124 031 Cycles ▲	**Key Generation Time**:	256 403 Cycles ▲
Signing Time:	333 013 Cycles ▲	**Signing Time**:	529 106 Cycles ▲
Verification Time:	118 412 Cycles ▲	**Verification Time**:	179 424 Cycles ▲

(a) Dilithium2 (L. 2) (b) Dilithium3 (L. 3)

Figure 3.17: Criteria for CRYSTALS-Dilithium from [42].

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▲	▲	▲	▲	▲
Feat. Ac.	▲	▲	▲	▲	▲	▲	▲
Sec. AC	▲	▲	▲	▲	▲	▲	▲
KeyGen oD	▲	▲	▲	▲	■	▼	■

(a) Dilithium2 (L. 2)

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▲	▲	▲	▲	▲
Feat. Ac.	▲	▲	▲	▲	▲	■	▲
Sec. AC	▲	▲	▲	▲	▲	■	▲
KeyGen oD	▲	▲	▲	▲	▼	▼	▼

(b) Dilithium3 (L. 3)

Figure 3.18: Use case suitability for CRYSTALS-Dilithium.

CRYSTALS-Dilithium [53] is a lattice-based signature scheme based on lattice problems over module lattices. The hard underlying problems are module learning with errors (MLWE) and module short integer solutions (MSIS). Signature and key sizes, and the performance of Dilithium are rather balanced, which opens it to a wide area of potential applications. In addition to the instantiations from Figure 3.17, Dilithium includes the variant Dilithium-AES that uses hardware supported AES-256 in counter mode instead of SHAKE. This improves the cycle counts by 25-40% compared to Figure 3.17 [42].

However, Dilithium relies on structured lattices, for which security analyses and parameter choices would benefit from further research [4].

Due to its usability as general purpose signature scheme, NIST decided to standardize Dilithium after Round 3 [5].

FALCON

Publication:	2017 ●
Standardization:	NIST standardization choice ▲
Security:	immature ■
Patents:	debated ■

Meta criteria

Number of Signatures:	∞ # ▲	**Number of Signatures:**	∞ # ▲
Public Key Size:	897 B ▲	**Public Key Size:**	1793 B ▲
Secret Key Size:	768 B ▲	**Secret Key Size:**	1280 B ▲
Signature Size:	666 B ▲	**Signature Size:**	1280 B ▲
Key Generation Time:	19 458 000 Cycles ■	**Key Generation Time:**	63 135 000 Cycles ▼
Signing Time:	386 678 Cycles ▲	**Signing Time:**	789 564 Cycles ▲
Verification Time:	82 340 Cycles ▲	**Verification Time:**	168 498 Cycles ▲

(a) Falcon-512 (L. 1) (b) Falcon-1024 (L. 5)

Figure 3.19: Criteria for Falcon from [81].

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▲	▲	▼	▲	▲
Feat. Ac.	▲	▲	▲	▲	▼	▲	▲
Sec. AC	▲	▲	▲	▲	▼	▲	▲
KeyGen oD	▲	▲	▲	▲	▼	▼	▲

(a) Falcon-512 (L. 1)

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▲	▲	▼	▲	▲
Feat. Ac.	▲	▲	▲	▲	▼	▼	▲
Sec. AC	▲	▲	▲	▲	▼	▼	▲
KeyGen oD	▲	▲	▲	▲	▼	▼	▼

(b) Falcon-1024 (L. 5)

Figure 3.20: Use case suitability for Falcon.

Falcon [81] is a lattice-based signature scheme based on the short integer solutions (SIS) problem over NTRU lattices. In terms of signature and key sizes, and performance, Falcon is comparable to Dilithium, while Falcon is harder to implement due to its usage of floating-point arithmetic.

Similar to Dilithium, the security of Falcon would benefit from further research due to its rather low CoreSVP security strength [4].

Due to its slightly better performance profile, NIST decided to standardize Falcon after Round 3. It is recommended for situations where the performance of Dilithium is not sufficient [5].

qTESLA

Publication:	2017
Standardization:	NIST Round 2
Security:	immature
Patents:	debated

Meta criteria

Number of Signatures:	∞ #
Public Key Size:	14 880 B
Secret Key Size:	5224 B
Signature Size:	2592 B
Key Generation Time:	2 358 600 Cycles
Signing Time:	2 299 000 Cycles
Verification Time:	814 300 Cycles

(a) qTESLA-p-I (L. 1)

Number of Signatures:	∞ #
Public Key Size:	38 432 B
Secret Key Size:	12 392 B
Signature Size:	5664 B
Key Generation Time:	13 151 400 Cycles
Signing Time:	5 212 300 Cycles
Verification Time:	2 102 300 Cycles

(b) qTESLA-p-III (L. 3)

Figure 3.21: Criteria for qTESLA from [20].

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▲	▲	▲	▲	▼
Feat. Ac.	▲	▲	▲	▲	▼	▼	▼
Sec. AC	▲	▲	▲	▲	▼	▼	▼
KeyGen oD	▲	▲	▲	▲	▼	▼	▼

(a) qTESLA-p-I (L. 1)

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▲	▲	▼	□	▼
Feat. Ac.	▲	▲	▲	▲	▼	▼	▼
Sec. AC	▲	▲	▲	▲	▼	▼	▼
KeyGen oD	▲	▲	▲	▲	▼	▼	▼

(b) qTESLA-p-III (L. 3)

Figure 3.22: Use case suitability for qTESLA.

qTESLA is a lattice-based signature scheme based on structured lattice assumptions. In NIST Round 2, 10 of the initially 12 parameter sets were retracted due to doubts about their security assumptions. Since the remaining two parameter sets (see Figure 3.21) gain no advantage over Dilithium or Falcon with regards to signature sizes, key sizes, or performance, qTESLA did not advance to NIST Round 3 [4].

3.5.2 KEM

CRYSTALS-Kyber

Publication:	2017 ●
Standardization:	NIST standardization choice ▲
Security:	immature ■
Patents:	debated ■

Meta criteria

Public Key Size:	800 B ▲	**Public Key Size**:	1184 B ▲
Secret Key Size:	1632 B ▲	**Secret Key Size**:	2400 B ▲
Ciphertext Size:	768 B ▲	**Ciphertext Size**:	1088 B ▲
Key Generation Time:	33 856 Cycles ▲	**Key Generation Time**:	52 732 Cycles ▲
Encapsulation Time:	45 200 Cycles ▲	**Encapsulation Time**:	67 624 Cycles ▲
Decapsulation Time:	34 572 Cycles ▲	**Decapsulation Time**:	53 156 Cycles ▲

(a) Kyber-512 (L. 1) (b) Kyber-768 (L. 3)

Figure 3.23: Criteria for CRYSTALS-Kyber from [43].

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▲	▲	▲
KeyGen oD	▲	▲	▲	▲	▲	▲
Ses. SC	▲	▲	▲	▲	▲	▲

(a) Kyber-512 (L. 1)

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▲	▲	▲
KeyGen oD	▲	▲	▲	▲	▲	▲
Ses. SC	▲	▲	▲	▲	▲	▲

(b) Kyber-1024 (L. 3)

Figure 3.24: Use case suitability for CRYSTALS-Kyber.

CRYSTALS-Kyber [21] is a lattice-based Key Encapsulation Mechanism that shares a common framework with the CRYSTALS-Dilithium signature scheme. Thus, it relies on the hardness of the module learning with errors (MLWE) problem. In comparison to other PQC schemes, Kyber offers a relatively fast performance and small key sizes, which potentially makes it a suitable candidate for many practical applications. In addition to the instantiations from Figure 3.23, the Kyber Round 3 submission includes the variant Kyber-90s that uses hardware supported AES-256 in counter mode instead of SHAKE. This improves the presented cycle counts by roughly 40% compared to Figure 3.23.

As for Dilithium, the relatively low CoreSVP security strength may present a concern on the security of Kyber [4]. Thus, more research on this topic is required.

NIST decided to standardize Kyber after Round 3 because it has confidence in assumptions based on the MLWE problem, the specification of the scheme provided an extended security analysis, and the performance of Kyber is very good [5].

FrodoKEM

Publication:	2016 ●
Standardization:	NIST Round 3 (alternate candidate), BSI Recommendation ▲
Security:	well-understood ▲
Patents:	none known ▲

Meta criteria

Public Key Size:	9616 B ▲	**Public Key Size**:	15 632 B ▲
Secret Key Size:	19 888 B ▲	**Secret Key Size**:	31 296 B ▲
Ciphertext Size:	9720 B ▲	**Ciphertext Size**:	15 744 B ▲
Key Generation Time:	1 387 000 Cycles ■	**Key Generation Time**:	2 846 000 Cycles ■
Encapsulation Time:	1 634 000 Cycles ■	**Encapsulation Time**:	3 047 000 Cycles ■
Decapsulation Time:	1 531 000 Cycles ■	**Decapsulation Time**:	2 894 000 Cycles ■

(a) FrodoKEM-640-AES (L. 1) (b) FrodoKEM-976-AES (L. 3)

Figure 3.25: Criteria for FrodoKEM from [74].

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▼	▼	▼
KeyGen oD	▲	▲	▲	▼	▼	▼
Ses. SC	▲	▲	▲	▼	▼	▼

(a) FrodoKEM-640-AES (L. 1)

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▼	▼	▼
KeyGen oD	▲	▲	▲	▼	▼	▼
Ses. SC	▲	▲	▲	▼	▼	▼

(b) FrodoKEM-976-AES (L. 3)

Figure 3.26: Use case suitability for FrodoKEM.

FrodoKEM [22] is a lattice-based Key Encapsulation Mechanism that relies on the hardness of the plain LWE problem. Thus, among the lattice-based NIST candidates, FrodoKEM uses the least amount of structure in its lattices, which means that is it less likely to be susceptible to algebraic attacks. However, this potential security advantage leads to worse performance and larger key sizes compared to other lattice-based schemes.

In addition to advancing to NIST Round 3 as alternate candidate, FrodoKEM has been recommended by the BSI [65]. However, among the lattice-based KEMs, NIST decided to only standardize Kyber after Round 3. Due to NIST's endeavor to not only rely on lattice problems, other lattice-based KEMs like FrodoKEM did not proceed to Round 4 [5].

LAC

Publication:	2017 ●
Standardization:	NIST Round 2 ■
Security:	immature ■
Patents:	debated ■

Meta criteria

Public Key Size:	544 B ▲	**Public Key Size**:	1056 B ▲
Secret Key Size:	1056 B ▲	**Secret Key Size**:	2080 B ▲
Ciphertext Size:	712 B ▲	**Ciphertext Size**:	1188 B ▲
Key Generation Time:	59 584 Cycles ▲	**Key Generation Time**:	119 246 Cycles ▲
Encapsulation Time:	89 055 Cycles ▲	**Encapsulation Time**:	137 653 Cycles ▲
Decapsulation Time:	103 229 Cycles ▲	**Decapsulation Time**:	224 249 Cycles ▲

(a) LAC.CCA-128 (L. 1) (b) LAC.CCA-192 (L. 3)

Figure 3.27: Criteria for LAC.CCA from [68].

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▲	▲	▲
KeyGen oD	▲	▲	▲	▲	▲	■
Ses. SC	▲	▲	▲	▲	▲	■

(a) LAC.CCA-128 (L. 1)

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▲	▲	▲
KeyGen oD	▲	▲	▲	■	▼	▼
Ses. SC	▲	▲	▲	■	▼	▼

(b) LAC.CCA-192 (L. 3)

Figure 3.28: Use case suitability for LAC.CCA.

LAC [69] is a lattice-based Key Encapsulation Mechanism based on the RLWE problem. Its design features error-correcting codes that can correct decryption failures, which means that a higher decryption failure rate can be tolerated. Although this feature enables very efficient performance, it allowed for several attacks that reduced the security of LAC during NIST Round 1 [4]. LAC was modified to mitigate these attacks, yet some more security issues were discovered in the NIST Round 2 submission. Thus, NIST decided not to consider LAC for Round 3 since much more research would be required to gain confidence in the security of the scheme.

NewHope

Publication:	2015
Standardization:	NIST Round 2
Security:	immature
Patents:	debated

Meta criteria

Public Key Size:	928 B
Secret Key Size:	1888 B
Ciphertext Size:	1120 B
Key Generation Time:	68 080 Cycles
Encapsulation Time:	109 836 Cycles
Decapsulation Time:	114 176 Cycles

(a) NewHope512-CCA-KEM (L. 1)

Public Key Size:	1824 B
Secret Key Size:	3680 B
Ciphertext Size:	2208 B
Key Generation Time:	129 670 Cycles
Encapsulation Time:	210 092 Cycles
Decapsulation Time:	220 864 Cycles

(b) NewHope1024-CCA-KEM (L. 5)

Figure 3.29: Criteria for NewHope from [80].

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▲	▲	▲
KeyGen oD	▲	▲	▲	▲	■	■
Ses. SC	▲	▲	▲	▲	■	■

(a) NewHope512

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▲	▲	▲
KeyGen oD	▲	▲	▲	▼	▼	▼
Ses. SC	▲	▲	▲	▼	▼	▼

(b) NewHope1024

Figure 3.30: Use case suitability for NewHope.

NewHope [9] is a lattice-based Key Encapsulation Mechanism based on the RLWE problem. Its performance profile and key sizes are competitive to other lattice-based KEMs, where the performance benefits from the structure of the lattices used in NewHope.

However, the fact that RLWE schemes are the most structured among the PQC candidates may lead to security concerns. Due to this, NIST developed a preference for MLWE schemes with less structure, since NewHope does not offer a significant advantage over schemes like DILITHIUM-Kyber [4]. Thus, NewHope was not considered for NIST Round 3.

Publication:	1996 ●
Standardization:	NIST Round 3 ▲
Security:	immature ■
Patents:	expired ▲

Meta criteria

Public Key Size:	931 B ▲	**Public Key Size**:	1230 B ▲
Secret Key Size:	1235 B ▲	**Secret Key Size**:	1592 B ▲
Ciphertext Size:	931 B ▲	**Ciphertext Size**:	1230 B ▲
Key Generation Time:	309 216 Cycles ▲	**Key Generation Time**:	431 667 Cycles ▲
Encapsulation Time:	83 519 Cycles ▲	**Encapsulation Time**:	98 809 Cycles ▲
Decapsulation Time:	59 729 Cycles ▲	**Decapsulation Time**:	75 384 Cycles ▲

(a) ntruhps2048677 (L. 1) (b) ntruhps4096821 (L.3)

Figure 3.31: Criteria for NTRU from [35].

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▲	▲	▲
KeyGen oD	▲	▲	▲	▼	▲	▲
Ses. SC	▲	▲	▲	▼	▲	▲

(a) ntruhps2048677 (L. 1)

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▲	▲	▲
KeyGen oD	▲	▲	▲	▼	▲	▲
Ses. SC	▲	▲	▲	▼	▲	▲

(b) ntruhps4096821 (L. 3)

Figure 3.32: Use case suitability for NTRU.

NTRU [58] is a lattice-based Key Encapsulation Mechanism that uses structured lattices, but differs from the RLWE or MLWE assumptions of other lattice-based candidates. It offers small key sizes and an efficient performance, yet it is outperformed by schemes like CRYSTALS-Kyber or SABER.

The long history of NTRU means that new algebraic attacks on its structured lattices and new patent claims are rather unlikely. However, parameters for the concrete instantiations of NTRU are based on two different cost metrics. It remains to be shown which cost metric is suitable in practice, and how they translate to the cost metrics of other lattice-based schemes. The numbers of Figure 3.31 refer to security levels in the more conservative cost metric.

Similar to FrodoKEM, NTRU did not proceed to NIST Round 4. However, NIST states that if potential patent problems with respect to Kyber cannot be resolved, it may standardize NTRU instead [5].

NTRU Prime

Publication:	2016 ●
Standardization:	NIST Round 3 (alternate candidate) ▲
Security:	immature ■
Patents:	none known ▲

(a) Meta criteria

Public Key Size:	994 B ▲
Secret Key Size:	1518 B ▲
Ciphertext Size:	897 B ▲
Key Generation Time:	716 209 Cycles ▲
Encapsulation Time:	44 155 Cycles ▲
Decapsulation Time:	55 778 Cycles ▲

(b) sntrup653 (L. 1)

Public Key Size:	1505 B ▲
Secret Key Size:	2254 B ▲
Ciphertext Size:	1349 B ▲
Key Generation Time:	1 523 540 Cycles ■
Encapsulation Time:	62 704 Cycles ▲
Decapsulation Time:	80 654 Cycles ▲

(c) sntrup953 (L. 3/4)

Figure 3.33: Criteria for NTRU Prime.

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▼	▲	▲
KeyGen oD	▲	▲	▲	▼	▲	▲
Ses. SC	▲	▲	▲	▼	▲	▲

(a) sntrup653 (L. 1)

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▼	▲	▲
KeyGen oD	▲	▲	▲	▼	▲	▲
Ses. SC	▲	▲	▲	▼	▲	▲

(b) sntrup953 (L. 3/4)

Figure 3.34: Use case suitability for NTRU Prime.

NTRU Prime [17] contains two lattice-based KEMs. Streamlined NTRU Prime is based on NTRU with a quotient structure in public keys, while NTRU LPRime is based on RLWE with a product structure in public keys. A central difference between NTRU Prime and other structured lattice-based schemes is the usage of a different ring structure, in particular, the abandonment of cyclotomic rings. This variation may be of special interest if attacks that specifically require the structure of cyclotomic rings are found.

Both KEM variants of NTRU Prime offer efficient performance and key sizes, which are competitive with other lattice-based KEMs. The data presented in Figure 3.33 refers to Streamlined NTRU Prime. In comparison to this, NTRU LPRime provides much faster key generation (roughly 20-40 times faster), slightly slower encapsulation and decapsulation (roughly 50% slower), and comparative key sizes.

We note that NIST voiced doubt regarding some of the security categories assigned to the NTRU Prime parameter sets [4].

Similar to FrodoKEM and NTRU, NTRU Prime did not proceed to NIST Round 4 [5].

Publication:	2018
Standardization:	NIST Round 2
Security:	immature
Patents:	unclear

Meta criteria

Public Key Size:	676 B
Secret Key Size:	16 B
Ciphertext Size:	740 B
Key Generation Time:	54 000 Cycles
Encapsulation Time:	67 000 Cycles
Decapsulation Time:	86 000 Cycles

(a) R5ND_1CCA_od (L. 1)

Public Key Size:	983 B
Secret Key Size:	32 B
Ciphertext Size:	1103 B
Key Generation Time:	82 000 Cycles
Encapsulation Time:	106 000 Cycles
Decapsulation Time:	133 000 Cycles

(b) R5ND_3CCA_od (L. 3)

Figure 3.35: Criteria for Round5-KEM from [54].

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▲	▲	▲
KeyGen oD	▲	▲	▲	▲	▲	▲
Ses. SC	▲	▲	▲	▲	▲	▲

(a) R5ND_1CCA_od (L. 1)

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▲	▲	▲
KeyGen oD	▲	▲	▲	▼	▼	▼
Ses. SC	▲	▲	▲	▼	▼	▼

(b) R5ND_3CCA_od (L. 3)

Figure 3.36: Use case suitability for Round5-KEM.

Round5 [13] is a lattice-based Key Encapsulation Mechanism based on the learning with rounding problem, and also features a PKE variant. It was formed through a merge of the NIST Round 1 candidates Round2 and Hila5. Round5 offers a very efficient performance as well as small key sizes, which are competitive to schemes like CRYSTALS-Kyber or SABER.

However, concerns were raised about the security of the scheme, which led to updates to the proposed parameters. Nevertheless, NIST reports that the rather complicated specification and a lack of confidence in some building blocks of Round5 resulted in not considering the scheme for NIST Round 3 [4]. Furthermore, the Round5 submission does not contain a royalty-free license.

Publication:	2017
Standardization:	NIST Round 3
Security:	immature
Patents:	debated

Meta criteria

Public Key Size:	672 B
Secret Key Size:	1568 B
Ciphertext Size:	736 B
Key Generation Time:	45 232 Cycles
Encapsulation Time:	62 236 Cycles
Decapsulation Time:	62 624 Cycles

(a) LightSABER (L. 1)

Public Key Size:	992 B
Secret Key Size:	2304 B
Ciphertext Size:	1088 B
Key Generation Time:	80 340 Cycles
Encapsulation Time:	103 204 Cycles
Decapsulation Time:	103 092 Cycles

(b) SABER (L. 3)

Figure 3.37: Criteria for SABER from [44].

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▲	▲	▲
KeyGen oD	▲	▲	▲	▲	▲	▲
Ses. SC	▲	▲	▲	▲	▲	▲

(a) LightSABER (L. 1)

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▲	▲	▲
KeyGen oD	▲	▲	▲	▲	■	■
Ses. SC	▲	▲	▲	▲	■	■

(b) SABER (L. 3)

Figure 3.38: Use case suitability for SABER.

SABER [45] is a lattice-based Key Encapsulation Mechanism based on the Module Learning With Rounding (MLWR) problem, which is a variant of MLWE. NIST states that a mild concern lies in the fact that reduction from MLWR to MLWE are not applicable to SABER [4]. SABER is among the fastest lattice-based KEMs with competitive key sizes, and hence is comparable to CRYSTALS-Kyber or Round5.

It should be noted that the applicability of a known patent to SABER is currently heavily debated on the NIST PQC forum [77]. Thus, this patent could potentially hamper the adaption of SABER in practice.

Similar to FrodoKEM and NTRU, SABER did not proceed to NIST Round 4 [5], due to NIST's preference for CRYSTALS-Kyber.

Three Bears

Publication:	2017 ●
Standardization:	NIST Round 2 ■
Security:	immature ■
Patents:	none known ▲

Meta criteria

Public Key Size:	804 B ▲
Secret Key Size:	40 B ▲
Ciphertext Size:	917 B ▲
Key Generation Time:	41 000 Cycles ▲
Encapsulation Time:	62 000 Cycles ▲
Decapsulation Time:	28 000 Cycles ▲

(a) BabyBear (L. 2)

Public Key Size:	1194 B ▲
Secret Key Size:	40 B ▲
Ciphertext Size:	1307 B ▲
Key Generation Time:	84 000 Cycles ▲
Encapsulation Time:	103 000 Cycles ▲
Decapsulation Time:	34 000 Cycles ▲

(b) MamaBear (L. 4)

Figure 3.39: Criteria for Three Bears from [57].

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▲	▲	▲
KeyGen oD	▲	▲	▲	▲	▲	▲
Ses. SC	▲	▲	▲	▲	▲	▲

(a) BabyBear (L. 2)

Use case	Public Key	Secret Key	Ciphertext	Key Gen.	Encaps.	Decaps.
Sec. AC	▲	▲	▲	▲	▲	▲
KeyGen oD	▲	▲	▲	▲	■	▲
Ses. SC	▲	▲	▲	▲	■	▲

(b) MamaBear (L. 4)

Figure 3.40: Use case suitability for Three Bears.

Three Bears is a lattice-based Key Encapsulation Mechanism based on the Integer Module Learning With Errors (I-MLWE) problem. It features a highly efficient performance and competitive key sizes, comparable to the most efficient lattice-based KEMs from above.

From a security point of view, however, Three Bears is rather immature. I-MLWE is a new variant of the MLWE problem that was introduced along with the Three Bears scheme. Although asymptotic security reductions between I-MLWE and MLWE exist, the I-MLWE problem has not been rigorously reviewed by the broader research community yet. Thus, NIST chose not to consider Three Bears for Round 3 of the standardization process.

3.5.3 Suitability for QuantumRISC

Lattice-based schemes feature the best overall performance profile. That is, schemes are relatively balanced between key sizes, signature sizes, and running times, which could be beneficial for many applications, even when including resource-constrained devices.

Among the lattice-based KEMs, CRYSTALS-Kyber and Saber feature the best performance profiles, while other schemes like NTRU, NTRU Prime, and FrodoKEM are designed more conservatively in terms of security considerations, and are therefore not as fast and compact.

Among the lattice-based signature schemes, CRYSTALS-Dilithium and FALCON are promising schemes that moved to Round 3 of the NIST standardization.

In QuantumRISC, we mainly focus on the CRYSTALS schemes due to their excellent performance profile, and FrodoKEM because of its more conservative security design and the BSI recommendation.

This mostly aligns with NIST's decisions to standardize CRYSTALS-Dilithium and CRYSTALS-Kyber as main recommendations [5].

3.6 Multivariate schemes

3.6.1 Signature

GeMSS

Publication:	2017 ●
Standardization:	NIST Round 3 (alternate candidate) ▲
Security:	immature ■
Patents:	none known ▲

Meta criteria

Number of Signatures:	∞ # ▲
Public Key Size:	352 190 B ▼
Secret Key Size:	128 B ▲
Signature Size:	258 B ▲
Key Generation Time:	38 700 000 Cycles ▼
Signing Time:	531 000 000 Cycles ▼
Verification Time:	106 000 Cycles ▲

(a) GeMSS128 (L. 1)

Number of Signatures:	∞ # ▲
Public Key Size:	1 237 960 B ▼
Secret Key Size:	192 B ▲
Signature Size:	411 B ▲
Key Generation Time:	175 000 000 Cycles ▼
Signing Time:	1 800 000 000 Cycles ▼
Verification Time:	304 000 Cycles ▲

(b) GeMSS192 (L. 3)

Figure 3.41: Criteria for GeMSS from [30].

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▲	▲	▼	▼	▲
Feat. Ac.	▲	▲	▲	▲	▼	▼	▲
Sec. AC	▲	▼	▲	▲	▼	▼	▲
KeyGen oD	▲	▼	▲	▲	▼	▼	■

(a) GeMSS128 (L. 1)

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▲	▲	▼	▼	▲
Feat. Ac.	▲	▲	▲	▲	▼	▼	▲
Sec. AC	▲	▼	▲	▲	▼	▼	▲
KeyGen oD	▲	▼	▲	▲	▼	▼	▼

(b) GeMSS192 (L. 3)

Figure 3.42: Use case suitability for GeMSS.

GeMSS is a multivariate-based signature scheme based on the HFEv construction. Its signature sizes are among the smallest and verification performance among the most efficient of all PQC schemes. However, the signing performance is comparably slow and public keys are large, which makes GeMSS appropriate mainly for offline signing applications. Furthermore, NIST notes that GeMSS is difficult to implement on low-end devices [4].

Like Rainbow, ROLLO, and RQC, GeMSS was subject to an algebraic attack. However, this attack did not break any of the GeMSS parameter sets (see [4]). GeMSS mainly competes with Rainbow, which offers faster signing and smaller public keys, while signatures are slightly larger.

Due to cryptanalytic progress, such as efficient key-recovery attacks [88], GeMSS did not advance to NIST Round 4 [5].

LUOV

Publication:	2017 ●
Standardization:	NIST Round 2 ■
Security:	immature ■
Patents:	none known ▲

Meta criteria

Number of Signatures:	∞ # ▲
Public Key Size:	11 500 B ▲
Secret Key Size:	32 B ▲
Signature Size:	239 B ▲
Key Generation Time:	1 100 000 Cycles ■
Signing Time:	224 000 Cycles ▲
Verification Time:	49 000 Cycles ▲

(a) LUOV Level 1

Number of Signatures:	∞ # ▲
Public Key Size:	35 400 B ▲
Secret Key Size:	32 B ▲
Signature Size:	337 B ▲
Key Generation Time:	4 600 000 Cycles ■
Signing Time:	643 000 Cycles ▲
Verification Time:	152 000 Cycles ▲

(b) LUOV Level 3

Figure 3.43: Criteria for LUOV from [19].

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▲	▲	▲	▲	▲
Feat. Ac.	▲	▲	▲	▲	▼	▲	▲
Sec. AC	▲	▲	▲	▲	▼	▲	▲
KeyGen oD	▲	▲	▲	▲	▼	▼	▲

(a) LUOV Level 1

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▲	▲	▲	▲	▲
Feat. Ac.	▲	▲	▲	▲	▼	▼	▲
Sec. AC	▲	▲	▲	▲	▼	▼	▲
KeyGen oD	▲	▲	▲	▲	▼	▼	▼

(b) LUOV Level 3

Figure 3.44: Use case suitability for LUOV.

LUOV [18] is a multivariate-based signature scheme based on a lifted variant of the Unbalanced Oil-Vinegar (UOV) signature scheme. This new construction allows for much smaller keys than in UOV.

However, LUOV was subject to a new differential attack, that broke the proposed parameter sets [52]. Although the lifting approach still appears promising, LUOV did not proceed to NIST Round 3 due to its immature security assumptions.

MQDSS

Publication:	2016 ●
Standardization:	NIST Round 2 ■
Security:	immature ■
Patents:	none known ▲

Meta criteria

Number of Signatures:	∞ # ▲
Public Key Size:	46 B ▲
Secret Key Size:	16 B ▲
Signature Size:	28 400 B ▲
Key Generation Time:	1 035 498 Cycles ■
Signing Time:	5 466 490 Cycles ■
Verification Time:	3 563 610 Cycles ■

(a) MQDSS-31-48 (L. 1-2)

Number of Signatures:	∞ # ▲
Public Key Size:	64 B ▲
Secret Key Size:	24 B ▲
Signature Size:	59 928 B ▲
Key Generation Time:	2 406 494 Cycles ■
Signing Time:	14 355 450 Cycles ■
Verification Time:	9 806 182 Cycles ■

(b) MQDSS-31-64 (L. 3-4)

Figure 3.45: Criteria for MQDSS from [86].

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▲	▲	▲	▲	▼
Feat. Ac.	▲	▲	▲	▲	▼	▼	▼
Sec. AC	▲	▲	▲	▲	▼	▼	▼
KeyGen oD	▲	▲	▲	▲	▼	▼	▼

(a) MQDSS-31-48 (L. 1-2)

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▲	▲	▲	▼	▼
Feat. Ac.	▲	▲	▲	▲	▼	▼	▼
Sec. AC	▲	▲	▲	▲	▼	▼	▼
KeyGen oD	▲	▲	▲	▲	▼	▼	▼

(b) MQDSS-31-64 (L. 3-4)

Figure 3.46: Use case suitability for MQDSS.

MQDSS [36] is a multivariate-based signature algorithm based on the multivariate quadratic (MQ) problem. However, its structure is more comparable to symmetric-based schemes like SPHINCS$^+$ and Picnic, rather than other multivariate schemes. MQDSS offers key sizes that are among the smallest of all PQC schemes, but large signature sizes and rather slow performance compared to other multivariate signatures.

MQDSS was subject to a forgery attack, that required the choice of much larger parameters, and therefore worse performance. Due to this, the competitors SPHINCS$^+$ and Picnic appear to be better candidates for standardization, and MQDSS did not advance to NIST Round 3 [4].

Rainbow

Publication:	2
Standardization:	NIST Round 3
Security:	immature
Patents:	none known

Meta criteria

Number of Signatures:	∞ #
Public Key Size:	157 800 B
Secret Key Size:	101 200 B
Signature Size:	528 B
Key Generation Time:	9 900 000 Cycles
Signing Time:	67 000 Cycles
Verification Time:	34 000 Cycles

(a) Rainbow-I (L. 1)

Number of Signatures:	∞ #
Public Key Size:	861 400 B
Secret Key Size:	611 300 B
Signature Size:	1312 B
Key Generation Time:	52 000 000 Cycles
Signing Time:	285 000 Cycles
Verification Time:	132 000 Cycles

(b) Rainbow-III (L. 3)

Figure 3.47: Criteria for Rainbow from [51].

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▼	▲	▼	▲	▲
Feat. Ac.	▲	▲	▲	▲	▼	▲	▲
Sec. AC	▲	▼	▼	▲	▼	▲	▲
KeyGen oD	▲	▲	▲	▲	▼	▲	▲

(a) Rainbow-I (L. 1)

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▼	▲	▼	▲	▲
Feat. Ac.	▲	▲	▼	▲	▼	▲	▲
Sec. AC	▲	▼	▼	▲	▼	▲	▲
KeyGen oD	▲	▼	▼	▲	▼	▼	▼

(b) Rainbow-III (L. 3)

Figure 3.48: Use case suitability for Rainbow.

Rainbow [49] is a multivariate-based signature scheme based on the Unbalanced Oil-Vinegar (UOV) signature scheme. Its additional structure allows for efficiency advantages compared to UOV. Rainbow features small signature sizes and efficient signing and verifying, but has large public keys and a rather slow key generation.

However, the additional structure opens Rainbow to more avenues for cryptanalytic advances. During NIST Round 2, the proposed parameters had to be increased due to cryptanalytic progress (see [4]). During Round 3, a more efficient key-recovery attack broke Rainbow instantiations from the Round 3 submission [**C:Beullens22**]. Thus, larger parameters and slower performance numbers than presented above (taken from the Round 3 submission) would be required to achieve sufficient security. However, due to the dwindling confidence in the security of Rainbow, and tweaks from the basic UOV construction in general, it did not proceed to NIST Round 4 [5].

3.6.2 Suitability for QuantumRISC

In general, multivariate signature schemes offer a very fast signature verification, and thus seem suitable for applications on constrained devices that only have to perform this task. On the other hand, public keys are large, which means that they may not fit in memory of a memory-constrained device. Nevertheless, especially Rainbow and LUOV appeared to be promising if this problem can be overcome.

However, recent attack improvements against GeMSS, LUOV, and Rainbow (see Section 3.6.1) further increased the parameter sizes, and significantly weakened the confidence in the security of multivariate signature schemes. Therefore, their usage is currently not recommended. An exception is the plain UOV signature scheme, since the mentioned attacks exploit optimizations that were only introduced for Rainbow resp. LUOV. However, UOV is less efficient, and therefore did not participate in the NIST standardization process.

3.7 Symmetric-/Hash-based schemes

3.7.1 Signature

Picnic

Publication:	2017 ●
Standardization:	NIST Round 3 (alternate candidate) ▲
Security:	immature ■
Patents:	none known ▲

Meta criteria

Number of Signatures:	∞ # ▲	**Number of Signatures**:	∞ # ▲
Public Key Size:	34 B ▲	**Public Key Size**:	48 B ▲
Secret Key Size:	17 B ▲	**Secret Key Size**:	24 B ▲
Signature Size:	12 595 B ▲	**Signature Size**:	27 551 B ▲
Key Generation Time:	N/A	**Key Generation Time**:	N/A
Signing Time:	18 590 000 Cycles ■	**Signing Time**:	37 560 000 Cycles ▼
Verification Time:	14 260 000 Cycles ■	**Verification Time**:	29 200 000 Cycles ▼

(a) Picnic3-L1 (L. 1) (b) Picnic3-L3 (L. 3)

Figure 3.49: Criteria for Picnic from [63] (no data for key generation given).

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▲	▲	●	▼	▼
Feat. Ac.	▲	▲	▲	▲	●	▼	▼
Sec. AC	▲	▲	▲	▲	●	▼	▼
KeyGen oD	▲	▲	▲	▲	●	▼	▼

(a) Picnic3-L1 (L. 1)

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▲	▲	●	▼	▼
Feat. Ac.	▲	▲	▲	▲	●	▼	▼
Sec. AC	▲	▲	▲	▲	●	▼	▼
KeyGen oD	▲	▲	▲	▲	●	▼	▼

(b) Picnic3-L3 (L. 3)

Figure 3.50: Use case suitability for Picnic.

Picnic [33, 90] is a signature scheme that uses a non-interactive zero-knowledge proof of knowledge of a secret key. In contrast to most of the other PQC proposals, it is not based on the hardness of a number-theoretic problem, but on assumptions on symmetric primitives, in particular, a hash function and a block cipher. The design of Picnic is modular, which means that hash functions and block ciphers can easily be substituted. Picnic features small public key sizes, but large signatures and slow signing and verifying.

Although the security assumptions on symmetric primitives are rather conservative, the design of Picnic and its LowMC block cipher are rather new. Thus, more research in this direction is required, and it did not proceed to NIST Round 4 [5].

SPHINCS$^+$

Publication:	2015 ●
Standardization:	NIST standardization choice ▲
Security:	well-understood ▲
Patents:	none known ▲

Meta criteria

Number of Signatures:	∞ # ▲
Public Key Size:	32 B ▲
Secret Key Size:	64 B ▲
Signature Size:	7856 B ▲
Key Generation Time:	84 964 790 Cycles ▼
Signing Time:	644 740 090 Cycles ▼
Verification Time:	861 478 Cycles ▲

(a) SPHINCS$^+$-SHA-256-128s-simple (L. 1)

Number of Signatures:	∞ # ▲
Public Key Size:	48 B ▲
Secret Key Size:	96 B ▲
Signature Size:	16 224 B ▲
Key Generation Time:	125 310 788 Cycles ▼
Signing Time:	1 246 378 060 Cycles ▼
Verification Time:	1 444 030 Cycles □

(b) SPHINCS$^+$-SHA-256-192s-simple (L. 3)

Figure 3.51: Criteria for SPHINCS$^+$ from [59].

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▲	▲	▼	▼	▼
Feat. Ac.	▲	▲	▲	▲	▼	▼	▼
Sec. AC	▲	▲	▲	▲	▼	▼	▼
KeyGen oD	▲	▲	▲	▲	▼	▼	▼

(a) SPHINCS$^+$-SHA-256-128s-simple (L. 1)

Use case	# Signatures	Public Key	Secret Key	Signature	Key Gen.	Signing	Verification
Sec. DL	▲	▲	▲	▲	▼	▼	▼
Feat. Ac.	▲	▲	▲	▲	▼	▼	▼
Sec. AC	▲	▲	▲	▲	▼	▼	▼
KeyGen oD	▲	▲	▲	▲	▼	▼	▼

(b) SPHINCS$^+$-SHA-256-192s-simple (L. 3)

Figure 3.52: Use case suitability for SPHINCS$^+$.

SPHINCS$^+$ is a *stateless* hash-based signature scheme. Thus, in contrast to LMS and XMSS, a signer does not have to keep track of any information of parts of a SPHINCS$^+$ tree that have been used for signing already. Although the number of signatures per private key is limited in theory, in practice these limits are usually large enough to consider this number unlimited. The security of SPHINCS$^+$ relies solely on a hash function that satisfies certain security requirements. E.g., these include second preimage resistance, which is a well-understood property.

While public key sizes are very small in SPHINCS$^+$, its signatures are comparably large and signing is rather slow. We note that tradeoffs between signature size and speed are possible.

Due to its very conservative security features, NIST chose to standardize SPHINCS$^+$ after Round 3 [5].

LMS

Publication:	1993 ●
Standardization:	NIST Recommendation, IETF Standard ▲
Security:	well-understood ▲
Patents:	expired ▲

Figure 3.53: Criteria for LMS.

LMS is a *stateful* hash-based signature scheme. This means that an LMS key pair can only be used for a limited number of signatures, since an instantiation contains a limited number of one-time key pairs. Thus, the signer must keep track of the state, which indicates which of the key pairs have already been used. Note that depending on the required number of signatures, the performance numbers and key/signature sizes significantly differ. Thus, we refrain from giving specific representative data on this, since the requirements of different application scenarios will lead to different data and ratings.

The security of LMS is well-understood, since it solely depends on security features of the respective hash function. LMS and XMSS have both been recommended by NIST [39].

XMSS

Publication:	2011 ●
Standardization:	NIST Recommendation, IETF Standard ▲
Security:	well-understood ▲
Patents:	none known ▲

Figure 3.54: Criteria for XMSS.

Like LMS, XMSS [25] is a stateful hash-based signature scheme, which means that only a limited number of messages can be signed per public key, and the signer has to keep track of which one-time key pairs have already been used. As for LMS, the security of XMSS relies on the underlying hash function, and is well-understood. LMS and XMSS share a similar design, but differ in several design choices. In general, XMSS and LMS share a similar performance profile, where XMSS allows for slightly smaller signatures, while being somewhat slower than LMS.

Similar to the case of LMS, the performance and key and signature sizes differ with the required number of signatures, and hence we refrain from giving representative data and ratings. In general, these numbers can be considered to lie in the realm of the numbers of SPHINCS$^+$, while the simpler structure of stateful schemes allows for a slightly faster performance and smaller signature sizes in both LMS and XMSS.

XMSS has been recommended by NIST along with LMS [39].

3.7.2 Suitability for QuantumRISC

Hash-based schemes are considered as being a conservative choice in terms of security, since they rely on certain well-understood features of cryptographic hash functions. Although they cannot compete, e.g., with lattice-based signature schemes in terms of performance and signature sizes, these schemes are promising options due to their well-understood security, and a high likeliness of being standardized. Furthermore, the performance of hash-based schemes can be improved by deploying (existing) hardware accelerators for faster hash evaluations.

If the use case allows for managing the state of the scheme, XMSS or LMS can be used. For both schemes there is is an IETF standard, and they are recommended by NIST. If a stateless scheme is required, SPHINCS$^+$ is a conservative option, which was chosen to be standardized after NIST Round 3.

The symmetric-based scheme Picnic relies on the security of both hash functions and block ciphers. Although it appears to be competitive with hash-based schemes, this new approach requires further analysis, e.g., with respect to its security and performance. Thus, we restrict to hash-based schemes in QuantumRISC.

Properties	Backend	Client(s)
Latency	<10 ms:<1 s	<10 ms:<1 s
# Executions over product lifetime	unlimited	unlimited
Size of processed data	≤64B:>1 GB	≤64B:>1 GB
Physical accessibility	untrusted:trusted	untrusted
Computational power	Server	embedded:IT
RAM availability	Server	embedded:IT
Storage availability	Server	embedded:IT
# Key pairs	≤1000	≤1000
Data dissipation	one-to-one	
Life time	≤1 year	
Current security level	≤128 bits	

Table 3.5: Requirements of the use case Session-based Secure Channel [78, Table 3.27].

4 Performance optimizations

4.1 Parameters for SQISign

This section aims at finding parameters for the SQISign signature scheme that allow for efficient implementations [24]. In recent years the tantalising problem of finding two large, consecutive, smooth integers has emerged in the context of instantiating efficient isogeny-based public key cryptosystems. Though the problem was initially motivated in the context of key exchange [40], a wave of polynomial time attacks has completely broken the isogeny-based key exchange scheme SIDH, leaving post-quantum signatures as the most compelling cryptographic application of isogenies at present. In terms of practical potential, the leading isogeny-based signature scheme is SQISign; it boasts the smallest public keys and signatures of all post-quantum signature schemes, at the price of a signing algorithm that is orders of magnitude slower than its post-quantum counterparts. Finding secure parameters for SQISign is related to the twin smooth problem mentioned above, with a large contributing factor to the overall efficiency of the protocol being the smoothness bound, B, of the rational torsion used in isogeny computations. This bound corresponds to the degree of the largest prime-degree isogeny computed in the protocol, for which the fastest algorithm runs in $\tilde{O}(\sqrt{B})$ field operations. Part of the reason for SQISign's performance drawback is that the problem of finding parameters with small B is difficult: the fastest implementation to date targets security comparable to NIST Level I and has $B = 3923$ [46].

We revisit the problem of finding two consecutive B-smooth integers by giving an optimized implementation of the Conrey-Holmstrom-McLaughlin "smooth neighbors" algorithm (CHM) [37]. Though the results are not large enough to be cryptographic parameters themselves, we feed them as input into known methods of searching for twins to yield cryptographic parameters that are much smoother than those given in prior works. Our methods seem especially well-suited to finding parameters for the SQISign signature scheme, particularly those that are geared towards high-security levels.

The CHM algorithm

Conrey, Holmstrom, and McLaughlin [37] present the following algorithm for producing many B-smooth values of $X(X+1)$. It starts with the initial set

$$S^{(0)} = \{1, 2, \ldots, B-1\}$$

of all integers less than B, representing the B-smooth twins $(1,2)$, $(2,3)$, …, $(B-1, B)$. Next, it iteratively passes through all pairs of distinct $r, s \in S^{(0)}, r < s$ and computes

$$\frac{t}{t'} = \frac{r}{r+1} \cdot \frac{s+1}{s},$$

writing $\frac{t}{t'}$ in lowest terms. If $t' = t + 1$, then clearly t also represents a twin smooth pair. The next set $S^{(1)}$ is formed as the union of $S^{(0)}$ and the set of all solutions t such that $t' = t + 1$. Now the algorithm iterates through all pairs of distinct $r, s \in S^{(1)}$ to form $S^{(2)}$ and so on. We call the process of obtaining $S^{(d)}$ from $S^{(d-1)}$ the d-th CHM iteration. Once $S^{(d)} = S^{(d-1)}$, the algorithm terminates.

Implementation and results

We implemented a somewhat optimized version of the pure CHM algorithm in C++. In particular, this implementation is parallelized, and avoids multiple checks of the same pairs of twin smooths (r, s). Furthermore, we iterate through smoothness bounds: We start with a small bound B_1 and the initial set $S_1^{(0)} = \{1, \ldots, B_1 - 1\}$, and use the CHM algorithm to iteratively compute sets $S_1^{(i)}$ until we reach some d_1 such that $S_1^{(d_1)} = S_1^{(d_1-1)}$. In the next iteration, we increase the smoothness bound to $B_2 > B_1$, define the initial set $S_2^{(0)} = S_1^{(d_1)} \cup \{B_1, \ldots, B_2 - 1\}$. Again we compute CHM iterations until we find d_2 such that $S_2^{(d_2)} = S_2^{(d_2-1)}$, where we avoid checking pairs (r, s) that have been processed in earlier iterations. Ideally, we could repeat this procedure until we reach a smoothness bound B_i for which the CHM algorithm produces large enough twin smooths for cryptographic purposes. However, our data suggests that this is infeasible in practice due to both runtime and memory limitations.

In particular, we ran this approach up to the smoothness bound $B = 547$, after which the set of twin smooths contains 82,026,426 pairs, whose bitlength distribution roughly equals a normal distribution centered around 58-bit. The largest pair has a bitlength of 122 bits. However, the sheer amount of twins smooths found means that we cannot continue with this approach to find large enough twin smooths for cryptographic applications.

Therefore, we implemented various other optimizations, such as only checking pairs of twin smooths (r, s) that satisfy $r < s < k \cdot r$ for $1 < k \leq 2$. This sacrifices the completeness of the CHM algorithm for the benefit of being able to move to larger smoothness bounds. With variants of this approach, we ran CHM up to $B = 2^{11}$, finding twin smooths of up to 145 bits.

Cryptographic primes of the form $p = 2r^n - 1$

For SQISign, we require primes p, such that $p^2 - 1$ has $\log T'$ bits of B-smoothness, where $T' = 2^f T$. Concretely, we have $\log p \in \{256, 384, 512\}$ for NIST Level I, III and V (respectively), $T \approx p^{5/4}$ and f as large as possible. In the current implementation of SQISign, $f \approx \lfloor \log \left(p^{1/4}\right) \rfloor$ (i.e., $T' \approx p^{3/2}$), and therefore, we aim for this when finding primes.

Fix a smoothness bound B and let $p_n(x) = 2x^n - 1$. We have $p_n(x)^2 - 1 = 4x^n(x^n - 1)$, and its factorization for $n = 2, \ldots, 6$ is given in Table 4.1.

We observe that for n even, both $x + 1$ and $x - 1$ appear in the factorization of $p_n(x) - 1$. In this case, for twin smooths $(r, r \pm 1)$, evaluating $p_n(x)$ at r guarantees that we have a smooth factor $4x^n(x \pm 1)$ in $p^2 - 1$. For n odd, we will only have that $x - 1$ appears in the factorization, and therefore only consider twins $(r, r - 1)$ to guarantee we have B-smooth factor $4x^n(x - 1)$.

n	$p_n(x)^2 - 1$
2	$4x^2(x-1)(x+1)$
3	$4x^3(x-1)(x^2+x+1)$
4	$4x^4(x-1)(x+1)(x^2+1)$
5	$4x^5(x-1)(x^4+x^3+x^2+x+1)$
6	$4x^6(x-1)(x+1)(x^2-x+1)(x^2+x+1)$

Table 4.1: Factorization of $p_n(x)^2 - 1$ for $n = 2, 3, 4, 5, 6$, where $p_n(x) = 2x^n - 1$.

Thus, we can plug twin smooths of the correct sizes found via CHM in $p_n(x)$, and obtain high probabilities for finding SQISign-friendly primes. The results are summarized in Table 4.2. Note that $\sqrt{B}/f$ is the signing cost metric, which roughly estimates the cost of signing.

We note that we can also pick larger values of n and simply compute $p_n(x)$ for smooth inputs x. For $n \in \{12, 18\}$, this yields good smoothness probabilities, yet the search space is relatively small, and can easily be exhausted. Thus, the respective primes in Table 4.2 cannot be improved, while we could find more primes through CHM by using more computational power and further optimizations for CHM.

NIST security level	n	r	$\lceil \log_2(p) \rceil$	f	B	$\sqrt{B}/f$	$\log_p(T)$
NIST-I	2	12114603117167727905665745290012917776	241	49	1091	0.67	1.28
		20910230141429718023578160841527132 16	243	49	887	0.61	1.28
	3	10227318375788227199589376	251	31	383	0.63	1.31
		21611736033260878876800000	254	31	421	0.66	1.28
		20461449125500374748856320	254	46	523	0.50	1.26
		26606682403634464748953600	255	40	547	0.58	1.28
	4	34848218231355211776*	261	77	2311	0.62	1.30
NIST-III	3	13740020350057131495504053433738485 76	362	37	1277	0.97	1.25
	4	513973487626239096407087308 8	370	45	11789	2.41	1.26
		12326212283367463507272925184	375	77	55967	3.07	1.31
		18080754980295452456023326720	377	61	95569	5.07	1.26
		27464400309146790228660255744	379	41	13127	2.79	1.29
	6	2628583629218279424	369	73	13219	1.58	1.27
		5417690118774595584	375	79	58153	3.05	1.27
		11896643388662145024	382	79	10243	1.28	1.30
	12	2446635904*	376	85	9187	1.13	1.29
NIST-V	4	1142167815485817094395128758012797911 04*	507	65	75941	4.24	1.26
		1237942743874742989127425438192425871 36*	508	41	15263	3.01	1.29
	6	9469787780580604464332800	499	109	703981	7.70	1.25
		1223346860574068600780800 0	502	73	376963	8.41	1.28
		2669797390044648368060825 6	508	85	150151	4.56	1.26
		3192974042794487000652185 6	510	91	550657	8.15	1.25
		4134024820090081905679360 0	512	67	224911	7.08	1.28
	12	5594556480768*	510	97	88469	3.07	1.29
	18	335835120*	511	73	24229	2.13	1.29

Table 4.2: A table of SQISign parameters $p = p_n(r)$ found using twin-smooth integers $(r, r \pm 1)$ at each security level. The f is the power of two dividing $(p^2 - 1)/2$ and B is the smoothness bound of the odd cofactor $T \approx p^{5/4}$. The r marked with an asterisk correspond to primes p not found using the CHM machinery.

5 Optimization of memory requirements and message sizes

5.1 Memory-constrained Classic McEliece

Deploying Classic McEliece in practical applications can be challenging due to the size of its public keys, especially in the context of embedded, memory-constrained devices. In the paper *Classic McEliece Implementation with Low Memory Footprint* [84] this is addressed and an implementation optimized to overcome memory constraints is presented. The results are evaluated and demonstrated for an ARM-Cortex M4 development board that runs with 168 MHz and has 256 kB of RAM.

The fundamental idea is to avoid storing the whole public key by utilizing smaller structures which can be used to retrieve parts of the public key ad-hoc as they are needed. This process is referred to as *streaming* the public key in analogy to streaming large video files: only small chunks are actually sent and processed at a time. While this concept can be employed for on-line communication protocols, it is also applicable, e.g., to store the public key memory efficiently on the flash storage by streaming the public key to the flash storage.

Algorithm 1 depicts the Classic McEliece key generation algorithm.

Algorithm 1: Classic McEliece Key Generation

Input: Classic McEliece Parameters $(n, t, m, q = 2^m, k = n - mt)$
Output: Private Key (Γ, s), Public Key T

1 Generate a uniform random monic irreducible polynomial $g(x) \in \mathbb{F}_q[x]$ of degree t.
2 Select a uniform random sequence $(\alpha_1, \alpha_2, ..., \alpha_n)$ of n distinct elements of $\mathbb{F}_q$
3 Compute the $t \times n$ matrix $\tilde{H} = h_{i,j}$ over $\mathbb{F}_q$, where $h_{i,j} = \alpha_j^{i-1}/g(\alpha_j)$ for $i = 1, ..., t$ and $j = 1, ..., n$.
4 Form an $mt \times n$ matrix $\hat{H}$ over $\mathbb{F}_2$ by replacing each entry $c_0 + c_1 z + ... + c_{m-1} z^{m-1}$ of $\tilde{H}$ with a column of t bits $c_0, c_1, ..., c_{m-1}$.
5 Reduce $\hat{H}$ to systematic form $(I_{n-k} \mid T)$, where I_{n-k} is an $(n-k) \times (n-k)$ identity matrix.
6 **if** *the previous step fails (i.e., the leftmost* $(n-k) \times (n-k)$ *submatrix of* $\hat{H}$ *is singular)* **then**
7 go back to Step 1.
8 **end**
9 Generate a uniform random n-bit string s.
10 Put $\Gamma = (g, (\alpha_1, \alpha_2, ..., \alpha_n))$ and output (Γ, s) as private key and T as public key.

Steps 1 and 2 generate the private key, i.e., a binary Goppa code. The following steps

derive the public key from the private key by forming the parity-check matrix for the binary Goppa code. Most notably, Gaussian elimination is used in Step 5 to reduce the parity-check matrix to systematic form (if possible). The generation of the private key can be done memory efficiently whereas the computation of the public key requires a lot of memory for storing the large matrix $\tilde{H}$ which is transformed into $\hat{H}$ and successively $H = (I_{n-k} \mid T)$. In order to compute the Gaussian elimination algorithm, a $(n-k) \times n$ matrix is required to be held in memory.

In order to both save the space that is required for storing the resulting public key T and for computing the Gaussian elimination algorithm, Steps 3 to 5 are replaced in the following extended private key generation algorithm, Algorithm 2. The public key is not an output any longer. Essentially, the key generation algorithm is split into the extended private key generation algorithm, which computes the $(n-k) \times (n-k)$ matrix S^{-1} over $\mathbb{F}_2$ instead of the public key T, and the public key column retrieval algorithm, Algorithm 3.

Algorithm 2: Extended Private Key Generation

Parameter: $n, t, m, q = 2^m, k = n - mt$
Output: $K_{priv\_ext} = ((g(x), (\alpha_1, ..., \alpha_n), s, S)$

1 Generate a uniform random irreducible polynomial $g(x) \in \mathbb{F}_q[x]$ of degree t.
2 Select a uniform random sequence $(\alpha_1, \alpha_2, ..., \alpha_n)$ of n distinct elements of $\mathbb{F}_q$.
3 Compute the $t \times (n-k)$ matrix $\tilde{S}^{-1}$ over $\mathbb{F}_q$ by letting $(\tilde{S}^{-1})_{i,j} = h_{i,j}$, where $h_{i,j} = \alpha_j^{i-1}/g(\alpha_j)$ for $i = 1, ..., t$ and $j = 1, ..., n-k$.
4 Form an $(n-k) \times (n-k)$ matrix S^{-1} over $\mathbb{F}_2$ by replacing each entry $c_0 + c_1 z + ... + c_{m-1} z^{m-1}$ of $\tilde{S}^{-1}$ with a column of m bits $c_0, c_1, ..., c_{m-1}$.
5 Compute S as the inverse of S^{-1}.
6 **if** *the previous step fails (i.e., S^{-1} is singular)* **then**
7 | go back to line 1.
8 **end**
9 Generate a uniform random n-bit string s.

The public key column retrieval algorithm can be used to compute single columns of the public key, given the extended private key as input. The extended private key replaces both the former public and private key. Computing single columns of the public key enables the streaming of the public key. In contrast to requiring memory for storing a $(n-k) \times n$ matrix for computing the public key, now only the $(n-k) \times (n-k)$ matrix in the extended private key is required. This is, depending on parameter sets, roughly 2.6 to 3.7 times less than would be required originally. For the *mceliece348864* parameter set, only $81\,268\,\mathrm{B}$ are required for the extended private key, in contrast to $267\,572\,\mathrm{B}$ for the public and private key.

In order to compute the inverse of S^{-1} in Step 5 in Algorithm 3, the inversion has to be done memory efficiently as well. Otherwise, the memory requirements of the key generation are unnecessarily high, perhaps preventing memory-constrained devices from performing it. Algorithm 4 depicts an efficient algorithm to invert the matrix almost in-place (the permutation matrix is additionally stored as an array, requiring $2(nk)$ bytes).

Algorithm 5 depicts the LU decomposition algorithm that is used. This is essentially the

Algorithm 3: Public Key Column Retrieval

Input: $K_{priv\_ext} = (\Gamma, s, S)$, Column c (Integer)
Parameter: $n, t, m, q = 2^m, k = n - mt$
Output: cth column of the public key: $T_{\bullet c}$

1 Compute $\tilde{H}_{\bullet c}$ as the t-dimensional vector over $\mathbb{F}_q$ with $\tilde{H}_{j,c} = \alpha_c^{j-1}/g(\alpha_c)$ for $j = 1, 2, ..., t$.
2 Compute $\hat{H}_{\bullet c}$ as the mt-dimensional vector over $\mathbb{F}_2$ that results by replacing each entry $c_0 + c_1 z + ... + c_{m-1} z^{m-1}$ of $\tilde{H}_{\bullet c}$ with a column of m bits $c_0, c_1, ..., c_{m-1}$.
3 Compute $H_{\bullet c} = S\hat{H}_{\bullet c}$.

Algorithm 4: LU-Decomposition-based Matrix Inversion

Input: $S^{-1} \in \mathbb{F}_2^{(n-k)\times(n-k)}$
Output: S

1 Find the LU decomposition of S^{-1}, i.e., $PS^{-1} = LU$ where $P, L, U \in \mathbb{F}_2^{(n-k)\times(n-k)}$ and P is a permutation matrix and L and U are lower and upper triangular matrices.
2 Invert L and U.
3 Compute the product $U^{-1}L^{-1}$.
4 Undo the permutation to obtain $S = U^{-1}L^{-1}P$.

"kij-variant" of the outer-product formulation of the Gaussian elimination algorithm. For the implementation, care has to be taken in order to fulfill the constant time property. The pivoting is most noteworthy here and Step 3 can for example be implemented by not explicitly branching but using all-one or all-zero masks for byte operations, respectively.

Step 3 of Algorithm 4, the multiplication of $U^{-1}L^{-1}$, might also require some further explaining. U^{-1} and L^{-1} are both stored in the memory where S^{-1} used to be – as upper and lower triangular matrix, respectively. To obtain an algorithm that multiplies both matrices in-place, the triangular structure of U^{-1} and L^{-1} is utilized. For convenience, define $\bar{A}$ as the matrix that contains L^{-1} and U^{-1} as a lower and an upper triangular matrix and A as $A := U^{-1}L^{-1}$. Each entry $A(i, j)$ can then be written as $A(i, j) = \sum_{k=\max(i,j)+1}^{n} \bar{A}(k, j) \cdot \bar{A}(i, k)$. By appropriately ordering the computations of entries of A, overriding values that are needed for future computations is prevented. More precisely, first compute the element in the top-left corner, i.e., the first diagonal element $A(1, 1)$. Then, the three remaining elements in the top-left 2×2-matrix can be computed in any order. Continuing like this, i.e., computing the remaining five elements in the top-left 3×3-matrix in any order, and so on, all elements of A can be computed. Each evaluation of the given formula only depends on values of $\bar{A}$ that have not been overwritten by elements of A yet. Therefore, the outlined approach can be implemented in-place.

With the presented algorithms, the (extended) private key holder can substantially reduce the memory requirements on their end. The public key can be streamed to a communication party, or to the flash memory, or any other place while requiring less memory than what would originally be required. Note that streaming columns instead of rows means that the public key is transposed when compared to the Classic McEliece

Algorithm 5: LU Decomposition

Input: $A \in \mathbb{F}_2^{(n-k)\times(n-k)}$

Output: $P, L, U \in \mathbb{F}_2^{(n-k)\times(n-k)}$, s.t. $PA = LU$. Return $\perp$ (error) if A is singular.

```
for k := 1 to n − 1 do
    for i := k + 1 to n do
        Swap row i with row k if the ith row has a non-zero entry at the kth column
         and update P accordingly (partial-pivoting).
    end
    if Pivoting fails (i.e., A is singular) then
        return ⊥
    end
    for i := k + 1 to n do
        for j := k + 1 to n do
            A(i, j) := A(i, j) − A(i, k) · A(k, j)
        end
    end
end
```

specification.

During on-line protocols – TLS as an example –, the memory requirements for the encapsulation operation can also be drastically reduced. The original encapsulation algorithm requires the complete public-key matrix T. Since the public key is essentially only used for a matrix-vector multiplication, it is trivial to reorder the operations such that single rows or single columns of the public key can be processed at a time. Each processed chunk of the public key can be used to update the intermediate result of the operation. Thus, the encapsulation algorithm can be performed while the public key is received, and the chunk of data can be processed and immediately deleted from memory. This makes a very efficient streaming approach for the encapsulation operation possible. Algorithm 6 describes this for the case of consuming single columns of the public key at a time. This is in accordance with the public key column retrieval algorithm, i.e., the sending party can use the public key column retrieval algorithm to memory efficiently stream the public key, and the receiving party can use the single-column encoding subroutine to memory-efficiently compute the syndrome of e.

5.2 Memory-constrained verification of PQC signatures

Cryptographic schemes from the five families of PQC have different advantages and disadvantages in terms of efficiency, e.g., memory and timing. Hence, in certain applications it can be challenging to deploy specific schemes because they do not fulfill the given requirements. In [56] we address the challenge of performing signature verification of post-quantum signature schemes with a large public key or signature in a highly memory-constrained environment. For our analysis, we chose an ARM Cortex-M3 board with 128 kB RAM and 1 MB Flash, an STM32 Nucleo-F207ZG, as the implementation platform. This

Algorithm 6: Classic McEliece Single-Column Encoding Subroutine

Input: weight-t Vector $e \in \mathbb{F}_2^n$
Parameter: $n, t, m, q = 2^m, k = n - mt$
Output: C_0

Initialize C_0 as an $n-k$-dimensional zero-vector.
Set $i := 1$.
while $i \leq n - k$ **do**
    Set $C_{0i} := e_i$.
    Set $i := i + 1$.
end
Set $i := 1$.
while $i \leq k$ **do**
    Read the ith column of the public key T from some location into $T_{\cdot i}$.
    Set $C_0 := C_0 + e_i \cdot T_{\cdot i}$.
    Set $i := i + 1$.
end

is a practical and widely deployed setup in, for instance, the automotive sector. However, this amount of memory is insufficient for most schemes.

We focused on the NIST PQC round-3 candidates Dilithium, Falcon, Rainbow, GeMSS, and SPHINCS$^+$, and studied performance improvements for their verification subroutines. We first revealed that for Rainbow and GeMSS, public keys are too big, while for SPHINCS$^+$, signatures do not fit in this memory. To make signature verification work also for these schemes, our approach is to stream the public key or the signature. We show that this way signature verification can be done keeping only small data packets in constrained memory. However, when streaming the public key, the device needs to securely store a hash value of the public key to verify the authenticity of the streamed public key. During signature verification, the public key is incrementally hashed, matching the data flow of the streamed public key.

We implemented and benchmarked the proposed public key and signature streaming approach for four different signature schemes (Dilithium, SPHINCS$^+$, Rainbow, and GeMSS). Although for Dilithium streaming the public key is not strictly necessary, the saved bytes allow us to keep more intermediate results in memory, which results in a speed-up. For comparison, we also implemented the lattice-based scheme Falcon for which streaming the public key or the signature is not necessary in our scenario as the entire public key and signature fit into RAM. We demonstrate that the proposed streaming approach is very well suited for constrained devices with a maximum utilization of 8 kB RAM and 8 kB Flash. The source code of our work is published and available at `https://git.fslab.de/pqc/streaming-pq-sigs`

Table 5.1 presents the speed results for our implementations. The studied signature schemes rely on either SHA-256 (rainbowI-classic, sphincs-sha256) or SHA-3/SHAKE (dilithium2, falcon-512, and gemss-128). In a typical HSM enabled device, SHA-256 would be available in hardware and SHA-3/SHAKE will also be available in the future.

Table 5.1: Cycle count for signature verification for a 33-byte message. Average over 1 000 signature verifications. Hashing cycles needed for verification of the streamed in public key (hashing and comparing to embedded hash) are reported separately. We also report the verification time on a practical HSM running at 100 MHz and also the total time including the streaming at 20 Mbit/s.

	w/o pk vrf.	w/ pk verification			w/ streaming
		pk vrf.	total	time[e]	20 Mbit/s
`sphincs-s`[a]	8 741k	0	8 741k	87.4 ms	90.6 ms
`sphincs-f`[b]	26 186k	0	26 186k	261.9 ms	268.7 ms
`rainbowI-classic`	333k	6 850k[d]	7 182k	71.8 ms	136.5 ms
`gemss-128`	1 619k	109 938k[c]	111 557k	1 115.6 ms	1 679.1 ms
`dilithium2`	1 990k	133k[c]	2 123k	21.2 ms	21.8 ms
`falcon-512`	581k	91k[c]	672k	6.7 ms	8.2 ms

[a] `-sha256-128s-simple` [b] `-sha256-128f-simple` [c] SHA-3/SHAKE [d] SHA-256
[e] At 100 MHz (no wait states)

However, on the Nucleo-F207ZG no hardware accelerators are available. Hence, we resort to software implementations instead. For SHA-256 we use the optimized C implementation from SUPERCOP.[1] For SHA-3/SHAKE, we rely on the ARMv7-M implementation from the XKCP.[2]

This work was inspired by the QuantumRISC result [84], where we studied streaming of the public key for the McEliece encryption scheme.

5.3 Memory-constrained SPHINCS$^+$ signing and verifying

While SPHINCS$^+$ offers small keys, the signatures can be up to around $50\,\mathrm{kB}$. This can be challenging for memory-constrained devices. Therefore, an approach is explored to reduce the memory requirements. Similarly to the Classic McEliece optimizations, the streaming of the signature is explored. The paper [56] demonstrates the practicability for some parameter sets of different PQ algorithms, including SPHINCS$^+$ and considers streaming as well. In contrast, [76] focuses on SPHINCS$^+$ entirely and encompasses all SPHINCS$^+$ parameter sets (apart from Haraka).

The first central observation in [76] is that a SPHINCS$^+$ signature can be written to and read from sequentially in n byte chunks where n is the output length (in bytes) of the underlying hash function. That is, in principle, accessing previously read or written chunks is not necessary in order to verify or generate the signature. The structure of a SPHINCS$^+$ signature can be seen in Figure 5.1.

A streaming interface is implemented which uses the abstraction that the signature is not a large buffer any more. Instead, a number of chunks of the signature can be requested to be read or written. The intention is that in the background, the streaming interface can then do the actual I/O operation that reads or writes the chunk(s). If beneficial for

[1] `https://bench.cr.yp.to/supercop.html`
[2] `https://github.com/XKCP/XKCP`

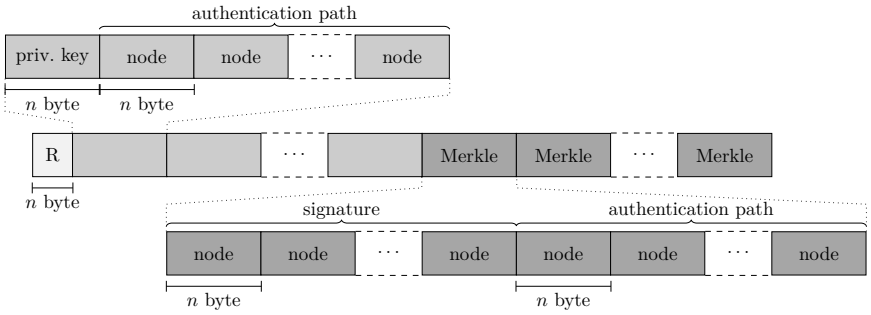


Figure 5.1: Structure of a SPHINCS$^+$ signature.

the concrete use case, the implementation can also choose to buffer some chunks, e.g., to fill an entire frame in an internet communication. It should be noted that the I/O can be anything that can be expressed by sequential read and write functions. The signature can simply be read from or written to a file, but the streaming interface could also be used to read or write the signature in a complex network protocol.

Necessary changes to the reference code are outlined and detailed measurements are given. As a stand-alone implementation, the memory requirements for the key generation, sign, and verify operations are less than $3\,\mathrm{kB}$ for every parameter set. Here memory requirements refer to the maximum stack size and no heap is used by the implementation. The memory requirements do not include the implementation of the signature streaming (actual I/O operations) since this is highly dependent on the type and implementation of the I/O.

As a proof of concept demonstration, the paper briefly explains and evaluates using the streaming SPHINCS$^+$ implementation with a TPM. For this, the authors integrate the streaming interface into the TPM 2.0 specification and communicate with the TPM implementation via SPI. Due to the speed of the SPI interface, there is a significant I/O overhead in this case. The streaming implementation buffers $1\,\mathrm{kB}$ of the signature, reducing the number of SPI transfers.

5.4 On-the-fly computation of twiddle factors for NTT

This section presents a technique to reduce memory consumption during the Number Theoretic Transform (NTT) computation. The NTT is a common optimization used in Module- and Ring-LWE based schemes, as it enables the multiplication of two polynomials in $n \log n$ steps instead of n^2 when common "school book" multiplication algorithms are used. As part of [8], this technique was adapted to lower the memory consumption of the Kyber and NewHope schemes. This version of the algorithm is often called Iterative NTT [67].

The Number Theoretic Transform (NTT) is a generalization of the discrete Fourier transform, and can be used to speed up the multiplication of polynomials, as done, for example, in the Kyber and NewHope ciphers. The basic step of a NTT takes a polynomial $f \pmod{X^n+1}$ and transforms it into two polynomials $f_1 \pmod{X^{\frac{n}{2}}-\zeta}$ and $f_2 \pmod{X^{\frac{n}{2}}+\zeta}$ with $f = f_1 + Xf_2$, if a primitive root with $\zeta^2 = 1$ exists. In principle, this transformation

can be repeated recursively on f_1 and f_2, as long as an n-th primitive root with $\zeta^n = 1$ in the underlying field exists. During the recursive transformation, powers of the primitive – in this context called *twiddle factors* – are used in butterfly operations, depending on the algorithm and butterfly either in ascending or bit-reversed order.

The order of the twiddle factors used during the NTT varies depending on the choice of the butterfly operation, e.g., Gentleman-Sande [55] or Cooley-Tukey [38], and the order of the input polynomial such as bit-reversed order or normal order, and also the order in which the coefficients are processed. The first design decision was not to use the bit-reversal process with the goal to reduce memory instructions. Thus, a forward NTT for normal order to bit-reversed order as well as an inverse NTT for bit-reversed order to normal order was implemented.

Although on-the-fly computation of twiddle factors reduces the memory usage, multiplication with reduction is more expensive than a single memory operation on most platforms. Therefore, this version of the algorithm is often used in FPGA implementations, where implementers can design efficient modular arithmetic circuits [85]. Software implementations usually opt for precomputation of all twiddle factors [23], or at least some part of them [7] to reduce the running time.

While the RISC-V ISA only contains general purpose computing instructions, the instruction code space also reserves opcodes for custom instructions. Hence both FPGA implementation tricks and the tricks used in implementations on constrained microcontrollers can be used on the RISC-V by providing instruction set extensions for modular arithmetic. Part of [8] was also a single cycle multiplication operation in $\mathbb{Z}_q$ that makes the on-the-fly computation of twiddle factors possible without any performance penalty. Furthermore, when interleaved, this multiplication does not stall the pipeline of the processor, unlike a memory load instruction.

The results show that using such an on-the-fly calculation together with an instruction set extensions for finite field arithmetic helps to reduce memory consumption not only of program code but also of data in situations where constants can be recomputed on-the-fly instead of storing them explicitly. This is particularly beneficial on architectures with small or slow memory.

6 Analysis of physical attacks

6.1 Safe-error attacks on SIKE and CSIDH

This section analyzes the resistance of SIKE and CSIDH towards safe-error attacks [26]. We present four safe-error attacks, two against SIKE and two against a constant-time implementation of CSIDH that uses dummy isogenies. The attacks use targeted bitflips during the respective isogeny-graph traversals. All four attacks lead to full key recovery. By using voltage and clock glitching, we physically carried out two of the attacks - one against each scheme -, thus demonstrate that full key recovery is also possible in practice.

Safe-error attacks

In so-called safe-error attacks, the adversary uses fault injections to perturb a specific memory location with the intention of not modifying the final result of the computation: The algorithm may overwrite or throw away modified values, making them safe errors. The presence or absence of an error then gives insight into which code path the algorithm executed. Two kinds of safe-error attacks exist: In a memory safe-error (M safe-error) attack, the attacker modifies the memory, i.e., in general these attacks focus on specific implementations. In a computational safe-error (C safe-error) attack, however, the computation itself is attacked through, e.g., skipping instructions. Hence, C safe-error attacks rather target algorithmic vulnerabilities.

Safe-error attacks on SIKE

SIKE private keys consist of an integer m, such that $P + [m]Q$ is the secret subgroup that forms the kernel of the secret isogeny. Thus, fault attacks can target the computation of $P + [m]Q$ to reveal the secret key m. SIKE uses a three-point ladder for this computation, which can be targeted for safe-error attacks. In general, such an attack initiates a SIKE key exchange, faults the execution of the three-point ladder in step k, and observes if the key exchange still succeeds. Due to secret-dependent behavior in the three-point ladder, this reveals the k-th bit of the secret key. A suitable fault injection can be achieved in the following ways.

Each step of the three-point ladder calls a function for a combined point doubling and addition. Depending on the secret key bit of the respective step, only two of the three point variables are overwritten in this function. Thus, an M safe-error attack can inject a fault to change the value of such a variable, and observe if the key exchange still succeeds. In this case, the variable has been overwritten after the point multiplication and addition, and otherwise this did not happen. Since this depends on the k-th secret key bit, this attack can reveal all bits by attacking the respective round k of the three-point ladder.

Similarly, a C safe-error attack can be used to launch such an attack. In particular, SIKE uses a constant-time point swap function, which swaps points if a secret key bit equals 1, and leaves them unchanged if it is 0. Thus, faulting the execution of this swap function, e.g., through an instruction skip, we can observe if the key exchange still succeeds. This again reveals bits of the secret key, and with the same strategy as above, we can recover the full secret key.

Safe-error attacks on CSIDH

Similar attacks can also be launched against CSIDH implementations. In particular, the following approach considers the constant-time implementations from [28, 79].

One approach for an M safe-error attack is to reveal dummy isogenies through memory faults. The implementations from [28, 79] use dummy isogenies to achieve the constant-time property. This means that some isogenies are computed, but their results are discarded. While a real isogeny updates the current curve coefficient A, a dummy isogeny simply keeps the previous value. Thus, injecting a memory fault in A during the isogeny computation will only cause computations to fail if a dummy isogeny is computed. Otherwise, A will be overwritten by the new valid curve coefficient at the end of the call to the isogeny function. Thus, analogous to the attacks on SIKE, we can detect dummy isogenies, and thus reveal the secret key up to the signs of the key elements, resp. the full key for the implementation of [72].

Similarly, [79] keeps two points throughout the computations. However, which of these variables is overwritten after an isogeny computation depends on the secret information if a dummy or real isogeny is computed. Thus, we can again launch an M safe-error attack, and observe whether the key exchange still succeeds. As above, repeating this procedure for each isogeny reveals the private key up to signs.

Practical experiments

All practical attacks were implemented using the ChipWhisperer tool chain (version 5.3.0) in Python (version 3.8.2) and performed on a ChipWhisperer-Lite board with a 32-bit STM32F303 ARM Cortex-M4 processor as target core. Based on available implementations, we wrote slightly modified ARM implementations of SIKEp434 and CSIDH-512 to make them suitable for our setup.

The attacks on SIKEp434 require 1,090 fault injections in order to reveal the private key with an accuracy of 99%. In CSIDH-512, we require 888 fault injections to reveal the secret key up to sign with an accuracy of 99%. The signs can be found by a simple meet-in-the-middle attack.

Countermeasures

As usual, redundancy can prevent the safe-error attacks from this section. That is, we can simply adapt the code to check if point swaps were executed correctly, and modify access patterns in order to prevent M safe-error attacks.

6.2 Disorientation fault attacks in CSIDH

This section presents a new class of fault-injection attacks against the CSIDH family of cryptographic group actions. Our *disorientation attacks* effectively flip the direction of some isogeny steps [**EPRINT:BKLMPRST22**]. We achieve this by faulting a specific subroutine, connected to the Legendre symbol or Elligator computations performed during the evaluation of the group action. These subroutines are present in almost all known CSIDH implementations. Post-processing a set of faulty samples allows us to infer constraints on the secret key. The details are implementation specific, but we show that in many cases, it is possible to recover the full secret key with only a modest number of successful fault injections and modest computational resources. We provide details for attacking the original CSIDH proof-of-concept software as well as the CTIDH constant-time implementation. Finally, we present a set of lightweight countermeasures against the attack and discuss their security.

Isogeny walks

The CSIDH group action consists of an isogeny walk containing a series of isogeny steps that we denote as $\mathfrak{l}_i^{\pm 1}$. A secret key is a vector of integers $(e_1, \ldots, e_n)$ that determines how often each $\mathfrak{l}_i^{\pm 1}$ is applied, and in which direction these steps are taken, i.e., if in positive or negative direction. The action of an $\mathfrak{l}_i^{\pm 1}$ is computed by finding a point of order ℓ_i of the correct orientation, which uniquely determines the ℓ_i-isogeny induced by $\mathfrak{l}_i^{\pm 1}$. The orientation of a point is positive, if it has coordinates (x, y) with $x, y \in \mathbb{F}_p$, and negative, if it has coordinates (x, y) with $x \in \mathbb{F}_p$ and $y \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$. Usually, the high cost for sampling points is amortized by combining several ℓ_i-isogenies of the same direction, i.e., for some $i \in S$ such that the secret key elements e_i have the same sign.

Attack scenario and disorientation faults

We assume physical access to some hardware device containing an unknown CSIDH private key $\mathfrak{a}$. In the basic version of the attack, we suppose that the device provides an interface to pass in a CSIDH public-key curve E and receive back the result $\mathfrak{a} * E$ of applying $\mathfrak{a}$ to the public key E as in the second step of the key exchange. Note that alternative scenarios, such as a hashed version of the attack are discussed in [**EPRINT:BKLMPRST22**].

We assume that the attacker is able to trigger an error during the computation of the orientation of a point in a specific round of the CSIDH algorithm: Whenever a point P with orientation $s \in \{-1, 1\}$ is sampled during the algorithm, we can flip the orientation $s \mapsto -s$ by injecting a fault in the Legendre symbol computation that determines the field of definition of the respective y-coordinate, and hence the orientation of the point. This leads to some isogenies being computed in the opposite direction throughout the round.

Recovering secret keys

Suppose we flip the orientation of a point in one round of the isogeny computation $E_B = \mathfrak{a} * E_A$ and the rest of the computation is performed correctly. The resulting curve E_t

is called a faulty curve. If the round was computing steps for isogenies in S with direction s, the resulting curve satisfies

$$E_B = \prod_{i \in S} \mathfrak{l}_i^{2s} * E_t,$$

that is, the faulty curve differs from the correct curve by an isogeny whose degree is given by the (squares of) primes ℓ_i for $i \in S$, the set S in the round we faulted.

Due to the point rejection probabilities in CSIDH, which results in effects like missing torsion and torsion noise, faulting the same round multiple times usually results in a set of different output curves. These curves have small distance, and hence a meet-in-the-middle search finds the connecting isogenies between them. Thus, each such set of curves of a round r results in a graph, which reveals information such as the orientation of primes ℓ_i.

Repeating this procedure for different rounds r gives us a set of different graphs. Again, a meet-in-the-middle search can connect these graphs, and connect them to the correct output curve E_B. This connected graph then gives us all the information to recover the secret key, see [**EPRINT:BKLMPRST22**].

Figure 6.1 gives an example for such a graph. For simplicity of representation, this graph stems from a toy implementation of CSIDH, using a 103-bit prime as parameter. The secret key in this example is
$(-1, +1, +2, +3, -2, +3, +2, +3, +1, +2, -3, -3, +2, +3, -2, -3, -2, +2, +1, -3,\ 0)$.

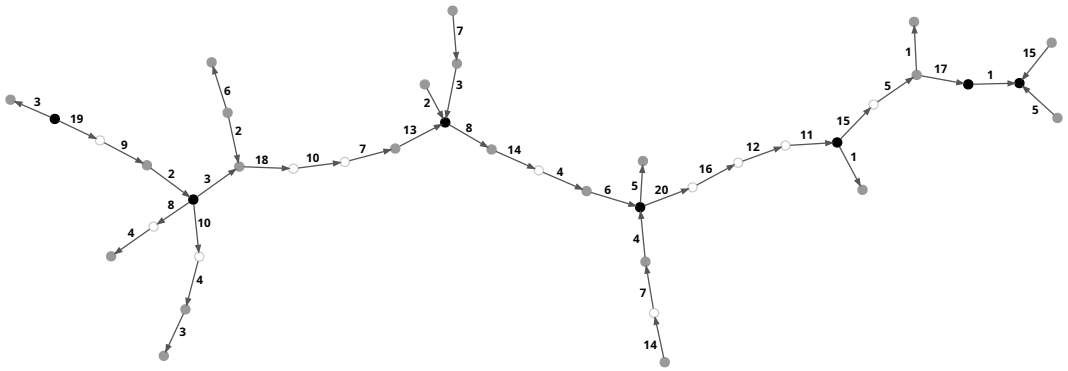


Figure 6.1: Example isogeny graph of faulty curves obtained from attacking the fictitious CSIDH-103 implementation.

Simulation

This approach of faulting the orientation of points applies to almost all implementations of CSIDH. As examples, [**EPRINT:BKLMPRST22**] simulates the fault for the CSIDH-512 [32] and CTIDH-512 [16] implementations. The reported number of required samples for recovering secret keys is roughly 128 for CSIDH-512 and 40 for CTIDH-512, respectively.

Countermeasures

While previous the literature presented a countermeasure that significantly complicates disorientation faults, its cost is rather high with an overhead of roughly 30% [28]. To avoid

this, we propose to use pseudo y-coordinates. In particular, we compute the orientation of a point in a way that ensures the correctness of its y-coordinate. A successful disorientation is then expected to require two successive faults with a success probability of $1/p$. For details, we refer to [**EPRINT:BKLMPRST22**]. The computational overhead for this is not noticeable in CSIDH-512, and roughly 5.5% in CTIDH-512.

6.3 Zero-value attacks and correlation attacks on CSIDH and SIKE

This section presents zero-value and correlation attacks on CSIDH and SIKE. We expand on previous zero-value attacks on SIKE [47] by analyzing the behavior of the zero curve E_0 and six curve E_6 in CSIDH and SIKE. We demonstrate an attack on static-key CSIDH and SIKE implementations that recovers bits of the secret key by observing via zero-value-based resp. exploiting correlation-collision-based side-channel analysis whether secret isogeny walks pass over the zero or six curve. We apply this attack to fully recover secret keys of SIKE and two state-of-the-art CSIDH-based implementations: CTIDH and SQALE. We show the feasibility of exploiting side-channel information for the proposed attacks based on simulations with various realistic noise levels. Additionally, we discuss countermeasures to prevent zero-value and correlation-collision attacks against CSIDH and SIKE in our attacker model.

We use the fact that both CSIDH and SIKE use elliptic curves in Montgomery form $E_a\colon y^2 = x^3 + ax^2 + x$ with the Montgomery coefficient $a \in \mathbb{F}_p$ in CSIDH resp. $a \in \mathbb{F}_{p^2}$ in SIKE. We exploit representations of the curves E_0 and E_6 that either contain zero values or observable correlations. Both schemes essentially perform secret isogeny walks. That is, they compute a sequence of secret isogenies that are determined by the secret key. The main idea is that we construct public keys such that the target performs a path that potentially passes such a curve. Whenever this happens, we can observe this via side-channel analysis (SCA), and recover bits of the secret key.

Recovering CSIDH keys

To decrease computational cost by avoiding costly inversions, in CSIDH the curve E_a is represented using *projective* coordinates for $a \in \mathbb{F}_p$. The following two are used in current CSIDH-based implementations:

- the Montgomery form $(A : C)$, such that $a = A/C$, with C non-zero,
- and the alternative Montgomery form $(A + 2C : 4C)$, such that $a = A/C$, with C non-zero.

This means that the zero curve E_0, which is a valid supersingular curve in CSIDH, is represented either as $(0 : C)$ or $(2C : 4C)$ with some random $C \in \mathbb{F}_p$.

The representation $(0 : C)$ contains a zero value, which cannot be prevented by coordinate randomization. Thus, whenever a secret isogeny walk passes over E_0, we can detect this zero value using standard SCA techniques, as, e.g., described in [47].

In the representation $(2C : 4C)$ with C non-zero, if $2C < p/2$, it is clear that $4C$ is a simple bit shift of $2C$. This can easily be observed via correlation-collision attacks.

A CSIDH isogeny walk consists of a series of isogeny steps that we denote as $\mathfrak{l}_i^{\pm 1}$. If the sequence of these isogenies is fixed, and the first $k-1$ key bits are known, it is trivial to construct public keys E_{PK} resp. $\tilde{E}_{\mathrm{PK}}$ such that paths pass through E_0 after step k if the k-th step applies $\mathfrak{l}_i$ resp. $\mathfrak{l}_i^{-1}$. Observing for which of these cases this occurs then leaks the k-th secret bit. Following this approach, we can recover CSIDH secret keys adaptively bit by bit. The general approach is depicted in Figure 6.2.

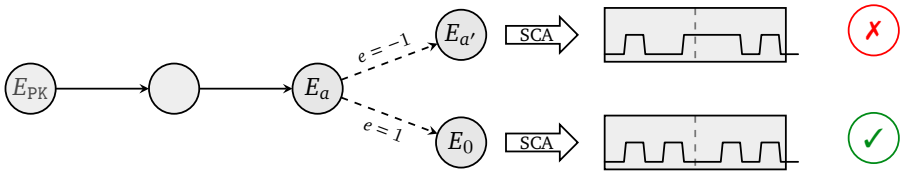


Figure 6.2: Generic approach to discover secret bits using side-channel information.

In CSIDH implementations, individual isogeny steps $\mathfrak{l}_i^{\pm 1}$ can fail with a certain probability, in which case paths do not pass E_0. However, we can repeat measurements several times for each bit to obtain a large probability for no failures to occur at least in one run.

When applied to the state-of-the-art constant-time implementations SQALE-2048 [34] resp. CTIDH-511 [16], a simulation of the attack requires an average of 8,273 resp. 85,000 measurements, see [27].

Recovering SIKE keys

SIKE implementations use a similar representation of curve coefficients. In particular, it uses Montgomery curves over $\mathbb{F}_{p^2}$ of the following form:

- The alternative Montgomery form $(A + 2C : 4C)$, such that $a = A/C$ with C non-zero. This representation is used for Alice's computations as it is the most efficient for computing 2-isogenies. It is often written as $(A_{24}^+ : C_{24})$ with $A_{24}^+ = A + 2C$ and $C_{24} = 4C$ so that $a = 2(2A_{24}^+ - C_{24})/C_{24}$.
- The form $(A+2C : A-2C)$, such that $a = A/C$, with C non-zero. This representation is used for Bob's computations as it is the most efficient for computing 3-isogenies. It is often written as $(A_{24}^+ : A_{24}^-)$ with $A_{24}^+ = A + 2C$ and $A_{24}^- = A - 2C$ so that $a = 2(A_{24}^+ + A_{24}^-)/(A_{24}^+ - A_{24}^-)$.

In both forms, the curve E_6 is represented as $(8C : 4C)$, which admits the same correlation as for CSIDH. As both values are defined over $\mathbb{F}_{p^2}$, they consist of two $\mathbb{F}_p$-values each, which simplifies their detection via SCA. Furthermore, SIKE uses primes of the form $p = 2^e * f - 1$ with $2^e \approx \sqrt{p}$, which means that modular reductions do not effect the correlation of half of the bits of $4C$ resp. $8C$.

Similar to the CSIDH attack, we assume knowledge of the first $k-1$ secret key bits, and construct public keys that lead the target over E_6 after the k-th step, and learn the k-th bit from this. Constructing such public keys is more involved than in CSIDH. However,

theory about dual isogenies allows for computing suitable public keys via backtracking, as detailed in [27]. In contrast to CSIDH, isogeny steps cannot fail in SIKE, which substantially simplifies the attack.

When applied to the SIKE Round 3 software, the number of required measurements in a simulation of the attack for different parameter sets is detailed in Table 6.1.

Scheme	SIKEp434	SIKEp503	SIKEp610	SIKEp751
Samples	228	265	320	398

Table 6.1: Required number of samples to reconstruct secret keys in simulations.

Countermeasures

Both attacks on CSIDH variants and SIKE are currently not mitigated by public key validation methods. In CSIDH, the attack provides valid supersingular curves as public keys, which means that the key validation cannot detect the attack. In SIDH and SIKE, the public keys are not honestly generated. However, full key validation is conjectured not to be possible, and the public keys satisfy all properties that can currently be validated.

Potential countermeasures include masking by precomposing the secret isogeny with an ephemeral isogeny of lower degree, and postcomposing by its dual. This steers isogeny paths away from vulnerable curves, and an attacker needs to correctly guess the ephemeral masking isogeny in order to learn key bits. On the other hand, it remains an open question to find curve representations that do not admit such correlation or zero-value attacks.

Bibliography

[1] Carlos Aguilar Melchor et al. *HQC*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2020 (cit. on p. 18).

[2] Carlos Aguilar Melchor et al. *HQC*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions. National Institute of Standards and Technology, 2022 (cit. on p. 7).

[3] Carlos Aguilar Melchor et al. *RQC*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. National Institute of Standards and Technology, 2019 (cit. on p. 21).

[4] Gorjan Alagic et al. *NISTIR 8309: Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*. https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8309.pdf. 2020 (cit. on pp. 10, 16, 18–21, 26–29, 31, 32, 34–36, 39, 41, 42).

[5] Gorjan Alagic et al. *NISTIR 8413: Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*. https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413.pdf. 2022 (cit. on pp. 10, 18, 23, 26, 27, 29, 30, 33, 34, 36, 38, 39, 42, 44, 45).

[6] Martin R. Albrecht et al. *Classic McEliece*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2022 (cit. on pp. 7, 17).

[7] Erdem Alkim, Philipp Jakubeit, and Peter Schwabe. "NewHope on ARM Cortex-M". In: *Security, Privacy, and Applied Cryptography Engineering — SPACE 2016*. Vol. 10076. Lecture Notes in Computer Science. 2016, pp. 332–349. DOI: 10.1007/978-3-319-49445-6_19 (cit. on p. 60).

[8] Erdem Alkim et al. "ISA Extensions for Finite Field Arithmetic". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.3 (2020). https://tches.iacr.org/index.php/TCHES/article/view/8589, pp. 219–242. ISSN: 2569-2925. DOI: 10.13154/tches.v2020.i3.219-242 (cit. on pp. 59, 60).

[9] Erdem Alkim et al. "Post-quantum Key Exchange - A New Hope". In: *USENIX Security 2016: 25th USENIX Security Symposium*. Ed. by Thorsten Holz and Stefan Savage. USENIX Association, Aug. 2016, pp. 327–343 (cit. on p. 32).

[10] Nicolas Aragon et al. *BIKE*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2020 (cit. on p. 15).

[11] Nicolas Aragon et al. *BIKE*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions. National Institute of Standards and Technology, 2022 (cit. on p. 7).

[12] Nicolas Aragon et al. *ROLLO*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. National Institute of Standards and Technology, 2019 (cit. on p. 20).

[13] Hayo Baan et al. "Round5: Compact and Fast Post-quantum Public-Key Encryption". In: *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*. Ed. by Jintai Ding and Rainer Steinwandt. Springer, Heidelberg, 2019, pp. 83–102. DOI: 10.1007/978-3-030-25510-7_5 (cit. on p. 35).

[14] Marco Baldi et al. *LEDAcrypt*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. National Institute of Standards and Technology, 2019 (cit. on p. 19).

[15] Marco Baldi et al. "LEDAcrypt: QC-LDPC Code-Based Cryptosystems with Bounded Decryption Failure Rate". In: *Code-Based Cryptography - 7th International Workshop, CBC 2019, Darmstadt, Germany, May 18-19, 2019, Revised Selected Papers*. Ed. by Marco Baldi, Edoardo Persichetti, and Paolo Santini. Vol. 11666. Lecture Notes in Computer Science. Springer, Heidelberg, 2019, pp. 11–43 (cit. on p. 19).

[16] Gustavo Banegas et al. "CTIDH: faster constant-time CSIDH". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.4 (2021). https://tches.iacr.org/index.php/TCHES/article/view/9069, pp. 351–387. ISSN: 2569-2925. DOI: 10.46586/tches.v2021.i4.351-387 (cit. on pp. 24, 64, 66).

[17] Daniel J. Bernstein et al. "NTRU Prime: Reducing Attack Surface at Low Cost". In: *SAC 2017: 24th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Carlisle Adams and Jan Camenisch. Vol. 10719. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2017, pp. 235–260. DOI: 10.1007/978-3-319-72565-9_12 (cit. on p. 34).

[18] Ward Beullens and Bart Preneel. "Field Lifting for Smaller UOV Public Keys". In: *Progress in Cryptology - INDOCRYPT 2017: 18th International Conference in Cryptology in India*. Ed. by Arpita Patra and Nigel P. Smart. Vol. 10698. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2017, pp. 227–246 (cit. on p. 40).

[19] Ward Beullens et al. *LUOV*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. National Institute of Standards and Technology, 2019 (cit. on p. 40).

[20] Nina Bindel et al. *qTESLA*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. National Institute of Standards and Technology, 2019 (cit. on p. 28).

[21] Joppe W. Bos et al. "CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM". In: *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*. IEEE, 2018, pp. 353–367 (cit. on pp. 7, 29).

[22] Joppe W. Bos et al. “Frodo: Take off the Ring! Practical, Quantum-Secure Key Exchange from LWE”. In: *ACM CCS 2016: 23rd Conference on Computer and Communications Security*. Ed. by Edgar R. Weippl et al. ACM Press, Oct. 2016, pp. 1006–1018. DOI: 10.1145/2976749.2978425 (cit. on p. 30).

[23] Leon Botros, Matthias J. Kannwischer, and Peter Schwabe. “Memory-Efficient High-Speed Implementation of Kyber on Cortex-M4”. In: *AFRICACRYPT 19: 11th International Conference on Cryptology in Africa*. Ed. by Johannes Buchmann, Abderrahmane Nitaj, and Tajje eddine Rachidi. Vol. 11627. Lecture Notes in Computer Science. Springer, Heidelberg, July 2019, pp. 209–228. DOI: 10.1007/978-3-030-23696-0_11 (cit. on p. 60).

[24] Giacomo Bruno et al. *Cryptographic Smooth Neighbors*. Cryptology ePrint Archive, Paper 2022/1439. https://eprint.iacr.org/2022/1439. 2022 (cit. on pp. 25, 49).

[25] Johannes A. Buchmann, Erik Dahmen, and Andreas Hülsing. “XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions”. In: *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*. Ed. by Bo-Yin Yang. Springer, Heidelberg, 2011, pp. 117–129. DOI: 10.1007/978-3-642-25405-5_8 (cit. on p. 46).

[26] Fabio Campos, Juliane Krämer, and Marcel Müller. “Safe-Error Attacks on SIKE and CSIDH”. In: *Security, Privacy, and Applied Cryptography Engineering - 11th International Conference, SPACE 2021, Kolkata, India, December 10-13, 2021, Proceedings*. Ed. by Lejla Batina, Stjepan Picek, and Mainack Mondal. Vol. 13162. Lecture Notes in Computer Science. Springer, Heidelberg, 2021, pp. 104–125 (cit. on p. 61).

[27] Fabio Campos et al. *Patient Zero and Patient Six: Zero-Value and Correlation Attacks on CSIDH and SIKE*. Cryptology ePrint Archive, Report 2022/904. https://eprint.iacr.org/2022/904. 2022 (cit. on pp. 66, 67).

[28] Fabio Campos et al. *Trouble at the CSIDH: Protecting CSIDH with Dummy-Operations against Fault Injection Attacks*. Cryptology ePrint Archive, Report 2020/1005. https://eprint.iacr.org/2020/1005. 2020 (cit. on pp. 62, 64).

[29] Anne Canteaut and François-Xavier Standaert, eds. *Advances in Cryptology – EUROCRYPT 2021, Part I*. Vol. 12696. Lecture Notes in Computer Science. Springer, Heidelberg, Oct. 2021.

[30] A. Casanova et al. *GeMSS*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2020 (cit. on p. 39).

[31] Wouter Castryck and Thomas Decru. *An efficient key recovery attack on SIDH (preliminary version)*. Cryptology ePrint Archive, Report 2022/975. https://eprint.iacr.org/2022/975. 2022 (cit. on p. 23).

[32] Wouter Castryck et al. “CSIDH: An Efficient Post-Quantum Commutative Group Action”. In: *Advances in Cryptology – ASIACRYPT 2018, Part III*. Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11274. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2018, pp. 395–427. DOI: 10.1007/978-3-030-03332-3_15 (cit. on pp. 24, 64).

[33] Melissa Chase et al. “Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives”. In: *ACM CCS 2017: 24th Conference on Computer and Communications Security*. Ed. by Bhavani M. Thuraisingham et al. ACM Press, 2017, pp. 1825–1842. DOI: 10.1145/3133956.3133997 (cit. on p. 44).

[34] Jorge Chávez-Saab et al. *The SQALE of CSIDH: Sublinear Vélu Quantum-resistant isogeny Action with Low Exponents*. Cryptology ePrint Archive, Report 2020/1520. https://eprint.iacr.org/2020/1520. 2020 (cit. on p. 66).

[35] Cong Chen et al. *NTRU*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2020 (cit. on p. 33).

[36] Ming-Shing Chen et al. “From 5-Pass MQ-Based Identification to MQ-Based Signatures”. In: *Advances in Cryptology – ASIACRYPT 2016, Part II*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10032. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2016, pp. 135–165. DOI: 10.1007/978-3-662-53890-6_5 (cit. on p. 41).

[37] J. B. Conrey, M. A. Holmstrom, and T. L. McLaughlin. “Smooth neighbors”. In: *Experimental Mathematics* 22.2 (2013), pp. 195–202 (cit. on p. 49).

[38] James W. Cooley and John W. Tukey. “An algorithm for the machine calculation of complex Fourier series”. In: *Mathematics of computation* 19.90 (1965), pp. 297–301 (cit. on p. 60).

[39] David Cooper et al. *NIST SP 800-208: Recommendation for Stateful Hash-Based Signature Schemes*. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-208.pdf. 2020 (cit. on p. 46).

[40] Craig Costello. “B-SIDH: Supersingular Isogeny Diffie-Hellman Using Twisted Torsion”. In: *Advances in Cryptology – ASIACRYPT 2020, Part II*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2020, pp. 440–463. DOI: 10.1007/978-3-030-64834-3_15 (cit. on pp. 25, 49).

[41] Craig Costello, Michael Meyer, and Michael Naehrig. “Sieving for Twin Smooth Integers with Solutions to the Prouhet-Tarry-Escott Problem”. In: *Advances in Cryptology – EUROCRYPT 2021, Part I*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. Lecture Notes in Computer Science. Springer, Heidelberg, Oct. 2021, pp. 272–301. DOI: 10.1007/978-3-030-77870-5_10 (cit. on p. 25).

[42] *CRYSTALS-Dilithium*. https://pq-crystals.org/dilithium/index.shtml, (accessed 18.02.2022) (cit. on p. 26).

[43] *CRYSTALS-Kyber*. https://pq-crystals.org/kyber/index.shtml, (accessed 18.02.2022) (cit. on p. 29).

[44] Jan-Pieter D'Anvers et al. *SABER*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2020 (cit. on p. 36).

[45] Jan-Pieter D'Anvers et al. "Saber: Module-LWR Based Key Exchange, CPA-Secure Encryption and CCA-Secure KEM". In: *AFRICACRYPT 18: 10th International Conference on Cryptology in Africa*. Ed. by Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi. Vol. 10831. Lecture Notes in Computer Science. Springer, Heidelberg, May 2018, pp. 282–305. DOI: 10.1007/978-3-319-89339-6_16 (cit. on p. 36).

[46] Luca De Feo, Antonin Leroux, and Benjamin Wesolowski. *New algorithms for the Deuring correspondence: SQISign twice as fast*. Cryptology ePrint Archive, Report 2022/234. https://eprint.iacr.org/2022/234. 2022 (cit. on pp. 25, 49).

[47] Luca De Feo et al. *SIKE Channels*. Cryptology ePrint Archive, Report 2022/054. https://eprint.iacr.org/2022/054. 2022 (cit. on p. 65).

[48] Luca De Feo et al. "SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies". In: *Advances in Cryptology – ASIACRYPT 2020, Part I*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12491. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2020, pp. 64–93. DOI: 10.1007/978-3-030-64837-4_3 (cit. on p. 25).

[49] Jintai Ding and Dieter Schmidt. "Rainbow, a New Multivariable Polynomial Signature Scheme". In: *ACNS 05: 3rd International Conference on Applied Cryptography and Network Security*. Ed. by John Ioannidis, Angelos Keromytis, and Moti Yung. Vol. 3531. Lecture Notes in Computer Science. Springer, Heidelberg, June 2005, pp. 164–175. DOI: 10.1007/11496137_12 (cit. on p. 42).

[50] Jintai Ding and Rainer Steinwandt, eds. *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*. Springer, Heidelberg, 2019.

[51] Jintai Ding et al. *Rainbow*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2020 (cit. on p. 42).

[52] Jintai Ding et al. "The Nested Subset Differential Attack - A Practical Direct Attack Against LUOV Which Forges a Signature Within 210 Minutes". In: *Advances in Cryptology – EUROCRYPT 2021, Part I*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. Lecture Notes in Computer Science. Springer, Heidelberg, Oct. 2021, pp. 329–347. DOI: 10.1007/978-3-030-77870-5_12 (cit. on p. 40).

[53] Léo Ducas et al. "CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018.1 (2018). https://tches.iacr.org/index.php/TCHES/article/view/839, pp. 238–268. ISSN: 2569-2925. DOI: 10.13154/tches.v2018.i1.238-268 (cit. on pp. 7, 26).

[54] Oscar Garcia-Morchon et al. *Round5*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. National Institute of Standards and Technology, 2019 (cit. on p. 35).

[55] W. Morven Gentleman and Gordon Sande. “Fast Fourier Transforms: for fun and profit”. In: *Proceedings of the November 7-10, 1966, fall joint computer conference*. ACM. 1966, pp. 563–578 (cit. on p. 60).

[56] Ruben Gonzalez et al. “Verifying Post-Quantum Signatures in 8 kB of RAM”. In: *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021*. Ed. by Jung Hee Cheon and Jean-Pierre Tillich. Springer, Heidelberg, 2021, pp. 215–233. DOI: 10.1007/978-3-030-81293-5_12 (cit. on pp. 56, 58).

[57] Mike Hamburg. *Three Bears*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. National Institute of Standards and Technology, 2019 (cit. on p. 37).

[58] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. “NTRU: A Ring-Based Public Key Cryptosystem”. In: *Third Algorithmic Number Theory Symposium (ANTS)*. Vol. 1423. Lecture Notes in Computer Science. Springer, Heidelberg, June 1998, pp. 267–288 (cit. on p. 33).

[59] Andreas Hulsing et al. *SPHINCS+*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2020 (cit. on pp. 7, 45).

[60] David Jao and Luca De Feo. “Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies”. In: *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*. Ed. by Bo-Yin Yang. Springer, Heidelberg, 2011, pp. 19–34. DOI: 10.1007/978-3-642-25405-5_2 (cit. on pp. 23, 25).

[61] David Jao et al. *SIKE*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2020 (cit. on p. 23).

[62] David Jao et al. *SIKE*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions. National Institute of Standards and Technology, 2022 (cit. on p. 7).

[63] Daniel Kales and Greg Zaverucha. *Improving the Performance of the Picnic Signature Scheme*. Cryptology ePrint Archive, Report 2020/427. https://eprint.iacr.org/2020/427. 2020 (cit. on p. 44).

[64] Neal Koblitz. “Elliptic curve cryptosystems”. In: *Mathematics of computation* 48.177 (1987), pp. 203–209 (cit. on p. 7).

[65] “Kryptographische Verfahren: Empfehlungen und Schlüssellangen”. In: *Technische Richtlinie TR-02102-1, Bundesamt fur Sicherheit in der Informationstechnik* (2020) (cit. on pp. 17, 30).

[66] Greg Kuperberg. “A Subexponential-Time Quantum Algorithm for the Dihedral Hidden Subgroup Problem”. In: 35.1 (2005), pp. 170–188 (cit. on p. 24).

[67] Charles Eric Leiserson et al. *Introduction to algorithms*. Vol. 6. MIT press Cambridge, MA, 2001 (cit. on p. 59).

[68] Xianhui Lu et al. *LAC*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. National Institute of Standards and Technology, 2019 (cit. on p. 31).

[69] Xianhui Lu et al. *LAC: Practical Ring-LWE Based Public-Key Encryption with Byte-Level Modulus*. Cryptology ePrint Archive, Report 2018/1009. https://eprint.iacr.org/2018/1009. 2018 (cit. on p. 31).

[70] Luciano Maino and Chloe Martindale. *An attack on SIDH with arbitrary starting curve*. Cryptology ePrint Archive, Report 2022/1026. https://eprint.iacr.org/2022/1026. 2022 (cit. on p. 23).

[71] Robert J. McEliece. *A public-key cryptosystem based on algebraic coding theory*. The Deep Space Network Progress Report 42-44. https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF. Jet Propulsion Laboratory, California Institute of Technology, 1978, pp. 114–116 (cit. on p. 17).

[72] Michael Meyer, Fabio Campos, and Steffen Reith. "On Lions and Elligators: An Efficient Constant-Time Implementation of CSIDH". In: *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*. Ed. by Jintai Ding and Rainer Steinwandt. Springer, Heidelberg, 2019, pp. 307–325. DOI: 10.1007/978-3-030-25510-7_17 (cit. on p. 62).

[73] Victor S. Miller. "Use of Elliptic Curves in Cryptography". In: *Advances in Cryptology – CRYPTO'85*. Ed. by Hugh C. Williams. Vol. 218. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1986, pp. 417–426. DOI: 10.1007/3-540-39799-X_31 (cit. on p. 7).

[74] Michael Naehrig et al. *FrodoKEM*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2020 (cit. on p. 30).

[75] National Institute of Standards and Technology (NIST). *Post-Quantum Cryptography Standardization*. https://csrc.nist.gov/Projects/post-quantum-cryptography/Post-Quantum-Cryptography-Standardization. 2016 (cit. on p. 7).

[76] Ruben Niederhagen, Johannes Roth, and Julian Wälde. *Streaming SPHINCS+ for Embedded Devices using the Example of TPMs*. Cryptology ePrint Archive, Report 2021/1072. https://eprint.iacr.org/2021/1072. 2021 (cit. on p. 58).

[77] *NIST PQC forum*. https://groups.google.com/a/list.nist.gov/g/pqc-forum (cit. on pp. 10, 36).

[78] David Noack et al. *QuantumRISC WP1 Report: Use Cases and Requirements*. 2020. URL: https://quantumrisc.org/results/quantumrisc-wp1-report.pdf (cit. on pp. 8, 12–16, 48).

[79] Hiroshi Onuki et al. "(Short Paper) A Faster Constant-Time Algorithm of CSIDH Keeping Two Points". In: *IWSEC 19: 14th International Workshop on Security, Advances in Information and Computer Security*. Ed. by Nuttapong Attrapadung and Takeshi Yagi. Vol. 11689. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2019, pp. 23–33. DOI: 10.1007/978-3-030-26834-3_2 (cit. on p. 62).

[80] Thomas Poppelmann et al. *NewHope*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. National Institute of Standards and Technology, 2019 (cit. on p. 32).

[81] Thomas Prest et al. *FALCON*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2020 (cit. on pp. 7, 27).

[82] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications of the Association for Computing Machinery* 21.2 (1978), pp. 120–126 (cit. on p. 7).

[83] Damien Robert. *Breaking SIDH in polynomial time*. Cryptology ePrint Archive, Report 2022/1038. https://eprint.iacr.org/2022/1038. 2022 (cit. on p. 23).

[84] Johannes Roth, Evangelos G. Karatsiolis, and Juliane Krämer. "Classic McEliece Implementation with Low Memory Footprint". In: *Smart Card Research and Advanced Applications - 19th International Conference, CARDIS 2020, Virtual Event, November 18-19, 2020, Revised Selected Papers*. Ed. by Pierre-Yvan Liardet and Nele Mentens. Vol. 12609. Lecture Notes in Computer Science. Springer, Heidelberg, 2020, pp. 34–49 (cit. on pp. 53, 58).

[85] Sujoy Sinha Roy et al. "Compact Ring-LWE Cryptoprocessor". In: *Cryptographic Hardware and Embedded Systems – CHES 2014*. Ed. by Lejla Batina and Matthew Robshaw. Vol. 8731. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2014, pp. 371–391. DOI: 10.1007/978-3-662-44709-3_21 (cit. on p. 60).

[86] Simona Samardjiska et al. *MQDSS*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. National Institute of Standards and Technology, 2019 (cit. on p. 41).

[87] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM J. Comput.* 26.5 (1997), pp. 1484–1509 (cit. on p. 7).

[88] Chengdong Tao, Albrecht Petzoldt, and Jintai Ding. "Efficient Key Recovery for All HFE Signature Variants". In: *Advances in Cryptology – CRYPTO 2021, Part I*. Ed. by Tal Malkin and Chris Peikert. Vol. 12825. Lecture Notes in Computer Science. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 70–93. DOI: 10.1007/978-3-030-84242-0_4 (cit. on p. 39).

[89] Bo-Yin Yang, ed. *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*. Springer, Heidelberg, 2011.

[90] Greg Zaverucha et al. *Picnic*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2020 (cit. on p. 44).