



# CONFIDENTIAL COMPUTING AND PRIVACY-PRESERVING TECHNOLOGIES FOR 6G

## D2.1 POST-QUANTUM ANALYSIS REPORT



## PROJECT INFORMATION

Call	HORIZON-JU-SNS-2022
Type of Action	HORIZON-JU-RIA HORIZON JU Research and Innovation Actions
Project start date	01/01/2023
Duration	36 months
GA No	101096435

## DELIVERABLE INFORMATION

Deliverable WP	WP2
Deliverable Task	Tasks T2.1
Deliverable Identifier	CONFIDENTIAL6G_D2.1
Deliverable Title	POST-QUANTUM ANALYSIS REPORT
Editor(s)	TU/e
Author(s)	TUE: Emanuele Di Giandomenico, Tanja Lange, Sven Schäge TU Graz: Lena Heimberger, Christian Rechberger, Fabian Schmid ZEN: Stevan Jokic
Reviewer(s)	Julian Kotzur (FAU), Thom Sijpesteijn (TNO)
Contractual Date of Delivery	31/12/2023
Submission Date	22/12/2023
Dissemination Level	PU
Status	Final
Version	1.0
File Name	D2.1 Post-quantum Analysis Report_v1.0

### Disclaimer

The information and views set out in this deliverable are those of the author(s) and do not necessarily reflect the official opinion of the European Union. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein.

### Note

It should be noted that this document has been produced with the use of Overleaf.

## D2.1 Post-quantum Analysis Report

Version	Date	Description of Change
v1.0	22/12/2023	Final version, submitted to EC through SyGMa

# Table of Contents

List of Acronyms and Abbreviations	6
List of Figures	7
List of Tables	8
Executive Summary	9
<b>1 Introduction</b>	<b>11</b>
1.1 Quantum Computers and their Capabilities	11
1.2 General Threats for Cryptography – Symmetric Crypto	11
1.3 General Threats for Cryptography – Asymmetric Crypto	12
1.4 Families of Post-Quantum Secure Cryptographic Systems	12
1.4.1 Code-based Cryptography	12
1.4.2 Lattice-based Cryptography	13
1.4.3 Multi-Variate Cryptography	13
1.4.4 Hash-based Cryptography	13
1.5 A New Dawn- The algorithms of NISTPQC	13
1.6 Construction Ideas and Alternative	14
1.6.1 KEMs	14
1.6.2 Digital Signatures	15
1.6.3 Hash-based Signatures and SPHINCS <sup>+</sup>	17
1.7 Background on Lattices	17
1.7.1 Gram-Schmidt Orthogonalization	18
1.7.2 The determinant	18
1.7.3 Minimum Distance	18
1.7.4 Average vs. Worst-case problems	19
1.8 Classical lattice problems are not reflected in state-of-the-art cryptosystems	19
1.8.1 Shortest Vector Problem (SVP)	19
1.8.2 Short Integers Solutions Problem (SIS)	19
1.8.3 Learning With Errors Problem (LWE)	20
1.9 Lattice Based Crypto and Regev System	21
1.10 Kyber	21
1.10.1 Key Encapsulation Mechanism	21
1.10.2 Notation and Parameters	21
1.10.3 Number-Theoretic Transform (NTT)	22
1.10.4 Algorithms	23
1.10.5 Kyber.CPAPKE	25
1.10.6 Kyber.CCAKEM	27
1.10.7 Instantiated Primitives	27
1.10.8 Security Assumption	28
1.11 Digital Signatures	28
1.12 Dilithium	29
1.12.1 Notation and Parameters	29
1.12.2 Hashing	29
1.12.3 Algorithms	30
1.12.4 Signature	31

## D2.1 Post-quantum Analysis Report

1.12.5	Security Assumption	31
1.13	SPHINCS <sup>+</sup>	32
1.13.1	WOTS <sup>+</sup> One-Time Signatures	32
1.13.2	The SPHINCS <sup>+</sup> Hypertree	35
1.13.3	FORS: Forest Of Random Subsets	40
1.13.4	The SPHINCS <sup>+</sup> Construction	43
1.13.5	Security Assumption	46
1.14	Privacy-Preserving Authentication	46
1.14.1	Existing Solutions - Direct Anonymous Attestation	47
1.14.2	Existing Group Signatures	47
1.14.3	Existing Oblivious Pseudorandom Functions	47
1.14.4	Existing Zero-Knowledge Proofs	48
2	Threats on the 6G Infrastructure	49
2.1	Identity Concealment	49
2.2	Symmetric Cryptography	50
3	Migration to PQC and Challenges	51
3.1	Strategies for Smooth Transitions	51

## List of Acronyms and Abbreviations

Abbreviation	Definition
AES	Advanced Encryption Standard
CBOM	Cryptographic Bill of Materials
DDH	Decisional Diffie-Hellman
ECIES	Elliptic Curve Integrated Encryption Scheme
FIPS	Federal Information Processing Standards
FORS	Forest Of Random Subsets
G,H	Hash Functions
IETF	Internet Engineering Task Force
IND-CCA2	Indistinguishability under Adaptive Chosen Ciphertext Attack
IND-CPA	Indistinguishability under Chosen Plaintext Attack
IMSI	International Mobile Subscriber Identity
IPsec	Internet Protocol Security
KDF	Key Derivation Function
KEM	Key Encapsulation Mechanism
LWE	Learning With Errors
MCC	Mobile Country Code
MDPC	Moderate Density Parity-Check
MLWE	Module-Learning With Errors
ML-DSA	Module-Learning Digital Signature Algorithm
ML-KEM	Module-Learning Key Encapsulation Mechanism
MSIN	Mobile Subscription Identification Number
NIST	National Institute of Standards and Technology
NTRU	NTRUEncrypt
NTT	Number Theoretic Transform
OPRF	Oblivious Pseudo-Random Function
OW-CPA	One-Way Chosen Plaintext Attack
PKE	Public Key Encryption
PQ	Post-Quantum
PQC	Post-Quantum Cryptography
PRF	Pseudo-Random Function
QC-LDPC	Quantum-safe Low-Density Parity-Check
QC-MDPC	Quantum-safe Moderate Density Parity-Check
QKD	Quantum Key Distribution
RLWE	Ring Learning With Errors
RSA	Rivest-Shamir-Adleman System
SHA3	Secure Hash Algorithm 3
SHAKE	Secure Hash Algorithm and KECCAK
SLH-DSA	Stateful Hash-Based Digital Signature Algorithm
SUPI	Subscription Permanent Identifier
TLS	Transport Layer Security
WOTS+	Winternitz One-Time Signature Plus
XMSS	eXtended Merkle Signature Scheme

## List of Figures

### List of Figures

1	Implementation of the Parse: $\mathcal{B}^* \rightarrow R_q^n$ function. . . . .	23
2	Implementation of the $\text{CBD}_\eta: \mathcal{B}^{64\eta} \rightarrow R_q$ function. . . . .	24
3	Implementation of the $\text{Decode}_\ell: \mathcal{B}^{32\ell} \rightarrow R_q$ function. . . . .	24
4	Key generation algorithm $\text{Kyber.CPAPKE.KeyGen}$ . . . . .	25
5	Encryption algorithm $\text{Kyber.CPAPKE.Enc}$ . . . . .	26
6	Decryption algorithm $\text{Kyber.CPAPKE.Dec}$ . . . . .	26
7	Key generation algorithm $\text{Kyber.CCAKEM.KeyGen}$ . . . . .	27
8	Encryption algorithm $\text{Kyber.CCAKEM.Enc}$ . . . . .	27
9	Decryption algorithm $\text{Kyber.CCAKEM.Dec}$ . . . . .	27
10	$\text{SampleInBall}(\varrho)$ generates a controlled pseudorandom binary sequence of length 256. . . . .	29
11	Algorithms for modular arithmetic $\text{Power2Round}_q$ , hint generation $\text{MakeHint}_q$ , and utilization $\text{UseHint}_q$ . Supplementary functions $\text{Decompose}_q$ , $\text{HighBits}_q$ , and $\text{LowBits}_q$ . .	30
12	Key pair generation $\text{Gen}$ , signature creation $\text{Sign}$ , and verification $\text{Verify}$ algorithms. .	31
13	Structure of a HT signature in the HT scheme. Each layer represents an XMSS signature, and the entire signature is a byte string of length $(h + d \cdot \text{len}) \cdot n$ . . . . .	39



## List of Tables

### List of Tables

1	Lattice-based KEMs with approximately NIST's security level I (120). . . . .	15
2	Lattice-based KEMs with approximately NIST's security level III (180). . . . .	16
3	Lattice-based KEMs with approximately NIST's security level V (260). . . . .	16

## Executive Summary

This document relates to WP 2.1 of the Confidential6G project. Its main purpose is to describe how 6G can be made resistant to quantum attackers and thus provide better long-term security with a special focus on data confidentiality and privacy-preserving computation enablers. To this end, this document identifies where 6G will require post-quantum secure building blocks to decrease the susceptibility against quantum attackers. Moreover, it sketches ways to migrate towards a post-quantum secure implementations and infrastructure. The analysis aims at extending existing constructions, where possible, to maximize code reuse. New cryptographic primitives should accommodate the specific constraints of 6G. Moreover, complex protocols should be proven secure while relying on post-quantum security models.

We identify three fundamental usage scenarios of asymmetric cryptography in 5G that will likely transfer to 6G: 1) TLS [47] and 2) IPSec [27] for securing the most critical parts of the radio-access- and core network, and 3) ECIES for identity concealment. These applications rely on key encapsulation mechanisms/public key encryption and digital signatures. Unfortunately, existing standards of these primitives (and in particular those used in 5G) are not secure in the presence of quantum computers and thus they have to be updated with post-quantum versions in 6G. For high interoperability, we recommend relying on the upcoming standards of NIST: Kyber, Dilithium, and SPHINCS+ which are currently still in the process of standardization by NIST. Since the final standards have not been published yet, we provide a detailed description of the underlying submissions.

There are some more KEM systems still under consideration by NIST in a fourth round as well as an additional round for Digital Signatures and some under consideration by ISO, but giving descriptions of all these systems is beyond the scope of this document. See [11] for descriptions of all round-3 candidates from NIST's competition. Symmetric primitives on the other hand are threatened generically by Grover's algorithm for pre-image search. For ciphers without any further structure, considering only the number of iterations in Grover's algorithm, the security level is halved. However, these iterations need to happen sequentially and thus require a very stable quantum computer. The NIST competition limited the depth of the quantum circuits which limits the power of Grover's attack even further. It is generally sufficient to increase the key sizes of the used symmetric primitives and if the depth is limited or the costs of the oracle calls are taken into account the key size less than doubles. To some extent, attack improvements under multi-target settings have matching quantum speedups but the benefits are smaller. So, 256-bit keys generally suffice to guarantee a security level of 128 bits even under quantum attacks.

We propose the following migration strategy: first identify the usage and purpose of cryptographic components, document those systems in well-defined Cryptographic Bill of Materials (CBOM), then replace public-key systems with corresponding ones that resist attacks by quantum computers and ensure that symmetric ciphers use sufficiently large keys. The substitution of public-key encryption with post-quantum variants has a higher priority than that of digital signatures. This is mainly due to the fact that signatures guarantee authenticity – assuring the communicating parties that they are really communicating with who they think they are – which matters only at the moment of the communication and thus as long as large-scale quantum computers do not exist, authenticity can be ensured with pre-quantum solutions. However, public-key encryption protects confidentiality of the communication and protect against “store-now-decrypt-later” attacks, which are a concern that requires urgent action as attacks may record current traffic and decrypt it later once a large-scale quantum computer is available. Furthermore, in modern cryptographic protocols, entity authentication does not need to rely on digital signatures at all and instead could be performed via pre-shared keys or KEMs as well.

We note that our changes when applied to TLS or IPSec will result in new non-standard

## D2.1 Post-quantum Analysis Report

protocols. At the same time, the community has already produced some suggestions for post-quantum versions of TLS and IPSec differing in which properties of the old protocols they preserve. Discussions continue in the IETF working groups and on the IETF “Post Quantum Cryptography discussion list” [pgc@ietf.org](mailto:pgc@ietf.org) and it is not clear when this process will lead to replacement protocols after the NIST standards are published. We believe that, ultimately, 6G should rely on these new standards. In the meantime, we propose to use the results of this document to design post-quantum secure versions of cryptographic protocols in WP 2.4 and in particular a sound, post-quantum secure version of TLS. This guarantees that 6G can readily rely on post-quantum secure building blocks. At the same time, we propose to use the insights gained from developing protocols for 6G to influence the standard development of TLS and IPSec towards better applicability in 6G.

## 1 Introduction

The advent of quantum computers will have major implications for the security of our digital infrastructure. It is important to analyze what these implications are and to develop new techniques to base our security on in the future. Let us begin by sketching the overall threat scenario introduced by quantum computers.

### | 1.1 Quantum Computers and their Capabilities

Quantum computers are computers that exploit principles of quantum mechanics to solve certain tasks much more efficiently than conventional computers. Whereas normal bits can exist in a state of either 0 or 1, quantum bits or qubits can exist in a superposition of both 0 and 1 simultaneously. Intuitively, this property allows quantum computers to explore multiple possibilities at the same time. That being said, it is important to note that (universal) quantum computers are not simply faster versions of classical computers. They may not provide an advantage for all types of computations, and there are still significant technical challenges to be overcome in building large-scale, error-tolerant quantum computers. However, they excel at solving certain types of problems and importantly, this includes many important problems in cryptography. Indeed, quantum computers threaten most of the cryptographic systems used in practice. This is mostly due to two quantum algorithms that have considerably lower runtime for breaking cryptographic systems compared to the best non-quantum algorithms for the same tasks: Grover's algorithm and Shor's algorithm. Whereas Grover's algorithm has more applicability, the speedup is more moderate. Shor's algorithm has more narrow applicability but leads to considerable speedups when solving important cryptographic problems that underly many cryptosystems, essentially breaking them with no way for recovery. It is unclear when we can expect powerful quantum computers to be available that can break today's cryptosystems in practice. Recent, careful expectations for the advent of powerful quantum computers range from one to two decades given that progress in quantum computing remains comparable in the future<sup>1</sup>. We note that the practical feasibility of such systems is ultimately still unclear.

### | 1.2 General Threats for Cryptography – Symmetric Crypto

Grover's algorithm by Lev Grover (STOC'96) [31] is an algorithm for unstructured search that finds the unique input to a function  $f : X \rightarrow Y$  mapping to some given output  $y$ . This can be used to generally invert functions. Whereas classical algorithms require  $O(T)$  applications of  $f$ , Grover's algorithm has a runtime of  $O(\sqrt{T})$  where  $T = |Y|$ . This results in a quadratic speedup. Although the speedup may appear moderate in asymptotic terms, it can be used to considerably reduce the difficulty of solving brute-force key searches in practice: for seemingly unmanageable key-spaces of size  $2^{128}$ , Grover's algorithm only requires a runtime of  $2^{64}$ . It is important to note that Grover's algorithm threatens cryptographic systems in general since it improves attacks on the core building block of cryptography, one-way functions. We note additionally that, depending on the exact attack scenario (single-target vs multi-target, single-key vs. multi-key) corresponding quantum speedups are possible (in case several quantum computers are available) [6].

NIST (National Institute for Standardization and Technology) in their call for post-quantum submissions [43] limits the depth, basically the maximum sequential running time, of the quantum circuit and with that associates a cost of  $2^{170}/\text{MAXDEPTH}$  operations to solving AES-128. To run Grover as described above would require  $2^{64}$  sequential calls to an oracle which itself contributes some extra depth. At optimal MAXDEPTH the algorithm still takes  $2^{92}$  qubit operations.

<sup>1</sup>[https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Quantencomputer/Entwicklungsstand\\_QC\\_V\\_2\\_0.pdf?\\_\\_blob=publicationFile&v=2](https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Quantencomputer/Entwicklungsstand_QC_V_2_0.pdf?__blob=publicationFile&v=2)

which is significantly larger than the  $2^{64}$  just mentioned. The main difference is that the costs for the oracle evaluation are also taken into account, which mostly come from implementing a quantum circuit for computing AES. In 2016, NIST used [30] for estimating the costs of these circuits and in the meantime significant work has gone into improving the estimates. The most recent publication [36] reduces the depth of the AES circuit from more than 12 000 to 40 while increasing the width only by a factor of 4. While these speedups are significant and change the basis for comparison with the submissions to the NIST competition, it is important to note that even large-scale stable quantum computers are not expected to be available for the optimal MAXDEPTH and that thus the quantum speedups on symmetric systems are even less significant. Nevertheless, it seems wise to move to 256-bit AES also in light of multi-target attacks without quantum computers.

### | 1.3 General Threats for Cryptography – Asymmetric Crypto

Shor's algorithm (1994) [53] can be used to efficiently (i.e. in polynomial time) find periods in finite abelian groups (which is an instantiation of the more general hidden subgroup problem that does not require commutativity per se). This polynomial-time algorithm can be used to find discrete logarithms in setting based on both finite fields and elliptic curves, and to factor large integers. Thus, these problems can be solved efficiently using Shor's algorithm. This threatens the security of our modern digital infrastructure since these problems are at the core of cryptographic algorithms like RSA [19] and ECIES [1]<sup>2</sup> that our daily life relies on. It is important to note that Shor's algorithm does not give a general speedup but only applies to specific problems. In cryptography, these problems are important in asymmetric cryptography and are widespread as a foundation of public-key encryption (PKE) or key encapsulation mechanisms (KEMs), digital signatures, and key exchange protocols.

Also for Shor's algorithm the oracle calls add to the complexity and need to be studied, but the total number of calls is polynomial and thus estimates such as [49] only matter for understanding what size quantum computer would lead to a complete break and do not give any extra protection.

### | 1.4 Families of Post-Quantum Secure Cryptographic Systems

With 6G technology on the horizon, incorporating PQC into network security becomes crucial for ensuring robust and secure communications [2]. So far standardization efforts have considered four major families of algorithms for post-quantum secure cryptography: code-based, lattice-based, multivariate, and hash-based cryptography. Each family has its unique characteristics.

#### Code-based Cryptography

Code-based cryptography relies on the hardness of decoding general linear codes, with the McEliece cryptosystem from 1978 being the most prominent example [40]. It offers fast encryption and decryption, along with resistance to known quantum attacks. However, the primary drawback is the large key sizes, which may pose challenges in 6G networks, where low-latency and resource-constrained devices are expected [10]. The McEliece public key cryptosystem utilizes the properties of error-correcting codes, specifically Goppa codes, to achieve its security guarantees. The encryption process involves introducing random errors in the plaintext, which are then corrected during decryption using the private key. This approach offers remarkable performance advantages, including fast encryption and decryption, and minimal computational overhead, making it an attractive option for the latency-sensitive and resource-constrained devices. Recent research efforts have focused on improving the efficiency and

<sup>2</sup>[https://www.shoup.net/papers/iso-2\\_1.pdf](https://www.shoup.net/papers/iso-2_1.pdf)

practicality of code-based cryptography for real-world applications. For example, variants of the McEliece cryptosystem have been proposed using different types of codes, such as quasi-cyclic low-density parity-check (QC-LDPC) codes and quasi-cyclic moderate-density parity-check (QC-MDPC) codes, which result in smaller key sizes and ciphertext expansion factors.

### Lattice-based Cryptography

Lattice-based cryptography encompasses a wide range of cryptographic primitives, such as encryption schemes, digital signatures, and key exchange protocols, all of which are built on the foundation of lattice problems. Notable lattice-based cryptographic schemes include NTRU [33], and Ring-LWE, and LWE-based key exchange protocols that we detail later. NTRU, for example, is a public-key cryptosystem that achieves its security from the difficulty of finding short vectors in a lattice. Ring-LWE is a variant of the LWE problem that uses polynomial rings to construct more efficient cryptographic primitives. These schemes have demonstrated significant advantages in terms of security and efficiency, making them potential candidates for integration into the CONFIDENTIAL6G project. The complexity of lattice-based cryptographic algorithms, e.g. when drawing Gaussian noise, can pose difficulties in their implementation and introduce vulnerabilities to side-channel attacks. Advantageously, several advanced, cryptographic systems like homomorphic encryption are typically based on lattice-based cryptography. Homomorphic encryption, in turn, is an important ingredient in multi-party computation. Both are powerful tools in the design of privacy-preserving cryptography. This makes the underlying mathematical structures very versatile for cryptographic applications in practice.

### Multi-Variate Cryptography

Multivariate cryptography utilizes the hardness of solving multivariate quadratic equations over finite fields [45]. This cryptographic family is founded on the hardness of solving multivariate quadratic equations over finite fields, which has demonstrated resistance to known quantum attacks. One of the primary concerns is the long-term security of these schemes against quantum attacks, as some multivariate cryptosystems are vulnerable to certain types of attacks, such as the Gröbner basis attack [24]. Additionally, the performance of multivariate cryptographic algorithms can considerably be affected by their parameter selection, which can negatively impact the efficiency of encryption and decryption processes, as well as the size of the resulting ciphertexts and signatures.

### Hash-based Cryptography

Hash-based cryptography relies on the security of cryptographic hash functions and can be used to build digital signature schemes. Unfortunately, strong theoretical results suggest that from hash-based cryptography alone we cannot build asymmetric encryption or key exchange [35]. Often, as we describe in detail later, hash-based schemes utilize a one-time signature concept and from that construct a binary hash tree, also known as a Merkle tree, to sign multiple messages using a single public key. This approach offers several advantages, such as simplicity, resistance to quantum attacks, and the ability to leverage existing, well-studied cryptographic hash functions. An advantage of using hash-based cryptography is that hash functions are used ubiquitously in cryptography, and so there is a potential for a very small code base. Overall, hash-based cryptography is attractive for securing communication where quantum-resilient security solutions are crucial.

## | 1.5 A New Dawn- The algorithms of NISTPQC

Despite, the uncertainty with respect to the feasibility of quantum computers in the near future, the cryptographic community has set out to develop cryptosystems that are not sus-

ceptible to quantum computers. At the forefront, NIST has organized a global competition to choose the next generation of key encapsulation mechanisms and digital signatures. With the apparent understanding that key exchange can efficiently be built from these primitives, the call for submissions has not explicitly asked for key exchange protocols. NIST has in 2022 selected CRYSTALS-Kyber [13, 51] as the winner in the category PKE/KEM and CRYSTALS-Dilithium [38], FALCON [46], SPHINCS+ [34] in the category digital signatures. The CRYSTALS algorithms and FALCON are based on hard lattice problems while SPHINCS+ is based on the security of hash functions. The competition is currently investigating other candidates for standardization. Moreover, there has been a new call for additional signature schemes. At the same time, NIST is currently preparing standards ML-KEM<sup>3</sup>, ML-DSA<sup>4</sup> and, SLH-DSA<sup>5</sup> based on the winners Kyber, Dilithium, and SPHINCS<sup>+</sup>. Moreover, NIST is also developing a FIPS that specifies a digital signature algorithm derived from FALCON as an additional alternative to these standards. Since the standardization is not finished at the time of writing this document, we focus instead on describing the most recent submission variants of Kyber<sup>6</sup>, Dilithium<sup>7</sup>, and SPHINCS<sup>+</sup><sup>8</sup>. We can expect the winners to be highly important in practice. Moreover, we can expect several standards of cryptographic protocols that rely on secure KEMs and digital signatures like the widespread security protocols TLS and IPsec to be updated in the near future. So far, lattice-based cryptosystems are very competitive among the set of post-quantum secure schemes with respect to the overall efficiency and in particular when focussing on message size. Hash-based signatures rely on more conservative security assumptions but are generally less efficient in terms of signature size.

## | 1.6 Construction Ideas and Alternative

Since the early submission of the NIST winners, research has naturally progressed. In particular, new algorithms have been developed that have additional, useful features. However, these algorithms have naturally so far been analyzed with less scrutiny, giving less confidence in their design.

### KEMs

A large class of new KEMs is designed in lattices while following a very fruitful recipe that will ultimately result in provably secure schemes based on lattice assumptions. Kyber follows this overall template as well. At the core of this recipe, there is a two-step approach. First, build an OW-CPA (or IND-CPA) secure PKE scheme based on some lattice-based security assumption like LWE (learning with errors), RLWE (ring learning with errors), MLWE (module learning with errors) or on one of the rounding-based versions LWR (learning with rounding), RLWE (ring learning with rounding), MLWE (module learning with rounding). Most designs follow the Regev cryptosystem or its extension by Peikert and Lindner. Roughly, the idea is to mimic the ElGamal cryptosystem [21] that classically is based on the Decisional Diffie Hellman (DDH) assumption using a new mechanism. For a square matrix  $A$ , public keys are now matrix vector pairs  $A, b$  where  $b$  is a noisy version of  $c = A * s$  and  $s$  is the secret key vector. The noise is introduced through a small random or deterministic error in the lower bits of  $c$ :  $b \approx c$ . Likewise, we have that  $A, b'$  is a public key where  $b'$  is a noisy version of  $c' = s'^T A$  for some secret key  $s'$ . The underlying lattice-based security assumption now states that  $A, b$  and  $A, b'$  are indistinguishable from  $A, \hat{d}$  where  $\hat{d}$  is a truly random vector. This setup can be used to compute a shared key by parties  $P_1$  and  $P_2$  that hold two such public keys  $A, b$  and  $A, b'$  with corresponding secret

<sup>3</sup><https://csrc.nist.gov/pubs/fips/203/ipd>

<sup>4</sup><https://csrc.nist.gov/pubs/fips/204/ipd>

<sup>5</sup><https://csrc.nist.gov/pubs/fips/205/ipd>

<sup>6</sup><https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf>

<sup>7</sup><https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf>

<sup>8</sup><https://sphincs.org/data/sphincs+-r3.1-specification.pdf>



Table 1: Lattice-based KEMs with approximately NIST's security level I (120).

Scheme	sk	pk	ct	DFP	Sec	$ K $	Assumption	no ECC
LizarMong	640	544	544	179	133		RLWE+RLWR	F
Round5		445	549	88	138	128	GLWR	F
Sable	800	608	672	139	114		MLWR	T
SMAUG	176	672	672	120	120		MLWE+MLWR	T
Round5		634	682	65	128	128	GLWR	T
NTRU	699	935	699	$\infty$	106		NTRU	T
Saber	832	672	736	120	118		MLWR	T
Kyber	1632	800	768	139	118		MLWE	T
Tiger	528	480	768	128	130	128	RLWR+RLWE	F
RLizard	385	4096	2080	188	147		RLWE+RLWR	T

keys  $s$  and  $s'$  and try to compute shared  $k$ : we have that  $k_{P_2} = s'^T b \approx s'^T A s \approx b^T s = k_{P_1}$ . Due to the small randomness, the most significant bits of  $k_{P_1}$  and  $k_{P_2}$  are equal and can be used to compute common key  $k$ .

The next major step is to apply a variant of the so-called Fujisaki-Okamoto transform that turns the resulting PKE scheme into an IND-CCA2 secure KEM. Virtually all post-quantum secure KEM candidates, including Kyber, use this technique.

One noteworthy general development is the usage of sparse secret keys, i.e. secret keys that have mostly zeroes in all positions and only in a few ones 1 or -1 (or additionally 2 and -2). This achieves small secret keys when only the non-zero entries have to be stored. This is advantageous in scenarios where the secret key needs to be protected physically since physical protection is usually expensive. Algorithms like SMAUG [16] and TiGER [44], which are both submissions in the South Korean KPQC competition<sup>9</sup>, use this approach. Moreover, multiplication with such a secret key can be highly efficient even without relying on the number-theoretic transform as Kyber does. However, it seems that these schemes tend to be more susceptible to combinatorial attacks<sup>10</sup> like the one of May [39].

We can now provide a comparison of the recent KEM schemes that have enough core-SVP hardness in Table 1, Table 2, and Table 3. The data in the tables is taken verbatim from <https://eprint.iacr.org/archive/2023/739/1684818419.pdf>. The sizes are given in bytes and the decryption failure probability (DFP) is given in  $\log_2(\text{prob})$ ; " $\infty$ " implies that it uses a perfectly correct PKE scheme as a building block. We left a blank space for the schemes that have not specified the sizes of their secret key. Security (Sec) is given in the classic core-SVP hardness reported in the paper of each scheme. The shared key entropy ( $|K|$ ) is given in bits if they are less than 256 bits. The assumption is given as "A+B" (A for key generation, B for encapsulation) or "A", if A=B. The last column shows if the scheme uses an error correction code (ECC) to reduce the DFP or not.

## Digital Signatures

Dilithium is based on a variant of the popular Fiat-Shamir transform [25] that turns 3-move zero-knowledge protocols into non-interactive zero-knowledge proofs (NIZK). The transform takes as input a zero-knowledge protocol with messages  $(a, c, s)$  for proving knowledge of some secret key  $sk$  that is executed between a prover (who initiates the protocol) and a verifier. The second message  $c$  – from verifier to prover – consists of a random challenge. Via the Fiat-Shamir transform the prover can compute this challenge locally by applying a hash function to  $a$  and context information. This makes the protocol non-interactive. By additionally hashing a given

<sup>9</sup><https://kpqc.or.kr/competition.html>

<sup>10</sup>[https://groups.google.com/g/kpqc-bulletin/c/GejJ\\_lp3GLI/m/1XjMhypoFwAJ](https://groups.google.com/g/kpqc-bulletin/c/GejJ_lp3GLI/m/1XjMhypoFwAJ)



Table 2: Lattice-based KEMs with approximately NIST's security level III (180).

Scheme	sk	pk	ct	DFP	Sec	$ K $	Assumption	no ECC
Round5		780	859	117	193	192	GLWR	F
Round5		909	981	71	192	192	GLWR	T
Sable	1152	896	1024	143	185		MLWR	T
SMAUG	236	1088	1024	136	181		MLWE+MLWR	T
Tiger	1056	800	1024	154	161		RLWR+RLWE	F
Saber	1248	992	1088	136	189		MLWR	T
Kyber	2400	1184	1088	164	183		MLWE	T
NTRU	1230	1590	1230	$\infty$	178		NTRU	T
RLizard	641	4096	4144	245	195		RLWE+RLWR	T

Table 3: Lattice-based KEMs with approximately NIST's security level V (260).

Scheme	sk	pk	ct	DFP	Sec	$ K $	Assumption	no ECC
Round5		972	1063	64	256		GLWR	F
LizardMong	1280	1056	1088	302	226		RLWE+RLWR	F
Tiger	1056	928	1152	200	263		RLWR+RLWE	F
Round5		1178	1274	64	256		GLWR	T
Saber	1664	1312	1472	165	260		MLWR	T
SMAUG	218	1792	1472	167	264		MLWE+MLWR	T
Kyber	3168	1568	1568	174	256		MLWE	T
AKCN-SEC		2240	1792	70	255		RLWE	F
OKCN-SEC		2336	1792	70	255		RLWE	F
NewHope	3680	1824	2208	216	257		RLWE	T
RLizard	769	8192	8256	305	318		RLWE+RLWR	T

input message besides  $a$ , the protocol is turned into a signature of knowledge. This transformation has proven very fruitful in the past, for example resulting in the extremely efficient Schnorr signature scheme [50]. However, in the lattice setting the vanilla Fiat-Shamir transform is not enough and Dilithium thus relies on what is commonly called the Fiat-Shamir with aborts technique [37]. Roughly, for certain choices of the secret key,  $a$ , and  $c$  the prover cannot compute  $s$ . In this case, the prover has to abort. However, since these events depend on  $sk$  they might reveal information about the secret key. This information could be accumulated and used to break the signature scheme. To avoid that, the final scheme artificially aborts with a certain probability thus effectively hiding when the aborting behavior is dependent on  $sk$ .

### Hash-based Signatures and SPHINCS<sup>+</sup>

It is well-known that digital signatures can be built from hash functions alone via so-called hash trees [41]. The idea is to create  $2^\ell$  random public keys of a so-called few-time signature scheme (FTS) that only supports a few applications per public key. Next, a full binary tree is generated where parent nodes consist of the hash value obtained when hashing the concatenation of both children (in SPHINCS<sup>+</sup> this is called hypertree). The root of the tree is the public key. When signing, the signer first chooses a random leaf that corresponds to a specific FTS public key  $pk$ . It uses the corresponding secret key to compute a FTS-signature. This is the first part of the signature. The second part consists of the co-path in the hash tree that leads to the root. Using this co-path, the verifier can check whether the leaf is indeed part of the hash tree. Signatures constructed in this way have the advantage that they do not rely on specific number-theoretic assumptions but only on the security of a symmetric hash function. However, the disadvantage is that signatures have at least  $\ell$  components. SPHINCS<sup>+</sup> is a particularly efficient instantiation of this general template.

## | 1.7 Background on Lattices

We indicate with capital bold letters the matrices, with lowercase bold letters the vectors, and with lowercase letters the scalars.

**Definition 1.1.** Let  $B = [b_1, \dots, b_n]$  be a matrix of column vectors in a vector space  $\mathcal{V}$ . We call  $B$  a basis of  $\mathcal{V}$  if every element of  $\mathcal{V}$  may be written in a unique way as a finite linear combination of column vectors of  $B$ . We can indicate  $\mathcal{V} = \text{span}(B)$ .

**Definition 1.2.** Let  $B = [b_1, \dots, b_n] \in \mathbb{R}^{m \times n}$  be a matrix of linearly independent vectors in  $\mathbb{R}^m$ . The lattice generated by  $B$  is the set

$$\mathcal{L}(B) = \{Bx \mid x \in \mathbb{Z}^n\} = \left\{ \sum_{i=1}^n x_i b_i \mid x_i \in \mathbb{Z} \right\}$$

of all the integer linear combinations of the columns of  $B$ . The matrix  $B$  is called a basis for the lattice  $\mathcal{L}(B)$ .

**Remark.** Notice that if  $B$  is a basis for the lattice  $\mathcal{L}(B)$ , then it is also a basis for the vector space  $\text{span}(B)$ .

We can visualize lattices as a grid or a repeating pattern of points that extend infinitely in all directions.

**Definition 1.3.** A matrix  $U \in \mathbb{Z}^{n \times n}$  is unimodular if there exist a matrix  $V \in \mathbb{Z}^{n \times n}$  such that  $VU = UV = \text{Id}$ .

**Proposition 1.1.** The unimodular matrices satisfy the following properties.



## D2.1 Post-quantum Analysis Report

- If  $U$  is unimodular, then  $U^{-1}$  is unimodular.
- If  $U$  and  $V$  are unimodular, then  $UV$  is unimodular.
- $U \in \mathbb{Z}^{n \times n}$  is unimodular if and only if  $\det(U) = \pm 1$ .

Theorem 1.1. Let  $B$  and  $C$  be two bases for lattices. Then  $\mathcal{L}(B) = \mathcal{L}(C)$  if and only if there exists a unimodular matrix  $U$  such that  $B = CU$ .

### Gram-Schmidt Orthogonalization

Definition 1.4. We say that  $x$  and  $y$  are orthogonal if they are perpendicular each other i.e. if the scalar product of the two vectors is zero,  $x \cdot y = 0$ .

Definition 1.5. Let  $B$  a basis for the vector space  $\mathcal{V}$ . We say that  $B$  is an orthogonal basis if all the vectors in  $B$  are mutually orthogonal.

From any basis  $B$  can be found an orthogonal basis  $B^*$  for the same vector space using the Gram-Schmidt orthogonalization method.

Definition 1.6. For any  $B = [b_1, \dots, b_n]$ , define a matrix of mutually orthogonalized column vectors  $B^* = [b_1^*, \dots, b_n^*]$  iteratively via the formula

$$b_i^* = b_i - \sum_{j < i} \mu_{i,j} b_j^*$$

where  $\mu_{i,j} = \frac{b_i \cdot b_j^*}{b_j^* \cdot b_j^*}$ .

### The determinant

Definition 1.7. Let  $B = [b_1, \dots, b_n] \in \mathbb{R}^{m \times n}$  be a basis for a lattice  $\mathcal{L}(B)$ . The determinant  $\det(\mathcal{L}(B))$  of the lattice is defined as

$$\det(\mathcal{L}(B)) = \prod_i^n \|b_i^*\|$$

where  $B^* = [b_1^*, \dots, b_n^*]$  is the Gram-Schmidt orthogonalization of  $B$ .

Since  $\|b_i^*\| \leq \|b_i\|$ , we can find a simple upper bound to the determinant,  $\det(\mathcal{L}(B)) \leq \prod_i^n \|b_i\|$ .

Theorem 1.2. For any lattice basis  $B \in \mathbb{R}^{m \times n}$ , we have  $\det(\mathcal{L}(B)) = \sqrt{\det(B^T B)}$ .

Theorem 1.3. Suppose  $B, C$  are bases of the same lattice,  $\mathcal{L}(B) = \mathcal{L}(C)$ . Then,  $\det(\mathcal{L}(B)) = \det(\mathcal{L}(C))$ .

### Minimum Distance

Definition 1.8. For any lattice  $\mathcal{L}$ , the minimum distance of  $\mathcal{L}$  is the smallest distance between any two lattice points,

$$\lambda(\mathcal{L}) = \inf\{\|x - y\| \mid x, y \in \mathcal{L}, x \neq y\}.$$

We observe that the minimum distance can be equivalently defined as the length of the shortest nonzero lattice vector,  $\lambda(\mathcal{L}) = \inf\{\|v\| \mid v \in \mathcal{L} \setminus \{0\}\}$ .

## Average vs. Worst-case problems

The majority of lattice-based security assumptions used in cryptography typically can be related to classical assumptions in lattices via a worst-case to average-case reduction. This means that breaking the lattice-based assumption on average is as hard as breaking a classical assumption in the worst-case. The classical assumption is usually a variant of the approximate shortest vector problem, which we will discuss in the next session.

- Average-case problems: in order for an adversary to break these problems, it must only succeed in solving the problem for some random instance.
- Worst-case problems: in order for an adversary to break these problems, it must only succeed in solving the problem on all instances of the problem.

## | 1.8 Classical lattice problems are not reflected in state-of-the-art cryptosystems

The average-to-worst-case reduction is a major factor in the attractiveness of lattice-based assumptions. It guarantees that, ultimately, a lattice-based scheme that relies on an LWE-like assumption can be based on a classical lattice assumption with worst-case guarantees. However, it is important to note that this implication is not reflected in modern cryptosystems. In particular, the reduction to classical assumptions have a large security loss (that quantifies the efficiency of the reduction to break the LWE-like assumption relative to the efficiency of the attacker against the cryptosystem). This has far-reaching consequences. Instantiating the parameter sizes for Regev-like schemes in a theoretically-sound way when relying on the underlying classical assumption can increase the dimension  $n$  of the lattices by a factor of (at least) ten as compared to what is used in practice [28]. This means that for the currently used parameters, to the best of our knowledge, all lattice-based finalists of the NIST competition do not reduce to some underlying classical worst-case assumption in a theoretically-sound way. Put differently, when applying the existing security reductions to the parameter sizes used in practice, they would merely imply that a classical lattice problem can be solved that has very small parameters.

### Shortest Vector Problem (SVP)

**Definition 1.9.** Given a lattice basis  $B$ , the SVP asks to find a shortest nonzero lattice vector, i.e. a vector  $v \in \mathcal{L}(B)$  with  $\|v\| = \lambda(\mathcal{L}(B))$ .

**Definition 1.10.** Given a lattice basis  $B$ , for  $\gamma \geq 1$  the  $\text{SVP}_\gamma$  asks to find a nonzero lattice vector  $v \in \mathcal{L}(B) \setminus \{0\}$  of norm at most  $\|v\| = \gamma \lambda(\mathcal{L}(B))$ .

**Definition 1.11.** Given a lattice basis  $B$  and a positive real  $d$ , the  $\text{GapSVP}_\gamma$  asks to determine if  $\lambda(\mathcal{L}(B)) \leq d$  or  $\lambda(\mathcal{L}(B)) \geq \gamma d$ .

### Short Integers Solutions Problem (SIS)

**Definition 1.12.** Let  $n, m, q, \beta$  be positive integers. Given a matrix  $A \in \mathbb{Z}_q^{n \times m}$ , the  $\text{SIS}_{n,m,q,\beta}$  problem asks to find a nonzero vector  $x \in \mathbb{Z}^m$  such that:

- $\|x\| \leq \beta$ .
- $Ax = 0 \in \mathbb{Z}_q^n$ .

Generally,  $m, q, \beta$  are functions of  $n$ . We claim  $\beta \leq q$ ; otherwise,  $x = (q, 0, \dots, 0) \in \mathbb{Z}^m$  is a solution of the problem.



## D2.1 Post-quantum Analysis Report

Geometric view of SIS problem. We define the lattice in  $\mathbb{R}^m$ ,  $\mathcal{W}^\perp(A) = \{x \in \mathbb{Z}^m \mid Ax = 0 \pmod{q}\}$ . Then,  $Ax = 0 \iff x \in \mathcal{W}^\perp(A)$ . Given a matrix  $A \in \mathbb{Z}_q^{n \times m}$ , solving  $\text{SIS}_{n,m,q,\beta}$  problem equals to find a short vector on lattice  $\mathcal{W}^\perp(A)$ .

Hardness of SIS problem.

- Large  $\beta$  means longer vectors in  $\mathcal{W}^\perp(A)$  are allowed, which makes the problem easier.
- Large  $q$  makes  $\mathcal{W}^\perp(A)$  sparser, which makes the problem easier.
- Large  $m$  means more column in  $A$  to form a 0 vector, which makes the problem easier.
- The average-case hardness of SIS problem is admitted by the worst-case hardness of GapSVP problem.

From the last point, we can make the following assumption. There is no PPT algorithm that can solve the average-case SIS problem with non-negligible probability.

Theorem 1.4. Assume the hardness of  $\text{SIS}_{n,m,q,\beta}$  where  $m > n \log_t q$  and  $\beta = t\sqrt{m}$  for some  $t$  function of  $n$ , then  $\mathcal{H}_{n,m,q,t} = \{f_A : T^m \rightarrow \mathbb{Z}_q^n \mid A \in \mathbb{Z}_q^{n \times m}\}$  is a collision-resistant hash function family, where  $f_A(z) = Az$  and  $T = \{0, \dots, t-1\}$ .

### Learning With Errors Problem (LWE)

Definition 1.13. Let  $n, q$  be positive integers and let  $\chi$  be a probability distribution on  $\mathbb{Z}_q$ . Choosing a vector  $a \in \mathbb{Z}_q^n$  uniformly at random and choosing  $e \in \mathbb{Z}_q$  according to  $\chi$ , define  $A_{s,\chi}$  a probability distribution on  $\mathbb{Z}_q^n \times \mathbb{Z}_q$  outputting  $(a, b = a \cdot s + e)$ .

Definition 1.14. Given an arbitrary number of independent samples from  $A_{s,\chi}$ , the LWE problem asks to find  $s \in \mathbb{Z}_q^n$ .

Definition 1.15. Given uniform distributed  $s \in \mathbb{Z}_q^n$ , the decision LWE problem asks to distinguish between samples from  $A_{s,\chi}$  and samples from the uniform distribution on  $\mathbb{Z}_q^n \times \mathbb{Z}_q$ .

Definition 1.16. Let  $d$  be a power of two and let  $q$  be a prime integer. Define  $R_q = \mathbb{Z}_q[x]/(x^d + 1)$ . Given samples of the form  $(a, b = as + e) \in R_q \times R_q$ , where  $a \in R_q$  is chosen uniformly random and  $e$  is a term chosen independently from some error distribution over  $R_q$ . The Ring-LWE problem (RLWE) asks to find  $s \in R_q$ .

Definition 1.17. Let  $n$  be an integer dimension, let  $d$  be a power of two, and let  $q$  be a prime integer. Define  $R = \mathbb{Z}[x]/(x^d + 1)$  and define  $R_q = R/qR$ . Given samples of the form  $(a, b = a \cdot s + e) \in R_q^n \times R_q$ , where  $a \in R_q^n$  is chosen uniformly random and  $e$  is a term chose independently from some error distribution over  $R_q$ . The Module-LWE problem (MLWE) asks to find  $s \in R_q^n$ .

Geometric view of LWE problem. We can see the problem with matrices. Then, let  $m$  be an integer integer; given some uniform  $A \in \mathbb{Z}_q^{n \times m}$  and  $b = A^T s + e$ , the LWE problem asks to find  $s$ .

The collection of  $b' = A^T s$  forms a lattice  $\mathcal{W}(A) = \{A^T s \mid s \in \mathbb{Z}_q^n\}$ , with some perturbation from  $e$  to it,  $b = A^T s + e$  is still close to  $b'$ , then the LWE problem asks to find  $b'$ .

Hardness of LWE problem.

- Large  $q$  makes  $\mathcal{W}(A)$  sparser, which makes the problem easier.
- Let  $\alpha$  be the deviation of  $\chi$ . Large  $\alpha$  makes the perturbation larger, which makes the problem harder.

## | 1.9 Lattice Based Crypto and Regev System

Now we introduce the Regev cryptosystem that relies on LWE. It is quite inefficient, but when based on RLWE, the system can be very truly practical.

The cryptosystem is parameterized by the security parameter  $n$ , the number of equation  $m$ , the modulus  $q$ , which are integers, and a real positive noise parameter  $e$ .

- The private key is a uniformly  $s \in \mathbb{Z}_q^n$ .
- The public key consist of samples  $(a_i, b_i)$  from  $A_{s, \chi}$  for  $i = 1, \dots, m$ .
- For any bit of the message, choose a random set  $S$  uniformly among all subsets of  $\{1, \dots, m\}$ . The encryption is  $(\sum_{i \in S} a_i, \sum_{i \in S} b_i)$  if the bit is 0 and  $(\sum_{i \in S} a_i, \lfloor q/2 \rfloor + \sum_{i \in S} b_i)$  if the bit is 1.
- The decryption of a pair  $(a, b)$  is 0 if  $b - a \cdot s$  is closer to 0 than to  $\lfloor q/2 \rfloor$  modulo  $q$ , and 1 otherwise.

## | 1.10 Kyber

We rely on the following specification:

<https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf>. Major parts of the description are taken verbatim from this source.

### Key Encapsulation Mechanism

A Key Encapsulation Mechanism is a cryptographic technique used to securely exchange symmetric keys between two parties over an insecure communication channel.

**Definition 1.18 (KEM).** A key encapsulation mechanism  $KEM := (KeyGen, Encaps, Decaps)$  consist of three polynomial-time algorithms:

- $(pk, sk) \leftarrow KeyGen(1^\kappa)$ : The probabilistic key generation algorithm  $KeyGen$  takes as input the security parameter  $\kappa$  in unary and returns a public key  $pk \in \mathcal{PK}$  and a secret key  $sk \in \mathcal{SK}$ .
- $(ct, K) \leftarrow Encaps(pk)$ : The probabilistic encapsulation algorithm  $Encaps$  takes as input a public key  $pk$  and returns a ciphertext  $ct \in \mathcal{C}$  and a key  $K \in \mathcal{K}$ .
- $K = Decaps(sk, ct)$ : The deterministic decapsulation algorithm  $Decaps$  takes as input a secret key  $sk$  and a ciphertext  $ct$  and returns a key  $K \in \mathcal{K}$ .

**Definition 1.19 (Correctness KEM).** We say that KEM is  $\varrho$ -correct, if for any  $(pk, sk) \leftarrow KeyGen(1^\kappa)$  we have:

$$\Pr[K = K' \mid (ct, K) \leftarrow Encaps(pk), K' = Decaps(sk, ct)] \geq \varrho$$

### Notation and Parameters

We denote  $R = \mathbb{Z}[x]/(X^n + 1)$  and  $R_q = \mathbb{Z}_q[x]/(X^n + 1)$ . The values of  $q$  and  $n$  are fixed to  $q = 3329$  and  $n = 256$ .

**Modular reductions.** For an even (resp. odd) positive integer  $\alpha$ , we define  $r' = r \bmod^{\pm} \alpha$  to be the unique element  $r'$  in the range  $-\frac{\alpha}{2} < r' \leq \frac{\alpha}{2}$  (resp.  $-\frac{\alpha-1}{2} \leq r' \leq \frac{\alpha-1}{2}$ ) such that  $r' \equiv r \bmod \alpha$ .

**Rounding.** For an element  $x \in \mathbb{Q}$  we denote by  $\lceil x \rceil$  rounding of  $x$  to the closest integer with ties being rounded up.

## D2.1 Post-quantum Analysis Report

**Sizes of elements.** For an element  $w \in \mathbb{Z}_q$ , we write  $\|w\|_\infty$  to mean  $|w \bmod \pm q|$ . For  $w = w_0 + w_1X + \dots + w_{n-1}X^{n-1} \in R$  we define

$$\|w\|_\infty = \max_i \|w_i\|_\infty,$$

$$\|w\| = \sqrt{\|w_0\|_\infty^2 + \dots + \|w_{n-1}\|_\infty^2}$$

and similarly for  $\mathbf{w} = (w_1, \dots, w_k) \in R^k$  we define

$$\|\mathbf{w}\|_\infty = \max_i \|w_i\|_\infty,$$

$$\|\mathbf{w}\| = \sqrt{\|w_1\|_\infty^2 + \dots + \|w_k\|_\infty^2}.$$

We write  $S_\eta$  to denote all elements  $w \in R$  such that  $\|w\|_\infty \leq \eta$ . We write  $\tilde{S}_\eta$  to denote the set  $\{w \bmod \pm 2\eta \mid w \in R\}$ .

**Compression and Decompression.** We now define a function  $\text{Compress}_q(x, d)$  that takes an element  $x \in \mathbb{Z}_q$  and outputs an integer in  $\{0, \dots, 2^{d-1}\}$  where  $d < \lceil \log_2(q) \rceil$ . We furthermore define a function  $\text{Decompress}_q$  such that  $x' = \text{Decompress}_q(\text{Compress}_q(x, d), d)$  is an element close to  $x$ , more specifically  $|x' - x \bmod \pm q| \leq B_q := \lceil \frac{q}{2^{d+1}} \rceil$ . The function satisfying these requirements are defined as

$$\text{Compress}_q(x, d) = \lceil (2^d/q) \cdot x \rceil \bmod 2^d,$$

$$\text{Decompress}_q(x, d) = \lceil (q/2^d) \cdot x \rceil.$$

The procedure is applied to each coefficient individually for  $x \in R_q$  or  $\mathbf{x} \in R_q^k$ .

**Symmetric primitives.** The design of Kyber makes use of a pseudo-random function PRF :  $\mathcal{B}^{32} \times \mathcal{B} \rightarrow \mathcal{B}^*$  and of an extendable output function XOF :  $\mathcal{B}^* \times \mathcal{B} \times \mathcal{B} \rightarrow \mathcal{B}^*$ . Kyber also makes use of two hash functions  $H : \mathcal{B}^* \rightarrow \mathcal{B}^{32}$  and  $G : \mathcal{B} \rightarrow \mathcal{B}^{32} \times \mathcal{B}^{32}$  and a key-derivation function KDF :  $\mathcal{B}^* \rightarrow \mathcal{B}^*$ .

$\mathcal{B}$  denote the set  $\{0, \dots, 255\}$ , i.e. the set of 8-bit unsigned integers (bytes). Consequently, we denote by  $\mathcal{B}^k$  the set of byte arrays of length  $k$  and by  $\mathcal{B}^*$  the set of byte arrays of arbitrary length.

## Number-Theoretic Transform (NTT)

Multiplication in  $R_q$  based on the number-theoretic transform (NTT) has multiple advantages: it is extremely fast, does not require additional memory, and can be done in very little code space. It needs  $q$  to be of the form  $q - 1 = 2^k \cdot p$  where  $q$  and  $p$  are primes and  $k \in \mathbb{Z}$ . Since Kyber uses the prime  $q = 3329$ , with  $q - 1 = 2^8 \cdot 13$ , NTT can be used.

The polynomial  $X^{256} + 1$  can be factorized into 128 polynomials of degree 2 modulo  $q$  as

$$X^{256} + 1 = \prod_{i=0}^{127} (X^2 - \zeta^{2i+1})$$

where  $\zeta = 17$  is the first primitive 256-th root of unity modulo  $q$ . Then the NTT of  $f \in R_q$  is given by

$$(f \bmod X^2 - \zeta^{2 \cdot 0 + 1}, \dots, f \bmod X^2 - \zeta^{2 \cdot 127 + 1}).$$

We define  $\text{NTT} : R_q \rightarrow R_q$  to be the bijection that maps  $f \in R_q$  to the polynomial with the aforementioned coefficient vector, then

$$\text{NTT}(f) = \hat{f} = \hat{f}_0 + \hat{f}_1X + \dots + \hat{f}_{255}X^{255}$$

with

$$\hat{f}_{2i} = \sum_{j=0}^{127} f_{2j} \zeta^{(2i+1)j},$$

$$\hat{f}_{2i+1} = \sum_{j=0}^{127} f_{2j+1} \zeta^{(2i+1)j}.$$

The natural algebraic representation of  $\text{NNT}(f)$  is as 128 polynomials of degree 1, that is

$$\text{NNT}(f) = \hat{f} = (\hat{f}_0 + \hat{f}_1 X, \dots, \hat{f}_{254} + \hat{f}_{255} X).$$

For  $f, g \in R_q$ , we can compute the product  $f \cdot g$  using NNT and its inverse  $\text{NNT}^{-1}$ ; the product can be computed as  $\text{NNT}^{-1}(\text{NNT}(f) \circ \text{NNT}(g))$  where  $\text{NNT}(f) \circ \text{NNT}(g) = \hat{f} \circ \hat{g} = \hat{h}$  denotes the basecase multiplication consisting of the 128 products

$$\hat{h}_{2i} + \hat{h}_{2i+1} X = (\hat{f}_{2i} + \hat{f}_{2i+1} X)(\hat{g}_{2i} + \hat{g}_{2i+1} X) \mod X^2 - \zeta^{2i+1}$$

of linear polynomials.

## Algorithms

Uniform sampling in  $R_q$ . Kyber uses a deterministic approach to sample elements in  $R_q$  that are statistically close to a uniformly random distribution.

---

Input: Byte stream  $B = b_0, b_1, \dots \in \mathcal{B}^*$   
Output: NTT-representation  $\hat{a} \in R_q$  of  $a \in R_q$

```

i := 0
j := 0
while j < n do
  d1 := b_i + 256 · (b_{i+1} mod +16)
  d2 := ⌊b_{i+1/16}⌋ + 16 · b_{i+2}
  if d1 < q then
    â_j := d1
    j := j + 1
  end if
  if d2 < q and j < n then
    â_j := d2
    j := j + 1
  end if
  i := i + 3
end while
return â_0 + â_1 X + ⋯ + â_{n-1} X^{n-1}

```

---

Figure 1: Implementation of the Parse:  $\mathcal{B}^* \rightarrow R_q^n$  function.

Sampling from a binomial distribution. The centered binomial distribution  $B_\eta$  sample  $(a_1, \dots, a_\eta, b_1, \dots, b_\eta) \leftarrow \{0, 1\}^{2\eta}$  and output  $\sum_{i=1}^\eta (a_i - b_i)$ . If a polynomial  $f \in R_q$  or a vector of such polynomials is sampled from  $B_\eta$ , it means that each coefficient is sampled from  $B_\eta$ .

For the specification of Kyber we need to define how a polynomial  $f \in R_q$  is sampled according to  $B_\eta$  deterministically from  $64\eta$ . This is done by the function CBD defined as described in the following algorithm.





## D2.1 Post-quantum Analysis Report

---

Input: Byte array  $B = (b_0, b_1, \dots, b_{64\eta-1}) \in \mathcal{B}^{64\eta}$   
Output: Polynomial  $f \in R_q$   
 $(\beta_0, \dots, \beta_{512\eta-1}) := \text{BytesToBits}(B)$   
for  $i$  from 0 to 255 do  
     $a := \sum_{j=0}^{\eta-1} \beta_{2i\eta+j}$   
     $b := \sum_{j=0}^{\eta-1} \beta_{2i\eta+\eta+j}$   
     $f_i := a - b$   
end for  
return  $f_0 + f_1X + f_2X^2 + \dots + f_{255}X^{255}$

---

Figure 2: Implementation of the  $\text{CBD}_\eta: \mathcal{B}^{64\eta} \rightarrow R_q$  function.

Encoding and decoding. In the following algorithm, we give a pseudocode description of the function  $\text{Decode}_\ell$ , which deserializes an array of  $32\ell$  bytes into a polynomial  $f = f_0 + f_1X + \dots + f_{255}X^{255}$  with each coefficient  $f_i \in \{0, \dots, 2^\ell - 1\}$ . We define the function  $\text{Encode}_\ell$  as the inverse of  $\text{Decode}_\ell$ . Whenever we apply  $\text{Encode}_\ell$  to a vector of polynomials we encode each polynomial individually and concatenate the output byte arrays.

---

Input: Byte array  $B \in \mathcal{B}^{32\ell}$   
Output: Polynomial  $f \in R_q$   
 $(\beta_0, \dots, \beta_{256\ell-1}) := \text{BytesToBits}(B)$   
for  $i$  from 0 to 255 do  
     $f_i := \sum_{j=0}^{\ell-1} \beta_{i\ell+j}2^j$   
end for  
return  $f_0 + f_1X + f_2X^2 + \dots + f_{255}X^{255}$

---

Figure 3: Implementation of the  $\text{Decode}_\ell: \mathcal{B}^{32\ell} \rightarrow R_q$  function.

## Kyber.CPAPKE

---

Output: Secret key  $sk \in \mathcal{B}^{12 \cdot k \cdot n / 8}$   
Output: Public key  $pk \in \mathcal{B}^{12 \cdot k \cdot n / 8 + 32}$

```

1:  $d \leftarrow \mathcal{B}^{32}$ 
2:  $(\varrho, \sigma) := G(d)$ 
3:  $N := 0$ 
4: for  $i$  from 0 to  $k - 1$  do
5:   for  $j$  from 0 to  $k - 1$  do
6:      $\hat{A}[i][j] := \text{Parse}(\text{XOF}(\varrho, j, i))$ 
7:   end for
8: end for
9: for  $i$  from 0 to  $k - 1$  do
10:   $s[i] := \text{CBD}_{\eta_1}(\text{PRF}(\sigma, N))$ 
11:   $N := N + 1$ 
12: end for
13: for  $i$  from 0 to  $k - 1$  do
14:   $e[i] := \text{CBD}_{\eta_1}(\text{PRF}(\sigma, N))$ 
15:   $N := N + 1$ 
16: end for
17:  $\hat{s} := \text{NTT}(s)$ 
18:  $\hat{e} := \text{NTT}(e)$ 
19:  $\hat{t} := \hat{A} \circ \hat{s} + \hat{e}$ 
20:  $pk := (\text{Encode}_{12}(\hat{t} \bmod {}^+q) \parallel \varrho)$ 
21:  $sk := \text{Encode}_{12}(\hat{s} \bmod {}^+q)$ 
22: return  $(pk, sk)$ 
```

---

Figure 4: Key generation algorithm Kyber.CPAPKE.KeyGen.

## D2.1 Post-quantum Analysis Report

---

Input: Public key  $pk \in \mathcal{B}^{12 \cdot k \cdot n/8 + 32}$   
 Input: Message  $m \in \mathcal{B}^{32}$   
 Input: Random coins  $r \in \mathcal{B}^{32}$   
 Output: Ciphertext  $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$

```

1:  $N := 0$ 
2:  $\hat{t} := \text{Decode}_{12}(pk)$ 
3:  $\varrho := pk + 12 \cdot k \cdot n/8$ 
4: for  $i$  from 0 to  $k - 1$  do
5:   for  $j$  from 0 to  $k - 1$  do
6:      $\hat{A}^T[i][j] := \text{Parse}(\text{XOF}(\varrho, i, j))$ 
7:   end for
8: end for
9: for  $i$  from 0 to  $k - 1$  do
10:   $r[i] := \text{CBD}_{\eta_1}(\text{PRF}(r, N))$ 
11:   $N := N + 1$ 
12: end for
13: for  $i$  from 0 to  $k - 1$  do
14:   $e_1[i] := \text{CBD}_{\eta_2}(\text{PRF}(r, N))$ 
15:   $N := N + 1$ 
16: end for
17:  $e_2 := \text{CBD}_{\eta_2}(\text{PRF}(r, N))$ 
18:  $\hat{r} := \text{NTT}(\mathbf{r})$ 
19:  $\mathbf{u} := \text{NTT}^{-1}(\hat{A}^T \circ \hat{r}) + \mathbf{e}_1$ 
20:  $\mathbf{v} := \text{NTT}^{-1}(\hat{t}^T \circ \hat{r}) + e_2 + \text{Decompress}_q(\text{Decode}_1(m), 1)$ 
21:  $c_1 := \text{Encode}_{d_u}(\text{Compress}_q(\mathbf{u}, d_u))$ 
22:  $c_2 := \text{Encode}_{d_v}(\text{Compress}_q(\mathbf{v}, d_v))$ 
23: return  $c = (c_1 || c_2)$ 

```

---

Figure 5: Encryption algorithm Kyber.CPAPKE.Enc.

---

Input: Secret key  $sk \in \mathcal{B}^{12 \cdot k \cdot n/8}$   
 Input: Ciphertext  $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$   
 Output: Message  $m \in \mathcal{B}^{32}$

```

1:  $\mathbf{u} := \text{Decompress}_q(\text{Decode}_{d_u}(c), d_u)$ 
2:  $\mathbf{v} := \text{Decompress}_q(\text{Decode}_{d_v}(c + d_u \cdot k \cdot n/8), d_v)$ 
3:  $\hat{s} := \text{Decode}_{12}(sk)$ 
4:  $m := \text{Encode}_1(\text{Compress}_q(\mathbf{v} - \text{NTT}^{-1}(\hat{s}^T \circ \text{NTT}(\mathbf{u})), 1))$ 
5: return  $m$ 

```

---

Figure 6: Decryption algorithm Kyber.CPAPKE.Dec.

## Kyber.CCAKEM

---

Output: Public key  $sk \in \mathcal{B}^{12 \cdot k \cdot n/8 + 32}$   
Output: Secret key  $pk \in \mathcal{B}^{24 \cdot k \cdot n/8 + 96}$   
1:  $z \leftarrow \mathcal{B}^{32}$   
2:  $(pk, sk') := \text{Kyber.CPAPKE.KeyGen}()$   
3:  $sk := (sk' || pk || H(pk) || z)$   
4: **return**  $(pk, sk)$

---

Figure 7: Key generation algorithm Kyber.CCAKEM.KeyGen.

---

Input: Public key  $pk \in \mathcal{B}^{12 \cdot k \cdot n/8 + 32}$   
Output: Ciphertext  $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$   
Output: Shared key  $K \in \mathcal{B}^*$   
1:  $m \leftarrow \mathcal{B}^{32}$   
2:  $\bar{m} \leftarrow H(m)$   
3:  $(\bar{K}, r) := G(m || H(pk))$   
4:  $c := \text{Kyber.CPAPKE.Enc}(pk, m, r)$   
5:  $K := \text{KDF}(\bar{K} || H(c))$   
6: **return**  $(c, K)$

---

Figure 8: Encryption algorithm Kyber.CCAKEM.Enc.

---

Input: Ciphertext  $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$   
Input: Secret key  $sk \in \mathcal{B}^{24 \cdot k \cdot n/8 + 96}$   
Output: Shared key  $K \in \mathcal{B}^*$   
1:  $pk := sk + 12 \cdot k \cdot n/8$   
2:  $h := sk + 24 \cdot k \cdot n/8 + 32 \in \mathcal{B}^{32}$   
3:  $z := sk + 24 \cdot k \cdot n/8 + 64$   
4:  $m' := \text{Kyber.CPAPKE.Dec}(sk, c)$   
5:  $(\bar{K}', r') := G(m' || h)$   
6:  $c' := \text{Kyber.CPAPKE.Enc}(pk, m', r')$   
7: **if**  $c = c'$  **then**  
8:     **return**  $K := \text{KDF}(\bar{K}' || H(c))$   
9: **else**  
10:    **return**  $K := \text{KDF}(z || H(c))$   
11: **end if**  
12: **return**  $K$

---

Figure 9: Decryption algorithm Kyber.CCAKEM.Dec.

## Instantiated Primitives

Kyber relies on a set of instantiated primitives to achieve its security and functionality. These carefully selected cryptographic building blocks, as demonstrated in the preceding algorithms, play crucial roles in various aspects of the Kyber protocol. Below are the key instantiated primitives utilized in Kyber:



## D2.1 Post-quantum Analysis Report

- XOF is instantiated with SHAKE-128.
- $H$  is instantiated with SHA3-256.
- $G$  is instantiated with SHA3-512.
- $\text{PRF}(s, b)$  is instantiated with  $\text{SHAKE-256}(s\|b)$ .
- KDF is instantiated with SHAKE-256.

### Security Assumption

The problem underlying the security of the schemes is MLWE. We apply the Definition 1.17 with the correct specification. It consist in distinguish uniform samples  $(a_i, b_i) \leftarrow R_q^k \times R_q$  from samples  $(a_i, b_i) \in R_q^k \times R_q$  where  $a_i \leftarrow R_q^k$  is uniform and  $b_i = a_i^t s + e_i$  with  $s \leftarrow B_\eta^k$  common to all samples and  $e_i \leftarrow B_\eta$  fresh for every sample.

For an adversary  $A$ , we define  $\text{Adv}_{m,k,\eta}^{\text{MLWE}}(A)$  as

$$\left| \Pr \left[ b' = 1 \mid \begin{array}{l} A \leftarrow R_q^{m \times k}; (s, e) \leftarrow B_\eta^k \times B_\eta^m; \\ b = As + e; b' \leftarrow A(A, b) \end{array} \right] - \Pr[b' = 1 \mid A \leftarrow R_q^{m \times k}; b \leftarrow R_q^m; b' \leftarrow A(A, b)] \right|.$$

Theorem 1.5. Suppose XOF and  $G$  are random oracles. For any adversary  $A$ , there exist adversaries  $B$  and  $C$  with roughly the same running time as that of  $A$  such that

$$\text{Adv}_{\text{Kyber.CPAPKE}}^{\text{CPA}}(A) \leq 2\text{Adv}_{k+1,k,\eta}^{\text{MLWE}}(B) + \text{Adv}_{\text{PRF}}^{\text{PRF}}.$$

Theorem 1.6. Suppose XOF,  $H$  and  $G$  are random oracles. For any classic adversary  $A$  that makes at most  $q_{RO}$  many queries to random oracles XOF,  $H$  and  $G$ , there exist adversaries  $B$  and  $C$  of roughly the same running time as that of  $A$  such that

$$\text{Adv}_{\text{Kyber.CCAKEM}}^{\text{CCA}}(A) \leq 2\text{Adv}_{k+1,k,\eta}^{\text{MLWE}}(B) + \text{Adv}_{\text{PRF}}^{\text{PRF}}(C) + 4q_{RO}\delta.$$

Note that the security bound is tight. The negligible additive term  $4q_{RO}\delta$  stems from Kyber.CPAPKE's decryption-failure probability  $\delta$ .

Theorem 1.7. Suppose XOF,  $H$  and  $G$  are random oracles. For any quantum adversary  $A$  that makes at most  $q_{RO}$  many queries to quantum random oracles XOF,  $H$  and  $G$ , there exist quantum adversaries  $B$  and  $C$  of roughly the same running time as that of  $A$  such that

$$\text{Adv}_{\text{Kyber.CCAKEM}}^{\text{CCA}}(A) \leq 4q_{RO} \cdot \sqrt{\text{Adv}_{k+1,k,\eta}^{\text{MLWE}}} + \text{Adv}_{\text{PRF}}^{\text{PRF}}(C) + 8q_{RO}^2\delta.$$

Unfortunately, the above security bound is non-tight and therefore can only serve as an asymptotic indication of Kyber.CCAKEM's CCA-security in the quantum random oracle model.

## | 1.11 Digital Signatures

Digital Signatures and global and private PKI. There is a solution on the horizon, but not short-term.

Definition 1.20. A signature scheme  $(\text{KeyGen}, \text{Sig}, \text{Ver})$  consist of three polynomial-time algorithms

- $(\text{sigk}, \text{verk}) \leftarrow \text{KeyGen}(1^\kappa)$ : The key generation algorithm  $\text{KeyGen}$  takes as input the security parameter  $\kappa$  in unary and returns a signature key  $\text{sigk} \in \mathcal{S}$  and a verification key  $\text{verk} \in \mathcal{V}$ .
- $s \leftarrow \text{Sig}(\text{sigk}, m)$ : The signing algorithm  $\text{Sig}$  takes as input a signature key  $\text{sigk}$  and a message  $m$  and returns a signed message  $s \in \mathcal{SM}$ .



## D2.1 Post-quantum Analysis Report

- $b = \text{Ver}(\text{verk}, m, s)$ : The verification algorithm  $\text{Ver}$  takes as input a verification key  $\text{verk}$ , a message  $m$  and a signed message  $s$  and returns  $b \in \{0, 1\}$ .

Definition 1.21 (Correctness). We say that a signature scheme is  $\varrho$ -correct, if for any  $(\text{sigk}, \text{verk}) \leftarrow \text{KeyGen}(1^\kappa)$  we have:

$$\Pr[\text{Ver}(\text{verk}, m, \text{Sig}(\text{sigk}, m)) = 1] \geq \varrho$$

## | 1.12 Dilithium

We rely on the following specification: <https://pq-crystals.org/dilithium/data/dilithium-specification-r.pdf>. Major parts of the description are taken verbatim from this source.

### Notation and Parameters

We denote  $R = \mathbb{Z}[x]/(X^n + 1)$  and  $R_q = \mathbb{Z}_q[x]/(X^n + 1)$ . The values of  $q$  and  $n$  are fixed to  $q = 8380417 = 2^{23} - 2^{13} + 1$  and  $n = 256$ . Due to this choice of  $q$ , it is possible to use NTT.

Modular reductions. For an even (resp. odd) positive integer  $\alpha$ , we define  $r' = r \bmod^{\pm\alpha}$  to be the unique element  $r'$  in the range  $-\frac{\alpha}{2} < r' \leq \frac{\alpha}{2}$  (resp.  $-\frac{\alpha-1}{2} \leq r' \leq \frac{\alpha-1}{2}$ ) such that  $r' \equiv r \bmod \alpha$ .

Sizes of elements. For an element  $w \in \mathbb{Z}_q$ , we write  $\|w\|_\infty$  to mean  $|w \bmod^{\pm q}|$ . For  $w = w_0 + w_1X + \dots + w_{n-1}X^{n-1} \in R$  we define

$$\|w\|_\infty = \max_i \|w_i\|_\infty,$$

$$\|w\| = \sqrt{\|w_0\|_\infty^2 + \dots + \|w_{n-1}\|_\infty^2}$$

and similarly for  $\mathbf{w} = (w_1, \dots, w_k) \in R^k$  we define

$$\|\mathbf{w}\|_\infty = \max_i \|w_i\|_\infty,$$

$$\|\mathbf{w}\| = \sqrt{\|w_1\|_\infty^2 + \dots + \|w_k\|_\infty^2}.$$

We write  $S_\eta$  to denote all elements  $w \in R$  such that  $\|w\|_\infty \leq \eta$ . We write  $\tilde{S}_\eta$  to denote the set  $\{w \bmod^{\pm 2\eta} \mid w \in R\}$ .

### Hashing

Hashing to a Ball. Let  $B_\tau$  denote the set of elements of  $R$  that have  $\tau$  coefficients that are either  $-1$  or  $1$  and the rest are  $0$ . We have  $|B_\tau| = 2^\tau \cdot \binom{256}{\tau}$ .

```

SampleInBall( $\varrho$ )
00 Initialize  $\mathbf{c} = c_0c_1 \dots c_{255} = 00 \dots 0$ 
01 for  $i := 256 - \tau$  to 255
02    $j \leftarrow \{0, 1, \dots, i\}$ 
03    $s \leftarrow \{0, 1\}$ 
04    $c_i := c_j$ 
05    $c_j := (-1)^s$ 
06 return  $\mathbf{c}$ 

```

Figure 10:  $\text{SampleInBall}(\varrho)$  generates a controlled pseudorandom binary sequence of length 256.

Expanding the Matrix A. The function `ExpandA` maps a uniform seed  $\varrho \in \{0,1\}^{256}$  to a matrix  $A \in R_q^{k \times l}$  in NTT domain representation.

Sampling the vectors  $s_1$  and  $s_2$ . The function `ExpandS`, used for generating the secret vectors in key generation, maps a seed  $\varrho'$  to  $(s_1, s_2) \in S_\eta^l \times S_\eta^k$ .

Sampling the vectors  $y$ . The function `ExpandMask`, used for deterministically generating the randomness of the signature scheme, maps a seed  $\varrho'$  and a nonce  $\kappa$  to  $y \in \tilde{S}_{\gamma_1}^l$ .

Hashing. The function  $H$  used in the signature scheme is the extended output function (XOF) SHAKE-256 mapping onto the specified domain.

## Algorithms

We present some useful algorithms and some of their properties.

<b>Power2Round<sub>q</sub>(r, d)</b> 00 $r := r \bmod^+ q$ 01 $r_0 := r \bmod^{\pm} 2^d$ 02 return $((r - r_0)/2^d, r_0)$	<b>Decompose<sub>q</sub>(r, α)</b> 11 $r := r \bmod^+ q$ 12 $r_0 := r \bmod^{\pm} \alpha$ 13 if $r - r_0 = q - 1$ 14   then $r_1 := 0; r_0 := r_0 - 1$ 15 else $r_1 := (r - r_0)/\alpha$ 16 return $(r_1, r_0)$
<b>MakeHint<sub>q</sub>(z, r, α)</b> 03 $r_1 := \text{HighBits}_q(r, \alpha)$ 04 $v_1 := \text{HighBits}_q(r + z, \alpha)$ 05 return $\llbracket r_1 \neq v_1 \rrbracket$	<b>HighBits<sub>q</sub>(r, α)</b> 17 $(r_1, r_0) := \text{Decompose}_q(r, \alpha)$ 18 return $r_1$
<b>UseHint<sub>q</sub>(h, r, α)</b> 06 $m := (q - 1)/\alpha$ 07 $(r_1, r_0) := \text{Decompose}_q(r, \alpha)$ 08 if $h = 1$ and $r_0 > 0$ return $(r_1 + 1) \bmod^+ m$ 09 if $h = 1$ and $r_0 \leq 0$ return $(r_1 - 1) \bmod^+ m$ 10 return $r_1$	<b>LowBits<sub>q</sub>(r, α)</b> 19 $(r_1, r_0) := \text{Decompose}_q(r, \alpha)$ 20 return $r_0$

Figure 11: Algorithms for modular arithmetic `Power2Roundq`, hint generation `MakeHintq`, and utilization `UseHintq`. Supplementary functions `Decomposeq`, `HighBitsq`, and `LowBitsq`.

Lemma 1.1. Suppose that  $q$  and  $\alpha$  are positive integers satisfying  $q > 2\alpha$ ,  $q \equiv 1 \pmod{\alpha}$  and  $\alpha$  even. Let  $r$  and  $z$  be vectors of elements in  $R_q$  where  $\|z\|_\infty \leq \alpha/2$ , and let  $h, h'$  be vectors of bits. Then the `HighBitsq`, `MakeHintq`, and `UseHintq` algorithms satisfy the following properties:

- $\text{UseHint}_q(\text{MakeHint}_q(z, r, \alpha), r, \alpha) = \text{HighBits}_q(r + z, \alpha)$ .
- Let  $v_1 = \text{UseHint}_q(h, r, \alpha)$ . Then  $\|r - v_1 \cdot \alpha\|_\infty \leq \alpha + 1$ . Furthermore, if the number of 1's in  $h$  is  $\omega$ , then all except at most  $\omega$  coefficients of  $r - v_1 \cdot \alpha$  will have magnitude at most  $\alpha/2$  after centered reduction modulo  $q$ .
- For any  $h, h'$ , if  $\text{UseHint}_q(h, r, \alpha) = \text{UseHint}_q(h', r, \alpha)$ , then  $h = h'$ .

Lemma 1.2. If  $\|s\|_\infty \leq \beta$  and  $\|\text{LowBits}_q(r, \alpha)\|_\infty < \alpha/2 - \beta$ , then  $\text{HighBits}_q(r, \alpha) = \text{HighBits}_q(r + s, \alpha)$ .

Lemma 1.3. Let  $(r_1, r_0) = \text{Decompose}_q(r, \alpha)$ ,  $(w_1, w_0) = \text{Decompose}_q(r + s, \alpha)$ , and  $\|s\|_\infty \leq \beta$ . Then  $\|s + r_0\|_\infty < \alpha/2 - \beta \iff w_1 = r_1 \wedge \|w_0\|_\infty < \alpha/2 - \beta$ .

## Signature

```

Gen
00  $\zeta \leftarrow \{0, 1\}^{256}$ 
01  $(\varrho, \varrho', K) \in \{0, 1\}^{256} \times \{0, 1\}^{512} \times \{0, 1\}^{256} := H(\zeta)$  //H is instantiated as SHAKE-256
02  $A \in R_q^{k \times \ell} := \text{ExpandA}(\varrho)$  //A is generated and stored in NTT Representation as  $\hat{A}$ 
03  $(s_1, s_2) \in S_\eta^\ell \times S_\eta^k := \text{ExpandS}(\varrho')$ 
04  $t := As_1 + s_2$  //Compute  $As_1$  as  $\text{NTT}^{-1}(\hat{A} \cdot \text{NTT}(s_1))$ 
05  $(t_1, t_0) := \text{Power2Round}_q(t, d)$ 
06  $tr \in \{0, 1\}^{256} := H(\varrho \| t_1)$ 
07 return  $(pk = (\varrho, t_1), sk = (\varrho, K, tr, s_1, s_2, t_0))$ 

Sign(sk, M)
08  $A \in R_q^{k \times \ell} := \text{ExpandA}(\varrho)$  //A is generated and stored in NTT Representation as  $\hat{A}$ 
09  $\mu \in \{0, 1\}^{512} := H(tr \| M)$ 
10  $\kappa := 0, (z, h) := \perp$ 
11  $\varrho' \in \{0, 1\}^{512} := H(K \| \mu)$  (or  $\varrho' \leftarrow \{0, 1\}^{512}$  for randomized signing)
12 while  $(z, h) = \perp$  do //Pre-compute  $\hat{s}_1 := \text{NTT}(s_1)$ ,  $\hat{s}_2 := \text{NTT}(s_2)$ , and  $\hat{t}_0 := \text{NTT}(t_0)$ 
13    $y \in S_{\gamma_1}^\ell := \text{ExpandMask}(\varrho', \kappa)$ 
14    $w := Ay$  //w :=  $\text{NTT}^{-1}(\hat{A} \cdot \text{NTT}(y))$ 
15    $w_1 := \text{HighBits}_q(w, 2\gamma_2)$ 
16    $\tilde{c} \in \{0, 1\}^{256} := H(\mu \| w_1)$ 
17    $c \in B_r := \text{SampleInBall}(\tilde{c})$  //Store c in NTT representation as  $\hat{c} = \text{NTT}(c)$ 
18    $z := y + c s_1$  //Compute  $c s_1$  as  $\text{NTT}^{-1}(\hat{c} \cdot \hat{s}_1)$ 
19    $r_0 := \text{LowBits}_q(w - c s_2, 2\gamma_2)$  //Compute  $c s_2$  as  $\text{NTT}^{-1}(\hat{c} \cdot \hat{s}_2)$ 
20   if  $\|z\|_\infty \geq \gamma_1 - \beta$  or  $\|r_0\|_\infty \geq \gamma_2 - \beta$ , then  $(z, h) := \perp$ 
21   else
22      $h := \text{MakeHint}_q(-c t_0, w - c s_2 + c t_0, 2\gamma_2)$  //Compute  $c t_0$  as  $\text{NTT}^{-1}(\hat{c} \cdot \hat{t}_0)$ 
23     if  $\|c t_0\|_\infty \geq \gamma_2$  or the # of 1's in h is greater than  $\omega$ , then  $(z, h) := \perp$ 
24      $\kappa := \kappa + \ell$ 
25 return  $\sigma = (\tilde{c}, z, h)$ 

Verify(pk, M,  $\sigma = (\tilde{c}, z, h)$ )
26  $A \in R_q^{k \times \ell} := \text{ExpandA}(\varrho)$  //A is generated and stored in NTT Representation as  $\hat{A}$ 
27  $\mu \in \{0, 1\}^{512} := H(H(\varrho \| t_1) \| M)$ 
28  $c := \text{SampleInBall}(\tilde{c})$ 
29  $w'_1 := \text{UseHint}_q(h, Az - c t_1 \cdot 2^d, 2\gamma_2)$  //Compute as  $\text{NTT}^{-1}(\hat{A} \cdot \text{NTT}(z) - \text{NTT}(c) \cdot \text{NTT}(t_1 \cdot 2^d))$ 
30 return  $[\|z\|_\infty < \gamma_1 - \beta]$  and  $[\tilde{c} = H(\mu \| w'_1)]$  and  $[\text{\# of 1's in h is } \leq \omega]$ 

```

Figure 12: Key pair generation Gen, signature creation Sign, and verification Verify algorithms.

## Security Assumption

**MLWE Problem.** For integers  $m, k$  and a probability distribution  $D : R_q \rightarrow [0, 1]$ , we say that the advantage of adversary  $A$  in solving the decisional  $\text{MLWE}_{m,k,D}$  problem over the ring  $R_q$  is

$$\text{Adv}_{m,k,D}^{\text{MLWE}} := |\Pr[b = 1 | A \leftarrow R_q^{m \times k}; t \leftarrow R_q^m; b \leftarrow A(A, t)] - \Pr[b = 1 | A \leftarrow R_q^{m \times k}; s_1 \leftarrow D^k; s_2 \leftarrow D^m; b \leftarrow A(A, As_1 + s_2)]|.$$

**MSIS Problem.** To an algorithm  $A$  we associate the advantage function  $\text{Adv}_{m,k,\gamma}^{\text{MSIS}}$  to solve the  $\text{MSIS}_{m,k,\gamma}$  problem over the ring  $R_q$  as

$$\text{Adv}_{m,k,\gamma}^{\text{MSIS}}(A) := \Pr[0 < \|y\|_\infty \leq \gamma \wedge [I|A] \cdot y = 0 | A \leftarrow R_q^{m \times k}; y \leftarrow A(A)].$$





## D2.1 Post-quantum Analysis Report

**SelfTargetMSIS Problem.** Suppose that  $H : \{0, 1\}^* \rightarrow B_\tau$  is a cryptographic hash function. To an adversary  $A$  we associate the advantage function  $\text{Adv}_{H,m,k,\gamma}^{\text{SelfTargetMSIS}}(A)$  as

$$\Pr \left[ \begin{array}{l} 0 \leq \|y\|_\infty \leq \gamma \\ \wedge H(\mu \| |A| \cdot y) = c \end{array} \mid A \leftarrow R_q^{m \times k}; \left( y := \begin{bmatrix} r \\ c \end{bmatrix}, \mu \right) \leftarrow A^{H(\cdot)}(A) \right].$$

**Theorem 1.8.** If  $H$  is a quantum oracle, the advantage of an adversary  $A$  breaking the SUF-CMA security of the signature scheme is

$$\text{Adv}_{\text{Dilithium}}^{\text{SUF-CMA}}(A) \leq \text{Adv}_{k,\ell,D}^{\text{MLWE}}(B) + \text{Adv}_{H,k,\ell+1,\zeta}^{\text{SelfTargetMSIS}}(C) + \text{Adv}_{k,\ell,\zeta'}^{\text{MSIS}}(D) + 2^{-254},$$

for  $D$  a uniform distribution over  $S_\eta$ , and

$$\zeta = \max\{\gamma_1 - \beta, 2\gamma_2 + 1 + 2^{d-1} \cdot \tau\},$$

$$\zeta' = \max\{2(\gamma_1 - \beta), 4\gamma_2 + 2\}.$$

Furthermore, if the running time and success probabilities of  $A, B, C, D$  are defined as  $t_A, t_B, t_C, t_D, \epsilon_A, \epsilon_B, \epsilon_C, \epsilon_D$ , then the lower bound on  $t_A/\epsilon_A$  is within a small multiplicative factor of  $\min t_i/\epsilon_i$  for  $i \in \{B, C, D\}$ .

## I 1.13 SPHINCS<sup>+</sup>

Our description relies on the document provided in <https://sphincs.org/data/sphincs+-r3.1-specification.pdf>. Major parts of the description are taken verbatim from this source.

### WOTS<sup>+</sup> One-Time Signatures

WOTS<sup>+</sup> is a one-time signature (OTS) scheme [42]; while a private key can be used to sign any message, each private key must not be used to sign more than a single message. In particular, if a private key is used to sign two different messages, the scheme becomes insecure.

**WOTS<sup>+</sup> Parameters.** WOTS<sup>+</sup> uses the parameters  $n$  and  $w$ ; they both take positive integer values. These parameters are summarized as follows:

- $n$ : the security parameter; it is the message length as well as the length of a private key, public key, or signature element in bytes.
- $w$ : the Winternitz parameter; it is an element of the set  $\{4, 16, 256\}$ .

These parameters are used to compute values  $\text{len}$ ,  $\text{len}_1$  and  $\text{len}_2$ :

- $\text{len}$ : the number of  $n$ -byte-string elements in a WOTS<sup>+</sup> private key, public key, and signature. It is computed as  $\text{len} = \text{len}_1 + \text{len}_2$ , with

$$\text{len}_1 = \left\lceil \frac{8n}{\log(w)} \right\rceil, \text{len}_2 = \left\lceil \frac{\log(\text{len}_1(w-1))}{\log(w)} \right\rceil + 1$$

**WOTS<sup>+</sup> Chaining Function (Function `chain`).** The chaining function computes an iteration of  $F$  on an  $n$ -byte input using a WOTS<sup>+</sup> hash address  $\text{ADRS}$  and a public seed  $\text{PK.seed}$ . The address  $\text{ADRS}$  must have the first seven 32-bit words set to encode the address of this chain. In each iteration, the address is updated to encode the current position in the chain before  $\text{ADRS}$  is used to process the input by  $F$ .

The chaining function takes as input an  $n$ -byte string  $X$ ; the chaining function returns as output the value obtained by iterating  $F$  for  $s$  times on input  $X$ .

## D2.1 Post-quantum Analysis Report

```
#Input: Input string X, start index i, number of steps s, public seed PK.seed, address
        ADRS
#Output: value of F iterated s times on X

chain(X, i, s, PK.seed, ADRS) {
    if ( s == 0 ) {
        return X;
    }
    if ( (i + s) > (w - 1) ) {
        return NULL;
    }
    byte[n] tmp = chain(X, i, s - 1, PK.seed, ADRS);

    ADRS.setHashAddress(i + s - 1);
    tmp = F(PK.seed, ADRS, tmp);
    return tmp;
}
```

Algorithm 1: chain – Chaining function used in WOTS<sup>+</sup>.

WOTS<sup>+</sup> Private Key (Function `wots_SKgen`). The WOTS<sup>+</sup> private key, denoted by `sk`, is a length `len` array of `n`-byte strings. This private key must not be used to sign more than one message. This private key is only implicitly used.

The following pseudocode describes an algorithm to generate a WOTS<sup>+</sup> private key.

```
#Input: secret seed SK.seed, address ADRS
#Output: WOTS+ private key sk

wots_SKgen(SK.seed, ADRS) {
    skADRS = ADRS; // copy address to create key generation address
    skADRS.setType(WOTS_PRF);
    skADRS.setKeyPairAddress(ADRS.getKeyPairAddress());
    for ( i = 0; i < len; i++ ) {
        skADRS.setChainAddress(i);
        skADRS.setHashAddress(0);
        sk[i] = PRF(SK.seed, skADRS);
    }
    return sk;
}
```

Algorithm 2: `wots_SKgen` – Generating a WOTS<sup>+</sup> private key.

WOTS<sup>+</sup> Public Key Generation (Function `wots_PKgen`). A WOTS<sup>+</sup> key pair defines a virtual structure that consists of `len` hash chains of length `w`. Each of the `len` strings of `n`-bytes in the private key defines the start node for one hash chain. The public key is the tweakable hash of the end nodes of these hash chains.

The following pseudocode describes an algorithm for generating the public key `pk`.

```
#Input: secret seed SK.seed, address ADRS, public seed PK.seed
#Output: WOTS+ public key pk

wots_PKgen(SK.seed, PK.seed, ADRS) {
    wotspkADRS = ADRS; // copy address to create OTS public key address
    skADRS = ADRS; // copy address to create key generation address
    skADRS.setType(WOTS_PRF);
    skADRS.setKeyPairAddress(ADRS.getKeyPairAddress());
    for ( i = 0; i < len; i++ ) {
        skADRS.setChainAddress(i);
        skADRS.setHashAddress(0);
        sk[i] = PRF(SK.seed, skADRS);
    }
}
```

## D2.1 Post-quantum Analysis Report

```

    ADRS.setChainAddress(i);
    ADRS.setHashAddress(0);
    tmp[i] = chain(sk[i], 0, w - 1, PK.seed, ADRS);
}
wotspkADRS.setType(WOTS_PK);
wotspkADRS.setKeyPairAddress(ADRS.getKeyPairAddress());
pk = T_len(PK.seed, wotspkADRS, tmp);
return pk;
}

```

Algorithm 3: wots\_PKgen – Generating a WOTS<sup>+</sup> public key.

WOTS<sup>+</sup> Signature Generation (Function `wots_sign`). A WOTS<sup>+</sup> signature is a length `len` array of  $n$ -byte strings. The WOTS<sup>+</sup> signature is generated by mapping a message  $M$  to `len` integers between 0 and  $w - 1$ . To this end, the message is transformed into `len1` base- $w$  numbers using the `base_w` function. Each of the base- $w$  integers is used to select a node from a different hash chain. The signature is formed by concatenating the selected nodes. A WOTS<sup>+</sup> hash address `ADRS`, a public seed `PK.seed`, and a secret seed `SK.seed` have to be provided by the calling algorithm. The pseudocode for generating a WOTS<sup>+</sup> signature `sig` is shown below.

*#Input: Message M, secret seed SK.seed, public seed PK.seed, address ADRS*  
*#Output: WOTS+ signature sig*

```

wots_sign(M, SK.seed, PK.seed, ADRS) {
    csum = 0;

    // convert message to base w
    msg = base_w(M, w, len_1);

    // compute checksum
    for ( i = 0; i < len_1; i++ ) {
        csum = csum + w - 1 - msg[i];
    }

    // convert csum to base w
    if( (lg(w) % 8) != 0 ) {
        csum = csum << ( 8 - ( ( len_2 * lg(w) ) % 8 ) );
    }
    len_2_bytes = ceil( ( len_2 * lg(w) ) / 8 );
    msg = msg || base_w(toByte(csum, len_2_bytes), w, len_2);

    skADRS = ADRS; // copy address to create key generation address
    skADRS.setType(WOTS_PRF);
    skADRS.setKeyPairAddress(ADRS.getKeyPairAddress());
    for ( i = 0; i < len; i++ ) {
        skADRS.setChainAddress(i);
        skADRS.setHashAddress(0);
        sk = PRF(SK.seed, skADRS);
        ADRS.setChainAddress(i);
        ADRS.setHashAddress(0);
        sig[i] = chain(sk, 0, msg[i], PK.seed, ADRS);
    }
    return sig;
}

```

Algorithm 4: wots\_sign – Generating a WOTS+ signature on a message  $M$ .

WOTS<sup>+</sup> Compute Public Key from Signature (Function `wots_pkFromSig`). SPHINCS<sup>+</sup> uses implicit signature verification for WOTS<sup>+</sup>. In order to verify a WOTS<sup>+</sup> signature `sig` on a message  $M$ , the verifier computes a WOTS<sup>+</sup> public key value from the signature. This can be done by

## D2.1 Post-quantum Analysis Report

“completing” the chain computations starting from the signature values, using the base- $w$  values of the message hash and its checksum. This step, called `wots_pkFromSig`, is described below. The result of `wots_pkFromSig` then has to be verified. In a standalone version, this would be done by simple comparison. When used in SPHINCS<sup>+</sup> the output value is verified by using it to compute a SPHINCS<sup>+</sup> public key.

A WOTS<sup>+</sup> hash address `ADRS` and a public seed `PK.seed` have to be provided by the calling algorithm.

**#Input:** Message `M`, WOTS<sup>+</sup> signature `sig`, address `ADRS`, public seed `PK.seed`  
**#Output:** WOTS<sup>+</sup> public key `pk_sig` derived from `sig`

```
wots_pkFromSig(sig, M, PK.seed, ADRS) {
    csum = 0;
    wotspkADRS = ADRS;

    // convert message to base w
    msg = base_w(M, w, len_1);

    // compute checksum
    for ( i = 0; i < len_1; i++ ) {
        csum = csum + w - 1 - msg[i];
    }

    // convert csum to base w
    csum = csum << ( 8 - ( ( len_2 * lg(w) ) % 8 ));
    len_2_bytes = ceil( ( len_2 * lg(w) ) / 8 );
    msg = msg || base_w(toByte(csum, len_2_bytes), w, len_2);
    for ( i = 0; i < len; i++ ) {
        ADRS.setChainAddress(i);
        tmp[i] = chain(sig[i], msg[i], w - 1 - msg[i], PK.seed, ADRS);
    }

    wotspkADRS.setType(WOTS_PK);
    wotspkADRS.setKeyPairAddress(ADRS.getKeyPairAddress());
    pk_sig = T_len(PK.seed, wotspkADRS, tmp);
    return pk_sig;
}
```

Algorithm 5: `wots_pkFromSig` – Computing a WOTS<sup>+</sup> public key from a message and its signature.

## The SPHINCS<sup>+</sup> Hypertree

In this section, we explain how the SPHINCS<sup>+</sup> hypertree is built.

(Fixed Input-Length) XMSS. XMSS is a method for signing a potentially large but fixed number of messages. It authenticates  $2^{h'}$  WOTS<sup>+</sup> public keys using a binary tree of height  $h'$ . Hence, an XMSS key pair for height  $h'$  can be used to sign  $2^{h'}$  different messages. Each node in the binary tree is an  $n$ -byte value which is the tweakable hash of the concatenation of its two child nodes. The leaves are the WOTS<sup>+</sup> public keys. The XMSS public key is the root node of the tree. In SPHINCS<sup>+</sup>, the XMSS secret key is the single secret seed that is used to generate all WOTS<sup>+</sup> secret keys.

An XMSS signature in the context of SPHINCS<sup>+</sup> consists of the WOTS<sup>+</sup> signature on the message and the so-called authentication path. The latter is a vector of tree nodes that allow a verifier to compute a value for the root of the tree starting from a WOTS<sup>+</sup> signature. A verifier computes the root value and verifies its correctness. A standalone XMSS signature also contains the index of the used WOTS<sup>+</sup> key pair. In the context of SPHINCS<sup>+</sup> this is not necessary as the SPHINCS<sup>+</sup> signature allows to compute the index for each XMSS signature contained.

## D2.1 Post-quantum Analysis Report

XMSS Parameters. XMSS has the following parameters:

$h'$  : the height (number of levels - 1) of the tree.

$n$  : the length in bytes of messages as well as of each node.

$w$  : the Winternitz parameter as defined for WOTS<sup>+</sup> in the previous Section.

There are  $2^{h'}$  leaves in the tree. XMSS signatures are denoted by  $SIG_{XMSS}$  ( $SIG\_XMSS$  in pseudocode). WOTS<sup>+</sup> signatures are denoted by  $sig$ .

XMSS parameters are implicitly included in the algorithm inputs as needed.

**XMSS Private Key.** In the context of SPHINCS<sup>+</sup>, an XMSS private key is the single secret seed  $SK.seed$  contained in the SPHINCS<sup>+</sup> secret key. It is used to generate the WOTS<sup>+</sup> secret keys within the structure of an XMSS key pair as described in subsubsection 1.13.1.

**TreeHash (Function `treehash`).** For the computation of the internal  $n$ -byte nodes of a Merkle tree, the subroutine `treehash` accepts a secret seed  $SK.seed$ , a public seed  $PK.seed$ , an unsigned integer  $s$  (the start index), an unsigned integer  $z$  (the target node height), and an address  $ADRS$  that encodes the address of the containing tree. The `treehash` algorithm returns the root node of a tree of height  $z$  with the leftmost leaf being the WOTS<sup>+</sup> pk at index  $s$ .

```
# Input: Secret seed SK.seed, start index s, target node height z, public seed PK.seed,
        address ADRS
# Output: n-byte root node - top node on Stack

treehash(SK.seed, s, z, PK.seed, ADRS) {
    if( s % (1 << z) != 0 ) return -1;
    for ( i = 0; i < 2^z; i++ ) {
        ADRS.setType(WOTS_HASH);
        ADRS.setKeyPairAddress(s + i);
        node = wots_PKgen(SK.seed, PK.seed, ADRS);
        ADRS.setType(TREE);
        ADRS.setTreeHeight(1);
        ADRS.setTreeIndex(s + i);
        while ( Top node on Stack has same height as node ) {
            ADRS.setTreeIndex((ADRS.getTreeIndex() - 1) / 2);
            node = H(PK.seed, ADRS, (Stack.pop() || node));
            ADRS.setTreeHeight(ADRS.getTreeHeight() + 1);
        }
        Stack.push(node);
    }
    return Stack.pop();
}
```

Algorithm 6: `treehash` – The TreeHash algorithm.

**XMSS Public Key Generation (Function `xmss_PKgen`).** The XMSS public key is computed as described in `xmss_PKgen`. In the context of SPHINCS<sup>+</sup> the XMSS public key  $PK$  is the root of the binary hash tree. The root is computed using `treehash`. The public key generation takes a secret seed  $SK.seed$ , a public seed  $PK.seed$ , and an address  $ADRS$ . The latter encodes the position of this XMSS instance within the SPHINCS<sup>+</sup> structure.

```
# Input: Secret seed SK.seed, public seed PK.seed, address ADRS
# Output: XMSS public key PK

xmss_PKgen(SK.seed, PK.seed, ADRS) {
    pk = treehash(SK.seed, 0, h', PK.seed, ADRS)
    return pk;
}
```

## D2.1 Post-quantum Analysis Report

}

Algorithm 7: `xmss_PKgen` – Generating an XMSS public key.

XMSS Signature. An XMSS signature is a  $((len + h') \cdot n)$ -byte string consisting of

- a WOTS<sup>+</sup> signature sig taking  $len \cdot n$  bytes,
- the authentication path AUTH for the leaf associated with the used WOTS<sup>+</sup> key pair taking  $h' \cdot n$  bytes.

An authentication path contains exactly one node on every layer  $0 \leq x \leq (h' - 1)$ . For the  $i$ th WOTS<sup>+</sup> key pair, counting from zero, the  $j$ th authentication path node is

$$AUTH[j] = N \left( j, \left\lfloor \frac{i}{2^j} \right\rfloor \oplus 1 \right)$$

XMSS Signature Generation (Function `xmss_sign`). To compute the XMSS signature of a message  $M$  in the context of SPHINCS<sup>+</sup>, the secret seed `SK.seed`, the public seed `PK.seed`, the index `idx` of the WOTS<sup>+</sup> key pair to be used, and the address `ADRS` of the XMSS instance are needed.

```
# Input: n-byte message M, secret seed SK.seed, index idx, public seed PK.seed, address
        ADRS
# Output: XMSS signature SIG_XMSS = (sig || AUTH)

xmss_sign(M, SK.seed, idx, PK.seed, ADRS)
    // build authentication path
    for ( j = 0; j < h'; j++ ) {
        k = floor(idx / (2^j)) XOR 1;
        AUTH[j] = treehash(SK.seed, k * 2^j, j, PK.seed, ADRS);
    }

    ADRS.setType(WOTS_HASH);
    ADRS.setKeyPairAddress(idx);
    sig = wots_sign(M, SK.seed, PK.seed, ADRS);
    SIG_XMSS = sig || AUTH;
    return SIG_XMSS;
}
```

Algorithm 8: `xmss_sign` – Generating an XMSS signature.

XMSS Compute Public Key from Signature (Function `xmss_pkFromSig`). SPHINCS<sup>+</sup> makes use of implicit signature verification of XMSS signatures. An XMSS signature is used to compute a candidate XMSS public key, i.e., the root of the tree. This is used in further computations (signature of the tree above) and implicitly verified by the outcome of that computation. Hence, this specification does not contain an `xmss_verify` method but the method `xmss_pkFromSig`.

The method `xmss_pkFromSig` takes an  $n$ -byte message  $M$ , an XMSS signature `SIG_XMSS`, a signature index `idx`, a public seed `PK.seed`, and an address `ADRS`.

```
# Input: index idx, XMSS signature SIG_XMSS = (sig || AUTH), n-byte message M, public seed
        PK.seed, address ADRS
# Output: n-byte root value node[0]

xmss_pkFromSig(idx, SIG_XMSS, M, PK.seed, ADRS){

    // compute WOTS+ pk from WOTS+ sig
    ADRS.setType(WOTS_HASH);
```



## D2.1 Post-quantum Analysis Report

```

    ADRS.setKeyPairAddress(idx);
    sig = SIG_XMSS.getWOTSSig();
    AUTH = SIG_XMSS.getXMSSAUTH();
    node[0] = wots_pkFromSig(sig, M, PK.seed, ADRS);

    // compute root from WOTS+ pk and AUTH
    ADRS.setType(TREE);
    ADRS.setTreeIndex(idx);
    for ( k = 0; k < h'; k++ ) {
        ADRS.setTreeHeight(k+1);
        if ( (floor(idx / (2^k)) % 2) == 0 ) {
            ADRS.setTreeIndex(ADRS.getTreeIndex() / 2);
            node[1] = H(PK.seed, ADRS, (node[0] || AUTH[k]));
        } else {
            ADRS.setTreeIndex((ADRS.getTreeIndex() - 1) / 2);
            node[1] = H(PK.seed, ADRS, (AUTH[k] || node[0]));
        }
        node[0] = node[1];
    }
    return node[0];
}

```

Algorithm 9: xmss\_pkFromSig – Computing an XMSS public key from an XMSS signature.

HT: The Hypertree. The SPHINCS<sup>+</sup> hypertree HT is a variant of XMSS<sup>MT</sup>. It is essentially a certification tree of XMSS instances. A HT is a tree of several layers of XMSS trees. The trees on top and intermediate layers are used to sign the public keys. Trees on the lowest layer are used to sign the actual messages, which are FORS public keys in SPHINCS<sup>+</sup>. All XMSS trees in HT have equal height.

Consider a HT of total height  $h$  that has  $d$  layers of XMSS trees of height  $h' = h/d$ . Then layer  $d - 1$  contains one XMSS tree, layer  $d - 2$  contains  $2^{h'}$  XMSS trees, and so on. Finally, layer 0 contains  $2^{h-h'}$  XMSS trees.

HT Parameters. In addition to all XMSS parameters, a HT requires the hypertree height  $h$  and the number of tree layers  $d$ , specified as an integer value that divides  $h$  without remainder. The same tree height  $h' = h/d$  and the same Winternitz parameter  $w$  is used for all tree layers.

HT Key Generation (Function ht\_PKgen). The HT private key is the secret seed SK.seed which is used to generate all the WOTS<sup>+</sup> private keys within the virtual structure spanned by the HT.

The HT public key is the public key (root node) of the single XMSS tree on the top layer. Its computation is explained below. The public key generation takes as input a private and a public seed.

```

# Input: Private seed SK.seed, public seed PK.seed
# Output: HT public key PK_HT

ht_PKgen(SK.seed, PK.seed){
    ADRS = toByte(0, 32);
    ADRS.setLayerAddress(d-1);
    ADRS.setTreeAddress(0);
    root = xmss_PKgen(SK.seed, PK.seed, ADRS);
    return root;
}

```

Algorithm 10: ht\_PKgen – Generating an HT public key.

HT Signature. A HT signature SIG<sub>HT</sub> is a byte string of length  $(h + d * 1en) * n$ . It consists of  $d$  XMSS signatures (of  $(h/d + 1en) * n$  bytes each).



The data format for a signature is given in Figure 13

XMSS signature $SIG_{XMSS}$ (layer 0) $((h/d + 1) \cdot n)$ bytes)
XMSS signature $SIG_{XMSS}$ (layer 1) $((h/d + 1) \cdot n)$ bytes)
...
XMSS signature $SIG_{XMSS}$ (layer $d - 1$ ) $((h/d + 1) \cdot n)$ bytes)

Figure 13: Structure of a HT signature in the HT scheme. Each layer represents an XMSS signature, and the entire signature is a byte string of length  $(h + d \cdot 1en) \cdot n$ .

HT Signature Generation (Function `ht_sign`). To compute a HT signature  $SIG_{HT}$  of a message  $M$  using, `ht_sign` described below uses `xmss_sign`. The algorithm `ht_sign` takes as input a message  $M$ , a private seed  $SK.seed$ , a public seed  $PK.seed$ , and an index  $idx$ .

Algorithm `ht_sign` uses `xmss_pkFromSig` to compute the root node of an XMSS instance after that instance was used for signing. The algorithm `ht_sign` as described below is just one way to generate a HT signature.

```
# Input: Message M, private seed SK.seed, public seed PK.seed, tree index idx_tree, leaf
        index idx_leaf
# Output: HT signature SIG_HT

ht_sign(M, SK.seed, PK.seed, idx_tree, idx_leaf) {
    // init
    ADRS = toByte(0, 32);

    // sign
    ADRS.setLayerAddress(0);
    ADRS.setTreeAddress(idx_tree);
    SIG_tmp = xmss_sign(M, SK.seed, idx_leaf, PK.seed, ADRS);
    SIG_HT = SIG_HT || SIG_tmp;
    root = xmss_pkFromSig(idx_leaf, SIG_tmp, M, PK.seed, ADRS);
    for ( j = 1; j < d; j++ ) {
        idx_leaf = (h / d) least significant bits of idx_tree;
        idx_tree = (h - (j + 1) * (h / d)) most significant bits of idx_tree;
        ADRS.setLayerAddress(j);
        ADRS.setTreeAddress(idx_tree);
        SIG_tmp = xmss_sign(root, SK.seed, idx_leaf, PK.seed, ADRS);
        SIG_HT = SIG_HT || SIG_tmp;
        if ( j < d - 1 ) {
            root = xmss_pkFromSig(idx_leaf, SIG_tmp, root, PK.seed, ADRS);
        }
    }
    return SIG_HT;
}
```

Algorithm 11: `ht_sign` – Generating an HT signature

HT Signature Verification (Function `ht_verify`). HT signature verification can be summarized as  $d$  calls to `xmss_pkFromSig` and one comparison with a given value. HT signature verification takes a message  $M$ , a signature  $SIG_{HT}$ , a public seed  $PK.seed$ , an index  $idx$  (split into a tree index and a leaf index, as above), and a HT public key  $PK_{HT}$ .



## D2.1 Post-quantum Analysis Report

```
# Input: Message M, signature SIG_HT, public seed PK.seed, tree index idx_tree, leaf index
        idx_leaf, HT public key PK_HT.
# Output: Boolean

ht_verify(M, SIG_HT, PK.seed, idx_tree, idx_leaf, PK_HT){
    // init
    ADRS = toByte(0, 32);

    // verify
    SIG_tmp = SIG_HT.getXMSSSignature(0);
    ADRS.setLayerAddress(0);
    ADRS.setTreeAddress(idx_tree);
    node = xmss_pkFromSig(idx_leaf, SIG_tmp, M, PK.seed, ADRS);
    for ( j = 1; j < d; j++ ) {
        idx_leaf = (h / d) least significant bits of idx_tree;
        idx_tree = (h - (j + 1) * h / d) most significant bits of idx_tree;
        SIG_tmp = SIG_HT.getXMSSSignature(j);
        ADRS.setLayerAddress(j);
        ADRS.setTreeAddress(idx_tree);
        node = xmss_pkFromSig(idx_leaf, SIG_tmp, node, PK.seed, ADRS);
    }
    if ( node == PK_HT ) {
        return true;
    } else {
        return false;
    }
}
```

Algorithm 12: `ht_verify` – Verifying a HT signature  $SIG_{HT}$  on a message  $M$  using a HT public key  $PK_{HT}$

## FORS: Forest Of Random Subsets

The SPHINCS<sup>+</sup> hypertree HT is not used to sign the actual messages but the public keys of FORS instances which in turn are used to sign message digests. FORS is a few-time signature scheme (FTS). For security, it is essential that the input to FORS is the output of a hash function. In the following, we describe FORS as acting on bit strings.

FORS Parameters. FORS uses the parameters  $n$ ,  $k$ , and  $t$ ; they all take positive integer values. These parameters are summarized as follows:

- $n$ : the security parameter; it is the length of a private key, public key, or signature element in bytes.
- $k$ : the number of private key sets, trees, and indices computed from the input string.
- $t$ : the number of elements per private key set, number of leaves per hash tree, and upper bound on the index values. The parameter  $t$  MUST be a power of 2. If  $t = 2^a$ , then the trees have height  $a$  and the input string is split into bit strings of length  $a$ .

Inputs to FORS are bit strings of length  $k \log t$ .

FORS Private Key (Function `fors_SKgen`). In the context of SPHINCS<sup>+</sup>, a FORS private key is the single private seed `SK.seed` contained in the SPHINCS<sup>+</sup> private key. It is used to generate the  $kt$   $n$ -byte private key values using PRF with a FORS key generation address.

```
#Input: secret seed SK.seed, address ADRS, secret key index idx = it+j
#Output: FORS private key sk
```

## D2.1 Post-quantum Analysis Report

```
fors_SKgen(SK.seed, ADRS, idx) {
    skADRS = ADRS; // copy address to create key generation address
    skADRS.setType(FORS_PRF);
    skADRS.setKeyPairAddress(ADRS.getKeyPairAddress());

    skADRS.setTreeHeight(0);
    skADRS.setTreeIndex(idx);
    sk = PRF(SK.seed, skADRS);

    return sk;
}
```

Algorithm 13: `fors_SKgen` – Computing a FORS private key value.

FORS TreeHash (Function `fors_treehash`). Before coming to the FORS public key, we have to discuss the computation of the trees. For the computation of the  $n$ -byte nodes in the FORS hash trees, the subroutine `fors_treehash` is used. It is essentially the same algorithm as `treehash` (Algorithm 6) in subsubsection 1.13.2. The two differences are how the leaf nodes are computed and how addresses are handled. However, as the addresses are similar, an implementation can implement both algorithms in the same routine easily.

Algorithm `fors_treehash` accepts a secret seed `SK.seed`, a public seed `PK.seed`, an unsigned integer  $s$  (the start index), an unsigned integer  $z$  (the target node height), and an address `ADRS` that encodes the address of the FORS key pair. As for `treehash`, the `fors_treehash` algorithm returns the root node of a tree of height  $z$  with the leftmost leaf being the hash of the private key element at index  $s$ .

```
# Input: Secret seed SK.seed, start index s, target node height z, public seed PK.seed,
        address ADRS
# Output: n-byte root node - top node on Stack

fors_treehash(SK.seed, s, z, PK.seed, ADRS) {
    if( s % (1 << z) != 0 ) return -1;
    for ( i = 0; i < 2^z; i++ ) {
        sk = fors_SKgen(SK.seed, ADRS, s+i)
        node = F(PK.seed, ADRS, sk);
        ADRS.setTreeHeight(1);
        ADRS.setTreeIndex(s + i);
        while ( Top node on Stack has same height as node ) {
            ADRS.setTreeIndex((ADRS.getTreeIndex() - 1) / 2);
            node = H(PK.seed, ADRS, (Stack.pop() || node));
            ADRS.setTreeHeight(ADRS.getTreeHeight() + 1);
        }
        Stack.push(node);
    }
    return Stack.pop();
}
```

Algorithm 14: The `fors_treehash` algorithm.

FORS Public Key (Function `fors_PKgen`). In the context of SPHINCS<sup>+</sup>, the FORS public key is never generated alone. It is only generated together with a signature. We include `fors_PKgen` for completeness, a better understanding, and testing. Algorithm `fors_PKgen` takes a private seed `SK.seed`, a public seed `PK.seed`, and a FORS address `ADRS`. The latter encodes the position of the FORS instance within SPHINCS<sup>+</sup>. It outputs a FORS public key.

```
# Input: Secret seed SK.seed, public seed PK.seed, address ADRS
# Output: FORS public key PK
```

## D2.1 Post-quantum Analysis Report

```
fors_PKgen(SK.seed, PK.seed, ADRS) {
    forspkADRS = ADRS; // copy address to create FTS public key address

    for(i = 0; i < k; i++){
        root[i] = fors_treehash(SK.seed, i*t, a, PK.seed, ADRS);
    }
    forspkADRS.setType(FORS_ROOTS);
    forspkADRS.setKeyPairAddress(ADRS.getKeyPairAddress());
    pk = T_k(PK.seed, forspkADRS, root);
    return pk;
}
```

Algorithm 15: fors\_PKgen – Generate a FORS public key.

FORS Signature Generation (Function `fors_sign`). A FORS signature is a length  $k(\log t + 1)$  array of  $n$ -byte strings. It contains  $k$  private key values,  $n$ -bytes each, and their associated authentication paths,  $\log t$   $n$ -byte values each.

The algorithm `fors_sign` takes a  $(k \log t)$ -bit string  $M$ , a private seed `SK.seed`, a public seed `PK.seed`, and an address `ADRS`. The latter encodes the position of the FORS instance within SPHINCS<sup>+</sup>. It outputs a FORS signature `SIG_FORS`.

*#Input: Bit string M, secret seed SK.seed, address ADRS, public seed PK.seed*  
*#Output: FORS signature SIG\_FORS*

```
fors_sign(M, SK.seed, PK.seed, ADRS) {
    // compute signature elements
    for(i = 0; i < k; i++){
        // get next index
        unsigned int idx = bits i*log(t) to (i+1)*log(t) - 1 of M;

        // pick private key element
        SIG_FORS = SIG_FORS || fors_SKgen(SK.seed, ADRS, i*t + idx) ;

        // compute auth path
        for ( j = 0; j < a; j++ ) {
            s = floor(idx / (2j)) XOR 1;
            AUTH[j] = fors_treehash(SK.seed, i * t + s * 2j, j, PK.seed, ADRS);
        }
        SIG_FORS = SIG_FORS || AUTH;
    }
    return SIG_FORS;
}
```

Algorithm 16: fors\_sign – Generating a FORS signature on string  $M$ .

FORS Compute Public Key from Signature (Function `fors_pkFromSig`). SPHINCS<sup>+</sup> makes use of implicit signature verification of FORS signatures. A FORS signature is used to compute a candidate FORS public key. This public key is used in further computations and implicitly verified by the outcome of that computation. Hence, this specification does not contain a `fors_verify` method but the method `fors_pkFromSig`.

The method `fors_pkFromSig` makes use of functions `SIG_FORS.getSK(i)` and `SIG_FORS.getAUTH(i)`. The former returns the  $i$ th secret key element stored in the signature, the latter returns the  $i$ th authentication path stored in the signature.

*# Input: FORS signature SIG\_FORS, (k lg t)-bit string M, public seed PK.seed, address ADRS*  
*# Output: FORS public key*

```
fors_pkFromSig(SIG_FORS, M, PK.seed, ADRS){
```

## D2.1 Post-quantum Analysis Report

```

// compute roots
for(i = 0; i < k; i++){
    // get next index
    unsigned int idx = bits i*log(t) to (i+1)*log(t) - 1 of M;

    // compute leaf
    sk = SIG_FORS.getSK(i);
    ADRS.setTreeHeight(0);
    ADRS.setTreeIndex(i*t + idx);
    node[0] = F(PK.seed, ADRS, sk);

    // compute root from leaf and AUTH
    auth = SIG_FORS.getAUTH(i);
    ADRS.setTreeIndex(i*t + idx);
    for ( j = 0; j < a; j++ ) {
        ADRS.setTreeHeight(j+1);
        if ( (floor(idx / (2^j)) % 2) == 0 ) {
            ADRS.setTreeIndex(ADRS.getTreeIndex() / 2);
            node[1] = H(PK.seed, ADRS, (node[0] || auth[j]));
        } else {
            ADRS.setTreeIndex((ADRS.getTreeIndex() - 1) / 2);
            node[1] = H(PK.seed, ADRS, (auth[j] || node[0]));
        }
        node[0] = node[1];
    }
    root[i] = node[0];
}

forspkADRS = ADRS; // copy address to create FTS public key address
forspkADRS.setType(FORS_ROOTS);
forspkADRS.setKeyPairAddress(ADRS.getKeyPairAddress());
pk = T_k(PK.seed, forspkADRS, root);
return pk;
}

```

Algorithm 17: `fors_pkFromSig` – Compute a FORS public key from a FORS signature.

### The SPHINCS<sup>+</sup> Construction

We now have all the ingredients to describe our main construction SPHINCS<sup>+</sup>. Essentially, SPHINCS<sup>+</sup> is an orchestration of the methods and schemes described before. It only adds randomized message compression and verifiable index generation.

Parameters. SPHINCS<sup>+</sup> has the following parameters:

- $n$  : the security parameter in bytes.
- $w$  : the Winternitz parameter as defined in subsection 1.13.1.
- $h$  : the height of the hypertree as defined in 1.13.2.
- $d$  : the number of layers in the hypertree as defined in 1.13.2.
- $k$  : the number of trees in FORS as defined in subsection 1.13.3.
- $t$  : the number of leaves of a FORS tree as defined in subsection 1.13.3.

All the restrictions stated in the previous sections apply. Recall that we use  $a = \log t$ . Moreover, from these values the values  $m$  and  $len$  are computed as

## D2.1 Post-quantum Analysis Report

- $m$ : the message digest length in bytes. It is computed as

$$m = \lfloor (k \log t + 7)/8 \rfloor + \lfloor (h - h/d + 7)/8 \rfloor + \lfloor (h/d + 7)/8 \rfloor.$$

While only  $h + k \log t$  bits would be needed, using the longer  $m$  as defined above simplifies implementations significantly.

- $\text{len}$ : the number of  $n$ -byte string elements in a WOTS<sup>+</sup> private key, public key, and signature. It is computed as  $\text{len} = \text{len}_1 + \text{len}_2$ , with

$$\text{len}_1 = \left\lceil \frac{8n}{\log(w)} \right\rceil, \text{len}_2 = \left\lceil \frac{\log(\text{len}_1(w-1))}{\log(w)} \right\rceil + 1$$

In the following, we assume that all algorithms have access to these parameters.

**SPHINCS<sup>+</sup> Key Generation (Function `spx_keygen`).** The SPHINCS<sup>+</sup> private key contains two elements. First, the  $n$ -byte secret seed `SK.seed` which is used to generate all the WOTS<sup>+</sup> and FORS private key elements. Second, an  $n$ -byte PRF key `SK.prf` is used to deterministically generate a randomization value for the randomized message hash.

The SPHINCS<sup>+</sup> public key also contains two elements. First, the HT public key, i.e. the root of the tree on the top layer. Second, an  $n$ -byte public seed value `PK.seed` which is sampled uniformly at random.

As `spx_sign` does not get the public key but needs access to `PK.seed` (and possibly to `PK.root` for fault attack mitigation), the SPHINCS<sup>+</sup> secret key contains a copy of the public key.

The description of algorithm `spx_keygen` assumes the existence of a function `sec_rand` which on input  $i$  returns  $i$ -bytes of cryptographically strong randomness.

```
# Input: (none)
# Output: SPHINCS+ key pair (SK,PK)

spx_keygen(){
    SK.seed = sec_rand(n);
    SK.prf = sec_rand(n);
    PK.seed = sec_rand(n);
    PK.root = ht_PKgen(SK.seed, PK.seed);
    return ( (SK.seed, SK.prf, PK.seed, PK.root), (PK.seed, PK.root) );
}
```

Algorithm 18: `spx_keygen` – Generate a SPHINCS<sup>+</sup> key pair.

**SPHINCS<sup>+</sup> Signature.** A SPHINCS<sup>+</sup> signature  $\text{SIG}_{\text{HT}}$  is a byte string of length  $(1 + k(a+1) + h + d\text{len})n$ . It consists of an  $n$ -byte randomization string  $R$ , a FORS signature  $\text{SIG}_{\text{FORS}}$  consisting of  $k(a+1)$   $n$ -byte strings, and a HT signature  $\text{SIG}_{\text{HT}}$  of  $(h + d\text{len})n$  bytes.

**SPHINCS<sup>+</sup> Signature Generation (Function `spx_sign`).**

```
# Input: Message M, private key SK = (SK.seed, SK.prf, PK.seed, PK.root)
# Output: SPHINCS+ signature SIG

spx_sign(M, SK){
    // init
    ADRS = toByte(0, 32);

    // generate randomizer
    opt = PK.seed;
    if(RANDOMIZE){
```

## D2.1 Post-quantum Analysis Report

```

    opt = rand(n);
}
R = PRF_msg(SK.prf, opt, M);
SIG = SIG || R;

// compute message digest and index
digest = H_msg(R, PK.seed, PK.root, M);
tmp_md = first floor((ka + 7) / 8) bytes of digest;
tmp_idx_tree = next floor((h - h/d + 7) / 8) bytes of digest;
tmp_idx_leaf = next floor((h/d + 7) / 8) bytes of digest;

md = first ka bits of tmp_md;
idx_tree = first h - h/d bits of tmp_idx_tree;
idx_leaf = first h/d bits of tmp_idx_leaf;

// FORS sign
ADRS.setLayerAddress(0);
ADRS.setTreeAddress(idx_tree);
ADRS.setType(FORS_TREE);
ADRS.setKeyPairAddress(idx_leaf);

SIG_FORS = fors_sign(md, SK.seed, PK.seed, ADRS);
SIG = SIG || SIG_FORS;

// get FORS public key
PK_FORS = fors_pkFromSig(SIG_FORS, md, PK.seed, ADRS);

// sign FORS public key with HT
ADRS.setType(TREE);
SIG_HT = ht_sign(PK_FORS, SK.seed, PK.seed, idx_tree, idx_leaf);
SIG = SIG || SIG_HT;

return SIG;
}

```

Algorithm 19: `spx_sign` – Generating a SPHINCS<sup>+</sup> signature

### SPHINCS<sup>+</sup> Signature Verification (Function `spx_verify`)

```

# Input: Message M, signature SIG, public key PK
# Output: Boolean

spx_verify(M, SIG, PK){
    // init
    ADRS = toByte(0, 32);
    R = SIG.getR();
    SIG_FORS = SIG.getSIG_FORS();
    SIG_HT = SIG.getSIG_HT();

    // compute message digest and index
    digest = H_msg(R, PK.seed, PK.root, M);
    tmp_md = first floor((ka + 7) / 8) bytes of digest;
    tmp_idx_tree = next floor((h - h/d + 7) / 8) bytes of digest;
    tmp_idx_leaf = next floor((h/d + 7) / 8) bytes of digest;

    md = first ka bits of tmp_md;
    idx_tree = first h - h/d bits of tmp_idx_tree;
    idx_leaf = first h/d bits of tmp_idx_leaf;

    // compute FORS public key

```

## D2.1 Post-quantum Analysis Report

```

    ADRS.setLayerAddress(0);
    ADRS.setTreeAddress(idx_tree);
    ADRS.setType(FORS_TREE);
    ADRS.setKeyPairAddress(idx_leaf);

    PK_FORS = fors_pkFromSig(SIG_FORS, md, PK.seed, ADRS);

    // verify HT signature
    ADRS.setType(TREE);
    return ht_verify(PK_FORS, SIG_HT, PK.seed, idx_tree, idx_leaf, PK.root);
}

```

Algorithm 20: `spx_verify` – Verify a SPHINCS<sup>+</sup> signature SIG on a message  $M$  using a SPHINCS<sup>+</sup> public key PK

### Security Assumption

We make a statistical assumption on the hash function  $F$ . Informally we require that every element in the image of  $F$  has at least two preimages, i.e.,

$$(\forall k \in \{0, 1\}^n)(\forall y \in \text{IMG}(F_k))(\exists x, x' \in \{0, 1\}^n : x \neq x' \wedge F_k(x) = f_k(x')). \quad (1)$$

Informally, we will prove the following Theorem where  $F$ ,  $H$ , and  $T$  are the cryptographic hash functions used to instantiate  $F$  and  $H$ ,  $T$  respectively.

Theorem 1.9. For security parameter  $n \in \mathbb{N}$ , parameters  $w, h, d, m, t, k$  as described above, SPHINCS<sup>+</sup> is existentially unforgeable under post-quantum adaptive chosen message attacks if

- $F$ ,  $H$ , and  $T$  are post-quantum distinct-function multi-target second-preimage resistant function families,
- $F$  fulfills the requirement of Eqn. 1,
- $\text{PRF}$ ,  $\text{PRF}_{\text{msg}}$  are post-quantum pseudorandom function families,
- $\text{PRF}_{\text{BM}}$  is modeled as a quantum-accessible random oracle, and
- $H_{\text{msg}}$  is a post-quantum interleaved target subset resilient hash function family.

More specifically, the insecurity function  $\text{InSec}^{\text{PQ-EU-CMA}}(\text{SPHINCS}^+; \xi, 2^h)$  describing the maximum success probability over all adversaries running in time  $\leq \xi$  against the PQ-EU-CMA security of SPHINCS<sup>+</sup> is bounded by

$$\begin{aligned} \text{InSec}^{\text{pq-eu-cma}}(\text{SPHINCS}^+; \xi) &\leq 2(\text{InSec}^{\text{pq-prf}}(\text{PRF}; \xi) + \text{InSec}^{\text{pq-prf}}(\text{PRF}_{\text{msg}}; \xi)) \\ &+ \text{InSec}^{\text{pq-itsr}}(H_{\text{msg}}; \xi) + \text{InSec}^{\text{pq-dm-spr}}(F; \xi) + \text{InSec}^{\text{pq-dm-spr}}(H; \xi) + \text{InSec}^{\text{pq-dm-spr}}(T; \xi) \end{aligned} \quad (2)$$

## | 1.14 Privacy-Preserving Authentication

Use-Case 3 of the Confidential6G project envisions secure vehicle-to-infrastructure communication. Like vehicle-to-vehicle communication, it deals with vehicles that continuously communicate with their environment. Early on it has been recognized that such a technology can threaten user privacy if vehicles can be uniquely identified. This can allow attackers to create movement profiles of users. At the same time, useful applications crucially require that vehicles provide authenticated information to avoid vandalism and safety threats through



fake vehicles. Reconciling these two requirements – authentication and privacy preservation – requires deviating from the application of mere digital signatures. A promising solution revolves around privacy-preserving signatures. These are signatures that prove that the signer (or rather her car) is a member of some set  $U$  of users but does not reveal who exactly she is among the members of  $U$ . In the scenario that we consider  $U$  can be the set of registered cars. The security properties now require that 1) it is hard to decide given a signature who exactly signed it and 2) it is hard to produce a valid signature for non-members of  $U$ . Generally, these types of privacy-preserving signatures come in many flavors and variants. Arguably the most well-known are ring signatures [48] and group signatures [9]. In a ring signature, the privacy guarantees cannot be revoked. In group signatures, a dedicated judge can be specified which has the power to revoke the anonymity of each group signature in case the need arises. In the vehicle-to-X scenario, this can help to identify drivers that put the safety of others at risk. Thus users can stay generally anonymous until evidence is presented to a judge that she has been responsible for a violation of laws. Post-quantum variants of these signature schemes have been proposed. Similar to vanilla signature schemes have less efficiency than their traditional counterparts.

### Existing Solutions - Direct Anonymous Attestation

Group signatures have successfully been used as building blocks for privacy-preserving higher-level protocols like anonymous credential systems [15] and direct anonymous attestation [14]. Some of the use cases of Confidential6G consider these higher-level protocols explicitly. However, so far post-quantum secure instantiations have not been standardized.

### Existing Group Signatures

In contrast to KEMs and digital signatures, there are also, to the best of our knowledge, no standardization efforts underway to standardize post-quantum secure group signatures. Moreover, these building blocks have received less scrutiny than the NIST algorithms and in general, are less mature for applications in practice. Given the range of choices for parameters in a group signature scheme, the selection of which scheme should be used highly depends on the exact application. Moreover, new research ideas might fit the applications considered in Confidential6G better and offer higher efficiency. Group signatures are much harder to build than usual digital signatures and in fact, they typically imply classical signatures. Moreover, they involve several sub-protocols that set up parameters, produce user and judge keys, sign messages, verify messages, and revoke anonymity. The underlying building blocks used to build group signatures include classical digital signatures, zero-knowledge proofs, and KEMs. We observe that in general, all underlying building blocks have to be post-quantum secure to achieve an overall post-quantum secure group signature. As an example of an interesting, recent, post-quantum secure group signature we refer to <https://eprint.iacr.org/2021/1575.pdf>.

### Existing Oblivious Pseudorandom Functions

Oblivious Pseudorandom Functions (OPRFs) are a fundamental building block in privacy-preserving computation. On a high level, the OPRF takes an input, usually from a user, and a key, usually from a server. From this, it computes a pseudorandom output and gives it to the user. The protocol is secure when the user learns nothing but the output, especially nothing about the server key, and the server learns nothing, especially not about the user input.

Several applications, from the OPAQUE protocol which enables a password-authenticated key exchange to private set intersection protocols, use OPRFs to get a cryptographically secure, high-entropy object from a low-entropy input in a privacy-preserving manner. Another nice application is PrivacyPass, which reduces the number of necessary CAPTCHAs by using a token system, and in turn, reduces the number of manually solved challenges by a factor of  $30\times$ . To



## D2.1 Post-quantum Analysis Report

get a better overview of OPRFs, we recommend this very good systematization of knowledge that can be found at <https://eprint.iacr.org/2022/302.pdf>.

There are currently standardization efforts for OPRFs from prime order groups. The security assumption does not hold in the quantum setting. While there are numerous proposals for post-quantum instantiations, they are all very new and have not been analyzed thoroughly. In the table below, we review several of the current instantiations.

work	assumption	rounds	comm. cost	model (C-S)	no preproc.	no trusted setup	verifiable	full impl. available
[57]	3-Hash SDHI	2	766 bits	●-●	✓	✓	✓	✓
[5]	(R)LWE+SIS	2	2MB	●-○	✓	✓	✗	✓
[5]	(R)LWE+SIS	2	> 128 GB	●-●	✓	✓	✓	✗
[52]	multivariate	3	$\gamma \cdot 13$ kB	●-○	✗	✓	✓	✗
[23]	AES+GC	2	6.79MB	●-○	✓	✓	✗	✓
[20]	mod(2,3)	2	1836 bits	●-○	✗	✗	✗	✗
[4]	mod(2,3)+lattices	2	2.5 MB+10 KB	●-○	✓	✓	✗	✗
[4]	mod(2,3)+lattices	2	2.5 MB+160 KB	●-○	✓	✓	✓	✗
[8]	Isogenies $\mathbb{F}_{p^2}$	2	3.0 MB	●-●	✓	✗	✗	✗
[8]	Isogenies $\mathbb{F}_{p^2}$	2	8.7 MB	●-●	✓	✗	✓	✗
[12]	Isogenies $\mathbb{F}_p$ + lattices	2	20.54 kB	●-○	✓	✗	✗	✗
[12]	Isogenies $\mathbb{F}_p$ + lattices	4	34.88 kB	●-○	✓	✗	✗	✗
[32]	Isogenies $\mathbb{F}_p$ + lattices + HE OT	2	640 kB	●-○	✓	✓	✗	✓
[32]	CSIDH	258	24.7 kB	●-○	✓	✓	✗	✓

### Existing Zero-Knowledge Proofs

Zero-knowledge proofs prove statements in a yes-no fashion without disclosing the actual value of the statement. Imagine visiting the cinema and wanting to prove you are of age: instead of telling your concrete birth date, it suffices to verify the yes-no statement of being over a certain age threshold.

In a cryptographic world, the statements are usually related to knowledge of a private key. These proofs are relatively simple in a pre-quantum version, and similar to signing a challenge value with a given key. In a post-quantum world, the availability of efficient zero-knowledge proofs depends on what statement should be proven. Zero-knowledge proofs may be used in protocols to secure against leakage induced by carefully crafted queries from malicious adversaries. In lattice problems, this could mean proving that an error term in an LWE problem was sampled correctly. An interactive explanation of a so-called range proof can be found at <https://lattice-zk.iaik.tugraz.at/>.

## 2 Threats on the 6G Infrastructure

Asymmetric cryptography is used in 5G to protect the access and core network via general-purpose protocols like TLS [47, 18, 22] and IPsec [27, 26, 7, 29] and additionally for identity concealment. These applications will likely be used in 6G as well. They guarantee basic security properties in the network. In their most general modes, the first two protocols rely on key exchange protocols to establish confidential message transfer and digital certificates to provide party authentication. Digital certificates rely on digital signatures and a public-key infrastructure that reflects a trust hierarchy. It is important to keep in mind that generally, these protocols rely on asymmetric cryptography to achieve long-term security – even when authentication is based on authentication via pre-shared symmetric keys! It is true that authentication can be achieved entirely by symmetric primitives via pre-shared keys (at the cost of pre-distributing symmetric keys), for example using MACs. However, the current protocols require asymmetric systems to achieve what is called “forward security” in the 5G standard and is referred to as “post-compromise security” in the academic literature: If an attacker knows the shared key  $k_n$  at time  $n$  they should not be able to predict the shared key  $k_{n+i}$  at time  $n+i$  for  $i > 0$ . This property is required in addition to what the academic literature calls “forward secrecy”, which guarantees that even after the corruption of long-term keys, past communication sessions cannot be decrypted. In principle this latter property can be achieved by updating keys using a one-way function, such as a hash function, to compute  $k_{i+1} = H(k_i)$  and deleting the previous key  $k_i$ , but the former requires public-key operations that introduce new randomness. Almost all network protocols, with the notable exception of Kerberos, involve fresh key exchanges using public-key cryptography for each new session.

### | 2.1 Identity Concealment

While the application of general-purpose security protocols like TLS and IPsec for the protection of the access and core network needs little explanation, we would like to briefly provide background information on identity concealment in 5G <sup>11</sup>. In 5G, each SIM card is assigned a Subscription Permanent Identifier (SUPI), a permanent unique identifier that identifies each subscriber. In 4G, the SUPI was called IMSI (International Mobile Subscriber Identity). The SUPI is a string (15 or 16 digits long) that contains three parts: a 3-bit MCC (Mobile Country Code), 2-3 digits MNC (Mobile Network Code), and 10 digits MSIN (Mobile Subscriber Identification Number) that identifies the user. In 4G, the identifier was transferred to the home network without any protection. This allowed for a MitM attack: an attacker could simply intercept and spoof IMSIs via a device called IMSI catcher. 5G addresses this problem by encrypting the SUPI before sending it to the home network. The encrypted SUPI is referred to as SUCI (Subscription Concealed Identifier). However, only the MSIN is encrypted, the MCC and MNC parts are left unprotected. The encryption algorithm used for the SUPI-to-SUCI transformation is ECIES (Elliptic Curve Integrated Encryption Scheme) an IND-CCA2-secure public-key encryption system using elliptic curves. ECIES can be considered as the elliptic curve version of DHIES [1]<sup>12</sup>. It uses a Diffie–Hellman key exchange to derive a symmetric encryption key which is then used for authentication and encryption. As such it also provides integrity protection. The home network now holds a public and secret key pair. The public key must be available to the user device. Concretely, the standard supports two profiles for ECIES, essentially two sets of parameters based on Curve25519 or secp256r1. The underlying symmetric encryption system is AES while MACs are computed using HMAC-SHA-256. We note that ECIES has the potential to make the linking of user identities impossible since each component of the ECIES ciphertext is drawn randomly in each application. However, recently, new attacks have been published [17] that show

<sup>11</sup>[https://www.etsi.org/deliver/etsi\\_ts/133500\\_133599/133501/15.04.00\\_60/ts\\_133501v150400p.pdf](https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/15.04.00_60/ts_133501v150400p.pdf)

<sup>12</sup>For an overview see <https://www.secg.org/sec1-v2.pdf>.

## D2.1 Post-quantum Analysis Report

that linking of certain ciphertext can still be achieved via attacks on the underlying symmetric primitives in 5G (called AKE encryption). With such attacks, the question of whether user  $u$  is online can be successfully answered.

### | 2.2 Symmetric Cryptography

The 5G technology uses proprietary symmetric algorithms as well as standardized ones like AES, HMAC or, SHA-256. In general, all these algorithms need to be revised in 6G. A simple adaption to the threat of quantum computers is to increase the key size to account for Grover's attack if the specifications of keyed algorithms allow it. For some applications, where the standard does not allow to increase key sizes further, new standards would be required. However, one needs to be careful to exactly analyze the attack scenario in practice. As a consequence, the security overhead of a post-quantum secure 6G will generally be higher than that of 5G. This has to be accounted for when designing features of 6G and 6G-based applications.

### 3 Migration to PQC and Challenges

There are several constraints under which migration should proceed. In Confidential6G must substitute any asymmetric cryptography used in 5G with algorithms that resist attacks by quantum computers, where algorithms to achieve confidentiality have the highest urgency or replacement. Since TLS and IPsec are general-purpose protocols we expect, after the NIST standards for Kyber, Dilithium, and SPHINCS+ have been finalized, that the working groups of TLS and IPsec will publish new standards of these corresponding protocols as well. Ideally, 6G should rely on the prospective new standards of TLS and IPsec. However, it is unclear how long the standardization processes will take. Thus we believe a two-pronged strategy is best suited in this case. 1) Investigate dedicated variants of TLS and IPsec that feature post-quantum security, like for example hybrid versions of TLS or IPsec. This ensures that sufficient schemes are readily available for 6G soon. 2) Use the insights from 1) to influence the (public) standardization process of TLS and IPsec. In case the standardization TLS and IPsec is finished before 6G is published, use these new standardized protocols.

The identity concealment mechanism is more specific to 5G/6G applications. Here a dedicated solution could be useful overall. However similar to the situation now, it would be beneficial to use a standardized algorithm for identity concealment as well. A promising candidate is of course Kyber. However, the ciphertext sizes are much longer than that of ECIES and it needs to be investigated further and more concretely whether the prospective transition to Kyber is feasible in practice. We note that exchanging KEMs is generally of higher priority than signature schemes. This is due to a store-now-decrypt-later threat scenario where encrypted messages of today will be stored and decrypted only when a quantum computer becomes available. The Snowden revelations have shown that the required massive storage capacity for such an endeavor is readily available to certain actors. Signatures on the other hand, and in particular digital certificates, should generally have a short validity period. This allows for a relatively flexible change.

#### | 3.1 Strategies for Smooth Transitions

It would be beneficial to rely on post-quantum secure building blocks that respect existing infrastructure and, for example, do not pass the maximum transmission unit (MTU) of IPsec packets in the first two moves (IPsec does not support fragmentation in these moves, which in KEM terminology send an ephemeral public key in the first move and a ciphertext in the second move). Although this would ease the transition to post-quantum secure building blocks, we cannot rely on such hopes. To the best of our knowledge, all post-quantum secure KEMs proposed so far violate the MTU of IPsec on the highest security level (NIST V). However, for lower security levels (NIST I and NIST III) the Kyber public key and ciphertexts are small enough to each fit the MTU and leave enough space to also fit an elliptic-curve key/ciphertext. Another option is to layer pre-quantum and post-quantum systems, keeping the initial handshake with elliptic curves and then sending the post-quantum keys and ciphertexts as part of the payload. These variants are often referred to as hybrid cryptography. For IPsec [55, 56] and TLS [54] Furthermore, the future standardization of protocols like IPsec will possibly change the MTU or fragmentation scheme to better fit post-quantum secure primitives. As stated before, active engagement in the standardization process will help to guide the standard development in a direction that is particularly useful for 6G.

Situations in which the public key is fixed, such as in identity concealment, might call for other post-quantum systems, such as Classic McEliece [3], which offer a different profile with much smaller ciphertexts and significantly larger public keys as those public keys would only be updated when the phone enters roaming.

When transitioning to post-quantum secure cryptography we will in general try to exploit

## D2.1 Post-quantum Analysis Report

synergies as much as possible. This in particular holds in use cases that require the development of new building blocks. In these cases, we have more design freedom. In particular, we will design primitives to fit the needs of Confidential6G and 6G in more general. As an example, we will try to maximize code re-use to obtain small implementations that can also be executed on devices with small resources.

## References

- [1] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle diffie-hellman assumptions and an analysis of DHIES. In David Naccache, editor, Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings, volume 2020 of Lecture Notes in Computer Science, pages 143–158. Springer, 2001.
- [2] Ian F. Akyildiz and Ahan Kak. The internet of space things/cubesats: A ubiquitous cyber-physical system for the connected world. *Comput. Networks*, 150:134–149, 2019.
- [3] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece. Technical report, National Institute of Standards and Technology, 2022. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions>.
- [4] Martin R. Albrecht, Alex Davidson, Amit Deo, and Daniel Gardham. Crypto dark matter on the torus: Oblivious PRFs from shallow PRFs and FHE. *Cryptology ePrint Archive*, Report 2023/232, 2023. <https://eprint.iacr.org/2023/232>.
- [5] Martin R. Albrecht, Alex Davidson, Amit Deo, and Nigel P. Smart. Round-optimal verifiable oblivious pseudorandom functions from ideal lattices. pages 261–289, 2021.
- [6] Gustavo Banegas and Daniel J. Bernstein. Low-communication parallel quantum multi-target preimage search. In Carlisle Adams and Jan Camenisch, editors, *Selected Areas in Cryptography - SAC 2017 - 24th International Conference*, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers, volume 10719 of *Lecture Notes in Computer Science*, pages 325–335. Springer, 2017.
- [7] R. Barnes, K. Bhargavan, B. Lipp, and C. Wood. Hybrid Public Key Encryption. RFC 9180, IETF, February 2022.
- [8] Andrea Basso. A post-quantum round-optimal oblivious PRF from isogenies. *SAC Selected Areas in Cryptography*, 2023.
- [9] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. pages 614–629, 2003.
- [10] Daniel J. Bernstein and Tanja Lange. Post-quantum cryptography—dealing with the fallout of physics success. *Cryptology ePrint Archive*, Paper 2017/314, 2017. <https://eprint.iacr.org/2017/314>.
- [11] Ward Beullens, Jan-Pieter D’Anvers, Andreas Hülsing, Tanja Lange, Lorenz Panny, Cyprien de Saint Guilhem, Nigel P. Smart, Evangelos Rekleitis, Angeliki Aktypi, and Athanasios-Vasileios Grammatopoulos. Post-quantum cryptography: Current state and quantum mitigation, 2021. <https://www.enisa.europa.eu/publications/post-quantum-cryptography-current-state-and-quantum-mitigation/@download/fullReport>.
- [12] Dan Boneh, Dmitry Kogan, and Katharine Woo. Oblivious pseudorandom functions from isogenies. pages 520–550, 2020.

- [13] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 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, pages 353–367. IEEE, 2018.
- [14] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. pages 132–145, 2004.
- [15] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. pages 93–118, 2001.
- [16] Jung Hee Cheon, Hyeongmin Choe, Dongyeon Hong, and MinJune Yi. Smaug: Pushing lattice-based key encapsulation mechanisms to the limits. Cryptology ePrint Archive, Paper 2023/739, 2023. <https://eprint.iacr.org/2023/739>.
- [17] Merlin Chlosta, David Rupprecht, Christina Pöpper, and Thorsten Holz. 5g suci-catchers: still catching them all? In Christina Pöpper, Mathy Vanhoef, Lejla Batina, and René Mayrhofer, editors, WiSec '21: 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks, Abu Dhabi, United Arab Emirates, 28 June - 2 July, 2021, pages 359–364. ACM, 2021.
- [18] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, IETF, August 2008.
- [19] W. Diffie and M. Hellman. New directions in cryptography. IEEE Transactions on Information Theory, 22(6):644–654, 1976.
- [20] Itai Dinur, Steven Goldfeder, Tzipora Halevi, Yuval Ishai, Mahimna Kelkar, Vivek Sharma, and Greg Zaverucha. MPC-friendly symmetric cryptography from alternating moduli: Candidates, protocols, and applications. pages 517–547, 2021.
- [21] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. pages 10–18, 1984.
- [22] P. Eronen and H. Tschofenig. Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279, IETF, December 2005.
- [23] Sebastian H. Faller, Astrid Ottenhues, and Johannes Ottenhues. Composable oblivious pseudo-random functions via garbled circuits. In Abdelrahman Aly and Mehdi Tibouchi, editors, Progress in Cryptology - LATINCRYPT 2023 - 8th International Conference on Cryptology and Information Security in Latin America, LATINCRYPT 2023, Quito, Ecuador, October 3-6, 2023, Proceedings, volume 14168 of Lecture Notes in Computer Science, pages 249–270. Springer, 2023.
- [24] Jean Charles Faugère. A new efficient algorithm for computing gröbner bases without reduction to zero (f5). In Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, ISSAC '02, page 75–83, New York, NY, USA, 2002. Association for Computing Machinery.
- [25] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. pages 186–194, 1987.
- [26] S. Fluhrer, P. Kampanakis, D. McGrew, and V. Smylov. Mixing Preshared Keys in the Internet Key Exchange Protocol Version 2 (IKEv2) for Post-quantum Security. RFC 8784, IETF, June 2020.



- [27] S. Frankel and S. Krishnan. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. RFC 6071, IETF, February 2011.
- [28] Joel Gärtner. Concrete security from worst-case to average-case lattice reductions. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, Progress in Cryptology - AFRICACRYPT 2023 - 14th International Conference on Cryptology in Africa, Sousse, Tunisia, July 19-21, 2023, Proceedings, volume 14064 of Lecture Notes in Computer Science, pages 344–369. Springer, 2023.
- [29] Stefan-Lukas Gazdag, Sophia Grundner-Culemann, Tobias Guggemos, Tobias Heider, and Daniel Loebenberger. A formal analysis of ikev2's post-quantum extension. In Proceedings of the 37th Annual Computer Security Applications Conference, ACSAC '21, page 91–105, New York, NY, USA, 2021. Association for Computing Machinery.
- [30] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. Applying grover's algorithm to AES: Quantum resource estimates. pages 29–43, 2016.
- [31] Lov K. Grover. A fast quantum mechanical algorithm for database search. pages 212–219, 1996.
- [32] Lena Heimberger, Tobias Hennerbichler, Fredrik Meisingseth, Sebastian Ramacher, and Christian Rechberger. Oprfs from isogenies: Designs and analysis. Cryptology ePrint Archive, Paper 2023/639, 2023. <https://eprint.iacr.org/2023/639>.
- [33] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe Buhler, editor, Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings, volume 1423 of Lecture Notes in Computer Science, pages 267–288. Springer, 1998.
- [34] Andreas Hülsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS+. Technical report, National Institute of Standards and Technology, 2022. available at <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>.
- [35] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. pages 44–61, 1989.
- [36] Qun Liu, Bart Preneel, Zheng Zhao, and Meiqin Wang. Improved quantum circuits for AES: Reducing the depth and the number of qubits. Cryptology ePrint Archive, Paper 2023/1417, 2023. <https://eprint.iacr.org/2023/1417>.
- [37] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. pages 598–616, 2009.
- [38] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>.
- [39] Alexander May. How to meet ternary LWE keys. pages 701–731, 2021.
- [40] R. J. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. Deep Space Network Progress Report, 44:114–116, January 1978.



- [41] Ralph C. Merkle. A digital signature based on a conventional encryption function. pages 369–378, 1988.
- [42] Ralph C. Merkle. A certified digital signature. pages 218–238, 1990.
- [43] Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, 2016. <https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf>.
- [44] Seunghwan Park, Chi-Gon Jung, Aesun Park, Joongeun Choi, and Honggoo Kang. Tiger: Tiny bandwidth key encapsulation mechanism for easy migration based on rlwe(r). Cryptology ePrint Archive, Paper 2022/1651, 2022. <https://eprint.iacr.org/2022/1651>.
- [45] Jacques Patarin. Hidden fields equations (HFE) and isomorphisms of polynomials (IP): Two new families of asymmetric algorithms. pages 33–48, 1996.
- [46] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022. available at <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>.
- [47] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, IETF, August 2018.
- [48] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. pages 552–565, 2001.
- [49] Martin Roetteler, Michael Naehrig, Krysta M. Svore, and Kristin E. Lauter. Quantum resource estimates for computing elliptic curve discrete logarithms. pages 241–270, 2017.
- [50] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. pages 239–252, 1990.
- [51] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>.
- [52] István András Seres, Máté Horváth, and Péter Burcs. The legendre pseudorandom function as a multivariate quadratic cryptosystem: security and applications. In AAECC. Springer, 01 2023.
- [53] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. pages 124–134, 1994.
- [54] Douglas Stebila, Scott Fluhrer, and Shay Gueron. Hybrid key exchange in tls 1.3. Internet-Draft draft-ietf-tls-hybrid-design-09, IETF Secretariat, September 2023.
- [55] C. Tjhai, M. Tomlinson, G. Bartlett, Scott Fluhrer, Daniel Van Geest, Oscar Garcia-Morchon, and Valery Smyslov. Multiple key exchanges in ikev2. Internet-Draft draft-ietf-ipsecme-ikev2-multiple-ke-12, IETF Secretariat, December 2022.
- [56] C. Tjhai, M. Tomlinson, grbartle@cisco.com, Scott Fluhrer, Daniel Van Geest, Oscar Garcia-Morchon, and Valery Smyslov. Framework to integrate post-quantum key exchanges into internet key exchange protocol version 2 (ikev2). Internet-Draft draft-tjhai-ipsecme-hybrid-qske-ikev2-04, IETF Secretariat, July 2019.

## D2.1 Post-quantum Analysis Report

- [57] Nirvan Tyagi, Sofia Celi, Thomas Ristenpart, Nick Sullivan, Stefano Tessaro, and Christopher A. Wood. A fast and simple partially oblivious PRF, with applications. pages 674–705, 2022.