ETSI TR 103 616 V1.1.1 (2021-09)

CYBER;
Quantum-Safe Signatures

Reference
DTR/CYBER-QSC-008

Keywords
algorithm, cybersecurity

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM®** and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Technical Report (TR) has been produced by ETSI Technical Committee Cyber Security (CYBER).

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document provides technical descriptions of the digital signature schemes submitted to the National Institute of Standards and Technology (NIST) for the third round of their post-quantum cryptography standardization process.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] NIST FIPS 197: "Advanced Encryption Standard (AES)".

[i.2] NIST FIPS 180-4: "Secure Hash Standard".

[i.3] NIST FIPS 202: "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions".

[i.4] NIST IR 8105: "Report on Post-Quantum Cryptography".

[i.5] NIST FIPS 186-4: "Digital Signature Standard (DSS)".

[i.6] NIST SP-56A: "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography".

[i.7] NIST SP-56B: "Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography".

[i.8] NIST: "Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process", December 2016.

NOTE: Available at https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[i.9] NIST: "Post-Quantum Cryptography Standardization: Round 1 Submissions".

NOTE: Available at https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions.

[i.10] NIST IR 8240: "Status Report on the First Round of the NIST Post-Quantum Standardization Process".

[i.11] NIST: "Post-Quantum Cryptography Standardization: Round 2 Submissions".

NOTE: Available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-2-submissions.

[i.12] NIST IR 8309: "Status Report on the Second Round of the NIST Post-Quantum Standardization Process".

[i.13] NIST: "Post-Quantum Cryptography Standardization: Round 3 Submissions".

NOTE: Available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions.

[i.14] L. Lamport: "Constructing digital signatures from a one way function". Technical Report SRI-CSL-98. SRI International Computer Science Laboratory. 1979.

[i.15] R. Merkle: "A Certified Digital Signature". CRYPTO "89, LNCS, Vol. 263. Springer, pages 218-238, 1989.

[i.16] A. Fiat and A. Shamir: "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". CRYPTO 86, LNCS, Vol. 435. Springer, pages 186-194, 1986.

[i.17] ETSI GR QSC 001: "Quantum-Safe Cryptography (QSC); Quantum-Safe Algorithmic Framework".

NOTE: Available at https://www.etsi.org/deliver/etsi_gr/QSC/001_099/001/01.01.01_60/gr_QSC001v010101p.pdf.

[i.18] V. Lyubashevsky, L. Ducas, E. Kiltz, T. Lepoint, P. Schwabe, G. Seiler and D. Stehlé: "CRYSTALS-DILITHIUM: Algorithm Specifications and Supporting Documentation". NIST round 3 post-quantum submission.

[i.19] V. Lyubashevsky: "Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures". Asiacrypt 2009, LNCS, Vol. 5912. Springer, pages 598-616.

[i.20] T. Prest, P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. White and Z. Zhang: "FALCON: Fast-Fourier Lattice-based Compact Signatures over NTRU". NIST round 3 post-quantum submission.

[i.21] C. Gentry, C. Peikert, and V. Vaikuntanathan: "Trapdoors for Hard Lattices and New Cryptographic Constructions". STOC 2008, ACM, pages 197-206.

[i.22] J. Hoffstein, J. Pipher and J. H. Silverman: "NTRU: A Ring-Based Public Key Cryptosystem", ANTS-III, LNCS, Vol. 1423. Springer, pages 267-288, 1988.

[i.23] D. Stehlé and R. Steinfeld: "Making NTRU as Secure as Worst-Case Problems over Ideal Lattices". EUROCRYPT 2011, LNCS, Vol. 6632, pages 27-47.

NOTE: Available at https://eprint.iacr.org/2019/893.

[i.24] J. Ding, M.-S. Chen, A. Petzoldt, D. Schmidt and B.-Y. Yang: "Rainbow: Algorithm Specification and Documentation". NIST round 3 post-quantum submission.

[i.25] A. Kipnis, J. Patarin and L. Goubin: "Unbalanced Oil and Vinegar Signature Schemes". EUROCRYPT 1999, LNCS, Vol. 1592. Springer, pages 206-222.

[i.26] A. Petzoldt, S. Bulygin, and J. Buchmann: "CyclicRainbow - a Multivariate Signature Scheme with a Partially Cyclic Public Key". INDOCRYPT 2010, LNCS, vol. 6498, pages 33 - 48. .

[i.27] A. Casanova, J.-C. Faugère, G. Macario-Rat, J. Patarin, L. Perret and J. Ryckeghem: "GeMSS: A Great Multivariate Signature Scheme". NIST round 3 post-quantum submission.

[i.28] T. Matsumoto and H. Imai: "Public Quadratic Polynomial-Tuples for Efficient Signature-Verification and Message-Encryption". EUROCRYPT 1988, LNCS, Vol. 330. Springer, pages 419-453.

[i.29] J. Patarin: "Hidden Field Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithm". EUROCRYPT 1996, LNCS, Vol. 1070. Springer, pages 33-48.

[i.30] J. von zur Gathen and J. Gerhard: "Modern Computer Algebra (3. Ed.)". Cambridge University Press 2013.

[i.31] J.-C Faugère, L. Perret and J. Ryckeghem: "Software Toolkit for HFE-based Multivariate Schemes". IACR Trans. Cryptogr. Hardw. Embed. Syst. 2019, Vol. 3., pages 257-304.

[i.32] G. Zaverucha, M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger and D. Slamanig: "The Picnic Algorithm Signature Specification". NIST round 3 post-quantum submission.

[i.33] I. Giacomelli, J. Madsen and C. Orlandi: "ZKBoo : Faster Zero-Knowledge for Boolean Circuits". USENIX Security 2016, USENIX Association, pages 1069-1083.

[i.34] J. Katz, V. Kolesnikov and X. Wang: "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures". ACM CCS 2018, ACM, pages 525-537.

[i.35] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen and M. Zohner: "Ciphers for MPC and FHE". EUROCRYPT 2015. LNCS, Vol. 9056. Springer, pages 430-454.

[i.36] D. Unruh: "Non-Interactive Zero-Knowledge Proofs in the Quantum Random Oracle Model". EUROCRYPT 2015. LNCS, Vol. 9056. Springer, pages 430-454.

[i.37] A. Hulsing, D. J. Bernstein, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, P. Kampanakis, S. Kolbl, T. Lange, M. M Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, P. Schwabe and J.-P. Aumasson: "SPHINCS+: Submission to the NIST Post-Quantum Project". NIST round 3 post-quantum submission.

[i.38] S. Kölbl, M. Lauridsen, F. Mendel and C. Rechberger: "Haraka v2 - Efficient Short-Input Hashing for Post-Quantum Applications". IACR Trans. Symmetric Cryptol., volume 2016, number 2, pages 1-29, 2017.

[i.39] W. Beullens: "Improved Cryptanalysis of UOV and Rainbow". EUROCRYPT 2021, LNCS, Vol. 12696. Springer, 348-373. 2021.

[i.40] C. Tao, A. Petzoldt and J. Ding: "Improved Key Recovery of the HFEv- Signature Scheme". Cryptology ePrint Archive: Report 2020/1424.

NOTE: Available at https://eprint.iacr.org/2020/1424.

[i.41] J.O. Shallit, G.S. Frandsen, and J.F. Buss: "The Computational Complexity of some Problems of Linear Algebra". BRICS series report, Aarhus, Denmark, RS-96-33, 1996.

[i.42] M. Øygarden and D. Smith-Tone and J. Verbel: "On the Effect of Projection on Rank Attacks in Multivariate Cryptography". Cryptology ePrint Archive: Report 2021/655.

NOTE: Available at https://eprint.iacr.org/2021/655.

[i.43] I. Dinur: "Cryptanalytic Applications of the Polynomial Method for Solving Multivariate Equation Systems over GF(2)". EUROCRYPT 2021, LNCS, Vol. 12696. Springer, 374-403.

[i.44] The Rainbow Team: "Response to Recent Paper by Ward Beullens", NIST PQC Forum, December 2020.

NOTE: Available at http://precision.moscito.org/by-publ/recent/response-ward.pdf.

[i.45] ETSI TR 103 823: "CYBER; Quantum-Safe Public-Key Encryption and Key Encapsulation".

3 Definition of terms, symbols and abbreviations

3.1 Terms

Void.

3.2 Symbols

For the purposes of the present document, the following symbols apply:

$x \coloneqq y$	Variable x is assigned the value of y
$x = y$	The values of x and y are equal
$x \neq y$	The values of x and y are not equal
$x \parallel y$	The concatenation of x and y
$\mathbb{F}$	A finite field
$\mathbb{F}_q$	A finite field modulo q
$\mathbb{Z}$	The ring of integers
$\mathbb{Z}_q$	The ring of integers modulo q
R	A ring of polynomials
R_q	A ring of polynomials modulo q
$R_q^{k \times k}$	The set of $k \times k$ matrices with coefficients in R_q
R_q^k	The set of $1 \times k$ matrices with coefficients in R_q
B_η	Centered binomial distribution of width η
$\mathrm{GL}_{n \times n}(\mathbb{F}_q)$	The set of $n \times n$ invertible matrices whose coefficients are over $\mathbb{F}_q$

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AES	Advanced Encryption Standard
CMA	Chosen Message Attack
CPU	Central Processing Unit
CTR	CounTeR
EUF	Existential Unforgeability
FFT	Fast Fourier Transform
FIPS	Federal Information Processing Standards
FORS	Forest Of Random Subsets
FS	Fiat-Shamir
GPV	Gentry-Peikert-Vaikuntanathan
GR	Group Report
HFE	Hidden Field Equation
IDS	Identification Scheme
KEM	Key Encapsulation Mechanism
KMA	Known Message Attack
KOA	Key Only Attack
MLWE	Module Learning With Errors
MPC	Multi-Party Computation
MQ	Multivariate Quadratic
NIST	National Institute of Standards and Technology
NTT	Number Theoretic Transform
OTS	One-Time Signature
PKE	Public-Key Encryption
PoSSo	Polynomial System Solving
PQC	Post-Quantum Cryptography
PRF	Pseudo Random Function
QROM	Quantum Random Oracle Model
QSC	Quantum-Safe Cryptography
ROM	Random Oracle Model
SHA	Secure Hash Algorithm
SHAKE	Secure Hash Algorithm and KECCAK
SIS	Short Integer Solution
SUF	Strong existential Unforgeability
UOV	Unbalanced Oil and Vinegar
UUF	Universal Unforgeability
WOTS	Winternitz One-Time Signature
XOF	eXtendable Output Function

XOR	eXclusive OR
ZK	Zero-Knowledge

4 Introduction

The National Institute of Standards and Technology (NIST), an agency of the U.S. Department of Commerce, is responsible for producing cryptographic standards for the protection of sensitive U.S. Federal Government information. NIST standards, such as the Advanced Encryption Standard (AES) [i.1] and Secure Hash Algorithm (SHA) standards [i.2] and [i.3], are used globally in many different protocols and products.

In April 2016, NIST announced their intention [i.4] to augment their existing portfolio of public-key cryptography standards [i.5], [i.6] and [i.7] by developing new standards for post-quantum cryptography. In December 2016, they initiated the so-called NIST Post-Quantum Cryptography (PQC) standardization process; a competition-like process with a call for proposals [i.8] for digital signatures, Public Key Encryption (PKE) schemes, and Key Encapsulation Mechanisms (KEMs), that will remain secure even in the presence of a cryptographically relevant quantum computer. The goal of the process is to perform several rounds of public evaluation over a three- to five-year period, and select one or more acceptable algorithms for standardization based on that evaluation.

NIST's deadline for submissions was November 2017. They received 69 candidates that met the minimum acceptance criteria and submission requirements: 20 digital signature schemes and 49 PKE/KEMs. Five submissions were quickly broken and formally withdrawn from the process by their designers. This left a total of 64 first round candidates [i.9]. In January 2019, NIST announced [i.10] that 26 of the first round candidates would progress to the second round of evaluation: 9 digital signature schemes and 17 PKE/KEMs [i.11].

In July 2020, NIST announced [i.12] that 15 candidate algorithms would progress to the third round of evaluation. These were split into seven finalists and eight alternate candidates. NIST described the finalists as the algorithms they consider to be the most promising for the majority of use cases, and the most likely to be ready for standardization soon after the end of the third round. The seven finalists [i.13] included three digital signature schemes and four PKE/KEMs. The alternate candidates were described as having potential for future standardization, but most likely after a fourth round of evaluation. The eight alternate candidates included three digital signature schemes and five PKE/KEMs.

In June 2021, NIST declared that the third round will be finalized by the beginning of 2022. Following recent attacks against multivariate schemes [i.39] and [i.40], NIST also announced that they were considering selecting an alternate signature for standardization at the end of third round and issuing a call for new digital signature submissions in 2022.

The purpose of the present document is to give concise descriptions of the six signature schemes remaining in the third round of NIST's standardization process. ETSI TR 103 823 [i.45] provides similar descriptions of the nine remaining PKE/KEMs.

The three digital signature finalists are:

- **Dilithium** (see clause 6.1)
- **FALCON** (see clause 6.2)
- **Rainbow** (see clause 6.3)

The three digital signature alternate candidates are:

- **GeMSS** (see clause 7.1)
- **Picnic** (see clause 7.2)
- **SPHINCS+** (see clause 7.3)

Each of these schemes has a different profile in terms of security properties and performance characteristics, so it is expected that some of these schemes will be more suited to specific deployment scenarios than others.

The descriptions provided in the present document are not intended to be substitutes for the detailed specifications submitted to NIST. Instead, the emphasis is on clear mathematical descriptions that facilitate easy comparison of the different schemes. Implementation details, such as how to encode polynomials as bit-strings, have been omitted wherever possible. As such, some of the descriptions differ from the submitted specifications, in terms of level of abstraction, use of notation, and choice of variable names.

It is expected that details of some of the schemes, such as specific parameter choices, will change during the third round of evaluation, so for consistency the descriptions are based on the official submission packages provided to NIST at the beginning of the third round [i.13].

5 Background

5.1 Terminology

A digital signature scheme consists of a triple of algorithms:

- **Key generation (KeyGen).** Outputs a new public and private key pair.
- **Sign.** Takes a private key and message as input and outputs a signature.
- **Verify.** Takes a public key, a message and a signature as input and outputs either 'accept' or 'reject'.

5.2 Families of post-quantum algorithms

The cryptosystems that have progressed to the third round of the NIST process can be classified into the following families:

- **Code-based schemes.** The security of code-based schemes depends on the difficulty of decoding vectors to find the closest codeword or shortest error vector. Code-based cryptography lends itself more naturally to the construction of PKE schemes and KEMs than to digital signature algorithms.
- **Isogeny-based schemes.** The security of isogeny-based schemes depends on the difficulty of recovering a secret isogeny between a pair of elliptic curves. Isogeny-based cryptography lends itself more naturally to the construction of PKE schemes and KEMs than to digital signatures, though there has been some progress in this area.
- **Lattice-based schemes.** The security of lattice-based schemes depends on the difficulty of finding vectors in a lattice that are relatively short, or relatively close to some target vector. Lattice-based signature schemes generally fall into two categories: NTRU-style [i.22] schemes, such as FALCON, which use lattices that have been specifically constructed to contain private short vectors, and Module Learning With Errors (MLWE) schemes such as Dilithium which use particular classes of random lattices. In many cases lattice-based schemes admit worst-case to average-case security reductions, though these reductions are often not relevant to proposed parameter sets (see ETSI TR 103 823 [i.45]).
- **Multivariate schemes.** The security of multivariate schemes depends on the difficulty of solving systems of quadratic or higher degree multivariate polynomials (PoSSo problem, also known as the MQ problem for quadratic equations). Multivariate cryptography lends itself more naturally to the construction of digital signatures than to PKE schemes or KEMs. Rainbow and G*e*MSS are multivariate-based signature schemes.
- **Symmetric schemes.** The security of such schemes depends on the security of symmetric cryptographic primitives such as hash functions and block ciphers. Symmetric cryptography only lends itself to the construction of digital signatures. Examples include SPHINCS+ and Picnic.

5.3 Security categories

NIST have provided guidance on the evaluation criteria they intend to apply to candidate submissions [i.8]. As part of this guidance they have defined the following security categories in terms of the (classical or quantum) resources required to attack different NIST-approved symmetric primitives:

- **Category 1.** Resources equivalent to or greater than key recovery for AES-128.
- **Category 2.** Resources equivalent to or greater than collision search for SHA3-256.
- **Category 3.** Resources equivalent to or greater than key recovery for AES-192.

- **Category 4.** Resources equivalent to or greater than collision search for SHA3-384.
- **Category 5.** Resources equivalent to or greater than key recovery for AES-256.

NIST recommended that submissions include parameter sets that meet the requirements for categories 1, 2 and/or 3, as they believe that these categories will provide sufficient security for the foreseeable future. However, to demonstrate flexibility, and to protect against future cryptanalytic breakthroughs, NIST also recommended that submissions include at least one parameter set that provides a substantially higher level of security. Submitters were asked to include justifications for the security categories claimed for their proposed parameter sets.

5.4 Security properties

Digital signatures are typically intended to provide authentication, integrity and non-repudiation of data. The main security goal that is relevant for signatures is existential unforgeability under chosen message attack (see annex A for further discussions on security goals). This is usually modelled as a game:

- **Existential Unforgeability under Chosen Message Attack (EUF-CMA) for signatures.** The attacker can request valid signatures of messages of their choice. The attacker's goal is to exhibit a valid signature for any message not previously queried. The scheme is EUF-CMA secure it the attacker cannot do this using less resources than the security level.

To construct some proofs of security it is necessary to make assumptions about or use idealized versions of certain cryptographic primitives; this can mean the proof does not apply to a concrete implementation. In particular, in the Random Oracle Model (ROM) hash functions are modelled as ideal entities, referred to as random oracles, which respond to new queries with answers selected uniformly at random from the output domain, and respond to previously seen queries with the answer that was given the first time the query was received.

In the ROM it is assumed that adversaries interact classically with random oracles, but in the Quantum Random Oracle Model (QROM) it is assumed that adversaries can query a random oracle in a quantum superposition of states. Although the QROM affords an adversary more computational power, it can be hard to compare proofs in the ROM to proofs in the QROM, particularly if it is possible to construct a tight proof in the ROM but not the QROM.

NIST have stated that they intend to standardize at least one EUF-CMA signature scheme. It is further assumed that an attacker has access to no more than 2^{64} chosen signed messages (though attacks requiring more messages will be taken into consideration). NIST place more emphasis on the ROM model rather than QROM. NIST has not required proof of EUF-CMA security as part of a submission, but does give consideration to such proofs.

5.5 Frameworks for constructing digital signatures

The digital signature algorithms that have progressed to the third round of the NIST process can be classified by family: lattice-based, multivariate-based or symmetric (see clause 5.2). Another way to categorize these schemes is to consider the framework used to construct these primitives (see also Table 1):

- **Hash-and-sign**. These schemes are constructed from trapdoor one-way functions. FALCON [i.20], G*e*MSS [i.27] and Rainbow [i.24] are examples of schemes within this framework.
- **Hash-based**. These schemes follow the work of Lamport [i.14] and Merkle [i.15] and construct a signature from a hash function. SPHINCS+ [i.37] is an example of a scheme within this framework.
- **Fiat-Shamir**. These schemes are constructed by using the Fiat-Shamir transform [i.16] together with a post-quantum Identification Scheme (IDS). Dilithium [i.18] and Picnic [i.32] are examples of schemes within this framework.

Table 1: Categorization of digital signature schemes based on their underlying hard problems and design frameworks

	Lattice-based	Multivariate-based	Symmetric-based
Hash-and-sign	FALCON	GeMSS Rainbow	
Hash-based			SPHINCS+
Fiat-Shamir	Dilithium		Picnic

Annex B provides further discussions on the categorization sketched in the present clause.

5.6 Finalists and alternate candidates at a glance

Table 2 contains a summary of each of the NIST digital signature finalists.

Table 2: Summary of finalists

Scheme	Family	Type	Structure	Categories 1	2	3	4	5	Security EUF-CMA	Comments
DILITHIUM	Lattice	Fiat-Shamir	Module		Y	Y		Y	Y	
FALCON	Lattice	Hash-and-sign	Ring	Y				Y	Y	
Rainbow	Multivariate	Hash-and-sign	UOV	Y		Y		Y	Y	Note
NOTE: A new result [i.39] challenges the claimed security for Rainbow (see clause C.2 for a short discussion).										

Table 3 contains a summary of each of the NIST digital signature alternates.

Table 3: Summary of alternate candidates

Scheme	Family	Type	Structure	Categories 1	2	3	4	5	Security EUF-CMA	Comments
GeMSS	Multivariate	Hash-and-sign	HFE	Y		Y		Y	Y	Note 1
Picnic	Symmetric	Fiat-Shamir	Block cipher	Y		Y		Y	Y	Note 2
SPHINCS+	Symmetric	Stateless hash-based	Hash function	Y		Y		Y	Y	
NOTE 1: A new result [i.40] challenges the claimed security of GeMSS (see clause C.2 for a short discussion). NOTE 2: Clause C.3 discusses the impact of a new result [i.43] on the security of Picnic.										

6 Finalists

6.1 Dilithium

6.1.1 Introduction

CRYSTALS is a package that includes KYBER**,** a Key Encapsulation Mechanism [i.45] and the digital signature scheme Dilithium [i.18]. Dilithium uses the Fiat-Shamir with abort [i.19] framework to derive a signature scheme from a compact lattice-based IDS (clause B.3).

The security of Dilithium is based on the hardness of MLWE. Let R_q denote the polynomial ring $\mathbb{Z}_q[x]/(x^n+1)$ for a power-of-two n and prime q. A MLWE sample is a pair of the form $(\boldsymbol{A}, \boldsymbol{As}+\boldsymbol{e})$, where $\boldsymbol{A} \in R_q^{k\times\ell}$ is a public matrix consisting of polynomials whose coefficients are sampled uniformly at random from $\mathbb{Z}_q$, and $\boldsymbol{s} \in R_q^{\ell}$, $\boldsymbol{e} \in R_q^{k}$ are private vectors of polynomials whose coefficients are sampled from a small distribution over $\mathbb{Z}_q$. The MLWE problem asserts that it is computationally hard to distinguish MLWE samples of the form $(\boldsymbol{A}, \boldsymbol{As}+\boldsymbol{e})$ from pairs of the form $(\boldsymbol{A}, \boldsymbol{u})$, where $\boldsymbol{u} \in R_q^k$ is a vector consisting of polynomials sampled uniformly at random from R_q.

6.1.2 Public parameters

The main parameters for Dilithium are:

- n, the dimension of the polynomial ring R_q;
- q, the modulus of the polynomial ring R_q;
- k and ℓ, ranks of the vectors over R_q;
- η, the bound on the size of the coefficients for key generation;

- γ_1 and γ_2, bounds on the sizes of the coefficients for signing; and
- β, the reduction in the bounds for signing and verification.

6.1.3 Auxiliary primitives

Dilithium makes use of several auxiliary, symmetric primitives:

- H, a 256-bit cryptographic hash function;
- PRF, a pseudorandom function; and
- XOF, an extendable output function.

The instantiations of these primitives from the specification are described in Table 4.

Table 4: Auxiliary symmetric primitives for Dilithium

Primitive	SHAKE variant	AES variant
H	SHAKE-256	
PRF	SHAKE-256	
XOF	SHAKE-128	AES-256 in CTR mode

The AES-based XOF is included to improve performance on platforms that provide hardware support for AES.

Each $\boldsymbol{w} \in R_q^k$ can be decomposed as:

$$\boldsymbol{w} \coloneqq \boldsymbol{w}_1 \cdot 2\gamma_2 + \boldsymbol{w}_0$$

where the coefficients of $\boldsymbol{w}_0$ lie in $\{-\gamma_2, \ldots, \gamma_2\}$. Dilithium defines the functions $\mathrm{HighBits}(\boldsymbol{w}, 2\gamma_2) \coloneqq \boldsymbol{w}_1$ and $\mathrm{LowBits}(\boldsymbol{w}, 2\gamma_2) \coloneqq \boldsymbol{w}_0$.

Dilithium also makes use of a hash function $\mathcal{H} : \{0,1\}^* \rightarrow \mathcal{B}_h$ that takes arbitrary bit strings as input and returns an element of $\mathcal{B}_h$, the set of polynomials in R_q that have h coefficients that are either -1 or 1, and the rest are 0. This is constructed from H and the XOF (see [i.18] for a precise description).

6.1.4 Dilithium.KeyGen

Input: Security level

Output: Public key $\mathrm{pk} \in \{0,1\}^{256} \times R_q^k$
Private key $\mathrm{sk} \in \{0,1\}^{256} \times R_q^k \times R_q^\ell \times R_q^k$

1) Use PRF to sample two vectors $\boldsymbol{s_1} \in R_q^\ell$ and $\boldsymbol{s_2} \in R_q^k$ with coefficients of size at most η.
2) Choose a uniformly random seed $\rho \in \{0,1\}^{256}$.
3) Expand the seed ρ using XOF to produce the public matrix $\boldsymbol{A} \in R_q^{k \times \ell}$.
4) Compute $\boldsymbol{t} \coloneqq \boldsymbol{A}\boldsymbol{s_1} + \boldsymbol{s_2} \in R_q^k$.

The public key is $\mathrm{pk} \coloneqq (\rho, \boldsymbol{t}) \in \{0,1\}^{256} \times R_q^k$. The private key is:

$$\mathrm{sk} \coloneqq (\rho, \boldsymbol{t}, \boldsymbol{s_1}, \boldsymbol{s_2}) \in \{0,1\}^{256} \times R_q^k \times R_q^\ell \times R_q^k.$$

NOTE: The Dilithium submission includes additional public key compression where pk only contains the most significant bits of $\boldsymbol{t}$.

6.1.5 Dilithium.Sign

Input: Private key sk $\in \{0,1\}^{256} \times R_q^k \times R_q^\ell \times R_q^k$
Message $M \in \{0,1\}^*$

Output: Signature sig $\in R_q^\ell \times R_q$

1) Parse the private key as sk $:= (\rho, \boldsymbol{t}, \boldsymbol{s_1}, \boldsymbol{s_2},)$.

2) Expand the seed ρ using XOF to produce the public matrix $\boldsymbol{A} \in R_q^{k \times \ell}$.

3) Sample $\boldsymbol{y} \in R_q^\ell$ with coefficients in $\{-\gamma_1, \ldots, \gamma_1\}$.

4) Compute $\boldsymbol{w_1} := \text{HighBits}(\boldsymbol{Ay}, 2\gamma_2) \in R_q^k$.

5) Compute $c := \mathcal{H}(M||\boldsymbol{w_1}) \in \mathcal{B}_h$.

6) Compute $\boldsymbol{z} := \boldsymbol{y} + c\boldsymbol{s_1} \in R_q^\ell$.

7) If $\| \boldsymbol{z} \|_\infty \geq \gamma_1 - \beta$ or $\| \text{LowBits}(\boldsymbol{Az} - c\boldsymbol{t}) \|_\infty \geq \gamma_2 - \beta$, then go to step 1.

The signature is sig $:= (\boldsymbol{z}, c) \in R_q^\ell \times R_q$.

NOTE 1: Signing can be deterministic or randomized depending on how $\boldsymbol{y} \in R_q^\ell$ is generated in step 3. In either case, it is expanded from a seed using XOF.

NOTE 2: When the public key is in compressed form then the signer sends the verifier a hint $\boldsymbol{h}$ that allows them to compute the high-order bits of $\boldsymbol{Az} - c\boldsymbol{t}$.

6.1.6 Dilithium.Verify

Input: Public key pk $\in \{0,1\}^{256} \times R_q^k$
Message $M \in \{0,1\}^*$
Signature sig $\in R_q^\ell \times R_q$

Output: $Accept$ or $Reject$

1) Parse the public key as pk $:= (\rho, \boldsymbol{t}) \in \{0,1\}^{256} \times R_q^k$.

2) Parse the signature as sig $:= (\boldsymbol{z}, c) \in R_q^\ell \times R_q$.

3) Expand the seed ρ using XOF to produce the public matrix $\boldsymbol{A} \in R_q^{k \times \ell}$.

4) Compute $\boldsymbol{w_1'} := \text{HighBits}(\boldsymbol{Az} - \boldsymbol{ct}, 2\gamma_2) \in R_q^k$.

5) If $c = \mathcal{H}(M||\boldsymbol{w_1'})$ and $\| \boldsymbol{z} \|_\infty < \gamma_1 - \beta$ then $Accept$, else $Reject$.

6.1.7 Parameters and performance

The Dilithium submission **[**i.18**]** proposes three parameter sets (see Table 5).

Table 5: Proposed parameters for Dilithium

Parameter set	n	q	(k, ℓ)	η	γ_1	γ_2	β	Claimed security
Dilithium-2	256	8 380 417	(4,4)	2	2^{17}	$(q-1)/88$	78	Category 2
Dilithium-3	256	8 380 417	(6,5)	4	2^{19}	$(q-1)/32$	196	Category 3
Dilithium-5	256	8 380 417	(8,7)	2	2^{19}	$(q-1)/32$	120	Category 5

The three parameter sets lead to the public key and signature sizes listed in Table 6.

Table 6: Dilithium public key and signature sizes

Parameter set	Public key	Signature
Dilithium-2	1 312 bytes	2 420 bytes
Dilithium-3	1 952 bytes	3 293 bytes
Dilithium-5	2 592 bytes	4 595 bytes

The Dilithium submission also proposes two challenge parameter sets with claimed security below Category 1 and two extended security parameter sets with claimed security significantly above Category 5.

The Dilithium submission package includes a reference implementation and optimized AVX2 implementation. All cycle counts in Table 7 were obtained from the AVX2 optimized implementation using SHAKE-128 for the XOF on a 2,6 GHz Intel® Core™ i7-6700 processor with TurboBoost disabled.

Table 7: Dilithium AVX2 performance figures

Parameter set	KeyGen	Sign	Verify
Dilithium-2	124 000 cycles	259 000 cycles	118 000 cycles
Dilithium-3	256 000 cycles	429 000 cycles	179 000 cycles
Dilithium-5	298 000 cycles	539 000 cycles	280 000 cycles

6.2 FALCON

6.2.1 Introduction

FALCON [i.20] is an optimized instantiation of the GPV framework described in clause B.1 with NTRU lattices that uses a new trapdoor sampler called "*Fast Fourier sampling*". The underlying hard problem is the Short Integer Solution (SIS) problem over NTRU lattices [i.17].

6.2.2 Public parameters

FALCON is parameterized by:

- n, the dimension of the polynomial ring $R \coloneqq \mathbb{Z}[x]/(x^n + 1)$;
- q, the modulus of the polynomial ring $R_q \coloneqq \mathbb{Z}_q[x]/(x^n + 1)$;
- σ, the standard deviation of the private polynomials; and
- β, the verification bound.

Following the provable NTRU version of [i.23], FALCON chooses n to be a power of two and q to be prime such that $q \equiv 1 \pmod{2n}$. The standard deviation for the private polynomials is $\sigma \coloneqq 1{,}17\sqrt{q/2n}$.

6.2.3 Auxiliary primitives

FALCON makes use of an auxiliary, symmetric primitive:

- XOF, an extendable output function.

The instantiation of this primitive from the specification is described in Table 8.

Table 8: Auxiliary symmetric primitive for FALCON

Primitive	Description
XOF	SHAKE256

FALCON makes use of a hash function $\mathcal{H}: \{0,1\}^* \rightarrow R_q$ constructed from the XOF (see [i.20] for a precise description).

6.2.4 Trapdoor sampling

FALCON signs a message by using $\mathcal{H}$ to hash it to polynomial $c \in R$ and then sampling small polynomials $s_1, s_2 \in R$ such that $s_1 + s_2 h = c$ in R_q, where $h \in R_q$ is the public key.

The polynomials s_1, s_2 are obtained from an efficient trapdoor sampler using a private basis:

$$B := \begin{bmatrix} g & -f \\ G & F \end{bmatrix}$$

for the public NTRU lattice:

$$L := \begin{bmatrix} -h & 1 \\ q & 0 \end{bmatrix}$$

defined by $h \in R_q$. The distribution of polynomials produced by the sampler will be indistinguishable from a discrete Gaussian distribution of standard deviation σ' provided that σ' is slightly larger than $\|B\|_{\mathrm{GS}}$, the Gram-Schmidt norm of the basis B.

NOTE 1: FALCON uses the Fast Fourier Transform (FFT) to implement the arithmetic in key generation and in the efficient trapdoor sampler used for signing. This is a complex-valued transform optimized for polynomials in $\mathbb{Q}[x]/(x^n + 1)$ and is not the same as the Number Theoretic Transform (NTT) in R_q.

NOTE 2: FALCON uses a recursive LDL^* decomposition in its fast Fourier sampler rather than the Gram-Schmidt decomposition used in the original Klein sampler. This decomposition is stored in a structure called a "FALCON tree".

6.2.5 FALCON.KeyGen

Input: Security level

Output: Public key $\mathrm{pk} \in R_q$
Private key $\mathrm{sk} \in R \times R \times R \times R$

1) Sample polynomials $f, g \in R$ from the discrete Gaussian distribution over R with standard deviation σ.

2) Restart if f is not invertible in R_q.

3) Find polynomials $F, G \in R$ such that $fG - gF = q$ in R.

4) Restart if the polynomials F and G were not found in step 3 or if:

$$\left\| \begin{bmatrix} g & -f \\ G & -F \end{bmatrix} \right\|_{\mathrm{GS}}^2 > 2n\sigma^2.$$

5) Compute $h := gf^{-1} \in R_q$.

The public key is $\mathrm{pk} := h \in R_q$. The private key is $\mathrm{sk} := (f, g, F, G) \in R \times R \times R \times R$.

NOTE 1: The private polynomials f, g, F and G are stored in the FFT domain.

NOTE 2: Key generation also includes computation of the FALCON tree used by the efficient trapdoor sampler. The FALCON tree forms part of the private key, and the nodes of the tree are stored in the FFT domain.

6.2.6 FALCON.Sign

Input: Private key $sk := (f, g, F, G) \in R \times R \times R \times R$
Message $M \in \{0,1\}^*$.

Output: Signature $\mathrm{sig} \in R \times \{0,1\}^{320}$.

1) Choose a uniformly random $r \in \{0,1\}^{320}$.

2) Hash $c := \mathcal{H}(r \parallel M) \in R_q$.

3) Use the trapdoor sampler to find small polynomials $s_1, s_2 \in R$ such that $s_1 + s_2 h = c$ in R_q.

4) Restart if $\|(s_1, s_2)\|^2 > \beta$.

The signature is sig $:= (s_2, r) \in R \times \{0,1\}^{320}$.

NOTE: The FALCON submission includes compression techniques to reduce the size of the signature.

6.2.7 FALCON.Verify

Input: Public key pk $:= h \in R_q$
Message $M \in \{0,1\}^*$
Signature sig $:= (s_2, r) \in R \times \{0,1\}^{320}$.

Output: $Accept$ or $Reject$.

1) Hash $c := \mathcal{H}(r \parallel M) \in R_q$.

2) Compute $s_1 := c - s_2 h \in R_q$ and lift to a polynomial in R with coefficients in $\{-(q-1)/2, \ldots (q-1)/2\}$.

3) If $\|(s_1, s_2)\|^2 \leq \beta$ then $Accept$, else $Reject$.

6.2.8 Parameters and performance

The FALCON submission [i.20] describes two set of parameters (see Table 9).

Table 9: Proposed parameters for FALCON

Parameter set	n	q	β	Claimed security
FALCON-512	256	12 289	34 034 726	Category 1
FALCON-1024	256	12 289	70 265 242	Category 5

The two parameter sets lead to the private key, public key and signature sizes listed in Table 10.

Table 10: FALCON private key, public key and signature sizes

Parameter set	Private key	Public key	Signature
FALCON-512	32 bytes	897 bytes	666 bytes
FALCON-1024	32 bytes	1 793 bytes	1 280 bytes

The submission package includes a reference implementation (that uses AVX2). The performance figures in Table 11 were measured on a 2,3 GHz Intel® Core™ i5-8259U.

Table 11: Falcon AVX2 performance figures

Parameter set	KeyGen	Sign	Verify
FALCON-512	18 722 000 cycles	386 678 cycles	82 340 cycles
FALCON-1024	63 135 000 cycles	789 564 cycles	168 498 cycles

6.3 Rainbow

6.3.1 Introduction

Rainbow is a multivariate signature scheme [i.24] that generalizes the Unbalanced Oil and Vinegar (UOV) scheme [i.25]. The public key of Rainbow is a set of non-linear equations; the verification process involves evaluating the public-key polynomials; and the signing process involves solving a system of linear equations. A recent result [i.39] seems to challenge the security of the parameters proposed in the Rainbow round 3 submission (see clause C.2 for a short discussion).

6.3.2 Public parameters

Rainbow is parameterized by:

- q, a prime power specifying the finite field $\mathbb{F}_q$;
- u, the number of layers;
- $v_1 < v_2 < \cdots < v_u < v_{u+1}$, the number of vinegar variables in each layer; and
- s, the bit length of the random salt.

6.3.3 Auxiliary primitives

Rainbow makes use of two auxiliary, symmetric primitives:

- H, a cryptographic hash function; and
- PRF, a pseudorandom function.

The instantiations of these primitives from the specification are described in Table 12.

Table 12: Auxiliary symmetric primitives for Rainbow

Primitive	Category 1	Category 3	Category 5
H	SHA-256	SHA-384	SHA-512
PRF	AES-256 in CTR mode		

Rainbow also makes use of a hash function $\mathcal{H}: \{0,1\}^* \rightarrow \mathbb{F}_q^m$ constructed from H (see [i.24] for a precise description).

Rainbow follows the general framework described in clause B.1. The inner polynomials $f_{v_1+1}, \ldots, f_n \in \mathbb{F}_q[x_1, \ldots, x_n]$ have a layered structure. Define the index sets $V_\ell := \{1, \ldots, v_\ell\}$ and $O_\ell := \{v_\ell + 1, \ldots, v_{\ell+1}\}$, for all $1 \leq \ell \leq u$. Let $o_\ell := |O_\ell|$ be the number of oil variables in the ℓth layer , $n := v_{u+1}$ be the total number of variables, and $m := n - v_1$ be the total number of equations. For all $1 \leq \ell \leq u$, the ℓth layer is a set of o_ℓ quadratic polynomials in $v_\ell + o_\ell$ variables indexed by variables in $V_\ell \cup O_\ell$. Each layer has a specific shape so that specializing the vinegar variables indexed by V_ℓ leads to a set of o_ℓ linear polynomials in the o_ℓ oil variables indexed by the set O_ℓ.

Given $H' = \mathbb{F}_q^m$, the main task in the signature process is to find a solution to the system:

$$\begin{aligned} f_{v_1+1}(x_1, \ldots, x_n) &= H'_1 \\ &\vdots \\ f_n(x_1, \ldots, x_n) &= H'_m \end{aligned}$$

if one exists.

Due to the special structure of the polynomials $f_{v_1+1}, \ldots, f_n \in \mathbb{F}_q[x_1, \ldots, x_n]$, this can be done in polynomial time. The signer fixes the vinegar variables and finds the remaining components by solving a system of linear equations. The function $\mathrm{Inv}(f_{v_1+1} - H'_1, \ldots, f_n - H'_n)$ corresponds to this process. It returns $(s'_1, \ldots, s'_n) \in \mathbb{F}_q^n$ such that $f_{v_1+1}(s'_1, \ldots, s'_n) = H'_1, \ldots, f_n(s'_1, \ldots, s'_n) = H'_m$ and 0 if no solution exists.

6.3.4 Rainbow.KeyGen

Input: Security level

Output: Public key $\mathrm{pk} := (p_1, \ldots, p_m, s) \in \mathbb{F}_q[x_1, \ldots, x_n]^m \times \mathbb{N}$
Private key $\mathrm{sk} := (S, U, s) \in \mathrm{GL}_{n \times n}(\mathbb{F}_q) \times \mathrm{GL}_{m \times m}(\mathbb{F}_q) \times \mathbb{N}$

1) Use PRF to sample a pair of invertible matrices $(S, U) \in \mathrm{GL}_{n \times n}(\mathbb{F}_q) \times \mathrm{GL}_{m \times m}(\mathbb{F}_q)$.

2) Use PRF to generate inner polynomials $f_{v_1+1}, \ldots, f_n \in \mathbb{F}_q[x_1, \ldots, x_n]$ with the Rainbow shape.

3) Compute the public key polynomials $(p_1, \ldots, p_m) \coloneqq (f_{v_1+1}((x_1, \ldots, x_n)S), \ldots, f_n((x_1, \ldots, x_n)S))U \in \mathbb{F}_q[x_1, \ldots, x_n]^m$.

The public key is $\mathrm{pk} \coloneqq (p_1, \ldots, p_m, s) \in \mathbb{F}_q[x_1, \ldots, x_n]^m \times \mathbb{N}$. The private key $\mathrm{sk} \coloneqq (S, U, s) \in \mathrm{GL}_{n\times n}(\mathbb{F}_q) \times \mathrm{GL}_{m\times m}(\mathbb{F}_q) \times \mathbb{N}$.

NOTE: The specification [i.24] describes three variants of Rainbow: a Standard version where the public key is explicitly described as polynomial equations and two variants (CZ-Rainbow and Compressed). The key-pair generation process described in the present clause corresponds to Standard Rainbow. In the CZ-Rainbow and Compressed variants, the size of the public key is reduced by partially generating the polynomials from a random seed [i.26].

6.3.5 Rainbow.Sign

Input: Private key $\mathrm{sk} \coloneqq (S, U, s) \in \mathrm{GL}_{n\times n}(\mathbb{F}_q) \times \mathrm{GL}_{m\times m}(\mathbb{F}_q) \times \mathbb{N}$
Message $M \in \{0,1\}^*$

Output: Signature $\mathrm{sig} \coloneqq (s_1, \ldots, s_n, r) \in \mathbb{F}_q^n \times \{0,1\}^\ell$

1) Use PRF to sample $r \in \{0,1\}^s$.

2) Compute $H \coloneqq (H_1, \ldots, H_m) = \mathcal{H}(\mathcal{H}(M) \parallel r)) \in \mathbb{F}_q^m$.

3) Compute $H' \coloneqq HU^{-1} \in \mathbb{F}_q^n$.

4) If $\mathrm{Inv}(f_{v_1+1} - H'_1, \ldots, f_n - H'_n) = 0$ then go to step 1.

5) Set $(s'_1, \ldots, s'_n) \coloneqq \mathrm{Inv}(f_{v_1+1} - H'_1, \ldots, f_n - H'_n) \in \mathbb{F}_q^n$.

6) Compute $(s_1, \ldots, s_n) \coloneqq (s'_1, \ldots, s'_n)\, S^{-1} \in \mathbb{F}_q^n$.

The signature is $\mathrm{sig} \coloneqq (s_1, \ldots, s_n, r) \in \mathbb{F}_q^n \times \{0,1\}^\ell$.

6.3.6 Rainbow.Verify

Input: Public key $\mathrm{pk} \coloneqq (p_1, \ldots, p_m, s) \in \mathbb{F}_q[x_1, \ldots, x_n]^m \times \mathbb{N}$
Message $M \in \{0,1\}^*$
Signature $\mathrm{sig} \coloneqq (s_1, \ldots, s_n, r) \in \mathbb{F}_q^n \times \{0,1\}^s$.

Output: $Accept$ or $Reject$.

1) Compute $H \coloneqq (H_1, \ldots, H_m) = \mathcal{H}(\mathcal{H}(M) \parallel r) \in \mathbb{F}_q^m$.

2) Compute $H' \coloneqq (p_1(s_1, \ldots, s_n), \ldots, p_m(s_1, \ldots, s_n)) \in \mathbb{F}_q^m$.

3) If $H' = H$ then $Accept$ else $Reject$.

6.3.7 Parameters and performance

The Rainbow submission [i.24] proposes three sets of parameters (see Table 13).

Table 13: Proposed parameters for Rainbow

Parameter set	q	u	v_1	o_1	o_2	Claimed security
Rainbow-I	2^4	2	36	32	32	Category 1
Rainbow-III	2^8	2	68	32	48	Category 3
Rainbow-V	2^8	2	68	32	48	Category 5

These parameter sets lead to the private key, public key and signature sizes listed in Table 14.

Table 14: Rainbow private key, public key and signature sizes

Parameter set	Private key	Public key	Signature	Comments
Rainbow-I	101 200 bytes	157 800 bytes	528 bits	Notes 1 and 4
Rainbow-III	611 300 bytes	861 400 bytes	1 312 bits	Notes 2 and 4
Rainbow-V	1 375 700 bytes	1 885 400 bytes	1 696 bits	Notes 3 and 4
NOTE 1: CZ-Rainbow reduces the size of the Category 1 public key to 58 800 bytes. NOTE 2: CZ-Rainbow reduces the size of the Category 2 public key to 258 400 bytes. NOTE 3: CZ-Rainbow reduces the size of the Category 3 public key to 523 600 bytes. NOTE 4: Compressed Rainbow reduces the size of the private key to 32 bytes.				

The submission package includes reference and optimized implementations using AVX2 instructions. Tables 15, 16 and 17 give the performance figures obtained for the optimized implementation on a 3,60 GHz Intel® Xeon® CPU E3-1275 processor.

Table 15: Rainbow AVX2 performance figures

Parameter set	KeyGen	Sign	Verify
Rainbow-I	9 900 000 cycles	67 000 cycles	34 000 cycles
Rainbow-III	52 000 000 cycles	285 000 cycles	132 000 cycles
Rainbow-V	192 000 000 cycles	739 000 cycles	392 000 cycles

Table 16: CZ-Rainbow AVX2 performance figures

Parameter set	KeyGen	Sign	Verify
Rainbow-I	10 700 000 cycles	67 000 cycles	3 500 000 cycles
Rainbow-III	64 000 000 cycles	285 000 cycles	20 000 000 cycles
Rainbow-V	235 000 000 cycles	739 000 cycles	47 000 000 cycles

Table 17: Compressed Rainbow AVX2 performance figures

Parameter set	KeyGen	Sign	Verify
Rainbow-I	10 700 000 cycles	7 000 000 cycles	3 500 000 cycles
Rainbow-III	64 000 000 cycles	41 000 000 cycles	20 000 000 cycles
Rainbow-V	235 000 000 cycles	118 000 000 cycles	47 000 000 cycles

NOTE: Due to the analysis in [i.39], the Round 3 parameters presented in Table 13 might need to be updated (see clause C.2 for a short discussion).

7 Alternate Candidates

7.1 GeMSS

7.1.1 Introduction

G*e*MSS [i.27] is a multivariate-based signature scheme designed from a variant of the Hidden Field Equations (HFE) [i.29] with vinegar and minus modifiers (HFEv-). The public key of G*e*MSS is a set of multivariate quadratic binary polynomials. The verification process involves evaluating the public key polynomials whilst producing a signature reduces to find the roots of a univariate polynomial. A recent result [i.40] challenges the security of the parameters proposed in the G*e*MSS round 3 submission (see clause C.2).

7.1.2 Public parameters

G*e*MSS is parameterized by:

- D, the degree of a secret univariate polynomial;

- n, the degree of a finite field extension;
- v, the number of vinegar variables;
- m, the number of equations in the public key ($m \leq n$); and
- k, a repetition factor;

7.1.3 Auxiliary primitives

G*e*MSS makes use of two auxiliary, symmetric primitives:

- H, a cryptographic hash function; and
- PRF, a pseudorandom function.

The instantiations of these primitives from the specification are described in Table 18.

Table 18: Auxiliary symmetric primitives for GeMSS

Primitive	Category 1	Category 2	Category 3	Category 5
H	SHA3-256		SHAKE-512	
PRF	SHAKE-128		SHAKE-256	

G*e*MSS also makes use of a hash function $\mathcal{H}: \{0,1\}^* \rightarrow \mathbb{F}_2^m$ constructed from H (see [i.27] for a precise description).

Following the terminology of clause B.1, the inner polynomials $f_1, ..., f_n \in \mathbb{F}_2[x_1, ..., x_{n+v}]$ are computed from a polynomial $F \in \mathbb{F}_{2^n}[X, x_{n+1}, \dots, x_{n+v}]$ with a special HFEv form:

$$F(X, x_{n+1}, \dots, x_{n+v}) \coloneqq \sum_{\substack{0 \leq i<j \leq n-1 \\ 2^{i+j} \leq D}} A_{i,j} X^{2^{i+j}} + \sum_{\substack{0 \leq i \leq n-1 \\ 2^i \leq D}} \beta_i(x_{n+1}, \dots, x_{n+v}) X^{2^i} + \gamma(x_{n+1}, \dots, x_{n+v})$$

where $A_{i,j} \in \mathbb{F}_{2^n}$, the maps $\beta_i : \mathbb{F}_2^v \rightarrow \mathbb{F}_{2^n}$ are linear and the map $\gamma: \mathbb{F}_2^v \rightarrow \mathbb{F}_{2^n}$ is quadratic. The variables $x_{n+1}, \dots, x_{n+v}$ are called *vinegar variables* and for each specialization of them, the polynomial F becomes a univariate polynomial with the following shape:

$$Fs(X) \coloneqq \sum_{\substack{0 \leq i \leq j \leq n-1 \\ 2^{i+j} \leq D}} A_{i,j} X^{2^{i+j}} + \sum_{\substack{0 \leq i \leq n-1 \\ 2^i \leq D}} B_i X^{2^i} + C \in \mathbb{F}_{2^n}[X],$$

where $A_{i,j}, B_i, C \in \mathbb{F}_{2^n}$. The roots of $Fs(X)$ can be found in quasi-linear time in nD [i.30].

A central task in the signature process is to find a solution to the system:

$$\begin{aligned} f_1(x_1, \dots, x_{n+v}) &= {H'}_1 \\ &\vdots \\ f_n(x_1, \dots, x_{n+v}) &= {H'}_n \end{aligned}.$$

This involves randomly sampling $({s'}_{n+1}, \dots, {s'}_{n+v}) \in {\mathbb{F}_2}^v$, setting $Fs(X) \coloneqq F(X, {s'}_{n+1}, \dots, {s'}_{n+v})$ and computing the roots of $Fs(X)$. The process is repeated until $Fs(X)$ has at least one root. The function $\mathrm{Inv}(f_1 - {H'}_1, ..., f_n - {H'}_n)$ corresponds to this process. It returns $({s'}_1, \dots, {s'}_{n+v}) \in \mathbb{F}_2^{n+v}$ such that $f_1({s'}_1, \dots, {s'}_{n+v}) = {H'}_1, \dots, f_n({s'}_1, \dots, {s'}_{n+v}) = {H'}_n$.

7.1.4 GeMSS.KeyGen

Input: Security level

Output: Public key $\mathrm{pk} \coloneqq (p_1, ..., p_m) \in \mathbb{F}_2[x_1, \dots, x_{n+v}]^m$
Private key $\mathrm{sk} \in \mathrm{GL}_{(n+v)\times(n+v)}(\mathbb{F}_2) \times \mathrm{GL}_{n\times n}(\mathbb{F}_2) \times \mathbb{F}_{2^n}[X, x_{n+1}, \dots, x_{n+v}]$

1) Use PRF to sample a pair of invertible matrices $(S, U) \in \mathrm{GL}_{(n+v)\times(n+v)}(\mathbb{F}_2) \times \mathrm{GL}_{n\times n}(\mathbb{F}_2)$.

2) Use PRF to sample a HFEv polynomial (see clause 7.1) $F \in \mathbb{F}_{2^n}[X, x_{n+1}, \dots, x_{n+v}]$.

3) Generate the secret inner polynomials $f_1, \ldots, f_n \in \mathbb{F}_2[x_1, \ldots, x_{n+v}]$ from F.

4) Compute the public-key polynomials $(p_1, \ldots, p_m, \ldots, p_n) = (f_1((x_1, \ldots, x_{n+v})S), \ldots, f_n((x_1, \ldots, x_{n+v})S))U \in \mathbb{F}_2[x_1, \ldots, x_{n+v}]^n$.

The public key is pk $:= (p_1, \ldots, p_m) \in \mathbb{F}_2[x_1, \ldots, x_{n+v}]^m$. The private key is sk $:= ((S^{-1}, U^{-1}), F) \in \mathrm{GL}_{(n+v)\times(n+v)}(\mathbb{F}_2) \times \mathrm{GL}_{n\times n}(\mathbb{F}_2) \times \mathbb{F}_{2^n}[X, x_{n+1}, \ldots, x_{n+v}]$.

NOTE: The G*e*MSS submission describes how to compress the private key by deriving everything from a seed.

7.1.5 GeMSS.Sign

Input: Private key sk $= ((S^{-1}, U^{-1}), F) \in \mathrm{GL}_{(n+v)\times(n+v)}(\mathbb{F}_2) \times \mathrm{GL}_{n\times n}(\mathbb{F}_2) \times \mathbb{F}_{2^n}[X, x_{n+1}, \ldots, x_{n+v}]$
Message $M \in \{0,1\}^*$.

Output: Signature sig $\in \mathbb{F}_2^{m+\mathrm{k}(n+v-m)}$.

1) Set $S_0 = (0, \ldots, 0) \in \mathbb{F}_2^m$.

2) For i from 1 to k:

2.1) Compute $H := \mathcal{H}^i(M \parallel i) + S_{i-1}, \in \mathbb{F}_2^m$, where $\mathcal{H}^i$ is the ith composition of $\mathcal{H}$.

2.2) Use a PRF to sample $H_{m+1}, \ldots, H_n \in \mathbb{F}_2^{n-m}$.

2.3) Computer $H' := (HU^{-1}, H_{m+1}, \ldots, H_n) \in \mathbb{F}_2^m$.

2.4) Set $(s'_1, \ldots, s'_{n+v}) := \mathrm{Inv}(f_1 - H'_1, \ldots, f_n - H'_n) \in \mathbb{F}_2^{n+v}$.

2.5) Compute $(s_1, \ldots, s_{n+v}) := (s'_1, \ldots, s'_{n+v})\, S^{-1} \in \mathbb{F}_2^{n+v}$.

2.6) Set $S_i := (s_1, \ldots, s_m) \in \mathbb{F}_2^m$ and $X_i := (s_{m+1}, \ldots, s_{n+v}) \in \mathbb{F}_2^{n+v-m}$.

The signature is sig $:= (S_k, X_k, \ldots, X_1) \in \mathbb{F}_2^{m+\mathrm{k}(n+v-m)}$.

7.1.6 GeMSS.Verify

Input: Public key pk $:= \mathbf{p} = (p_1, \ldots, p_m) \in \mathbb{F}_2[x_1, \ldots, x_n]^m$
Message $M \in \{0,1\}^*$
Signature sig $:= (S_k, X_k, \ldots, X_1) \in \mathbb{F}_2^{m+\mathrm{k}(n+v-m)}$

Output: $Accept$ or $Reject$.

1) For i from k to 1:

1.1) Compute $S_{i-1} := \mathbf{p}(S_i, X_i) + \mathcal{H}^i(M \parallel i)$.

2) If $S_0 = (0, \ldots, 0)$ then $Accept$ else $Reject$.

7.1.7 Parameters and performance

Table 19 lists the families of proposed parameter sets added in the round 3 G*e*MSS submission [i.27]. See annex D for families of parameter sets from earlier versions of the submission.

Table 19: Proposed parameters for round 3 GeMSS

Parameter set	D	k	n	m	v	Claimed security
WhiteGeMSS-128	513	3	175	163	12	Category 1
CyanGeMSS-128	129	3	177	163	14	Category 1
MagentaGeMSS-128	17	3	178	163	15	Category 1
WhiteGeMSS-192	513	3	268	247	21	Category 3
CyanGeMSS-192	129	3	270	247	23	Category 3
MagentaGeMSS-192	17	3	271	247	24	Category 3
WhiteGeMSS-256	513	3	364	333	29	Category 5
CyanGeMSS-256	129	3	364	333	32	Category 3
MagentaGeMSS-256	17	3	366	333	33	Category 5

These parameter sets lead to the private key, public key and signature sizes listed in Table 20.

Table 20: GeMSS private key, public key and signature sizes

Parameter set	Private key	Public key	Signature
WhiteGeMSS-128	16 bytes	358 170 bytes	235 bits
CyanGeMSS-128		369 720 bytes	244 bits
MagentaGeMSS-128		381 460 bytes	253 bits
WhiteGeMSS-192	24 bytes	1 293 847 bytes	373 bits
CyanGeMSS-192		1 320 801 bytes	383 bits
MagentaGeMSS-192		1 348 033 bytes	391 bits
WhiteGeMSS-256	32 bytes	3 222 690 bytes	513 bits
CyanGeMSS-256		3 272 016 bytes	522 bits
MagentaGeMSS-256		3 321 716 bytes	531 bits

The submission package [i.27] includes a reference implementation, optimized implementation and additional implementation. MQsoft [i.31] is an efficient library in C for HFE-based schemes such as G*e*MSS by improving the complexity of several fundamental building blocks for such schemes. It uses SSSE2, SSSE3 and the AVX2 instructions sets. Table 21 gives the performance figures [i.27], obtained with MQsoft on a 3,40 GHz Intel® Core™ i7-6600U processor.

Table 21: GeMSS AVX2 performance figures

Parameter set	KeyGen	Sign	Verify
WhiteGeMSS-128	20 000 000 cycles	436 000 000 cycles	91 700 cycles
CyanGeMSS-128	18 500 000 cycles	49 800 000 cycles	91 000 cycles
MagentaGeMSS-128	16 700 000 cycles	1 820 000 cycles	101 000 cycles
WhiteGeMSS-192	73 100 000 cycles	1 330 000 000 cycles	263 000 cycles
CyanGeMSS-192	68 200 000 cycles	131 000 000 cycles	269 000 cycles
MagentaGeMSS-192	60 300 000 cycles	4 530 000 cycles	274 000 cycles
WhiteGeMSS-256	163 000 000 cycles	1 920 000 000 cycles	516 000 cycles
CyanGeMSS-256	159 000 000 cycles	190 000 000 cycles	535 000 cycles
MagentaGeMSS-256	148 000 000 cycles	7 610 000 cycles	535 000 cycles

Additional parameters and performance figures are given in annex D.

NOTE: Due to the analysis in [i.40], the Round 3 parameters presented in Table 19 might need to be updated (see clause C.2).

7.2 Picnic

7.2.1 Introduction

The Picnic signature scheme [i.32] is constructed from a Zero-Knowledge Identification Scheme (ZK-IDS) following the Fiat-Shamir framework (see clause B.3). Its security is based on the hardness of key-recovery against a block-cipher. The basic idea of Picnic is to use a generic ZK proof system for constructing a digital signature scheme. The first version Picnic used ZKB++, a variant of ZKBoo [i.33]. A new variant of ZKBoo was then introduced by Katz, Kolesnikov and Wang [i.34] which reduced the size of the signature.

The public key in Picnic is given by a pair $(\mathbf{p}, \mathbf{C} = E_{\mathrm{sk}}(\mathbf{p})) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$, where E is the LowMC [i.35] block-cipher. The private key $\mathrm{sk} \in \mathbb{F}_2^n$ of Picnic corresponds to the secret key of the block-cipher E. [i.32] derives a ZK-IDS of the private key knowledge.

7.2.2 Public parameters

The Picnic scheme is parameterized by:

- k, the security parameter;
- n, the LowMC key and blocksize in bits;
- m, the number of LowMC S-boxes;
- r, the number of LowMC rounds;
- ℓ_h, the length of the hash function output in bits;
- S, the length of seeds for generating the shares and blinding values for notional MPC participants; and
- T, the number of repetitions of the zero-knowledge proof.

7.2.3 Auxiliary primitives

Picnic makes use of several auxiliary, symmetric primitives:

- $\mathcal{H}$, a cryptographic hash function;
- XOF, an extendable output function; and
- E, a block cipher with n-bit keys and an n-bit blocksize.

The instantiations of these primitives from the specification are described in Table 22.

Table 22: Auxiliary symmetric primitive for Picnic

Primitive	Category 1	Category 3	Category 5
$\mathcal{H}$	SHAKE-128	SHAKE-256	SHAKE-256
XOF	SHAKE-128	SHAKE-256	SHAKE-256
E	The LowMC [i.35] block cipher.		

E is implemented in a manner that supports blinded, secure, multi-party computation such as the Yao garbled circuit construction [i.32]. In **Picnic,** a 3-party implementation is used with inputs and outputs $\boldsymbol{t}$ to smaller functions divided into shares $\boldsymbol{t} = \boldsymbol{t}_0 + \boldsymbol{t}_1 + \boldsymbol{t}_2$. Some share values are passed between the parties blinded with random XORs.

7.2.4 Picnic.KeyGen

Input: Security level

Output: Public key pk $\in \mathbb{F}_2^n \times \mathbb{F}_2^n$
Private key sk $\in \mathbb{F}_2^n$

1) Randomly sample a plaintext $\mathbf{p} \in \mathbb{F}_2^n$ and a private key $\mathrm{sk} \in \mathbb{F}_2^n$.

2) Set the public key as $\mathrm{pk} \coloneqq (\mathbf{p}, \mathbf{C} = E_{\mathrm{sk}}(\mathbf{p})) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$.

7.2.5 Picnic.Sign

Input: Public key $\mathrm{pk} \coloneqq (\mathbf{p}, \mathbf{C} = E_{\mathrm{sk}}(\mathbf{p})) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$
Private key $\mathrm{sk} \in \mathbb{F}_2^n$
Message $M \in \{0,1\}^*$.

Output: Signature $\mathrm{sig} \in \mathbb{F}_2^{2T} \times \mathbb{F}_2^{256} \times (\mathbb{F}_2^{\ell_h} \times \mathbb{F}_2^{3rm+2S+n})^T$

1) Generate a 256-bit $salt$ and a list of $3T$ seeds of length S for the XOF. Use the seeds to generate $3T$ random streams $rand[j][t] \coloneqq \mathrm{XOF}(H(seed[j][t]) \,||\, salt \,||\, t \,||\, j \,||\, output\_length)$.

2) For t from 0 to $T-1$:

   2.1) Generate three shares of sk by setting $x[0]$ to be the first n bits of $\mathrm{rand}[0][t]$, $x[1]$ to be the first n bits of $rand[0][t]$ and $x[2] \coloneqq \mathrm{sk} + x[0] + x[1]$.

   2.2) Execute the 3-party computation $E_{(x[0]+x[1]+x[2])}(\boldsymbol{p})$ for all three parties drawing blinding values for the parties from $rand[0][t]$, $rand[1][t]$ and $rand[2][t]$ respectively. Keep a transcript of all communications that each party would receive.

   2.3) Keep a copy of the final output shares of the computation.

   2.4) Form a commitment $C[t][i] \coloneqq \mathcal{H}(\mathcal{H}(seed[i]), input\_share[i] \,||\, transcript[i] \,||\, output\_share[i])$ for each party's view of the computation.

3) Form a list $e \in \mathbb{F}_3^T$ of T challenges by applying the XOF the full collection of output shares concatenated with the full collection of commitments, the $salt$, pk and M. Two-bit outputs 00, 01, 10, and 11 are treated as 0, 1, 2 and generate another output.

4) For t from 0 *to* $T-1$:

   4.1) Set the responses $b[t] \coloneqq C[t][(i+2) \bmod 3]$ and $z[t]$ to be the collection of transcript, seeds and shares required to verify the computation of parties i and $(i+1) \bmod 2$.

5) Set $\mathrm{sig} \coloneqq (e, salt, ((b[0], z[0]), \ldots, (b[T-1], z[T-1]))$.

7.2.6 Picnic.Verify

Input: Public key $\mathrm{pk} \coloneqq (\mathbf{p}, \mathbf{C} = E_{\mathrm{sk}}(\mathbf{p})) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$
Message $\mathrm{M} \in \{0,1\}^*$
Signature $\mathrm{sig} \in \mathbb{F}_2^{2T} \times \mathbb{F}_2^{256} \times (\mathbb{F}_2^{\ell_h} \times \mathbb{F}_2^{3rm+2S+n})^T$.

Output: $Accept$ or $Reject$.

1) Parse sig into $(e, salt, (b[0], z[0], \ldots, b[T-1], z[T-1]))$. If deserialization fails output Reject.

2) For t from 0 to $T-1$:

   2.1) Use $z[t]$ to initialize sk-shares and random streams of two parties.

2.2) Execute the 3-party computation $E_{(x[0]+x[1]+x[2])}(\boldsymbol{p})$ for the two parties that can be reconstructed using their input shares and random streams to create blinding values. Keep a transcript of all communications that each of the two parties would receive.

2.3) For the parties $i = e_t$ and $i = (e_t + 1) \bmod 3$ compute:

$$\mathcal{H}(\mathcal{H}(seed[i]), input\_share[i] \,||\, transcript[i] \,||\, output\_share[i]).$$

Use these together with $b[t]$ to reconstruct putative commitments $C[t]$ per step 2.4 of the signing process.

2.4) Compute the third output share $C + output\_share[e_t] + output\_share[(e_t + 1) \bmod 3]$.

3) Reconstruct the putative set of challenges e' per step 3 of the signing process.

4) If $e' = e$ then return $Accept$ else return $Reject$.

NOTE 1: Picnic-FR refers to the Fiat-Shamir variant of Picnic, Picnic-full is the same as the Fiat-Shamir variant but with different LowMC parameters. Finally, Picnic-UR is the variant using Unruh transformation.

NOTE 2: Picnic.Sign and Picnic.Verify describe signing and verification as performed in Picnic-{L1,L3,L5}-{FS,full} and partly as done by the UR variant.

7.2.7 Parameters and performance

The Picnic submission [i.32] proposes twelve sets of parameters (see Table 23).

Table 23: Proposed parameters for Picnic

Parameter set	k	m	n	r	ℓ_h	S	T	Claimed security
Picnic-L1-FS/UR	128	10	128	20	256	128	219	Category 1
Picnic-L1-full	128	43	129	4	256	128	219	Category 1
Picnic3-L1	128	43	129	4	256	128	250	Category 1
Picnic-L3-FS/UR	192	10	192	30	384	192	329	Category 3
Picnic-L3-full	192	64	192	4	384	192	329	Category 3
Picnic3-L3	192	64	192	4	384	192	419	Category 3
Picnic-L5-FS/UR	256	10	256	38	512	256	438	Category 5
Picnic-L5-full	255	85	255	4	512	256	438	Category 5
Picnic3-L5	255	85	255	4	512	256	601	Category 5

These parameter sets lead to the private key, public key and signature sizes listed in Table 24.

Table 24: Picnic private key, public key and signature sizes

Parameter set	Private key	Public key	Signature
Picnic-L1-FS	16 bytes	32 bytes	34 032 bytes
Picnic-L1-UR	16 bytes	32 bytes	53 961 bytes
Picnic-L1-full	17 bytes	34 bytes	32 061 bytes
Picnic3-L1-FS	17 bytes	34 bytes	13 802 bytes
Picnic-L3-FS	24 bytes	48 bytes	76 772 bytes
Picnic-L3-UR	24 bytes	48 bytes	121 845 bytes
Picnic-L3-full	24 bytes	48 bytes	71 179 bytes
Picnic3-L3-FS	24 bytes	48 bytes	35 024 bytes
Picnic-L5-FS	32 bytes	64 bytes	132 856 bytes
Picnic-L5-UR	32 bytes	64 bytes	209 506 bytes
Picnic-L5-full	32 bytes	64 bytes	126 286 bytes
Picnic3-L5-FS	32 bytes	64 bytes	61 024 bytes

The **Picnic** submission package includes a reference implementation and an optimized implementation that uses AVX2 instruction sets. Table 25 gives performance figures [i.32] for the optimized implementation on a 3,60 GHz Intel® Core™ i7-4790 processor.

Table 25: Picnic AVX2 performance figures

Parameter set	KeyGen	Sign	Verify
Picnic-L1-FS	5 599 cycles	4 924 342 cycles	3 982 244 cycles
Picnic-L1-UR	6 418 cycles	6 502 156 cycles	5 305 718 cycles
Picnic-L1-full	3 680 cycles	3 601 664 cycles	2 871 980 cycles
Picnic3-L1-FS	4 151 cycles	18 252 055 cycles	13 811 201 cycles
Picnic-L3-FS	10 479 cycles	11 509 382 cycles	9 452 289 cycles
Picnic-L3-UR	10 964 cycles	14 875 862 cycles	12 764 570 cycles
Picnic-L3-full	6 567 cycles	7 008 817 cycles	5 626 166 cycles
Picnic3-L3-FS	6 567 cycles	37 595 772 cycles	29 243 365 cycles
Picnic-L5-FS	15 255 cycles	20 085 119 cycles	16 722 292 cycles
Picnic-L5-UR	17 118 cycles	25 178 763 cycles	20 998 784 cycles
Picnic-L5-full	6 701 cycles	11 351 041 cycles	9 217 982 cycles
Picnic3-L5-FS	9 504 cycles	65 555 710 cycles	46 887 830 cycles

7.3 SPHINCS+

7.3.1 Introduction

SPHINCS+ [i.37] is a stateless hash-based signature scheme constructed using a tweakable hash function. The security of the scheme depends on properties of the tweakable hash function such as distinct-function multi-target second-preimage resistance.

SPHINCS+ uses several different components:

- The Winternitz One-Time Signature (WOTS) scheme;
- The Merkle signature scheme and its hierarchical version; and
- The Forest Of Random Subsets (FORS) few-time signature scheme.

There are two different instantiations of SPHINCS+: a simple variant which has a non-tight security proof in the ROM and a robust variant which has a tight security proof with fewer random oracle assumptions.

7.3.2 Public parameters

The main parameters for SPHINCS+ are:

- n, the length of the tweakable hash function output;
- w, the length of the hash chains in the WOTS one-time signature;
- ℓ, the number of hash chains in the WOTS one-time signature;
- k, the number of trees in the FORS few-time signature;
- a, the height of the trees in the FORS few-time signature;
- h, the total height of the tree in the hierarchical Merkle signature; and
- d, the number of layers in the hierarchical Merkle signature.

7.3.3 Auxiliary functions

7.3.3.1 Symmetric primitives

SPHINCS+ makes use of several auxiliary symmetric primitives:

- H, an n-bit tweakable hash function;

- H_{msg}, an m-bit cryptographic hash function where $m \coloneqq ka + h$; and
- PRF, a pseudorandom function.

These primitives are instantiated in the specification using either SHA-256, SHAKE-256 or Haraka.

NOTE: Every call to the tweakable hash function includes a public seed and unique hash address to prevent multi-target attacks.

7.3.3.2 One-time signature scheme

The WOTS one-time signature scheme is as follows:

- **KeyGen:** The one-time private key is a length-ℓ sequence of n-bit values. The private values are used as the starting values for ℓ hash chains of length w. The one-time public key is the n-bit hash of the values at the ends of the ℓ chains.
- **Sign:** The n-bit message and its checksum are used to select positions in each of the ℓ hash chains. The one-time signature is the sequence of n-bit intermediate chain values at these positions.
- **Verify:** The sequence of intermediate values from the signature are used to complete the hash chains. The signature is valid if the hash of the values recovered at the ends of the chains matches the public key.

NOTE: The WOTS one-time private keys are all derived from a single n-bit seed using PRF.

7.3.3.3 Merkle signature scheme

The Merkle signature scheme is as follows:

- **KeyGen:** The Merkle private key is a sequence of $2^{h'}$ one-time private keys. The one-time public keys form the leaf nodes of a binary hash tree. The Merkle public key is the n-bit value at the root node of the tree.
- **Sign:** The n-bit message with message index i is signed by the ith one-time private key. The Merkle signature is the one-time signature together with an authentication path consisting of intermediate nodes from the tree.
- **Verify:** The one-time signature and authentication path are used to recover the root node of the tree. The signature is valid if the recovered value matches the public key.

NOTE: SPHINCS+ uses a hierarchical Merkle signature scheme with d layers of trees of height $h' \coloneqq h/d$. The root node of a tree in layer i is signed by a one-time signature from a tree in layer $i + 1$. The hierarchical signature is the sequence of d Merkle signatures. The hierarchical public key is the root node of the tree in layer d.

7.3.3.4 Few-time signature scheme

The FORS few-time signature is as follows:

- **KeyGen:** The few-time private key is k length-2^a sequences of n-bit values. The private values are hashed to form the leaf nodes of k binary hash trees of height a. The few-time public key is the sequence of n-bit values at the root nodes of the trees.
- **Sign:** The ak-bit message digest is used to select positions in each of the k sequences. The few-time signature is the sequence of k private values at these positions together with the corresponding authentication paths.
- **Verify:** The private values and authentication paths in the few-time signature are used to recover the root nodes of the trees. The signature is valid is the recovered values match the public key.

NOTE: The FORS few-time private keys are all derived from a single n-bit seed using PRF.

7.3.4 SPHINCS+.KeyGen

Input: None

Output: Public key $\mathrm{pk} \in \{0,1\}^n \times \{0,1\}^n$
Private key $\mathrm{sk} \in \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n$.

1) Sample two uniformly random n-bit private seeds $s_0, s_1 \in \{0,1\}^n$.

2) Sample a uniformly random n-bit public seed $p_0 \in \{0,1\}^n$.

3) Compute the Merkle public key p_1 for the single tree at level d derived from the private seed s_0 and public seed p_0.

The public key is $\mathrm{pk} := (p_0, p_1)$. The private key is $\mathrm{sk} := (s_0, s_1, p_0, p_1)$.

7.3.5 SPHINCS+.Sign

Input: Private key $\mathrm{sk} \in \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n$
Message $M \in \{0,1\}^*$

Output: Signature $\mathrm{sig} \in \{0,1\}^n \in \{0,1\}^{(a+1)kn} \times \{0,1\}^{(h+d\ell)n}$

1) Parse the private key as $\mathrm{sk} := (s_0, s_1, p_0, p_1)$.

2) Compute an n-bit randomizer R from the private seed s_1 and the message M using PRF.

3) Compute the message digest and index $(md, idx) := H_{\mathrm{msg}}(R \parallel p_0 \parallel p_1 \parallel M)$.

4) Compute the few-time signature sig_0 of the message digest md using the few-time private key at index idx derived from the private seed s_0 and public seed p_0.

5) Compute the few-time public key pk_0 at index idx derived from the private seed s_0 and public seed p_0.

6) For i from 1 to d:

   6.1) Compute the Merkle signature sig_i of the public key pk_{i-1} using the one-time private key at index idx on level i derived from the private seed s_0 and public seed p_0.

   6.2) Compute the Merkle public key pk_i for the tree at index $\mathrm{Floor}(idx/2^{h/d})$ on level i derived from the private seed s_0 and public seed p_0.

   6.3) Set $idx := \mathrm{Floor}(idx/2^{h/d})$.

The signature is $\mathrm{sig} := (R, \mathrm{sig}_0, \mathrm{sig}_1, \ldots, \mathrm{sig}_d)$.

NOTE: The SPHINCS+ specification allows the option of including additional input to the PRF when computing the randomizer in step 2.

7.3.6 SPHINCS+.Verify

Input: Public key $\mathrm{pk} \in \{0,1\}^n \times \{0,1\}^n$
Message $M \in \{0,1\}^*$
Signature $\mathrm{sig} \in \{0,1\}^n \times \{0,1\}^{(a+1)kn} \times \{0,1\}^{(h+d\ell)n}$

Output: $Accept$ or $Reject$.

1) Parse the public key as $\mathrm{pk} := (p_0, p_1)$.

2) Parse the signature as $\mathrm{sig} := (R, \mathrm{sig}_0, \mathrm{sig}_1, \ldots, \mathrm{sig}_d)$.

3) Compute the message digest and index $(md, idx) := \mathcal{H}_{\mathrm{msg}}(R \parallel p_0 \parallel p_1 \parallel M)$.

4) Recover the few-time public key pk'_0 at index idx from the few-time signature sig_0 for md using the public seed p_0.

5) For i from 1 to d:

   5.1) Recover the Merkle public key pk'_i for the tree at index $\text{Floor}(idx/2^{h/d})$ on level i from the Merkle signature sig_i for pk'_{i-1} using the public seed p_0.

   5.2) Set $idx \coloneqq \text{Floor}(idx/2^{h/d})$.

6) If $pk'_d = p_1$ then return $Accept$ else return $Reject$.

7.3.7 Parameters and performance

The SPHINCS+ submission [i.37] describes six main parameter sets (see Table 26). Each of these can then be instantiated with a choice of three different hash functions (SHA-256, SHAKE-256 or Haraka [i.38]) and two different tweakable hash function constructions (simple or robust) giving a total of 36 parameter sets.

Table 26: Proposed parameters for SPHINCS+

Parameter set	n	w	ℓ	k	a	h	d	Claimed security
128s-L1	128	16	35	14	12	63	7	Category 1
128f-L1				33	6	66	22	Category 1
192s-L3	192		51	17	14	63	7	Category 3
192f-L3				33	8	66	22	Category 3
256s-L5	256		67	22	14	64	8	Category 5
256f-L5				35	9	68	17	Category 5

These parameters lead to the private key, public key and signature sizes listed in Table 27.

Table 27: SPHINCS+ private key, public key and signature sizes

Parameter set	Private key	Public key	Signature
128s-L1	64 bytes	32 bytes	7 856 bytes
128f-L1			17 088 bytes
192s-L3	96 bytes	48 bytes	16 224 bytes
192f-L3			35 664 bytes
256s-L5	128 bytes	64 bytes	29 792 bytes
256f-L5			49 216 bytes

Tables 28 and 29 present the performance results for the SHAKE and SHA2 parameter sets. Additional parameters and performance are given for the Haraka hash function in annex E. All cycle counts were obtained from the AVX2 optimized implementation on an Intel® Xeon™ E3-1275 V3 processor with TurboBoost and Hyperthreading disabled.

Table 28: SPHINCS+ AVX2 performance figures (SHAKE)

Parameter set	KeyGen	Sign	Verify
SHAKE256-128s-simple-L1	144 000 000 cycles	1 100 000 000 cycles	1 190 000 cycles
SHAKE256-128s-robust-L1	274 000 000 cycles	2 080 000 000 cycles	2 410 000 cycles
SHAKE256-128f-simple-L1	2 250 000 cycles	56 900 000 cycles	3 350 000 cycles
SHAKE256-128f-robust-L1	4 270 000 cycles	106 000 000 cycles	6 680 000 cycles
SHAKE256-192s-simple-L3	206 000 000 cycles	1 910 000 000 cycles	1 650 000 cycles
SHAKE256-192s-robust-L3	398 000 000 cycles	3 550 000 000 cycles	3 300 000 cycles
SHAKE256-192f-simple-L3	3 220 000 cycles	89 900 000 cycles	4 780 000 cycles
SHAKE256-192f-robust-L3	6 180 000 cycles	167 000 000 cycles	9 330 000 cycles
SHAKE256-256s-simple-L5	136 000 000 cycles	1 650 000 000 cycles	2 560 000 cycles
SHAKE256-256s-robust-L5	258 000 000 cycles	2 980 000 000 cycles	4 670 000 cycles
SHAKE256-256f-simple-L5	8 540 000 cycles	177 000 000 cycles	5 030 000 cycles
SHAKE256-256f-robust-L5	16 300 000 cycles	330 000 000 cycles	9 720 000 cycles

Table 29: SPHINCS+ AVX2 performance figures (SHA2)

Parameter set	KeyGen	Sign	Verify
SHA-256-128s-simple-L1	85 000 000 cycles	645 000 000 cycles	861 000 cycles
SHA-256-128s-robust-L1	176 000 000 cycles	1 330 000 000 cycles	1 830 000 cycles
SHA-256-128f-simple-L1	1 330 000 cycles	33 700 000 cycles	2 150 000 cycles
SHA-256-128f-robust-L1	2 750 000 cycles	68 500 000 cycles	4 800 000 cycles
SHA-256-192s-simple-L3	125 000 000 cycles	1 250 000 000 cycles	1 440 000 cycles
SHA-256-192s-robust-L3	261 000 000 cycles	2 520 000 000 cycles	3 100 000 cycles
SHA-256-192f-simple-L3	1 930 000 cycles	55 300 000 cycles	3 490 000 cycles
SHA-256-192f-robust-L3	4 060 000 cycles	113 000 000 cycles	7 550 000 cycles
SHA-256-256s-simple-L5	80 900 000 cycles	1 030 000 000 cycles	1 990 000 cycles
SHA-256-256s-robust-L5	339 000 000 cycles	3 910 000 000 cycles	8 290 000 cycles
SHA-256-256f-simple-L5	5 070 000 cycles	109 000 000 cycles	3 560 000 cycles
SHA-256-256f-robust-L5	21 300 000 cycles	436 000 000 cycles	14 900 000 cycles

Annex A:
Security properties

Digital signatures are typically intended to provide authentication, integrity and non-repudiation of data. Assurance of achieving these goals is provided by demonstrating the resilience of a signature scheme under an attack model. Besides EUF-CMA, there are various attack models for digital signature schemes. These attacks are classified according to the resources available to the attacker and the capability that is targeted by an attacker.

Examples of attacker resources include (in order of increasing resources):

- **Key Only Attack (KOA)**. The attacker is provided with the public verification key.
- **Known Message Attack (KMA)**. The attacker can request valid signatures for a large number of messages, but is not able to choose the messages that are signed.
- **(Adaptive) Chosen Message Attack** (**CMA**). The attacker can request valid signatures of messages that are chosen by the attacker. This includes requesting valid signatures of messages chosen based on previous responses as well as requesting multiple signatures of the same message.

Examples of targeted capabilities include (in order of difficulty of obtaining the capability):

- **Key recovery**. The attacker can obtain the private signing key or an equivalent.
- **Universal Unforgeability (UUF).** Attacker's goal is to create valid signatures for any messages that were not included in a signature request.
- **Existential Unforgeability (EUF).** Attacker's goal is to create a valid signature for some message that was not included in a signature request.
- **Strong existential Unforgeability (SUF).** Attacker's goal is to create a valid signature for some message where this signature is not the same as any signatures returned by signature requests for that message.

The SUF condition allows the attacker to exhibit alternative signatures for a given message, which could be possible using signature malleability. Attack models are usually specified in abbreviated form giving the targeted capability and the resources in that order, e.g. the UUF-KMA attack model seeks to defend against an attacker able to perform known message attacks with the goal of creating selected (universal) forgeries.

Annex B:
Frameworks for constructing digital signatures

B.1 Hash-and-sign

A hash-and-sign signature scheme takes the following general format: the message is hashed to a target challenge and the signature is a solution to this challenge that the signer can compute using some trapdoor. The verifier hashes the message and checks that the signature is indeed a solution to the challenge.

G*e*MSS and Rainbow fall in this category and both follow the same multivariate signature design principle introduced by Matsumoto-Imai [i.28]. In both cases, the public key is given by a set of m multivariate quadratic polynomials $p_1, ..., p_m \in \mathbb{F}_q[x_1, ..., x_n]$. This public key is derived (via a secret affine change of variables) from a secret set of polynomials $f_1, ..., f_m \in \mathbb{F}_q[x_1, ..., x_n]$ that have a specific structure. Namely, for all $(y_1, \dots, y_m) \in \mathbb{F}_q^m$, it is possible to find a solution $(x_1, \dots, x_n) \in {\mathbb{F}_q}^n$ to the system:

$$\begin{aligned} f_1(x_1, \dots, x_n) &= y_1 \\ &\vdots \\ f_m(x_1, \dots, x_n) &= y_m \end{aligned}$$

in polynomial time.

A signature is essentially generated by inverting the polynomials $f_1, ..., f_m \in \mathbb{F}_q[x_1, \dots, x_n]$ while the verification process requires evaluating the public polynomials. The main difference between G*e*MSS and Rainbow is the method for constructing the secret inner polynomials $f_1, ..., f_m \in \mathbb{F}_q[x_1, ..., x_n]$. This choice can lead to different trade-offs between the size of the signature, the size of the public key and the efficiency of the schemes.

FALCON [i.20] uses the Gentry-Peikert-Vaikuntanathan (GPV) framework [i.21] for constructing hash-and-sign lattice-based signature schemes. This approach can be sketched as follows:

- The public key includes a full-rank matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n \times \mathrm{m}}$ $(m \geq n)$ generating a q-ary lattice.
- The corresponding private key is a matrix $\boldsymbol{B} \in \mathbb{Z}_q^{\mathrm{m} \times \mathrm{m}}$ generating the lattice orthogonal to the public lattice modulo q, i.e. such that $\boldsymbol{B}\boldsymbol{A}^{\boldsymbol{T}} = \boldsymbol{0}$.
- The signature of a message $M \in \{0,1\}^*$ is a vector s $\in \mathbb{Z}_q^{\mathrm{m}}$ that is short (in the sense of having a small Euclidian norm) and such that $s\ \boldsymbol{A}^{\boldsymbol{T}} = \mathcal{H}(M||r)$, where $\mathcal{H}$ is a hash function and $r \in \{0,1\}^*$ is random string (or salt).

The signer first computes an arbitrary $c_0 \in \mathbb{Z}_q^{\mathrm{m}}$ such that $c_0\ \boldsymbol{A}^{\boldsymbol{T}} = \mathcal{H}(M||r)$. This requires solving a linear system. The c_0 that will be found is not short enough in general. The matrix $\boldsymbol{B}$ is then used to generate a vector $v \in \mathbb{Z}_q^{\mathrm{m}}$ in the lattice orthogonal to the public lattice and which is close to c_0. Thus, $s = c_0 - v \in \mathbb{Z}_q^{\mathrm{m}}$ is short and verifies $s\boldsymbol{A}^{\boldsymbol{T}} = c_0\boldsymbol{A}^{\boldsymbol{T}} - v\ \boldsymbol{A}^{\boldsymbol{T}} = \mathcal{H}(M||r)$, i.e., s is a valid signature. The process of computing $v \in \mathbb{Z}_q^{\mathrm{m}}$ is called "*trapdoor sampling*" and an appropriate trapdoor sampler is used to ensure that the signature $s \in \mathbb{Z}_q^{\mathrm{m}}$ does not leak information about the short matrix $\boldsymbol{B}$.

B.2 Hash-based

Hash-based signatures [i.17] originate from the works by Lamport [i.14] and Merkle [i.15] in the late 1970s. The basic approach is to start from a One-Time Signature scheme (OTS) and build a binary hash tree, called Merkle tree, of height h on top of the hashes of 2^h OTS public keys. The hash at the root of the tree is the public key of the scheme.

Hash-based signatures are divided into two categories: *stateful* and *stateless* [i.17]. The number of messages that can be signed by a keypair in stateful hash-based signatures is strictly limited by the number of one-time signatures available. Due to this limit and the corresponding state management issues, stateful hash-based signatures were not considered by the NIST PQC standardization process.

The main advantage of hash-based signatures compared to other quantum-resistant signatures is the minimal security assumptions. These are typically variants of well-understood properties of hash functions such as pseudorandomness or second-preimage resistance.

B.3 Fiat-Shamir

The general idea of Fiat-Shamir signature schemes is to transform an Identification Scheme (IDS) into a signature via a generic transformation. An IDS is described as a set of interactions between a *prover* and a *verifier*. The verifier challenges the prover with several questions. By correctly answering the questions, the prover demonstrates to the verifier that they have knowledge of a particular secret.

Classically, an IDS can be turned into a signature scheme via the Fiat-Shamir technique [i.16], i.e. the signature is derived from the interactions between a prover and a simulated verifier. In general, the Fiat-Shamir paradigm produces signature schemes that are provably secure in the ROM. The Unruh transform [i.36] transforms a ZK-IDS into a digital signature scheme that is secure in the QROM.

The security of IDS-based signatures can be reduced in the ROM to the security of commitment schemes and random instances of a computational hard problem: finding a short vector in structured lattices for Dilithium or recovering the secret key of a block-cipher for Picnic.

Annex C:
Recent cryptanalysis results

C.1 Introduction

The present annex discusses recent attacks against round 3 candidates. This is not intended to be exhaustive but rather to mention the most impactful results that appeared during the writing of the present document.

C.2 Improved MinRank attacks against G*e*MSS and Rainbow

The security of G*e*MSS and Rainbow relies on the computational hardness of the PoSSo problem [i.17] and the MinRank problem [i.41]. Given a set of input matrices and a target rank r, the MinRank problem asks an attacker to find a linear combination of the input matrices that has rank at most r.

In [i.39], the author described an improved MinRank attack against Rainbow. The attack applies to the round 3 parameter sets of Rainbow (see Table 13) and reduces the security level of Rainbow I, III and V by 20 bits, 40 bits and 55 bits respectively (leading to 127, 177 and 226 bits of security, respectively). These are below the NIST security targets for the categories claimed in the submission (see clause 5.3). In a rebuttal [i.44], the Rainbow team confirmed that the attack was correct [i.39] but claimed that the Rainbow parameter sets from the submission were still sufficiently secure if the memory requirements were included in the cost of the attack. At this stage, the Rainbow submission [i.24] has not been updated to take this new attack into account.

In [i.40], the authors describe an improved MinRank attack against G*e*MSS. The attack significantly reduces the security of the parameters sets of G*e*MMS (see Table 19) below the NIST targets for the categories claimed in the submission (see clause 5.3). In [i.42], the authors suggest a tweak of G*e*MMS that mitigates the new attack. At this stage, the G*e*MSS submission [i.27] has not been updated to take these new results into account.

C.3 Algebraic attack against Picnic

In [i.43], the author describes an attack against Picnic using a new algorithm for solving PoSSo to recover the LowMC secret key. The algorithm [i.43] is faster than exhaustive search and reduces the claimed security of several Picnic parameters (see Table 23). However, the memory required by [i.43] is significant and so the impact of the attack is not clear. At this stage, the Picnic submission [i.32] has not been modified to take this new attack into account. The attack only affects LowMC instances with full Sbox layer and hence only applies to the Picnic3 and Picnic-L{ 1,3,5}-full instances.

Annex D:
Additional parameters for GeMSS

Table D.1: GeMSS private key, public key and signature sizes for additional parameters

Parameter set	Private key	Public key	Signature
GeMSS128	16 bytes	352 190 bytes	258 bits
BlueGeMSS128		363 610 bytes	270 bits
RedGeMSS128		375 210 bytes	282 bits
GeMSS192	24 bytes	1 237 960 bytes	411 bits
BlueGeMSS192		1 264 120 bytes	423 bits
RedGeMSS192		1 290 540 bytes	435 bits
GeMSS256	32 bytes	3 040 700 bytes	576 bits
BlueGeMSS256		3 087 960 bytes	588 bits
RedGeMSS256		3 135 590 bytes	600 bits

Table D.2: GeMSS AVX2 performance figures for additional parameters

Parameter set	KeyGen	Sign	Verify
GeMSS128	19 600 000 cycles	608 000 000 cycles	106 000 cycles
BlueGeMSS128	18 400 000 cycles	67 200 000 cycles	134 000 cycles
RedGeMSS128	16 300 000 cycles	2 050 000 cycles	141 000 cycles
GeMSS192	69 400 000 cycles	1 800 000 000 cycles	304 000 cycles
BlueGeMSS192	65 000 000 cycles	252 000 000 cycles	325 000 cycles
RedGeMSS192	57 100 000 cycles	5 970 000 cycles	335 000 cycles
GeMSS256	158 000 000 cycles	2 490 000 000 cycles	665 000 cycles
BlueGeMSS256	152 000 000 cycles	248 000 000 cycles	680 000 cycles
RedGeMSS256	143 000 000 cycles	8 760 000 cycles	709 000 cycles

Annex E:
Haraka parameters for SPHINCS+

Table E.1: SPHINCS+ AVX2 performance figures (Haraka)

Parameter set	KeyGen	Sign	Verify
Haraka-128s-simple-L1	30 075 604 cycles	240 763 926 cycles	308 774 cycles
Haraka-128s-robust-L1	37 113 806 cycles	304 905 780 cycles	432 066 cycles
Haraka-128f-simple-L1	482 332 cycles	12 196 792 cycles	799 808 cycles
Haraka-128f-robust-L1	587 548 cycles	15 176 760 cycles	1 072 774 cycles
Haraka-192s-simple-L3	46 369 950 cycles	481 682 614 cycles	480 264 cycles
Haraka-192s-robust-L3	63 387 838 cycles	718 896 354 cycles	759 952 cycles
Haraka-192f-simple-L3	732 770 cycles	21 433 286 cycles	1 205 698 cycles
Haraka-192f-robust-L3	998 446 cycles	30 866 288 cycles	1 799 300 cycles
Haraka-256s-simple-L5	28 822 310 cycles	451 164 660 cycles	696 980 cycles
Haraka-256s-robust-L5	40 954 800 cycles	677 039 436 cycles	1 046 096 cycles
Haraka-256f-simple-L5	1 809 078 cycles	41 973 226 cycles	1 252 598 cycles
Haraka-256f-robust-L5	2 599 368 cycles	61 706 762 cycles	1 854 540 cycles

History

Document history		
V1.1.1	September 2021	Publication