NIST Internal Report
NISTIR 8214C 2pd

NIST First Call for Multi-Party Threshold Schemes

Second Public Draft

Luís T. A. N. Brandão
René Peralta

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8214C.2pd

NATIONAL INSTITUTE OF
STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

NIST Internal Report
NISTIR 8214C 2pd

NIST First Call for Multi-Party Threshold Schemes

Second Public Draft

Luís T. A. N. Brandão [1*,2], René Peralta [1]

[1] Computer Security Division, Information Technology Laboratory, NIST, Gaithersburg, Maryland, USA
[2] Strativia, Largo, Maryland, USA

[*] NIST Associate (Foreign Guest Researcher, Contractor[2])

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8214C.2pd

March 2025

U.S. Department of Commerce
Howard Lutnick, Secretary

National Institute of Standards and Technology
Craig Burkhardt, Acting Under Secretary of Commerce for Standards and Technology and Acting NIST Director

NIST Technical Series Publications: https://www.nist.gov/nist-research-library/nist-publications

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in federal information systems.

References to Other Publications

There may be references in this publication to other publications currently under development by NIST in accordance with its assigned statutory responsibilities. The information in this publication, including concepts and methodologies, may be used by federal agencies even before the completion of such companion publications. Thus, until each publication is completed, current requirements, guidelines, and procedures, where they exist, remain operative. For planning and transition purposes, federal agencies may wish to closely follow the development of these new publications by NIST.

Organizations are encouraged to review all draft publications during public comment periods and provide feedback to NIST. Many NIST cybersecurity publications, other than the ones noted above, are available at https://csrc.nist.gov/publications

Non-Endorsement Disclaimer

Certain equipment, instruments, software, or materials, commercial or non-commercial, are identified in this paper in order to specify the experimental procedure adequately. Such identification does not imply recommendation or endorsement of any product or service by NIST, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

Call for Patent Claims

This public review includes a call for information on essential patent claims (claims whose use would be required for compliance with the guidance or requirements in this Information Technology Laboratory (ITL) draft publication). Such guidance and/or requirements may be directly stated in this ITL Publication or by reference to another publication. This call also includes disclosure, where known, of the existence of pending U.S. or foreign patent applications relating to this ITL draft publication and of any relevant unexpired U.S. or foreign patents. ITL may require from the patent holder, or a party authorized to make assurances on its behalf, in written or electronic form, either:

1. assurance in the form of a general disclaimer to the effect that such party does not hold and does not currently intend holding any essential patent claim(s); or
2. assurance that a license to such essential patent claim(s) will be made available to applicants desiring to utilize the license for the purpose of complying with the guidance or requirements in this ITL draft publication either:
    (a) under reasonable terms and conditions that are demonstrably free of any unfair discrimination; or
    (b) without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

Such assurance shall indicate that the patent holder (or third party authorized to make assurances on its behalf) will include in any documents transferring ownership of patents subject to the assurance, provisions sufficient to ensure that the commitments in the assurance are binding on the transferee, and that the transferee will similarly include appropriate provisions in the event of future transfers with the goal of binding each successor-in-interest.

The assurance shall also indicate that it is intended to be binding on successors-in-interest regardless of whether such provisions are included in the relevant transfer documents.

Such statements should be addressed to: nistir-8214C-comments@list.nist.gov

Publication History

The present version is the Second Public Draft.
Supersedes NISTIR 8214C ipd (January 2023), DOI:10.6028/NIST.IR.8214C.ipd.

Suggested Citation

Luís T. A. N. Brandão, René Peralta (2025). NIST First Call for Multi-Party Threshold Schemes. (National Institute of Standards and Technology, Gaithersburg, MD) NIST Internal Report (NISTIR) 8214C 2pd (Second Public Draft). DOI:10.6028/NIST.IR.8214C.2pd

Author Names and ORCID Identifiers

Luís T. A. N. Brandão (0000-0002-4501-089X); René Peralta (0000-0002-2318-7563).

Public Comments Period: March 27, 2025 – April 30, 2025

Public Comments Contact: nistir-8214C-comments@list.nist.gov

All comments are subject to release under the Freedom of Information Act (FOIA)

Abstract

This document calls for public submissions of multi-party threshold schemes, and other related crypto-systems, to support the United States National Institute of Standards and Technology (NIST) in gathering a public body of reference material on advanced cryptography. In a threshold scheme, an underlying cryptographic primitive (e.g., signature, encryption, decryption, key generation) is computed in a distributed manner, while a private/secret key is or becomes secret shared across various parties. Threshold schemes submitted in reply to this "NIST Threshold Call" should produce outputs that are "interchangeable" with a reference conventional (non-threshold) primitive of interest, from various categories organized into two classes: Class **N**, for selected **N**IST-specified primitives; and Class **S**, for **s**pecial primitives that are not specified by NIST but are threshold-friendlier or have useful functional features. The scope of Class S also includes fully-homomorphic encryption, zero-knowledge proofs, and auxiliary gadgets. This document specifies the requirements for submission (including specification, implementation, and evaluation), along with phases and deadlines. The ensuing public analysis will support the elaboration of a characterization report, which may help assess new interests beyond the cryptographic techniques currently standardized by NIST, and may include recommendations for subsequent processes.

Keywords

Crypto-systems; distributed systems; fully-homomorphic encryption (FHE); post-quantum cryptography (PQC); secure multi-party computation (MPC); threshold cryptography; threshold encryption; threshold schemes; threshold signatures; zero-knowledge proofs (ZKP).

Table of Contents

List of Tables

Preface

The exploration of "advanced" cryptography is a challenging endeavor from a standardization perspective, possibly requiring "advanced" or at least customized processes. *What and when to standardize or issue recommendations? Focus on building blocks or protocols? Which security properties to require? How to navigate a space of numerous possibilities?*

This "NIST First Call for Multi-Party Threshold Schemes" (or simply the "Threshold Call") establishes a process for international community engagement aligned with the development of reference material. The call stands at the threshold of advanced cryptography, putting forward a proactive exploration of advanced cryptographic techniques of increasing relevance and utility. Primarily, the call deals with threshold cryptography and related multi-party computation (MPC) techniques, allowing distribution of trust in the implementation of cryptographic primitives, such as those standardized by NIST. The call is also open to special types of primitives not standardized by NIST, including fully-homomorphic encryption (FHE) and zero-knowledge proofs (ZKP). The process aims to establish a high-quality body of reference material, to be analyzed with public engagement.

In order to promote transparency, involvement and collaboration, the Threshold Call:

- was widely disseminated in advance, via the publication of an initial public draft in 2023 and subsequent presentations to the community;
- was open to public comments, discussed in various workshops, and subject to further review via a second public draft;
- promotes a collaborative environment, by allowing a submission package to propose multiple crypto-systems, possibly developed by different subteams;
- encourages the early presentation of plans ("previews") for future package submissions, to enable public awareness and discussion while teams can still adapt.

Overall, this document is intended for: cryptography experts interested in providing constructive technical feedback; or in collaborating in the development of open reference material, technicians engaged in the development of recommendations for threshold schemes and advanced cryptography; and those, in academia, industry, government and the general public, who are interested in future recommendations about threshold schemes.

Acknowledgments

The authors thank their NIST colleagues Harold Booth, Chris Celi, Lily Chen, Michael Davidson, John Kelsey, Dustin Moody, Ray Perlner, Andrew Regenscheid, Angela Robinson, Hamilton Silberg, and Meltem Sönmez Turan, for their feedback on diverse aspects of this call. The authors also thank Isabel Van Wyk, from NIST, for various editorial comments. Additionally, the authors thank the 32 people that submitted feedback [NIST-IR8214C-ipd-comms] in response to the January 2023 initial public draft (ipd) of this publication, and the many other researchers who demonstrated interest throughout various conversations at events and through invitations for related presentations.

Note to the Reviewers

Do not submit threshold schemes yet. This publication [NIST-IR8214C-2pd] is a second public draft (2pd). Submissions should wait until the final version is published.

The initial public draft (ipd) [NIST-IR8214C-ipd] was released in January 2023, requesting public comments and encouraging early preparation of potential submissions. A compilation of received comments is available on the publication webpage hosted by the NIST Computer Security Resource Center (CSRC). Further public feedback was obtained during the NIST Multi-Party Threshold Schemes (MPTS) 2023 workshop [MPTS-2023].

This 2pd has substantial changes (see list in Appendix E) compared with the ipd, although the type of material needed for submission is largely similar. The publication of this new public draft opens an additional period for public comments, before the final version is released. Related comments for community discussion are welcome via the MPTC-forum. However, public comments intended for consideration during the final revision of the document should be sent by April 30, 2025 to nistir-8214C-comments@list.nist.gov. Teams are encouraged to prepare for upcoming submissions using the present draft as a baseline.

1. Introduction

1.1. A Variety of Cryptographic Schemes and Primitives

Traditional techniques. For several decades, the United States National Institute of Standards and Technology (NIST) has standardized important cryptographic schemes, in various Federal Information Processing Standards (FIPS) publications, and in Special Publications (SP) in Computer Security (the SP 800 series). For example, they specify digital signatures [FIPS-186-5], public-key encryption (e.g., for key-encapsulation) [SP800-56B-Rev2], pair-wise key-agreement [SP800-56A-Rev3], symmetric-key encryption (e.g., based on block-ciphers [FIPS-197]), hashing [FIPS-180-4; FIPS-202], and message authentication [SP800-224-ipd; SP800-185; SP800-38B].

Recent standards. In the last decade, NIST also started the Post-Quantum Cryptography (PQC) process [Proj-PQC] to devise new standards for post-quantum key-encapsulation methods [FIPS-203] and (stateless) signatures [FIPS-204; FIPS-205], and started the Lightweight Cryptography (LWC) process [Proj-LWC] to select primitives for a new standard for authenticated encryption with associated data and hashing-style primitives [NIST-IR8454].

Advanced cryptography. Beyond standardization, NIST has also taken an exploratory interest in the areas of privacy-enhancing cryptography [Proj-PEC] and multi-party threshold cryptography [Proj-MPTC]. These areas of advanced cryptography deal with secure multi-party computation (MPC), which enables collaborations while ensuring correctness, privacy and composability within more complex systems. Other important techniques in scope are fully homomorphic encryption (FHE) and zero-knowledge proofs (ZKP). The related state of the art continues evolving based on important developments by the cryptography community, including from academia and industry.

Secret/private keys. The mentioned schemes include: key-generation (KeyGen) primitives, which output a private or secret key; key-based primitives (e.g., signing, decryption, enciphering, which require a private or secret key as input); and key-less primitives (e.g., hashing) that can nonetheless be applied to keys with secrecy requirements. In a traditional specification or implementation of these schemes/primitives, the operations are usually considered as performed by an individual party (e.g., a computing device) with access to the private or secret key. In a conventional implementation, that party is a *single-point of failure* for confidentiality, integrity and availability.

Threshold schemes. Modern cryptography enables a multi-party implementation paradigm based on developments in the fields of threshold cryptography MPC and distributed systems. In a (multi-party) *threshold* scheme, multiple parties perform a distributed computation that emulates the operation of a cryptographic algorithm, but without combining the private/secret key in any single place. This paradigm enables the decentralization of trust regarding the creation, storage and use of the private/secret keys. For appropriately defined notions of security, the interaction remains secure as long as the number of corrupted parties does not exceed a certain corruption *threshold*.

1.2. The NIST Threshold Call

The development of recommendations for threshold schemes, tapping into the domain of advanced cryptography, is an important step in addressing various challenges in cybersecurity and privacy. The increasing relevance of advanced cryptography warrants a proactive exploration [NIST-IR7977], in this case by opening a solicitation for structured inputs, to be open for for public analysis.

The present "NIST First Call for Multi-Party Threshold Schemes" (or simply the "Threshold Call") is expected to motivate broad community engagement and result in a diverse set of high-quality submissions, followed by expert public scrutiny. The submissions will include specifications (with technical description and security analysis), implementations and experimental evaluation. The collection to be gathered is intended to form a public body of open reference material, whose analysis will help identify sound approaches, best practices, and reusable building blocks. The results will help shape future recommendations and guidelines. More concretely, the Threshold Call has the following goals:

1. **Reference material:** Create a basis of properly motivated, specified, implemented and analyzed threshold schemes, to support future recommendations and guidelines.
2. **Threshold feasibility:** Assess the viability of threshold implementations of various cryptographic primitives of interest, including selected NIST-specified primitives.
3. **Pertinence of other primitives:** In the threshold context, facilitate an initial assessment of the merits of other cryptographic primitives that may be mature for adoption.
4. **Quantum resistance and other features:** Examine the threshold readiness of post-quantum cryptography and other advanced functional features.

Two classes in scope. To assess the viability of threshold schemes for cryptographic primitives, the scope of the Threshold call is organized into two classes, each with various categories (as listed in Table 1) of primitives in consideration for thresholdization:

- **Class N: N**IST-specified cryptographic primitives (including quantum-vulnerable and post-quantum) used in digital signature schemes, public-key encryption schemes, symmetric-key encryption, message-authentication codes, hashing, and key-generation (also including elliptic-curve based primitives for pair-wise key-establishment).

- **Class S: S**pecial primitives not specified by NIST, including threshold-friendly primitives for schemes of the same type as in Class N, and others for fully-homomorphic encryption, zero-knowledge proofs of knowledge, and auxiliary gadgets.

Table 1. Multiple categories per class

	Sign	**PKE**	**Symm**	**KeyGen**	**FHE**	**ZKPoK**	**Gadgets**
Class N	N1	N2	N3	N4	—	—	—
Class S	S1	S2	S3	S4	S5	S6	S7

Legend: Class N = **N**IST-specified; Class S = **S**pecial, not specified by NIST. FHE = Fully-Homomorphic Encryption. KeyGen = Key Generation. PKE = Public-Key Encryption. Symm = Symmetric. ZKPoK = Zero-Knowledge Proof of Knowledge.

The analysis of threshold schemes for NIST-specified primitives (i.e., in Class N) will help assess threshold friendliness and develop future recommendations highlighting reference approaches, techniques, building blocks, and best practices. The analysis in Class S will help assess new interests in primitives that are not currently standardized by NIST, and characterize the possible alignment between (i) threshold-friendliness, (ii) post-quantum readiness, and (iii) additional useful features. This may also become useful input to assess the readiness to deploy MPC for applications with advanced privacy requirements.

Overall, this incursion is intended to clarify the feasibility and security of various techniques in advanced cryptography, including an analysis of threshold friendliness.

Dissemination and feedback. Recent NIST context leading to the formulation of this Threshold Call can be found on the Multi-Party Threshold Cryptography (MPTC), and Privacy-Enhancing Cryptography (PEC) project websites, the NIST-IR8214A (2020) with considerations toward criteria, the MPTC-Call2021a for feedback on criteria for multi-party threshold schemes (MPTS), the MPTS 2020 workshop webpage, the NIST-IR8214B-ipd on threshold EdDSA/Schnorr signatures (2022), and the MPTS 2023 workshop webpage.

Since the publication of the initial public draft in January 2023, the idea of the "Threshold Call" has also been widely disseminated and socialized with the cryptography community, though talks and conversations at international conferences, including conferences organized by the International Association for Cryptologic Research (IACR).

Organization. Section 2 calls for submissions and explains the partition into two classes. Section 3 conveys a vision about the collaborative process, including the context of post-quantum migration, interchangeability, provable security, and a variety of options. Section 4 enumerates the phases and tentative deadlines. Section 5 explains the logistic requirements for the written specification. Section 6 discusses the required reference implementation. Section 7 asks for an experimental evaluation. Section 8 presents requirements about classic and quantum security strength and threshold security. Sections 9 and 10 specify submission requirements per category. Appendices A and B give informative details about primitives in scope. Appendix C suggests a baseline system model for the threshold setting, and comments on threshold security, profiles, and input/output interfaces. Appendix D defines the acronyms used in the document. Appendix E lists changes between the two public drafts.

2. Scope of the Call: Two Classes

This is a public **call** for high-quality submissions of multi-party threshold schemes for selected types of primitive across two classes: Class N (**N**IST-specified) and Class S (**s**pecial others). The latter also includes auxiliary components, such as zero-knowledge proofs of knowledge (ZKPoKs) and threshold-useful *gadgets*, which do not need to be thresholdized.

The term "**crypto-system**" is used in an encompassing sense to denote a main system submitted for analysis. A crypto-system can be a threshold scheme, a ZKPoK, a gadget, a Class S conventional scheme, or a family thereof (i.e., including variants or modes).

A submitted crypto-system ***shall*** be organized in a package that includes (i) a written specification with design rationale, technical description and security characterization, (ii) an open-source reference implementation with instructions, and (iii) a performance evaluation. The submission ***shall*** follow the baseline expectations set forth in Section 4.4, including a disclosure of known related patent claims associated with any team member. Submitted packages that are accepted for posting on the NIST-MPTC website are expected to benefit from exposure to public analysis. The collected reference material will inform the development of NIST recommendations and future processes, and will be referenced in a future NIST publication.

2.1. Class N: NIST-Specified Primitives

Class N consists of selected NIST-specified cryptographic primitives. As a mnemonic, "N" is associated with "**N**IST". The class is organized into four categories: N1 for signing; N2 for [regular] public-key encryption (PKE); N3 for symmetric-key and hashing-related primitives; and N4 for KeyGen and for non-PKE PKC-primitives useful for pair-wise key-agreement (2KA). The primitives from some recent and emerging standards for post-quantum (PQ) signatures and PKE (selected by the NIST PQC project [Proj-PQC]), and for "lightweight" (for constrained environments) symmetric-key and hashing-related schemes (selected by the NIST LWC project [Proj-LWC]), also fit into Class N.

Categories. Table 2 lists the various categories and their scope, including various "families of specifications". Each such family may include diverse primitives and modes/variants (including the options for input/output interfaces mentioned in Appendix C.4).

Table 2. Families of specifications of interest in categories of Class N

Category: Type	**Subtype**	**Families† of specifications**	**Sections in this Call**
N1: Signing	QV	EdDSA sign, ECDSA sign, RSADSA sign	9.1, A.1
	PQ	ML-DSA sign, HBS (SLH-DSA and stateful) sign	
N2: PKE	QV	RSA encrypt, decrypt	9.2, A.2
	PQ	ML-KEM encrypt, decrypt	
N3: Symmetric	Blockcipher	AES encipher, decipher	9.3, A.3
	AEAD	Ascon-AEAD encrypt, decrypt	
	Hash/XOF	SHA2, SHA3, SHAKE, Ascon-{Hash,XOF}	
	MAC	C/G/H/K-MAC	
N4: KeyGen	QV-PKC	ECC KeyGen (including for 2KA), RSA KeyGen	9.4, A.4
	PQ-PKC	ML KeyGen, HBS KeyGen	
	RBG	Random bit generation (e.g., bitstring, integer)	

See more details in Section 9 and Appendix A. **Legend:** 2KA = Pair-Wise Key Agreement. AES = Advanced Encryption Standard. C/G/H/K-MAC= Cipher/Galois/Hash/Keccak-based Message Authentication Code. ECC = Elliptic Curve Cryptography. ECDSA = Elliptic Curve DSA. EdDSA = Edwards-curve DSA. HBS = Hash-Based Signatures. KEM = Key Encapsulation Mechanism. ML = Module Lattice. PKC = Public-Key Cryptography. PKE = Public-Key Encryption. PQ = Post-Quantum. QV = Quantum Vulnerable. RBG = Random-bit generation. RSA = Rivest–Shamir–Adleman. RSADSA = RSA DSA. SHA = Secure Hash Algorithm. SHAKE = SHA with Keccak. SLH = Stateless Hash. XOF = eXtendable Output Function.

In each category, each family of specifications relates to at least one NIST publication:

- N1 for the signing operation in EdDSA, ECDSA, RSADSA [FIPS-186-5], ML-DSA [FIPS-204], SLH-DSA [FIPS-205], and Stateful HBS [SP800-208]. Future signature schemes (e.g., FN-DSA [FIPS-206-ipd]) may be added here once NIST-standardized.
- N2 for encryption and decryption primitives used in PKE schemes based on RSA [SP800-56B-Rev2] or ML-KEM [FIPS-203]. The scope may be broadened in the future, to encompass upcoming NIST-specified PQ-KEMs (e.g., based on HQC).
- N3 for symmetric-key AES encipher/decipher [FIPS-197] and Ascon-AEAD encrypt/decrypt [SP800-232-ipd]; for hashing and XOF'ing [FIPS-180-4; FIPS-202; SP800-

232-ipd]; and for MAC'ing based on ciphers [SP800-38B; SP800-38D], hash functions [SP800-224-ipd], and the Keccak permutation [SP800-185].

- N4 for KeyGen for ECC primitives [FIPS-186-5; SP800-56A-Rev3; SP800-186], RSA [FIPS-186-5; SP800-56B-Rev2], PQ-PKE (ML, SLH, FN, HBS), and random-bit generation (bitstrings or integers) [SP800-90A-R1; SP800-90B; SP800-90C-4PD].

2.2. Class S: Special Primitives Not Specified by NIST

The goal of Class S is to enable submissions that make a strong case for relevant primitives that are not standardized by NIST. As a mnemonic, "S" is associated with "**s**pecial". Submissions of threshold schemes for primitives in Class S ***shall*** be justified on the basis of differentiating features of the primitives (i.e., in comparison with those in Class N), such as: (i) being threshold-friendlier (TF); (ii) relying on alternative cryptographic assumptions (e.g., pairings), possibly PQ (e.g., lattice-based); (iii) having useful properties (e.g., deterministic, probabilistic, or enabling a useful homomorphism); or/and (iv) being more efficient in a relevant metric.

Categories. Class S has four regular categories (i.e., matching the categories in Class N), and three others (FHE, ZKPoK and Gadgets). Table 3 lists the categories in the first column. The second column shows corresponding types of conventional schemes (i.e., not threshold). The third column gives examples of primitives of interest. The ZKPoK and gadgets categories do not have to consider threshold versions of their underlying primitives.

Table 3. Examples of primitives in categories of Class S

Category: Type	**Example types of scheme**	**Example primitives**	**Sections in this Call**
S1: Signing	TF (threshold-friendly)-PQ signatures	Signing	10.1
|	TF succinct & verifiably-determ. signatures	|	|
S2: PKE	TF-PQ public-key encryption (PKE)	Decrypt, Encrypt	10.2
S3: Symmetric	[Keyed] TF cipher/PRP	Encipher, Decipher	10.3
|	[Keyed] TF PRF/MAC	Tag Generate (Gen)	10.3
|	[Keyless] TF hashing and XOF'ing	Hash, XOF	|
S4: KeyGen	Any of the above	KeyGen	10.4
S5: FHE	Fully-homomorphic Encryption (FHE)	Decrypt, KeyGen	10.5, B.1
S6: ZKPoK	ZKPoK of private key, ZKPoK of signature	ZKPoK.Gen	10.6, B.2
S7: Gadgets	Garbled circuit (GC)	GC.Gen, GC.Evaluate	10.7

See more details in Section 10 and Appendix B. **Legend:** determ. = deterministic. MAC = Message Authentication Code. PRF/PRP= Pseudorandom Function/Permutation [family]. PQ = Post-Quantum. XOF= Extendable Output Function. ZKPoK = Zero-Knowledge Proof of Knowledge.

The following list enumerates the categories identified in Table 3:

- S1 for signing (e.g., TF-PQ, or succinct and verifiably deterministic).
- S2 for PKE (e.g., TF-PQ decryption and encryption).
- S3 for "symmetric" primitives, including keyed (e.g., TF enciphering/deciphering and MAC'ing) and keyless (e.g., hashing and XOF'ing).
- S4 for KeyGen for primitives in other categories, including non-PKE PKC-primitives usable for multi-party key-establishment.
- S5 for **f**ully-**h**omomorphic **e**ncryption (FHE).
- S6 for **z**ero-**k**nowledge **p**roofs (or arguments) of **k**nowledge (ZKPoK) of secret information (e.g., a private key, consistent with a public key or with a correct secret-sharing setup) relatable to the threshold setting or other categories of the call.
- S7 for other auxiliary "gadgets" deemed useful to support the threshold setting, namely useful for implementation of threshold schemes in scope.

3. Vision

The scope of the Threshold Call is expected to motivate the submission of a variety of MPC techniques for threshold schemes, and of various underlying primitives that are not NIST-standardized (including ZKPoK, FHE, and gadgets). Overall, this call initiates a process whose success relies on the public engagement and collaboration of many experts (see Section 3.1). The combination of PQ and QV techniques (see Section 3.2) will enable an observation of the "quantum gap" (the distance between PQ and QV solutions), which is especially relevant in the ongoing setting of PQC-migration. There are also possible tradeoffs across different threshold schemes that are *interchangeable* with the same conventional primitive (see Section 3.3). The submissions are expected to include security modeling, and proofs of security, taking advantage of the wealth of knowledge in modern cryptography (see Section 3.4). The variety of choices (e.g., in system models, threshold profiles, security formulations) made across submissions will also contribute to an exploration of the threshold space (see Section 3.5).

3.1. Reliance on Contributions and Collaboration

The success of the process envisioned by this Threshold Call depends on: **high-quality submissions** by teams with cryptography expertise, including in the areas of multiparty computation and distributed systems; **expert public scrutiny**, including assessments of security; and **comments on pertinence**, by stakeholders of applications of threshold schemes.

A complex submission can specify various crypto-systems proposed by distinct sub-teams (see Section 5.4). This possibility can be followed as a way of facilitating collaboration between teams, better modularization of crypto-systems, reducing redundancy in submissions and corresponding analysis, and promoting consistency in terminology and notation.

Given the collaborative process, it is also important to set clear expectations about the reviewability and usability of submitted material, the ability to produce derivative work and redistribute it, and the applicability of known related patent claims (see Section 4.4).

3.2. Post-Quantum and Quantum-Vulnerable Cryptography

This Threshold Call welcomes submissions of PQ solutions (i.e., with security resistant against an adversary with a quantum computer) and quantum-vulnerable (QV) solutions (i.e., yet secure with respect to adversaries without a quantum computer). However, submitters should be aware of the ongoing PQC-migration context, namely the planned "disallowance" of NIST-standardized QV-signatures and QV-PKE, by 2035 [NCCoE-PQC; NIST-IR8547-ipd]. Correspondingly, the standalone use of threshold schemes for those QV primitives will eventually not be recommended by NIST. Still, "disallowance" in this context does not preclude a future use of "hybrid" schemes, in which a QV scheme and a PQ scheme coexist, with the security of either ensuring the security of the composite system.

While QV primitives are in use, their threshold implementation can be valuable. Furthermore, assessing the state of the art in QV threshold schemes and in QV threshold-friendly primitives can be useful as a reference to set future goals for PQ threshold cryptography.

This Threshold Call values the exploration of various combinations of post-quantum readiness and quantum vulnerability, across conventional primitives and their threshold schemes. Each combination can be of interest when properly motivated. Four examples:

1. **QV-QV:** A **QV threshold scheme** with security reducible to the same crypto assumptions as required by the **QV conventional primitive** being thresholdized.
2. **PQ-PQ:** A **PQ threshold scheme** used to distribute trust in a PQ application of a **PQ conventional primitive** being thresholdized.
3. **QV-PQ:** A **QV threshold scheme** for an application that requires PQ only for the final output of a **PQ conventional primitive** (e.g., PQ signature), whereas the adversary is assumed to be pre-quantum during the threshold protocol execution.

4. **PQ-QV:** A **PQ distributed-KeyGen (DKG)** for generating secret-shared private and public keys (i.e., a DKG) of a **QV conventional KeyGen** primitive being thresholdized, so that the QV public-key is not exposed to quantum adversaries.

3.3. Interchangeability

This Threshold Call is interested in threshold schemes whose output can be *interchangeably* used (see §2.4 of NIST-IR8214A) by subsequent operations (e.g., signature verification) of a conventional (i.e., non-threshold) primitive (e.g., signing) in scope. EdDSA signing provides a notable example of the relevance of defining *interchangeability*: in the threshold setting, producing a valid randomized signature can be much more efficient than obtaining the standardized pseudorandom one, and both are *interchangeable* with respect to the conventional/standardized EdDSA verification algorithm (see NIST-IR8214B-ipd).

3.4. Provable Security

The security of submitted threshold schemes (see Section 8.2 and Appendix C.2.3) is expected to be assessed based on multi-party protocol analysis, which is supported by a substantial body of knowledge in *provable security*. This is different from the extensive cryptanalysis that would be required in a call for basic primitives based on new cryptographic assumptions. That said, the security of threshold schemes is still recognized as multi-dimensional, depending on security formulation (e.g., which ideal functionalities or security games to choose), implementation (e.g., susceptibility to side-channels), and deployment suitability.

3.5. A Variety of Options

The domain space of multi-party threshold schemes is considerably wider than that of the primitives (e.g., digital signatures) being thresholdized. Acknowledging this, the present Threshold Call allows leeway for submitters to select from a variety of system models, threshold configurations, security formulations, technical approaches, and benchmarking focuses. Intentionally, this Threshold Call does **not** put forward a rigid "apples-to-apples" criteria (e.g., specific number of parties, common programming language, application programming interface) for comparison across submissions. Nonetheless, the submissions are expected to adhere to certain criteria, with respect to technical documentation (see Sections 5, 6, and 7), and security requirements (see Section 8), such as security against active corruptions in the threshold setting. The options followed across submissions will be informative.

4. Phases and Deadlines

Table 4 lists the phases and corresponding deadlines for (i) the submission of *previews* (i.e., early plans for package submission), (ii) the submission of packages, and (iii) the analysis process. See details in Sections 4.1, 4.2, and 4.3. Section 4.4 further discusses some agreements implied by submitting a package.

Table 4. Submission phases and tentative deadlines

Phase	Subphase	Required?	Deadline
Ph1: Previews	Ph1.1: Preview 1	No	$\approx X+2$
	Ph1.2: Preview 2	No	$\approx X+5$
Ph2: Packages	Ph2.1: Preliminary submission	No	$\approx X+6$
	Ph2.2: Regular submission	**Yes**	$\approx X+8$
Ph3: Analysis	Ph3.1: Public presentations	**Yes**	(2026)
	Ph3.2: Package updates	—	—
	Ph3.3: NIST-MPTC report	**Yes**	($\approx$ 2027)

Tentative deadlines are relative (unit in months) to the publication date (X) of the final version of this Call.

On deadlines. If, during the process, NIST-MPTC [Proj-MPTC] has a compelling reason to extend a deadline, the corresponding update will be publicly conveyed via the MPTC-forum. Similar communication will be used for other deadlines that may emerge during the unfolding process (e.g., eventual phases for updating packages).

4.1. Ph1: Previews

The "Previews" phase provides two opportunities for each team to publicly share in advance their plan (or preliminary plan) to submit full packages. This is intended to (i) facilitate collaboration across teams; (ii) promote the identification of opportunities for teams to strengthen their composition; (iii) and form an early expectation of the coverage of categories of the Threshold Call, which may help determine useful mergers or differentiation of packages. The submission and public presentation of a preview are **strongly encouraged**, aligning well with the open and collaborative spirit of the process.

- **Ph1.1: Preview 1 (Optional).** A preview is accomplished by submitting a short writeup (the plan) and later giving a related presentation in a public session.

  - **Submission of a plan.** By $\approx X + 2$, send an email to MPTC-submissions@list.nist.gov, attaching a document in portable document format (PDF), with no more than four pages (letter size, 1 inch margins, 12-point font size), with a title, a list of confirmed team members, and a summarized description of the "package" planned for later submission (Ph2). Suggested items to include: (i) a list of the crypto-systems to be proposed (i.e., chosen names, and (sub)categories), (ii) an outline of the main technical considerations (system model, protocol approach, security properties), (iii) comments on the intended reference implementation, and (iv) a list of relevant bibliographic references. The references do not count for the 4-page limit.
  - **Public presentation.** Shortly after the Preview 1 submission deadline, NIST-MPTC will organize a public session for each team to present their plan for package submission, and showcase their preparation status. For each presentation, there will be time for comments and questions from the public. NIST-MPTC will post online (publicly available) the writeup of the preview plan, the presentation slides, and the audio-video recordings.

- **Ph1.2: Preview 2 (Optional).** Similar to Preview 1 but with a new deadline: $\approx X + 5$. This is open to present new plans, and to revise or give a status update on previously presented plans.

4.2. Ph2: Packages

A complete and proper package ***shall*** contain the following main components:

- **Written specification:** A technical specification (including security analysis) of the crypto-systems (threshold scheme, ZKP, gadget, or/and Class S conventional scheme) proposed for analysis (see Section 5).
- **Reference implementation:** A software implementation of the proposed crypto-systems, including the open-source code, with an open-source license, comments for clarity, scripts, and instructions (see Section 6).
- **Experimental evaluation:** A report describing an experimental setting, measuring performance, and interpreting the results (see Section 7).

Submission medium. The submission of packages requires sending an email to MPTC-submissions@list.nist.gov, with each of the requested components. The reference implementation will be submitted by indicating a cryptographic hash and URL to a public Git-compatible repository controlled by the team, as described in Imp2 (see Section 6.2).

Subphases. The phased of package submissions is organized as follows:

- **Ph2.1: (Optional) Preliminary submission.** Packages received by NIST-MPTC by $\approx X + 6$ will be subject to a superficial review for completeness. Within 30 days, the submitters will be notified of identified deficiencies or other suggestions, to allow for amendments before the deadline.
- **Ph2.2: Regular submission.** Packages received by NIST-MPTC by the submission deadline ($\approx X + 8$) will be considered in the process of analysis. After a period expected to be no longer than 30 days, the accepted packages will be hosted on a NIST repository, and corresponding hyperlinks will be posted on the NIST-MPTC project website [Proj-MPTC].

4.3. Ph3: Public Analysis of Crypto-Systems

After the public posting of accepted packages, a period of public analysis will follow.

- **Ph3.1: Public presentations/discussion.** NIST-MPTC intends to host a seminar series for thorough presentations of the proposed crypto-systems, and to consider comments from the public.
- **Ph3.2: Future updates.** The NIST repository will allow public download of the hosted packages, and will also list links to the team's repository (external to NIST). After the submissions, each team can use the MPTC-forum to announce discovered issues, and possible updates incorporated in the team's repositories. As the process unfolds, NIST-MPTC may, infrequently, specify periods for optional submission of amendments or improvements, to update the body of NIST-hosted reference material.
- **Ph3.3: NIST-MPTC report.** It is expected that a follow-up NIST report will characterize the set of proposed crypto-systems, and assess a possible interest in future processes with more-focused analysis. This should clarify distinctions across primitives, threshold schemes, building blocks, and composition techniques.

4.4. Expectations about Submitted Material

This Threshold Call is intended to gather a public body of reference material, open for review and evaluation, to serve as a basis for potential developments and improvements, and foster a widespread, secure use of cryptographic techniques. Therefore, the process requires its participants to abide by baseline expectations about the openness of submitted material.

By submitting a package in reply to this Threshold Call, the submitting team (i.e., all its members, possibly across various subteams) **acknowledges and agrees** that the submitted material — written specification (document), reference implementation (code), and experimental evaluation (document) — is in accordance with the following expectations:

4.4.1. Original work. The submitted content is the original work of the submitters, except (where applicable) for the properly referenced and credited (i) fair-use inclusion of externally developed text from other publications, and/or (ii) externally developed open-source code (e.g., compilers and some libraries).

4.4.2. Available specification, and evaluation report. The submitted specification and experimental evaluation report (PDF files) are made freely available worldwide for public review and evaluation purposes. To enable this, by making a submission to NIST, the submitting team agrees that NIST is explicitly granted the right to post submitted materials online within the scope of the Threshold process, and the public is explicitly granted the right to use the material posted by NIST, including for commercial purposes. To allow further redistribution by other parties, teams are **encouraged** (but not required) to submit their documents with a license, such as the Creative Commons "CC BY 4.0 International" (Attribution 4.0 International) license [CC-BY-4.0]. If a different license is used, for example choosing to include share-alike terms, it ***shall*** not attempt to restrict use for commercial purposes.

4.4.3. Availability of the implementation code. The submitted implementation code ***shall*** be accompanied with an open source license (consistent with an "Open Source Initiative" approved license [OSI-lic]), as further explained in Section 6.3 (Imp3).

4.4.4. Disclosure of Patent Claims. Since patent claims may affect the adoption and development of techniques [ITL-Patent-Policy], it is important to solicit their disclosure. Therefore, the submitted specification ***shall*** include a statement (in Cv4; see Section 5.1) that: (i) discloses any known issued or pending patent (foreign or domestic) that does or could have claims that may cover the contents of the submission, where any team member is one of the inventors, applicants, or assignees, or is sponsored by or affiliated with an entity that holds the corresponding patent rights, or (ii) states that no such patent claims are known to exist. Regarding known related patents held by third-party stakeholders, i.e., not including anyone in the submitting team, the disclosure is encouraged to take place via the MPTC-forum or/and in public presentations along the process.

4.4.5. Security guarantees. It is expected that a good faith technical effort is made to ensure the security of the proposed schemes. If the team becomes aware of vulnerabilities after the submission, then it ***shall*** communicate them via the MPTC-forum.

5. Package Component: Written Specification

The first main component of a submission package is the written specification. This section describes its organization in "parts" and "sections".

PDF File. The written specification ***shall*** be submitted as a single digital file, in PDF, preferably named "`<team-name>-spec-v1.pdf`". The document ***shall*** be written in English, aided by mathematical notation where appropriate. An (optional to follow) LaTeX-based template will be provided when the final version of the Threshold Call is published, to help achieve the intended structure, and exemplify accessibility features (e.g., PDF bookmarks for easy navigation, hyper-references, tooltipped acronyms, tagged content).

Content organization. The content ***shall*** be organized into various parts, as follows:

- Cover and verso (see Cv1–Cv4 in Section 5.1);
- One *front matter* (see Fm1–Fm2 in Section 5.2);
- One *main matter* part called "Preliminaries" (see Pre1–Pre4 in Section 5.3);
- One *main matter* part for each crypto-system (see CSx.1–CSx.6 in Section 5.4);
- One *back matter* (see Bm1–Bm2 in Section 5.5).

The use of this common structure across specifications will facilitate their parsing. Authors are encouraged to further organize the content into subsections, sub-subsections and even some paragraphs with numbered headings for easy reference. (The Cv, FmX, PreX, CSx.X and BmN indices shown here are not meant for the actual specification.)

5.1. Covers and Verso

A sequence of unnumbered pages, as follows:

Cv1. Cover page: A cover page with: title (and optional subtitle) of the submission, submission date, submission version ("1.0" by default), team title, names of the submitters, names (and (sub)category indices) of the proposed crypto-systems, and optional document licensing (e.g., "CC BY 4.0 International"; see Section 4.4).

Cv2. Contacts: A page that (i) repeats the submission title and subtitle, the team title, the submitters' names, and the names of proposed crypto-systems, and (ii) additionally includes the team's mailing list (i.e., single email address that relays emails to all team members); hyperlinked ORCID identifiers and applicable affiliations of all team members, and (iii) identifies a primary contact person and (optionally) up to two secondary contact persons (identifying their email address and postal address). This page can include additional disclosures of contributions and funding. In case of multiple submitted crypto-systems, the page can identify one primary contact per crypto-system.

Cv3. Submission scope: A page that enables a quick glimpse at the submission scope. For each proposed crypto-system, it will identify the system name and the corresponding (sub)category(ies) of the Threshold Call, the "part" in which it is specified in the document, the proposing subteam, and (as applicable) a list of the main protocols or building blocks that compose each crypto-system (or family), and the [sub][sub]sections in which they are specified.

Cv4. Patents disclosure: A section disclosing the known related patent claims where any team member is one of the inventors, applicants, or assignees, or is sponsored by or affiliated with an entity that holds the corresponding patent rights (see Section 4.4).

5.2. Front Matter

A sequence of unnumbered sections (in roman-numbered pages), as follows:

Fm1. Abstract: A text with 200 to 350 words, describing the technical scope of the submission, and hinting at their main features, cryptographic assumptions and performance highlights of the submitted crypto-systems.

Fm2. Index of contents: A table of contents (TOC, i.e., index of sections, subsections, etc.); and (as applicable) lists of tables (LOT), figures (LOF), algorithms (LOA), and other relevant indexed components (e.g., list of design decisions), including corresponding hyper-references and page numbers.

Fm3. Preface: A 1-page section, without mathematical notation, briefly describing the motivation for the submission, including: the envisioned relevance (utility, applicability, deployability) of the proposed crypto-systems (e.g., in industry and for societal applications), considering the state of the art. It can also comment on the process and personal context of package development.

Fm4. Acknowledgments: An optional section with acknowledgments.

Fm5. Executive summary: An abridged explanation of the package content, highlighting relevant properties of the proposed threshold schemes, their applicability and performance, and other key insights. It should also describe the main challenges addressed while preparing the submission, including in the specification (e.g., in proving security), the open-source implementation, and the performance evaluation. Mathematical symbols should be used sparingly in this section, if at all.

5.3. Main Matter — Preliminaries

The main matter starts with a “Preliminaries” part, containing a sequence of decimal-numbered sections, in decimal-numbered pages, as follows:

Pre1. Introduction: An introduction that identifies the involved cryptographic primitives, the proposed crypto-systems, hints at the used technical approaches, and describes the high-level structure of the document (to help potential readers prioritize which sections to read).

Pre2. Notation: A section that lists and explains the used acronyms (§2.1), mathematical symbols (§2.2), and terms (§2.3). It ***may*** contain subsections to explain certain relatioships between symbols (e.g., sets, elements, operations).

Pre3. Related work and design decisions: A section with rationale about the proposed system model(s) and crypto-system(s). It ***shall*** **identify** building blocks, techniques, and ideas known to have been developed or authored in prior or related work, and that are used in or have directly influenced the specification of the crypto-systems proposed in the submission. It will include proper attribution and citations (see also Bm1), clarifying which works/authors (whether or not part of the submitting team) developed the discussed techniques. As deemed useful, this section can intertwine descriptions of related work, and related design choices. This modularized section is intended to allow later parts of the specification to be more straightforward.

Pre4. Conventional primitives and building blocks: A section that explains the interface, and properties of the building blocks and/or other technical components that the authors prefer to modularize away from CSx.3, but which are nonetheless used in at least one proposed crypto-system. For example, this can apply to recalling the conventional (non-threshold) Class N primitive that will be the subject to a proposal of threshold scheme. The *explanation* of some building blocks here is meant to be thorougher than a possible *identification* in Pre3. It should also be understandable without reading Pre3, which is more concerned with providing proper attribution and rationale.

5.4. Main Matter — Crypto-Systems

Each crypto-system (i.e., threshold scheme, ZKPoK, gadget, or Class S conventional scheme) proposed for standalone analysis ***may*** itself be a family of protocol/primitive variants (e.g., a family of threshold schemes for one conventional primitive, using different protocols to handle different threshold profiles, or with different probabilistic features). Teams are encouraged to favor modularity and team collaboration, possibly identifying different sub-teams for different crypto-systems. Each crypto-system ***shall*** satisfy the applicable requirements indicated in Sections 8, 9, and 10, and ***shall*** be specified within a standalone "Part" of the specification document, including the following sections:

CSx.1. Cover page: A cover page that indicates: the crypto-system name, the version and date of its last revision, the subteam composition (if different from the entire team); the names of the main algorithms, protocols, and/or variants being proposed; and the names of the building blocks (which can be referenced to the Preliminaries part or another crypto-system).

CSx.2. System model: A description of the system model (see Appendix C.1), including participants and their activation, communication network, and adversary.

Note: This section is intended to be straightforward about the chosen system model (in contrast with Pre3 that explains related work and design decisions). It can also give hints about the security formulation (e.g., whether a protocol abort is a valid outcome) and the critical safety properties of the intended system (e.g., key secrecy and unforgeability). However, a formal security formulation (functionalities and security games) should be deferred to CSx.4.

CSx.3. Proposed crypto-system: A detailed description of the algorithms and/or protocols that constitute the crypto-system (possibly a family) proposed for analysis. The building blocks used in the algorithms/protocols need to be understandable from at least one of the following (whichever applies): (i) interface and properties Pre4 described in the "Preliminaries" part); (ii) a thorough specification in another "crypto-system" part of this specification; or (iii) a modular specification in a (sub)subsection of this section (i.e., of CSx.3). The protocol can also be described with various phases (e.g., offline, online, secret resharing), which may have differentiated requirements.

CSx.4. Security analysis: A security assessment of the proposed crypto-systems, covering the requirements in Section 8. It will include a security formulation (e.g., ideal functionalities or games); an identification of assumed ideal components and other assumptions (cryptographic, and/or of trusted setup); a security proof sketch (small subsection) and a thorough security proof (which can go into Bm2); a discussion of security consequences of instantiating ideal components in a realistic (possibly not ideal) manner, and of deployment in environments that are (e.g., without synchrony) different from those assumed in the system models.

CSx.5. Analytic complexity: An analytical estimation of the (i) memory complexity, (ii) computational complexity, (iii) communication complexity, and (iv) round complexity of each proposed crypto-system. See Section 7.2 about experimental evaluation. As applicable, the estimates ***should*** include: a breakdown across various phases of the protocol; the complexity per party and for the entire system; and the functional dependence on configurable parameters, e.g., security strength, number of parties and the thresholds.

CSx.6. Deployment: A set of deployment requirements and recommendations, including those related to security, as well as a list of known and proposed applications of the submitted crypto-systems.

5.5. Back Matter

Bm1. References: A list of external references cited throughout the document. Where possible, each reference ***should*** include a persistent identifier (e.g., DOI, and ia.cr) hyperlinked to a preferably free and publicly available version of the reference. The use of author-year format is suggested for citation tags.

Bm2. Appendices: Optional sections with auxiliary elements that may be deemed too detailed or cumbersome for the main matter. For example, this may include complicated proofs of lemmas needed by the proof(s) of security in CSx.4).

6. Package Component: Reference Implementation

The second main component of a submission package is the open-source Packaged Codebase, which comprises the Team's Core Code and Bundled Dependencies. The Reference Implementation is what emerges from combining the Packaged Codebase with well-identified External Dependencies and compiling them in a Baseline Platform. The Reference Implementation ***should*** enable testing the main features of each specified crypto-system.

Terminology. The following implementation-related components are distinguished:

- **Team's Core Code:** The code developed by the team to execute the specified crypto-systems. Its compilability depends on a software environment (with dependencies).
- **Bundled Dependencies:** Externally developed **open-source** libraries (i.e,. not part of the team's core code) that, for convenience, have been bundled together with the Team's Core Code (i.e., cloned from another repositories into the team's repository).

- **Packaged Codebase (or "submitted code"):** The combination of the Team's Core Code and the Bundled Dependencies, made publicly available in a Git-compatible repository in the team's control.
- **External Dependencies:** Externally developed **open-source** libraries that were not included in the Packaged Codebase, and are not part of the operating system, but are needed for compiling or executing the Reference Implementation.
- **Deployment Package:** The combination of the Packaged Codebase and the External Dependencies. For distinction purposes, the operating system is not considered part of the Deployment Package, but rather part of the Baseline Platform.
- **Reference Implementation:** The result of compiling and/or running the Deployment Package within a Baseline Platform (i.e., hardware and operating system), to test a proposed crypto-system.

Summary of requirements. The Packaged Codebase ***shall*** satisfy the following:

- Imp1: Implements the proposed crypto-systems (see Section 6.1)
- Imp2: Is publicly available (see Section 6.2)
- Imp3: Is licensed as open-source (see Section 6.3 and 4.4)
- Imp4: Is compatible with a Baseline Platform (see Section 6.4)
- Imp5: Its External Dependencies are open source and well-identified (Section 6.5)
- Imp6: Is clear, including inline comments (see Section 6.6)
- Imp7: Includes useful scripts (see Section 6.7)
- Imp8: Includes useful instructions (see Section 6.8)

6.1. Imp1. Crypto-system(s) Implementation

When compiled together with External Dependencies and executed in the Baseline Platform, the open-source Packaged Codebase ***shall*** constitute a reference implementation of the crypto-systems (see Section 5.4) proposed in the submitted specification, including the applicable building blocks. In the case of a multi-party threshold scheme, the software ***should*** enable running each "party" as one process (or more), or within a software virtual container, separate from the other parties.

Networking versus cryptography. There can be significantly different challenges between (i) implementing networking between parties (see Appendix C.1.2) and (ii) implementing certain mathematical operations (cryptographic building blocks) per party. Neglecting any of these implementation aspects can lead to serious vulnerabilities. Therefore, a **strong alignment** between the proposed system model (see CSx.2 and Appendix C.1) and the provided implementation is **strongly encouraged**, notwithstanding possible virtualizations to enable execution in a personal computer (the Baseline Platform). For example, if a protocol specification relies on broadcast, then the provided implementation ***should*** instantiate it in alignment with the assumptions of the proposed system model. If the proposed system model depends on special hardware components (e.g., a router) beyond the threshold "parties", then the submission ***may*** include code for simulating the special component.

6.2. Imp2. Code Availability

The Packaged Codebase ***shall*** be publicly available via a public Git repository, where "Git" denotes the distributed version-control system. The **email** with the package submission ***shall*** include a SHA256 or SHA3-256 cryptographic commitment to a .zip archive of the Packaged Codebase, and the following auxiliary metadata: a **URL** to the .zip file; the **byte-size** of the .zip file (for preliminary checking), the SHA-256 or SHA3-256 hash of the .zip file, the **Git-commit hash** (an identifier string with at least 40 hexadecimal characters) of the given commit stage of the Git project; and a **cloning command** (valid long-term) for cloning the repository at the given commit-stage (i.e., with content matching the one in the committed .zip file). The URL to the .zip file is expected to have a syntax similar to:

```
https://<repo-hosting-server>/<team-name>/<repo-name>/archive/<Git-commit-hash>.zip
```

6.3. Imp3. Code Licensing and Posting

The Packaged Codebase ***shall*** be explicitly licensed as open-source [OSI-def], consistent with an "Open Source Initiative" (OSI) approved license [OSI-lic]. Thus, the code will be freely distributable. In particular, NIST-MPTC will publicly post the code on a NIST-controlled repository, and will include a reference to the team's public Git-repository (which can be continuously updated by the team).

If the license of the submitted code is found to have an issue preventing the intended posting, then NIST-MPTC may request the team to adapt the license.

6.4. Imp4. Compatibility With a Baseline Platform

The Deployment Package (i.e., the bundle of Packaged Codebase and External Dependencies) ***shall*** be compilable and executable as a Reference Implementation in a *baseline* **p**lat**f**orm consisting of a modern personal computer (possibly virtualized) equipped with:

1. **Central processing unit (CPU):** Eight x64 (64-bit) processing cores
2. **Fast primary memory:** 32 gigabytes (e.g., of random-access memory [RAM])
3. **Secondary memory (storage):** Four terabytes (e.g., in a solid state drive [SSD])
4. **Operating system:** Ubuntu Desktop 24.04.1 LTS (codenamed "Noble Numbat" and offering **l**ong-**t**erm **s**upport) [Ubuntu]

This is meant to be a platform on which all submitted implementations can be analyzed, but is not an indication of preference or recommendation for which platforms are best suitable for implementing any crypto-system. Teams are welcome, but not required, to report (e.g., in Imp7 and Imp8) lighter or easier setups (e.g., lighter open-source operating systems) with which their submitted code can be compiled and executed. Experimental evaluation results can also be provided for additional platforms (see Section 7.1).

6.5. Imp5. External dependencies

The compilation of the Reference Implementation ***may*** use External Dependencies (e.g. a compiler), which ***shall*** be licensed as open-source [OSI-lic]. Their exclusion from the Packaged Codebase may be motivated by license incompatibilities (e.g., copyleft versus permissive), or another reason (e.g., an inconveniently large size in bytes).

Precise version identification. For testing and future reproducibility, the open-source External Dependencies ***shall*** be publicly available. The code ***should*** be precisely identified (e.g., their version) and (preferably) available in a Git-compatible public repo. See connection with the requirements about a build script (Imp7.X1) and compilation instructions (Imp8.Inst1). However, this precise version identification ***should*** be considered with **practical wisdom**: an effort is expected, but if too challenging (e.g., due to unclear nested external dependencies) it should not hinder achieving a good Reference Implementation. Further recommendations about this aspect may emerge from public feedback and the upcoming experience of analysis of implementations.

Containers. Teams are welcome (but not required) to make available or explain how to build container images and/or virtual machines that would run their code in an exact environment, including all dependencies with specific versions.

6.6. Imp6. Clear Code

The Team's Core Code ***shall*** promote clarity about the Reference Implementation of the proposed crypto-systems, even if at detriment of some performance. It ***should*** accompany most functions/modules with auxiliary explanatory comments, Optionally, additional code that is **optimized for performance** but less clear can be included to showcase better experimental performance. Similar considerations are possible for the selected externally developed dependencies. The Packaged Codebase ***shall*** clearly distinguish between the Team's Core Code and Bundled Dependencies, and explain the functionalities that are added by the External Dependencies.

Language, compiler and API. This Threshold Call intentionally refrains (see Section 3.5) from specifying a concrete concrete programming language, compiler, or application programming interface (API). However, the Packaged Codebase ***should*** include rationale for the choices made, which ***should*** not come at the cost of clarity.

Validation and verification. This Threshold Call does not require formal verification or validation of implementations. However, it is expected that, during the phase of analysis, the public scrutiny of submitted implementations will contribute to clarifying suitable testing mechanisms across various types of submitted crypto-systems, which can promote the production of high-assurance software. The specification of a parameter set per security level may help reduce the complexity of required testing combinations. For example, if a hash function is used as a building block, then by specifying a specific one the testing may be simpler than in the case where all NIST-approved hash functions would have to be tested. The webpage of the NIST Cryptographic Algorithm Validation Program (CAVP) [CAVP] includes information about validation testing for various NIST-approved cryptographic algorithms.

6.7. Imp7. Useful Scripts (X)

The Team's Core Code ***shall*** incorporate a set of useful scripts, as follows:

X1. Build script: A script, which can be executed with a single command in the Baseline Platform, to automatically download the needed External Dependencies (if applicable), and perform the code compilation required to later execute/test the proposed crypto-systems. Teams are encouraged to strive for a script that can obtain the External Dependencies with a specific version, in order to favor reproducible results (see Inst1). The team ***may*** include an additional script designed to use the most-up-to-date version of the External Dependencies (which may later lead to non-working implementations, absent further adjustments of the Packaged Codebase).

X2. KAT-script: A script to automatically execute the crypto-systems in a way that reproduces the set of known-answer test (KAT) values provided for sanity checking (see Inst3 and Inst4 in Section 6.8).

X3. Benchmark script: A script to automatically benchmark the crypto-system in the Baseline Platform, to produce performance measurements (similar to those required in M3, in Section 7) for various configurations. If the Packaged Codebase includes additional code optimized for performance, and whose performance results are reported in M3, then the corresponding scripts ***should*** also be provided, to facilitate reproducibility of results.

X4. Other scripts (optional): Additional scripts that are useful for gaining insights or better testing the crypto-systems or underlying primitives.

6.8. Imp8. Useful Instructions (Inst)

The Reference Implementation ***shall*** include a set of useful instructions:

Inst1. Compilation instructions: A `README.md` file that explains:

- (a) How clone and checkout from the team's Git-compatible repository the submitted version of the Packaged Codebase.
- (b) How to execute the build script (X1) that downloads the External Dependencies (specific or most recent versions) and compiles the Reference Implementation.
- (c) Which files configure the parameters (see Inst2) for crypto-system execution and/or testing, and which files (see Inst3) describe how to execute/test the proposed crypto-systems.

Inst2. Parametrization instructions: A file (possibly named `PARAMETERS`) or files that explain how to configure execution parameters, such as the number of parties, the corruption threshold, the type of communication channels, some adversarial choices, and some client choices (e.g., input to the cryptographic primitive, such as message to be signed). Preferably, the configuration of each parameter ***should*** be possible via the editing of a human-readable text file, and/or command line arguments.

Inst3. Execution instructions: A file (or files) that explains how to run the benchmark script (see X3), and test various phases/modules/primitives of the crypto-systems.

Inst4. KAT values and API: With discretionary depth and thoroughness, a set of KAT values, and an API description, to facilite (i) testing, correctness verification, and interoperability, (ii) use in higher-level applications, (iii) performance comparison with other implementations with similar API.

7. Package Component: Experimental Evaluation

The third main component of a submission package is an experimental evaluation of the Reference Implementation. Its report (a PDFfile) ***shall*** describe the experimental setting (see §7.1), provide performance measurements (see §7.2), and interpret the results (see §7.3).

7.1. Experimental Setting. The report ***shall*** describe the relevant characteristics of the implementation platform, namely the (possibly emulated) hardware, including the processor (e.g., instruction set, number of processors, and clock frequency), communication network (e.g., bandwidth, and latency), and memory (e.g., speed, and space). Preferably, the use platform ***should*** be similar to the Baseline Platform. If applicable, the report ***shall*** identify noteworthy differences, and explain whether/how they are expected to affect performance. The experimentation ***may*** also include additional platforms.

7.2. Measurements. The experimental evaluation ***should*** report on:

- **Perf1. Memory complexity** (in number of bytes simultaneously stored).
- **Perf2. Processing time** (in seconds and/or number of cycles).
- **Perf3. Communication complexity** (in number of communicated bytes).
- **Perf4. Round complexity** (in number of inbound and outbound messages).

Each metric ***should*** be evaluated across a representative set of configurations supported by each proposed crypto-system. The measurements ***should*** be reported: (i) per main phase of the protocol, and in total across an execution; (ii) per party and collectively. There ***should*** be at least one comparison between a run with all honest parties, and a run with at least one corrupted party. It may be insightful to also identify the cases where processing and communication are or can be pipelined to reduce latency.

The results can be reported across various configurations, such as various numbers of parties, and various security strengths. The batch of measurements ***should*** be obtainable automatically by running a simple command for executing the benchmark script (see X3 and X4).

7.3. Analysis. The performance analysis ***should*** include a written explanation of the experimental results, interpreting the expected and unexpected observations, namely in comparison with the analytic complexity described in CSx.5 (see Section 5.4). For example, a correlation may be expected between a complexity metric and the number of parties in a threshold scheme. The analysis of results across different configurations is expected to be useful to understand, test of confirm scalability and tradeoffs. The analysis ***may*** also include comparisons with the known performance of other relatable schemes.

8. Security Requirements

The submission of a crypto-system ***shall*** instantiate (i.e., specify, implement and measure) at least one concrete parametrization. Section 8.1 discusses general goals of security strength, with regard to computational and statistical complexity. Section 8.2 specifies requirements about the threshold setting.

Critical safety properties. The security requirements in this section are meant to apply to critical safety properties, to be identified in CSx.2 (see Section 5.4). For example, key secrecy is always assumed to be critical. The criticality of other properties depend on the crypto-system at stake. For example, unforgeability is a critical safety property for signature schemes. Properties not deemed critical ***may*** be sacrificed (e.g., a security with abort notion sacrifices availability in the presence of a malicious adversary). (Naturally, in practice the criticality of some properties may also depend on the use case.)

8.1. Security Strength Levels

Table 5 lists three parameters of security strength: classic computational, quantum computational, and statistical. For each parameter, there is a required lower bound of security strength, and a suggested lower bound for an optional second instantiation (for comparison).

Table 5. Security strength parameters

Security parameter	**Required (1st case)**	**Suggested (2nd case)**
κ (Classic computational)	$\geq$128	$\geq$192
θ (Quantum computational, if claimed PQ)	$\geq$1	$\geq$3
σ (statistical)	$\geq$40	$\geq$64

8.1.1. Computational Security

Classic security levels. A submitted crypto-system ***shall*** include (i.e., specify, implement, and evaluate) at least one instantiation with classic security strength κ approximate to or larger than 128 bits (i.e., $\kappa \gtrsim 128$). Preferably (when applicable, but not required), the submission includes two instantiations, one with $\kappa \approx 128$, and another with $\kappa \gtrsim 192$.

Quantum security levels. The five PQC security categories [PQC-Call-2016, §4.A.5], with levels $\theta \in \{1, 2, 3, 4, 5\}$ represent the computational resources required to break AES-128, SHA3-256, AES-192, SHA3-384, and AES-256, respectively. Here, a break means key-recovery for AES, and finding a collision for SHA3.

A submitted crypto-system claimed to be PQ ***shall*** include at least one instantiation with PQ security $\theta \geq 1$. Preferably (but not required), when applicable, the submission presents two instantiations: one with $\theta \leq 2$, and another with $\theta \geq 3$.

Parameter sets. For each category in Class N, the parameter sets in scope already satisfy $\kappa \gtrsim 128$ or $\theta \geq 1$ (see Section 9). For Class S, the submission needs to specify at least one such parameter set. This applies to ZKPoKs, gadgets, submitted threshold schemes, and their conventional primitives.

In the interest of research, the computational security of a submitted threshold scheme does not need to be as high as that of the primitive being thresholdized. Also (see Section 3.2), the PQ/QV property of the threshold scheme does not have to match the one of the conventional primitive. In any case, submissions ***should*** make a strong case for the adoptability of the proposed instantiations.

Two contrasting examples: A submission of threshold AES-256 enciphering (A.3.1):

- ***May*** use a QV threshold scheme with classical security $\kappa \approx 128$ bits.
- ***May*** use a PQ threshold scheme with quantum security $\theta = 5$.

8.1.2. Statistical Security

The security of a protocol (e.g., some threshold schemes and interactive ZKPoKs) can also depend on a statistical parameter σ (the additive inverse of the binary logarithm of the probability that a security property is broken during a protocol execution). A submitted scheme ***shall*** aim to achieve $\sigma \gtrsim 40$. Preferably, it ***should*** have $\sigma \gtrsim 64$. See Appendix B.2.4 on transforming statistical security into computational security.

8.2. Security of Threshold Schemes

The following applies to threshold schemes submitted as crypto-systems (see Section 5.4).

8.2.1. Threshold Profile

- The system model (CSx.2) ***shall*** define at least one "threshold" profile applicable to the threshold scheme. See informative notes in Appendix C.3. The term "threshold" is used for convenience, but the actual access structure ***may*** be different.
- The security analysis (CSx.4) ***shall*** clarify which thresholds apply to which main security properties. The analsyis ***should*** also characterize the breakdown that occurs when threshold-profile assumptions are broken.

8.2.2. Type of Adversary

The security analysis (CSx.4) ***shall*** specify an adversary, that is:

1. **active (malicious)**, i.e., able to corrupt parties up to one (or various) corruption threshold(s), controlling them to deviate from the prescribed multi-party protocol;
2. **adaptive**, i.e., able to choose which parties to corrupt after observing some of the protocol execution; and
3. **mobile**, i.e., persistently attempting to corrupt parties across multiple executions of the main protocol.

8.2.3. Security Against an Adversary

The required security analysis (CSx.4) entails formulating an **ideal functionality** (e.g., in the ideal-real simulation paradigm, within the universal composability framework) or/and an idealized **game** (or set of games) that defines the capabilities and goals of an adversary. Since security analysis is a multi-dimensional exercise, it ***may*** include several security formulations/idealizations, which serve as reference to assess the security of a crypto-system.

With regard to critical safety properties, and considering the confines of at least one threshold profile specified by the team, a proposed threshold scheme ***shall*** aim to achieve security against the modeled adversary, as follows:

1. **Active security (against active corruptions).** Various active security nuances are possible, including security with abort, where a malicious party has the ability to break the availability of the cryptographic primitive. The latter is permissible for settings where availability is considered a non-critical property.

2. **Adaptive security (against adaptive corruptions).** There is a strong preference for **adaptive security**, in contrast to *static* only, with respect to critical safety properties, even if some other security properties are only satisfied against a static adversary. (See Appendix C.2.2 for notes on practical feasibility.)
3. **Compatibility with recovery mechanisms (against mobile attacks).** A submitted threshold scheme is **not required** to include recovery mechanisms that attempt to identify, remove or replace (recover) corrupted parties. However, the submission ***should*** discuss how it envisions possible augmentations to integrate mechanisms for **proactive** or **reactive** recovery, which are important for handling a persistent **mobile** adversary that continuously attempts to corrupt more parties. For example, with respect to refreshing secret shares, a solution can be based on a modularized phase of secret-resharing (see S7), while also specifying the needed conditions (e.g., requirement of some initial/final agreement by a qualified quorum) for its integration.

9. Requirements for Class N Schemes

As listed in Section 2.1 (Table 2), Class N considers four categories (types of primitive). The present section specifies requirements for the submission of threshold schemes for primitives in each of those categories: signing (N1; see Section 9.1), PKE (N2; see Section 9.2), symmetric (N3; see Section 9.3), and KeyGen (N4; see Section 9.4).

9.1. Category N1: Signing

Signing primitives in scope. The third column of Table 6 lists the various signing primitives of interest. For table succinctness, "[Hash]" denotes optional pre-hashing (to differentiate between "pure" and "pre-hashed" versions); "[Det-]" indicates a possible deterministic variant; $\{...\}$ denotes a set of options. Appendix A.1 provides additional details.

Interchangeability and security level. A submission within category N1 ***shall*** specify a threshold signature that is *interchangeable* (see Section 3.3) w.r.t. (with regard to) verification of the conventional NIST-specified signature scheme, and ***shall*** provide at least one implementation with $\kappa \gtrsim 128$, or $\theta \geq 1$, consistent with the parameters described in Table 6. The security analysis (CSx.4) ***shall*** also characterize the type of unforgeability achieved, and whether a non-aborting adversary can bias the signature value.

Table 6. Signing primitives in category N1

Subcategory: Specification	**NIST reference**	**Signing primitives to thresholdize**	**Cryptographic parameters**		**§ in this call**
			$\kappa \approx 128$ **or** $\theta = 1$	$\kappa \gtrsim 192$ **or** $\theta \geq 3$	
N1.1: EdDSA	[FIPS-186-5]	[Hash]EdDSA.Sign	Edwards25519	Edwards448	A.1.1
N1.2: ECDSA	|	[Det-]ECDSA.Sign	P-256	P-{384,521}	A.1.2
N1.3: RSASSA	|	RSASSA-PSS	$\|N\| = 3,072$	$\|N\| \geq 7,680$	A.1.3
|	|	RSASSA-PKCS-v1.5.Sign	|	|	|
N1.4: ML-DSA	[FIPS-204]	[Hash]ML-DSA.Sign_Internal	ML-DSA-44	ML-DSA-{65,87}	A.1.4
N1.5: SLH-DSA	[FIPS-205]	[Hash]SLH-DSA.sign	{SHA-256, SHAKE128}	{SHA-512, SHAKE256}	A.1.5.1
N1.5: LMS, XMSS	[SP800-208]	{LMS, XMSS}.Sign	{SHA-256, SHAKE256}	—	A.1.5.2

Legend: See related legend of Table 2. Det = **det**erministic. LMS = Leighton-Micali Signature. PSS = Probabilistic Signature Scheme. PKCS = Public-Key Cryptography Standards. RSASSA = RSA Signature Scheme with Appendix. XMSS= eXtended Merkle Signature Scheme. The elliptic curves (Edwards and P) are specified in SP800-186. [Hash] = Optional consideration of the pre-hashed variant.

Probabilistic versus deterministic signature schemes. In a probabilistic mode (e.g., RSASSA-PSS), two signings of the same input message yield two different signatures. Deterministic modes may be verifiably deterministic (e.g., RSASSA-PKCS-v1.5) or not (e.g., EdDSA, Det-ECDSA). In the case of (i) **conventional** probabilistic signatures, and (ii) **conventional** non-verifiably-deterministic signatures, a submission of **threshold scheme** (interchangeable w.r.t. verification) ***may*** opt between probabilistic and pseudorandom (PR) modes, including Prob, Q-PR, and F-PR (described ahead).

Threshold modes w.r.t. (non-)determinism. The mechanism by which the secret randomness (or pseudorandomness) is selected in the threshold signing scheme, combining contributions from the various parties, determines one of the following possible modes:

1. **Prob: Prob**abilistic (via a random or hybrid contribution per party)
2. **Q-PR: P**seudo**r**andom per **q**uorum (e.g., via a ZKP of PR contribution per party)
3. **F-PR: F**ully **p**seudo**r**andom (e.g., based on a distributed PRF computation)

If the conventional signature is deterministic per standard, but not verifiably so, then the F-PR mode (deterministic even if the quorum changes) can still be distinguished between:

- Functionally equivalent (FE), distributing the PRF computation (e.g., via MPC)
- Not FE, yet fully deterministic (e.g., by implementing a threshold-friendlier PRF)

Note: In the ML-DSA case, the signing primitive of interest for thresholdization is **ML-DSA.Sign_internal**. Submissions are also welcome to showcase an extension to for [Hash]ML-DSA signatures.

9.2. Category N2: PKE (Encryption/Decryption)

PKE primitives in scope. The third column of Table 7 lists the PKE-related encryption and decryption primitives of interest to thresholdize. From RSA-based pair-wise key-exchange (2KE) [SP800-56B-Rev2], the primitives in focus for thresholdization are the RSAEP and RSADP exponentiations from the "textbook" RSA crypto-scheme. From ML-KEM [FIPS-203], the primitives in focus for thresholdization are those from the underlying K-PKE scheme. However, the encryption primitive K-PKE can be considered in a probabilistic variant that ignores the seed and, where applicable, uses threshold determined randomness (or some other pseudorandomness). Appendix A.2 provides additional details.

Table 7. PKE primitives in category N2

Subcategory: Specification	**NIST reference**	**PKE primitives to thresholdize**	**Cryptographic parameters**		**§ in this call**
			$\kappa \approx 128$ **or** $\theta = 1$	$\kappa \gtrsim 192$ **or** $\theta \geq 3$	
N2.1: RSA-2KE	[SP800-56B-Rev2]	RSAEP	$\|N\| = 3,072$	$\|N\| \geq 7,680$	A.2.1
|	|	RSADP	|	|	|
N2.2: ML-KEM	[FIPS-203]	K-PKE.Encrypt	ML-KEM-512	ML-KEM-{768,1024}	A.2.2
|	|	K-PKE.Decrypt	|	|	|

Legend: See related legend of Table 2. 2KE = Pair-wise Key exchange. PKE = Public-Key Encryption. K-PKE = ML-KEM-related Public-Key Encryption. RSADP = RSA Decryption Primitive. RSAEP = RSA Encryption Primitive.

Threshold interfaces and security level. A submission within category N2 ***shall***, for at least one of the primitives listed in Table 7, specify at least one of the following:

- **Threshold scheme for** `Encrypt`, which ***shall*** be SSI w.r.t. the plaintext m, and ***should*** be NSS w.r.t. the public encryption key. This provides a secret-sharing protection of the secret plaintext before encryption, without hiding the "public" key. See Appendix A.2.2 about also hiding internal randomness, when applicable. An SSO mode w.r.t. the ciphertext ***may*** also be considered.
- **Threshold scheme for** `Decrypt`, which ***shall*** be SSI w.r.t. the private decryption key; ***may*** be SSI or not (default) w.r.t. the ciphertext; ***may*** be SSO or not w.r.t. the plaintext. The SSO-plaintext mode can be useful for a threshold receiver in a 2KE.

Additionally, the submission ***shall*** provide at least one implementation with $\kappa \gtrsim 128$, or $\theta \geq 1$, consistent with the parameters described in Table 7.

Threshold higher-level primitives. A threshold scheme in N2 ***may*** also showcase how to use or adapt it to the corresponding higher-level primitives:

- RSASVE.{Generate, Recover} or RSA-OAEP.{Encrypt, Decrypt} (see §A.2.1.2), based on {RSAEP,RSADP}.
- ML-KEM.{Encaps, Decaps} (see §A.2.2.2), based on K-PKE.{Encrypt,Decrypt}.
- NIST-standardized KAS/KTS/KEM. However, an interchangeable threshold implementation of one side of a full-fledged NIST-standardized 2KE protocol would require thresholdizing threshold-unfriendly symmetric primitives (N3) used for key-derivation and/or key-confirmation steps.

Threshold modes w.r.t. (non-)determinism. A submitted threshold scheme for decryption (the core primitive or the higher-level operation) ***shall*** be functionally equivalent to the standardized one (RSADP, RSASVE.Recover, RSA-OAEP.Decrypt, K-PKE.Decrypt, ML-KEM.Decaps), with possible negligible differences. A submitted threshold scheme for public-key encryption ***may*** follow a probability distribution different from the standardized one (RSASVE.Generate, RSA-OAEP.Encrypt, K-PKE.Encrypt, ML-KEM.Encaps), as long as it is interchangeable w.r.t. standardized decryption (and preserves the usual security notions of interest). In particular, a threshold scheme for non-deterministic or not-verifiably deterministic encryption ***may*** be in any of the modes {Prob, Q-PR, F-PR} (as enumerated in Section 9.1 for threshold schemes for non-verifiably deterministic schemes).

9.3. Category N3: Symmetric Primitives

Symmetric primitives in scope. The "symmetric" category (in Class N) includes:

1. AES Encipher/Decipher (see §A.3.1)
2. Ascon-AEAD Encrypt/Decrypt (§A.3.2)
3. Hash and XOF (see §A.3.3), assuming a secret-shared input message.
4. MAC TagGen (see §A.3.4). The NIST-approved MAC functions are based on primitives (cipher, hash function, Keccak permutation) from the above mentioned schemes.

Table 8 lists the NIST-specified primitives. Appendix A.3 provides additional details.

Threshold interfaces and security level. A submission within category N3 ***shall*** specify a threshold scheme for at least one primitive listed in Table 8, and ***shall*** provide one implementation with $\kappa \gtrsim 128$, consistent with the parameters in the table. If the package proposes a threshold-MAC (subcategory N3.4), then it ***should*** first specify and implement the corresponding threshold-cipher/AEAD/hash/XOF, and then use it to implement the MAC.

Table 8. "Symmetric" primitives in category N3

Subcategory: Type	NIST reference	Spec or family	Primitive type	Cryptographic parameters		§ in this call
				$\kappa \approx 128$ **or** $\theta = 1$	$\kappa \gtrsim 192$ **or** $\theta \geq 3$	
N3.1: Blockcipher	[FIPS-197]	AES	{Enc,Dec}	AES-128	AES-{192,256}	A.3.1
N3.2: AEAD	[SP800-232-ipd]	Ascon	{Enc,Dec}	Ascon-AEAD128	—	A.3.2
N3.3: Hash, XOF	[FIPS-180-4]	SHA2	Hash	SHA-256	SHA-{384,512}	A.3.3
|	[FIPS-202]	SHA3	|	SHA3-256	SHA3-{384,512}	|
|	|	SHAKE	XOF	SHAKE128	SHAKE256	|
|	[SP800-185]	cSHAKE	|	cSHAKE128	cSHAKE256	|
|	[SP800-232-ipd]	Ascon	Hash	Ascon-Hash256	—	|
|	|	|	XOF	Ascon-[C]XOF128	—	|
N3.4: MAC	[SP800-38B]	CMAC	TagGen	AES-128	AES-256	A.3.4
|	[SP800-38D]	GMAC	|	|	|	|
|	[SP800-224-ipd]	HMAC	|	SHA[3]-256	SHA[3]-{384,512}	|
|	[SP800-185]	KMAC	|	cSHAKE128	cSHAKE256	|
|	(The 4 above)	MAC	|	$\|key\| = 128$	$\|key\| = 256$	|

Legend: AEAD = Authenticated Encryption with Associated Data. AES = Advanced Encryption Standard. cSHAKE = Customizable SHAKE. [C]XOF = XOF or CXOF (the "C" denotes customizable). Dec = Decipher (if AES) or Decrypt (if Ascon). Enc = Encipher (if AES) or Encrypt (if Ascon). MAC = Message Authentication Code. SHA[3]- = {SHA-, SHA3-}. SHAKE- = SHA with Keccak (XOF). TagGen = Tag Generation. XOF = eXtendable Output Function.

Input lengths. The experimental evaluation ***should*** at least benchmark the case of one input that can be processed with a single primitive evaluation. For example: for AES enciphering, the plaintext length would be 128 bits, since it is the block size of the block-cipher; for SHA-256 hashing, the message can be up to 447 bits, since the minimum required padding of 65 bits leads it to the block size of 512 bits. To help clarify possible complexity amortization, the implementation ***may*** also benchmark the threshold execution of many (e.g., 256) operations. This may help clarify the feasibility of the threshold approach for some mode of operation that repeats the evaluation of many building blocks.

Input/Output interface. For hashing and XOF'ing (keyless primitives), the threshold scheme ***shall*** consider an SSI mode w.r.t. the message. The hashing/XOF'ing result (output) ***may*** be obtained in NSS or SSO mode. In the case of customizable XOFs, the additional inputs ***may*** be in the clear or secret shared. For the keyed-primitives (i.e., AES, Ascon-AEAD, MAC), the threshold scheme ***shall*** consider the key is secret-shared. For the remaining input/output, the threshold scheme ***may*** consider SSI and SSO modes, such as

- **For enciphering/encryption:** SSI w.r.t. message, nonce, and/or associated data (if applicable); and/or SSO w.r.t. ciphertext and/or tag; and vice-versa for deciphering/decryption.

- **For MAC tag generation:** SSI w.r.t. message, and/or SSO w.r.t. tag. For example, these can be useful when using MAC for key-derivation and/or confirmation, within a 2KE protocol.

9.4. Category N4: KeyGen for Class N schemes

On "private" and "secret" keys. Various NIST publications use "secret key" in the context of symmetric-key primitives, and "private key" in the context of public-key cryptography (to denote the non-"public" element of a private/public key-pair). Since this Threshold Call deals with both symmetric-key and public-key schemes, the expressions "secret key" and (occasionally) "private or secret key" are sometimes used to encompass both contexts.

Table 9 lists the subcategories used to organize the KeyGen primitives in scope. Table 10 exemplifies keys that can be generated via a KeyGen operation. Variations are possible. Appendix A.4 provides informative details.

Table 9. KeyGen in schemes of Class N

Subcategory # Type of KeyGen	Related operations	Sections in this Call
N4.1: ECC KeyGen	Scalar multiplication (in additive notation)	9.4.1, A.4.1
N4.2: RSA KeyGen	Generate modulus and/or key-pair	9.4.2, A.4.2
N4.3: ML KeyGen	ML-DSA.KeyGen, K-PKE.KeyGen	9.4.3, A.4.3
N4.4: HBS KeyGen	Generate hash trees (for SLH-DSA, LMS, XMSS)	9.4.4, A.4.3
N4.5: Secret RBG	RBG for secret bit-strings or integers	9.4.5, A.4.5

Legend: ECC = elliptic curve cryptography. HBS = hash-based signatures. KEM = Key-Encapsulation Mechanism. K-PKE = ML-KEM-related Public-Key Encryption. ML = Module Lattice. LMS = Leighton-Micali Signature. XMSS= eXtended Merkle Signature Scheme. RBG = **r**andom-**b**it **g**eneration. RSA = Rivest–Shamir–Adleman.

Threshold KeyGen (DKG). Threshold KeyGen schemes are usually known as Distributed Key Generation (DKG) protocols. They enable a set of parties to collaborate to generate a secret sharing of a fresh secret or private key, such that the key is never assembled in one place. When applicable, the parties also obtain the public key (e.g., an RSA modulus obtained from the product of two secret primes; usually not secret shared), and/or commitments of everyone's private keys. Some domain parameters are agreed upon before the DKG (e.g., elliptic curve, security strength κ, and RSA public encryption key e).

Table 10. Examples of KeyGen purposes

KeyGen purpose (subsequent operation)	Secret or private key	Public elements	Section (in this Call)
EdDSA signing	Secret keys $(s, \nu) = \text{Hash}(d)$	$Q = s \cdot G$ (EC point)	9.4.1
ECDSA signing	Exponent d (integer mod n)	$Q = d \cdot G$ (EC point)	|
ECC-CDH for 2KE	$P = (h \cdot d_A) \cdot Q_B$	$Q_A = d_A \cdot G$	A.4.1.2
RSA signing, Enc/Dec	Primes (p, q)	Modulus $N = p \cdot q$	9.4.2
|	Exponent $d = e^{-1} \bmod \phi_N$	Exponent e	|
RSA encryption for 2KE	Bit-string Z	$c = \text{RSAEP}((N, e), Z)$	|
ML for K-PKE Enc/Dec	Secret vectors $\hat{\mathbf{s}}, \hat{\mathbf{e}}$	$\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$	9.4.3
ML for ML-DSA Sign	Secret vectors $\mathbf{s_1}, \mathbf{s_2}$	$\mathbf{t} = \text{NTT}^{-1}(\hat{\mathbf{A}} \circ \widehat{\mathbf{s_1}}) + \mathbf{s_2}$	|
AES Enc/Dec	Bit-string k	—	9.4.5
Key derivation	|	$k = \text{KDM}(Z, \ldots)$	—
Key confirmation	|	MacTag $T = \text{KC}(\ldots, k, \ldots)$	—

Enc/Dec = encrypt/decrypt. h = cofactor. KC = key confirmation. KDM = key derivation mechanism.

Interchangeability of random values. In a DKG protocol, the secret key to be output in secret-shared form is obtained by combining contributions of randomness (or pseudorandomness) from several parties. The (pseudo)randomness from each party ***may*** be obtained using NIST-specified RBG methods. A submitted DKG ***shall*** be interchangeable w.r.t. a subsequent operation of interest (e.g., signing or encryption). The specification ***shall*** also explain why the obtained randomness is appropriate, from a security perspective, considering the conventional (non-threshold) KeyGen. The implementation ***shall*** generate secret-shared keys consistent with one set of NIST-approved parameters, to ensure security level $k \gtrsim 128$ or $\theta \geq 1$.

9.4.1. Subcategory N4.1: ECC KeyGen

The goal of a DKG for an ECC scheme is to produce a secret-sharing $[d]$ of the private key d, produce commitments of the shares d_i of each party, and calculate the public key $Q = d \cdot G$. The “commitments” ***may*** be simple public-key shares $Q_i = d_i \cdot G$, or fancier semantically hiding commitments. The KeyGen ***may*** include additional elements related to the commitments (e.g., a ZKPoK of each secret share). A submission in subcategory N4.1, i.e., with a threshold scheme for an ECC-based primitive in Class N, ***shall*** include an implementation based on at least one elliptic curve that is NIST-approved for the scheme, namely from: Edwards{25519,448} for EdDSA, P-{256,384,521} for ECDSA, and P-{256,384,521} for ECC-2KE. Appendix A.4.1 has additional details.

This subcategory also includes the ECC-based CDH and MQV primitives. In the threshold setting, they require computing the core ECC operation: scalar multiplication of a group element, when the scalar is secret shared. This is somewhat similar to computing a public ECC key from a secret-shared private key (i.e., the scalar). A submission in this subcategory ***may*** focus on these ECC-based primitives. A full-fledged thresholdization of one-side of a NIST-approved ECC-based 2KE protocol would additionally require thresholdizing specific non-ECC primitives for key-derivation/confirmation, which are threshold-unfriendly.

9.4.2. Subcategory N4.2: RSA KeyGen

A submission of RSA DKG ***should*** obtain a modulus of size at least $|N| = 3072$, for $\kappa \gtrsim 128$. Considering the possible applications of an RSA modulus without known factorization, an RSA-DKG scheme ***may*** be submitted as a standalone threshold scheme (i.e., independent of subsequent RSA signing, encryption and decryption operations). Appendix A.4.2.1 has additional details.

One complexity challenge of RSA DKG is the threshold handling of rejection sampling of candidate primes. For the sake of exploration, submissions ***may*** choose to sample the primes, and/or the private exponent, using criteria different from what is described in §A.4.2.2. However, any such differences ***shall*** be well-documented and motivated. For example, it is acceptable for the RSA modulus to be biased toward being (or even restricted to be) a Blum integer (i.e., with both primes being 3 mod 4), as their properties are useful in some applications. Submissions that follow a generation method (e.g., direct biprimality testing) different from what is described in the NIST publications ***shall*** present a rationale to convey adequacy (e.g., adequate number of rounds).

9.4.3. Subcategory N4.3: ML KeyGen

A DKG for NIST-specified schemes based on Module-Lattices (ML) ***shall*** obtain a random secret-shared private key, and a corresponding public key, that are suitable for at least one of the approved parameter sets (i.e., for ML-DSA{44,65,87} or ML-KEM{512,768,1024}). While KeyGen is different between K-PKE (the PKE in ML-KEM) and ML-DSA, they both produce lattice-related elements, and may have some commonalities in the threshold setting. Appendix A.4.3 has additional details.

As long as the interchangeability requirement is met, the DKG ***may*** handle (pseudo)randomness differently from the conventional case:

- **Random sampling instead of pseudorandom.** The distributed computation of NIST-specified threshold-unfriendly pseudorandom generations is expensive. Therefore, a submitted ML-DKG ***may*** use more-efficient threshold randomness-sampling, provided that it retains the functionally and security of the final key.

- **Different intermediate encodings/representations.** The optimization of threshold schemes may suggest different encodings or representations, which would still be mathematically equivalent (e.g., whether/when to use the NTT and its inverse NTT^{-1}). This ***may*** be done, provided that the keys are interchangeable w.r.t. a subsequent threshold operation (i.e., signing, decryption or encryption).

For any of the two ML-based schemes, the parties ***may*** use a secure coin-flipping protocol to collaboratively determine the public seed ρ, and then use it in the clear to pseudorandomly generate the public matrix $\hat{\mathbf{A}}$. The parties ***may*** then interact in a threshold manner to distributively generate a secret sharing (across the parties) of the needed vector terms: $(\hat{\mathbf{s}},\hat{\mathbf{e}})$ for K-PKE, and $(\mathbf{s_1},\mathbf{s_2})$ for ML-DSA. Finally, the parties ***may*** distributively compute the result of the linear operation between the matrix and the vectors: $\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$, for K-PKE; or $\mathbf{t} = \text{NTT}^{-1}(\hat{\mathbf{A}} \circ \text{NTT}(\mathbf{s_1})) + \mathbf{s_2}$, for ML-DSA. In the case of ML-DSA, the element K would also be distributively produced as a secret sharing. The private and public keys are then the applicable encodings of the computed elements.

9.4.4. Subcategory N4.4: HBS KeyGen

A DKG for stateless (SLH-DSA) or stateful (LMS and XMSS) hash-based signatures (HBS) depends on the intended type of secret sharing of the private key (e.g., of a hash tree), to facilitate a subsequent threshold operation. Practical threshold schemes for HBS will likely be based on threshold-friendlier PRFs, which would thus fit in Class S. Appendix A.4.3 has additional details.

9.4.5. Subcategory N4.5: Secret RBG

If a scheme requires a simple random value (e.g., a bit-string, or an integer) for a secret-key, then the DKG essentially needs to produce a secret sharing of such type of random value. The protocol may also produce public commitments of the shares of each party, even if the original primitive did not produce a public key. These commitments may change the security guarantees of the key. For example, AES-256 is considered PQ, but committing to its key a la ECC-KeyGen using an ECC-based commitment of the AES key would make the overall scheme QV. Appendix A.4.5 has additional details.

10. Requirements for Class S Schemes

Class S considers cryptographic schemes not standardized by NIST. However, its categories (already enumerated in Section 2.2 and Table 3) are not intended for the submission of every type of academically interesting scheme. A submission ***shall*** be motivated by a serious intention of proposing for technical consideration a scheme that, besides being secure and practical, is believed to have a high potential for adoption in the real world. In particular, Class S welcomes the submission of threshold schemes for primitives of conventional (non-threshold) schemes that have been previously or are being thoroughly specified elsewhere, such as in other standards, or standards' proposals.

Regular categories in Class S. The first four categories of Class S are S1 for signing, S2 for public-key encryption, S3 for symmetric primitives, and S4 for KeyGen. They are called "regular" in the sense that the types of primitives match those of the Class N categories.

The specification document ***shall*** include a motivating comparison with the NIST standardized primitives of the same type (e.g., signature, encryption, cipher). In particular, the proposed primitive ***should*** have a distinctive feature, such as being threshold friendlier or based on different cryptographic assumptions, or have better efficiency (the conventional scheme or its threshold version) in some useful metric (e.g., succinctness, or communication complexity). If the motivation relates to an additional algorithm (e.g., allowing batch verification, or being ZKP friendly), then the corresponding algorithm ***should*** also be explained.

Other categories in Class S. The scope of Class S also includes primitives from other types of schemes not standardized by NIST. Correspondingly, Class S has the additional categories S5 for FHE (see Sections 10.5 and B.1), S6 for ZKPoKs (see Sections 10.6 and B.2), and S7 for gadgets (see Section 10.7). In submitted proposals of ZKPoKs (S6) or gadgets (S7), a conventional scheme (i.e., non-threshold) suffices, being optional the specification of a corresponding threshold scheme. The FHE and ZKPoK cases are also of specific interest to the NIST Privacy-Enhancing Cryptography (PEC) project [Proj-PEC].

Combinations. A submission ***may*** include crypto-systems from multiple categories, e.g.:

- A threshold scheme for a primitive in a non-KeyGen category (e.g., S1–S3, S5).
- A DKG protocol (N4 or S4) for generating the secret-shared key.
- A related ZKPoK (S6).

Specification of conventional primitives. In a submission of a threshold scheme for a primitive in Class S, the specification document ***shall*** explain the conventional scheme, namely in sufficient detail to derive the interchangeability requirement (i.e., to establish what is a valid output of the threshold scheme), and its setup (e.g., properties about the initial key). For example, a submission of threshold scheme for a signing primitive not specified by NIST needs to explain the verification and KeyGen primitives, besides the conventional signing primitive. If the conventional scheme has additional features/algorithms that can benefit its assessment, then they ***should*** also be specified. The required specification of the conventional scheme ***may*** be done (i) thoroughly, as a standalone proposed crypto-system (in 5.4); or (ii) at a high-level (in Pre4), but explaining at least the notation, interface, and security properties, while including a reference to an authoritative specification that is freely available to the public. Also, the specification document ***shall*** propose at least one concrete set of parameters for implementation.

10.1. Category S1: Signing

Category S1 is for submissions of threshold schemes for the signing primitive of digital signature schemes (QV or PQ) that are not standardized by NIST. Example motivating comparisons with NIST-standardized signatures:

1. Threshold friendlier and PQ.
2. Succincter and verifiably deterministic (i.e., a function of the message and the public-key), even if QV (e.g., based on pairings).
3. ZKP-friendlier (e.g., easier to generate unlinkable ZKPoKs of a signature).
4. Blinding friendly (i.e., having a structure that efficiently enables a protocol for blind signing, with concurrent security and low communication complexity).
5. Aggregatable (i.e., allowing the aggregation of multiple signatures into a sublinear size result, such as just one signature).
6. Batch verifiable (i.e., enabling the efficient verification of many signatures at once).

Recall that a signature is essentially a ZKPoK of a private key, while binding the proof to the message. Therefore, a proposed threshold signature scheme can also be framed as a threshold ZKPoK of a distributed secret (i.e., the signing key).

10.2. Category S2: PKE

Category S2 is for submissions of threshold schemes for (non-keygen) primitives of "regular" public-key encryption (PKE) schemes that are not standardized by NIST. There is a particular interest in threshold-friendly PQ PKE schemes. Submitted threshold schemes ***may*** be applied to decryption when the private key is secret shared, or/and encryption when the plaintext is secret-shared (e.g., when used for key encapsulation).

10.3. Category S3: Symmetric

Symmetric primitives (such as those in N3) have traditionally been designed to be efficient in a single-party setting. However, they often do not lend themselves naturally to efficient threshold implementations. The category S3 enables proposals of threshold-friendlier (TF) symmetric primitives. It is also of interest to consider friendliness w.r.t. FHE (see Appendix B.1) and ZKP (see Appendix B.2).

Families of primitives of interest. A crypto-system proposed in S3 ***should*** fit one of the following indexed family of primitives: S3.1 PRP (e.g., for enciphering); S3.2 PRF (e.g., for MAC'ing); S3.3 Hash function or XOF. If a design principle allows building primitives for various families, then a submission ***may*** propose a corresponding family of "symmetric" primitives and their threshold schemes.

Efficiency goal. This category is not meant for proposals of conventional symmetric primitives that would only marginally improve efficiency in the threshold paradigm, as compared to a threshold scheme for primitives in N3. Rather, proposed conventional primitives ***should*** yield an order of magnitude or more of improvement in the threshold setting. This improvement ***may*** refer to a single evaluation, or to an amortized setting (e.g., with large input/output, or with many evaluations of the underlying primitive).

Example use for key derivation/confirmation. The full-fledged 2KE NIST-specified protocols are not threshold friendly (despite the use of threshold-friendly PKE primitives), because of their use of threshold-unfriendly key-derivation/confirmation primitives. The present category S3 can, for example, be used to propose threshold friendlier symmetric primitives that could be used for alternative key-derivation/confirmation components.

Interest in commitment schemes. One application of interest for TF symmetric primitives is a commitment scheme, with hiding, binding and non-malleable properties, with either succinct commitment or opening, and possibly ZKP-friendly w.r.t. selective disclosure. Such additional specification is allowed in this category only if based on building blocks used to first specify one of the above mentioned family of primitives of interest. Alternatively, it can be submitted in the category of gadgets (S7).

10.4. Category S4: Keygen

As in N4, the category S4 considers the KeyGen for schemes with primitives in other categories of Class S. This category ***should*** be identified in a submission that proposes a DKG as an alternative to using a dealer for the initial secret-sharing of a secret key.

If a proposed DKG is usable for primitives in both Class N and Class S, then the submission can indicate suitability with both categories N4 and S4.

Single-party primitives to support KA. This KeyGen category also includes single-party non-PKE PKC-primitives for use in multi-party key-agreement. A corresponding submission ***should*** be justified based on different assumptions (e.g., possibly PQ), or even for allowing efficient key-agreement between more than two parties. The scope excludes PKE encryption/decryption primitives, which are already covered by the PKE category (S2).

10.5. Category S5: FHE

Category S5 relates to fully-homomorphic encryption (FHE), which is a special type of encryption that allows for arbitrary computation over encrypted data. Given one or more ciphertexts produced using the same key (public or secret) under an FHE scheme, it is then possible, from the ciphertext(s) alone (i.e., without the original plaintext and the decryption key), to produce a ciphertext that encrypts the result of an intended operation over the original plaintexts. Appendix B.1 has additional details.

Threshold scheme. The submission ***shall*** specify (in CSx.3) at least how to perform (i) threshold decryption (i.e., with a secret-shared key), or (ii) threshold encryption of a secret-shared value. The thresholdization of other primitives (e.g., KeyGen) is optional.

Conventional scheme. The specification of the conventional FHE scheme, either (i) thoroughly in Pre4, or (ii) at a high-level (in CSx.3) and supported on a thorougher reference, ***shall*** explain at least the interface and properties of the four main algorithms: `KeyGen`, `Enc`, `Dec`, `hom`. (The latter can be a set of algorithms, covering various homomorphic operations.) Depending on the FHE scheme, it may be useful to modularize the specification of other auxiliary algorithms, such as for refreshing a ciphertext (a.k.a. bootstrapping, producing a new ciphertext that encrypts the same element as encrypted by the original ciphertext, but with reduced "noise").

Benchmarking of the conventional scheme. Since FHE is a type of encryption scheme that has not been previously standardized by NIST, but is of high exploratory interest, its conventional primitives ***should*** also be benchmarked, i.e., in addition to the benchmarking of the threshold schemes. The selection of the benchmarking use-cases to evaluate performance (in M3) is left to the discretion of the submitters of an FHE scheme. Yet, submissions are encouraged to use benchmarking approaches emerging, reviewed or endorsed by community efforts. It is known that different FHE schemes have different applications or use cases in which they excel, depending on the type of arithmetic (e.g., Boolean, modular integer, or fixed-point) to be homomorphically evaluated. Each submission ***should*** at least: (i) showcase performance for one use case in which it performs well, and (ii) explain the anticipated real-world adoptability of that application (see CSx.6). For comparison, the benchmarking is encouraged to also measure performance for some operation that is anticipated to perform better by a different FHE scheme.

10.6. Category S6: ZKPoK

Category S6 allows for the submission of zero-knowledge proofs of knowledge (ZKPoKs) that are relatable to the other categories. These proofs enable proving knowledge (possibly in a secret-shared sense) of a private value (e.g., a secret/private key, or some other confidential input or output of a cryptographic operation), without disclosing it. The ZKPoK of a private value needs to relate to some "public" value known by the verifier, such as: a a public key, the public commitments of secret shares, or the output of a cryptographic operation (e.g., signature, encryption, or hashing). In a threshold ZKPoK generation, a distributed prover can interact to produce a ZKPoK of a secret-shared value, without ever reconstructing it. Appendix B.2 has additional details.

Proofs and Arguments. When referring to ZKPoKs, this call uses the term "proof" in a broad sense that also encompasses *"arguments"* (with computational soundness). Any submission of a ZKPoK ***shall*** clarify its soundness type (to allow for differentiation between "proof" and "argument").

Conventional versus threshold. A submission of ZKPoK ***shall*** at least specify a conventional (i.e., non-threshold) ZKPoK. Optionally, the submission ***may*** also specify a corresponding threshold scheme. In the latter, the secret input (the witness) is secret shared across a distributed prover. If a threshold ZKPoK generation is proposed, then it ***shall*** be interchangeable w.r.t. verification of an explained conventional ZKPoK.

ZKPoK scope. A submission proposing a ZKPoK ***shall*** showcase an instantiation related to a primitive in the scope of another category (e.g., a ZKPoK of a signature valid w.r.t. a public key). Table 15 lists several examples (in Appendix B.2.1). The proposed ZKPoK system can be tailored to the primitive in question, or be a general ZKPoK system (e.g., applicable to any non-deterministic polynomial problem, with a given representation, such as a circuit or some other constraint system; see Appendix B.2.5). In the general case, the instantiation ***should*** be achieved by modularly specifying (in a standalone file) a concrete system of constraints, to be parsed by the proof generator and the proof verifier. The specific ZKPoK object (what is being proven) can then be changed by simply changing that file.

ZKPoK security and characterization. A ZKPoK submission (conventional or/and threshold) ***shall*** indicate a parameter set for achieving at least one profile of security strength for soundness and zero-knowledge, satisfying $(\kappa, \sigma) \gtrsim (128, 40)$. It is encouraged that a second parameter set is also proposed, for comparison purposes, achieving $(\kappa, \sigma) \gtrsim (192, 64)$. See additional notes in Appendix B.2.4.

A submission of ZKPoK ***should*** motivate the achieved features (e.g., when applicable, transferability or deniability, interactivity, succinctness). The instantiation of some of them may affect some aspects of composability, which ***should*** also be discussed.

10.7. Category S7: Gadgets

When deemed useful for other threshold schemes, Category S7 allows for the submission of auxiliary primitives, which this Threshold Call refers to as *gadgets*. They can be conventional or threshold. See related notes on §5.3.1 of NIST-IR8214A, and §5.5.2 of NIST-IR8214B-ipd.

Specification. A gadget ***may*** be specified as a standalone crypto-system (i.e., a formal "part" of the specification; see CSx.3 in Section 5.4), in which case it ***shall*** make a strong case for why it can be used to support crypto-systems in other categories. Alternatively, a gadget ***may*** be specified directly as a module (in Pre4 or CSx.3) of a more complex crypto-system, and still referenced as a gadget in the specification scope. Preferably, gadgets that are very simple (e.g., Shamir secret-sharing and Lagrange interpolation) and expected to appear in many independent submissions ***should*** be specified in the latter manner.

Implementation and evaluation. To be considered as a standalone gadget during the public analysis phase, the gadget ***shall*** (i) be implemented with corresponding scripts and instructions for testing the gadget, and (ii) be analyzed in the experimental performance.

Gadgets in particular categories. If a submission specifies a gadget that is directly in the scope of another category, and the submission also includes a threshold scheme for it, then the gadget ***should*** be organized in said category, instead of S7. For example, a threshold-friendly hash function fits better in S3.3, within the category Class S (symmetric).

Gadget examples: Sophisticated secret-sharing variants, such as verifiable or publicly-verifiable; garbled circuits; oblivious transfer; generation of correlated randomness; commitment schemes; secret resharing (possibly for new values f and n); multiplicative-to-additive share conversion; linearly homomorphic encryption; vector oblivious linear evaluation; verifiable random functions; consensus and broadcast.

1606 References

[CAVP] National Institute of Standards and Technology (2025). *Cryptographic Algorithm Validation Program (CAVP)*. https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program.

[CC-BY-4.0] Creative Commons (2013). *CC BY 4.0: Attribution 4.0 International*. https://creativecommons.org/licenses/by/4.0.

[FIPS-180-4] National Institute of Standards and Technology (2015). *Secure Hash Standard (SHS)*. (U.S. Department of Commerce) Federal Information Processing Standards (FIPS) 180-4. 10.6028/NIST.FIPS.180-4.

[FIPS-186-5] National Institute of Standards and Technology (2023). *Digital Signature Standard (DSS)*. (U.S. Department of Commerce) Federal Information Processing Standards (FIPS) 186-5. DOI:10.6028/NIST.FIPS.186-5.

[FIPS-197] National Institute of Standards and Technology (2023). *Advanced Encryption Standard (AES)*. (Update, without technical changes, of the original version from November 2001). DOI:10.6028/NIST.FIPS.197-upd1.

[FIPS-202] National Institute of Standards and Technology (2015). *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. (U.S. Department of Commerce) Federal Information Processing Standards (FIPS) 202. DOI:10.6028/NIST.FIPS.202.

[FIPS-203] National Institute of Standards and Technology (2024). *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. (U.S. Department of Commerce) Federal Information Processing Standards (FIPS) 203. DOI:10.6028/NIST.FIPS.203.

[FIPS-204] National Institute of Standards and Technology (2024). *Module-Lattice-Based Digital Signature Standard*. (U.S. Department of Commerce) Federal Information Processing Standards (FIPS) 204. DOI:10.6028/NIST.FIPS.204.

[FIPS-205] National Institute of Standards and Technology (2024). *Stateless Hash-Based Digital Signature Standard*. (U.S. Department of Commerce) Federal Information Processing Standards (FIPS) 205. DOI:10.6028/NIST.FIPS.205.

[FIPS-206-ipd] National Institute of Standards and Technology (2025). *Falcon-Based Digital Signature Standard*. An initial public draft, labeled FIPS-206 ipd, is expected to appear later in 2025, based on the Falcon submission to the NIST PQC standardization process.

[HES] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan (2018). *Homomorphic Encryption Standard*. Published by HomomorphicEncryption.org. Available at https://homomorphicencryption.org/standard.

[IG-FIPS-140-2] National Institute of Standards and Technology and Canadian Centre for Cyber Security (2023). *Implementation Guidance for FIPS 140-2 and the Cryptographic Module Validation Program (CMVP)*. Version updated on 2023-Oct-30. https://csrc.nist.gov/Projects/cryptographic-module-validation-program/announcements.

[ITL-Patent-Policy] National Institute of Standards and Technology (2019). *Information Technology Laboratory (ITL) Patent Policy*. https://www.nist.gov/itl/publications-0/itl-patent-policy-inclusion-patents-itl-publications.

[MPTC-Call2021a] Luís T. A. N. Brandão (2021). *NIST-MPTC Call 2021a for Feedback on Criteria for Threshold Schemes*. Available at https://csrc.nist.gov/csrc/media/projects/threshold-cryptography/documents/MPTC-call2021a-feedback.pdf. Public comments: https://csrc.nist.gov/csrc/media/projects/threshold-cryptography/documents/MPTC-Call2021a-Feedback-compilation.pdf.

[MPTS-2020] National Institute of Standards and Technology (2020). *NIST Workshop on Multi-Party Threshold Schemes 2020*. Virtual conference.

[MPTS-2023] National Institute of Standards and Technology (2023). *NIST Workshop on Multi-Party Threshold Schemes 2023*. Virtual conference.

[NCCoE-PQC] National Cybersecurity Center of Excellence (NCCoE) (2025). *Migration to Post-Quantum Cryptography*. https://www.nccoe.nist.gov/crypto-agility-considerations-migrating-post-quantum-cryptographic-algorithms.

[NIST-IR7977] NIST Cryptographic Technology Group (2016). *NIST Cryptographic Standards and Guidelines Development Process*. (National Institute of Standards and Technology) NIST Internal Report (NISTIR) 7977. DOI:10.6028/NIST.IR.7977.

[NIST-IR8214A] Luís T. A. N. Brandão, Michael Davidson, and Apostol Vassilev (2020). *NIST Roadmap Toward Criteria for Threshold Schemes for Cryptographic Primitives*. (National Institute of Standards and Technology (NIST) Internal Report (NISTIR) 8214A. DOI:10.6028/NIST.IR.8214A. Public comments: https://csrc.nist.gov/publications/detail/nistir/8214a/final.

[NIST-IR8214B-ipd] Luís T. A. N. Brandão and Michael Davidson (2022). *Notes on Threshold EdDSA/Schnorr Signatures*. (National Institute of Standards and Technology) NIST Internal Report (NISTIR) 8214B ipd (initial public draft). DOI:10.6028/NIST.IR.8214B.ipd.

[NIST-IR8214C-ipd] Luís T. A. N. Brandão and René Peralta (2023). *NIST First Call for Multi-Party Threshold Schemes*. (National Institute of Standards and Technology) NIST Internal Report (NISTIR) 8214C ipd (Initial Public Draft). DOI:10.6028/NIST.IR.8214C.ipd.

[NIST-IR8214C-ipd-comms] NIST Multi-Party Threshold Cryptography Project (2023). *Compilation of Public Comments on NISTIR 8214C ipd*. https://csrc.nist.gov/files/pubs/ir/8214/c/ipd/docs/nistir-8214c-ipd-public-feedback.pdf. Includes copied comments submitted by a total of 32 authors.

[NIST-IR8214C-2pd] Luís T. A. N. Brandão and René Peralta (2025). *NIST First Call for Multi-Party Threshold Schemes*. (National Institute of Standards and Technology) NIST Internal Report (NISTIR) 8214C 2pd (Second Public Draft). DOI:10.6028/NIST.IR.8214C.2pd.

[NIST-IR8454] Meltem Sönmez Turan, Kerry McKay, Donghoon Chang, Lawrence Bassham, Jinkeon Kang, Noah Waller, John Kelsey, and Deukjo Hong (2023). *Status Report on the Final Round of the NIST Lightweight Cryptography Standardization Process*. (National Institute of Standards and Technology) NIST Internal Report (NISTIR) 8454. DOI:doi.org/10.6028/NIST.IR.8454.

[NIST-IR8547-ipd] Dustin Moody, Ray Perlner, Andrew Regenscheid, Angela Robinson, and David Cooper (2024). *Transition to Post-Quantum Cryptography Standards*. National Institute of Standards and Technology (NIST) Internal Report (NISTIR) 8547. DOI:10.6028/NIST.IR.8547.ipd.

[OSI-def] Open Source Initiative (2025). *The Open Source Definition*. https://opensource.org/osd.

[OSI-lic] Open Source Initiative (2025). *OSI approved licenses*. https://opensource.org/licenses.

[PQC-Call-2016] NIST-PQC project (2016). *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. https://csrc.nist.gov/csrc/media/projects/post-quantum-cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[Proj-CC] National Institute of Standards and Technology (2025). *NIST Project on Circuit Complexity (CC)*. https://csrc.nist.gov/projects/circuit-complexity. See list of circuits at https://csrc.nist.gov/projects/circuit-complexity/list-of-circuits, and repository of circuits at GitHub:usnistgov/Circuits. Accessed in March 2025.

[Proj-LWC] National Institute of Standards and Technology (2025). *NIST Project on Lightweight Cryptography (LWC)*. https://csrc.nist.gov/projects/lightweight-cryptography.

[Proj-MPTC] National Institute of Standards and Technology (2025). *NIST Project on Multi-Party Threshold Cryptography (MPTC)*. https://csrc.nist.gov/projects/threshold-cryptography.

[Proj-PEC] National Institute of Standards and Technology (2025). *NIST Project on Privacy-Enhancing Cryptography (PQC)*. https://csrc.nist.gov/projects/pec.

[Proj-PQC] National Institute of Standards and Technology (2025). *NIST Project on Post-quantum Cryptography (PQC)*. https://csrc.nist.gov/projects/post-quantum-cryptography.

[RFC7748] Adam Langley and Mike Hamburg and Sean Turner (2016). *Elliptic Curves for Security*. Internet Research Task Force (IRTF) Request for Comments: RFC 7748. DOI:10.17487/RFC7748.

[RFC8017] Kathleen Moriarty and Burt Kaliski and Jakob Jonsson and Andreas Rusch (2016). *PKCS #1: RSA Cryptography Specifications Version 2.2*. Internet Engineering Task Force (IETF) Request for Comments: RFC 8017. DOI:10.17487/RFC8017.

[RFC8391] Andreas Huelsing and Denis Butin and Stefan-Lukas Gazdag and Joost Rijneveld and Aziz Mohaisen (2018). *XMSS: eXtended Merkle Signature Scheme*. Internet Research Task Force (IETF) Request for Comments: RFC 8391. DOI:10.17487/RFC8391.

[RFC8554] David McGrew and Michael Curcio and Scott Fluhrer (2019). *Leighton-Micali Hash-Based Signatures*. Internet Research Task Force (IETF) Request for Comments: RFC 8554. DOI:10.17487/RFC8554.

[SP800-38B] Morris Dworkin (2005). *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-38B. Updated on October 2016. DOI:10.6028/NIST.SP.800-38B.

[SP800-38D] Morris Dworkin (2007). *Recommendation for Block Cipher Modes of Operation: Galois/ Counter Mode (GCM) and GMAC.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-38D. DOI:10.6028/NIST.SP.800-38D.

[SP800-56A-Rev3] Elaine Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, and Richard Davis (2018). *Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-56A Rev. 3. DOI:10.6028/NIST.SP.800-56Ar3.

[SP800-56B-Rev2] Elaine Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, Richard Davis, and Scott Simon (2019). *Recommendation for Pair-Wise Key-Establishment Using Integer Factorization Cryptography.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-56B Rev. 2. DOI:10.6028/NIST.SP.800-56Br2.

[SP800-90A-R1] Elaine Barker and John Kelsey (2015). *Recommendation for Random Number Generation Using Deterministic Random Bit Generators.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-90A Rev. 1. DOI:10.6028/NIST.SP.800-90Ar1.

[SP800-90B] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry McKay, Mary Baish, and Michael Boyle (2018). *Recommendation for the Entropy Sources Used for Random Bit Generation.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-90B. DOI:10.6028/NIST.SP.800-90B.

[SP800-90C-4PD] Elaine Barker, John Kelsey, Kerry McKay, Allen Roginsky, and Meltem Sönmez Turan (2024). *Recommendation for Random Bit Generator (RBG) Constructions. Fourth Public Draft.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-90C 4PD. DOI:10.6028/NIST.SP.800-90C.4pd.

[SP800-108-Rev1] Lily Chen (2022). *Recommendation for Key Derivation Using Pseudorandom Functions.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-108 Rev. 1-Upd. 1. DOI:10.6028/NIST.SP.800-108r1-upd1.

[SP800-132] Meltem Sönmez Turan, Elaine Barker, William Burr, and Lily Chen (2010). *Recommendation for Password-Based Key Derivation: Part 1: Storage Applications.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-132. DOI:10.6028/NIST.SP.800-132.

[SP800-185] John Kelsey, Shu-jen Chang, and Ray Perlner (2016). *SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-185 (Draft). DOI:10.6028/NIST.SP.800-185.

[SP800-186] Lily Chen, Dustin Moody, Andrew Regenscheid, Angela Robinson, and Karen Randall (2023). *Recommendations for Discrete Logarithm-Based Cryptography: Elliptic Curve Domain Parameters.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-186. DOI:10.6028/NIST.SP.800-186.

[SP800-208] David Cooper, Daniel Apon, Quynh Dang, Michael Davidson, Morris Dworkin, and Carl Miller (2020). *Recommendation for Stateful Hash-Based Signature Schemes.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-208. DOI:10.6028/NIST.SP.800-208.

[SP800-224-ipd] Meltem Sönmez Turan and Luís T.A.N. Brandão (2024). *Keyed-Hash Message Authentication Code (HMAC): Specification of HMAC and Recommendations for Message Authentication (Initial Public Draft).* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-224 ipd. DOI:10.6028/NIST.SP.800-224.ipd. Its future final version is expected to supersede FIPS 198-1 and portions of SP 800-107 Rev. 1.

[SP800-232-ipd] Meltem Sönmez Turan, Kerry McKay, Donghoon Chang, Jinkeon Kang, and John Kelsey (2024). *Ascon-Based Lightweight Cryptography Standards for Constrained Devices: Authenticated Encryption, Hash, and Extendable Output Functions (Initial Public Draft).* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-232 ipd. DOI:10.6028/NIST.SP.800-232.ipd.

[Ubuntu] Canonical and friends (2024). *Ubuntu 24.04.1 LTS*. File available at https://releases.ubuntu.com/24.04/ubuntu-24.04.1-desktop-amd64.iso (SHA256 hash `c2e6f4dc 37ac944e 2ed507f8 7c6188dd·4d3179bf 4a3f9e11 0d3c88d1 f3294bdc`, 6,203,355,136 bytes). Release details at https://releases.ubuntu.com/24.04.1.

[ZkpComRef] ZKProof (2022). *ZKProof Community Reference*. Published by ZKProof.org.

Appendix A. Notes on Class N Categories

This section includes informative notes about some of the categories, subcategories, and primitives within Class N. Related requirements for submissions are discussed in Section 9.

A.1. Category N1: NIST-Specified Signing Primitives

Category N1 is for submissions of threshold schemes for NIST-specified signing primitives: See submission requirements in Section 9.1. The conventional signature schemes of interest are: EdDSA (N1.1, see §A.1.1), ECDSA (N1.2, see §A.1.2), RSADSA (N1.3, see §A.1.3), ML-DSA (N1.4, see §A.1.4), and SLH-DSA and Stateful HBS (N1.5, see §A.1.5).

Threshold verification of signatures (secondary interest). While some applications may meet a privacy goal by performing a threshold verification in SSI mode w.r.t. the signature and/or the message, this is of secondary interest in this Threshold Call. A specification of such threshold verification is welcome (but not required) to accompany a submitted specification of threshold signing. This is somewhat analogous to threshold PKE-encryption of a secret-shared message, but signatures usually relate to the protection of integrity instead of confidentiality.

A.1.1. Subcategory N1.1: EdDSA Signing

Conventional signature. EdDSA [FIPS-186-5, §7], has pseudorandom signatures. The standardized signing $\mathtt{Sign}_n[s, \nu](M)$ of a message M, requires a private signing key s and a nonce-derivation key ν. Ignoring some encoding details, the signing outputs a signature $\sigma = (R, S)$, where $R = r \cdot G$, G is the base-point of the elliptic curve, $r = H(\nu, M)$, H is a cryptographic hash function, $S = r + \chi \cdot s$, $\chi = H(R, Q, M)$ is the challenge, and $Q = s \cdot G$ is the public key. Per specification, the secret keys (s, ν) are obtained from the two halves of a hash of d, where d is a random secret key. HashEdDSA is a signing variant that considers pre-hashed messages as input.

A.1.2. Subcategory N1.2: ECDSA Signing

Conventional signature. ECDSA [FIPS-186-5, §6] has a default signing mode that is probabilistic (§6.3.1), and also has a deterministic mode (§6.3.2), here abbreviated as Det-ECDSA. Table 11 compares the conventional notation of EdDSA and of ECDSA.

The ECDSA signing $\mathtt{Sign}_n[d](M)$ of a message M outputs a signature $\sigma = (r, s)$, where (ignoring some encoding details), d is the private signing key, $Q = d \cdot G$ is the public

Table 11. Notation of EdDSA versus ECDSA (in FIPS 186-5)

Scheme	Signature (output)	Private key	Public key	Secret nonce	Nonce commitment	Challenge	"Precursor" private key
EdDSA	(R, S)	s	Q	r	R	χ	d
ECDSA	(r, s)	d	Q	k	r	e	—

key; G is the base-point of the elliptic curve, the "challenge" $e = \text{Encode}_n^{(1)}(\text{Hash}(M))$ is an encoding (mod n) of the hash of the message being signed; n is the order of G; $k \xleftarrow{\$} [1, \ldots, n-1]$ is (in the probabilistic version) a uniformly selected secret nonce; $R = k \bullet G$ is the "nonce commitment" and $r = \text{Encode}_n^{(2)}(R)$ is a corresponding encoding (mod n); and $s = k^{-1} \cdot (e + r \cdot d) \pmod{n}$. In Det-ECDSA the secret nonce k is instead obtained pseudorandomly from Hash(M) and d, requiring several calls to HMAC.

A.1.3. Subcategory N1.3: RSADSA Signing

RSA signature modes are specified in §5.4 of FIPS-186-5, by reference to IETF RFC8017, and with some constraints on the KeyGen and the possible hash functions. The standard specified two signature schemes with appendix (SSA):

1. RSASSA-PSS (probabilistic signature scheme), using an approved hash function or XOF
2. RSASSA-PKCS-v1.5 (deterministic), using an approved hash function

Both PSS and PKCS-v1.5 schemes are of the SSA type. This means the message itself is required to verify the signature, rather than the standalone signature enabling message recovery. However, they use different encoding mechanisms for the signature with appendix (EMSA), namely EMSA-PSS and EMSA-PKCS-v1.5, respectively. Other than that, they are then based on the same core primitives: RSASP1 for signing and RSAVP1 for verification.

A.1.4. Subcategory N1.4: ML-DSA Signing

Conventional signature. ML-DSA [FIPS-204] is one of the two first NIST-standardized stateless PQ signing schemes. It is a PQ signature scheme with a construction inspired by Schnorr signatures, but based on lattices and with various tweaks for efficiency and security. The standard defines three parametrizations: ML-DSA{44,65,87}, equating them to PQC security categories (θ) {2, 3, 5}, respectively. Correspondingly, the internal random-bit generations are required to use security strength κ at least {128, 192, 256}. However, if ML-DSA-44 is implemented with randomness with strength $128 \leq \kappa < 192$, then its claimed PQC security category is decreased to $\theta = 1$ [FIPS-204, §3.6.1].

Regular, pre-hashed, and internal function. ML-DSA has two main versions: "pure" (ML-DSA) and "pre-hashed" (HashML-DSA). Both versions accept an input argument with content (i.e., a byte-string ctx with length between 0 and 255). The pre-hashed version also receives as input an identifier of which hash function to use. The specification modularizes an internal algorithm ML-DSA.Sign_internal(sk, M', rnd), with the input M' already integrating the context and the plaintext message or its hash.

Probabilistic and deterministic modes in the conventional algorithm. The ML-DSA scheme allows (similar to ECDSA, in N1.2) probabilistic and deterministic variants:

- A default ("hedged") probabilistic mode, where rnd is a random 32-byte string
- A deterministic mode (Det-), where rnd is fixed to be the all-zeros 32-byte string

A standalone ML-DSA signature does not reveal which mode was used. In particular, the deterministic mode is not verifiably deterministic. Per FIPS 204 (Algorithm 7, step 7): the internal signing algorithm ML-DSA.Sign_internal(sk, M', rnd) obtains a secret "random-or-pseudorandom" value $\rho'' \leftarrow \mathsf{H}(K||rnd||\mu, 64)$, where K is derived from the private key, rnd is all zeros or random, and μ is derived from the message and the public key.

One peculiarity of ML-DSA signing is that it requires a rejection sampling loop. Without it, there would be a noticeable probability that a signature would reveal information about the secret key. Therefore, the rejection sampling keeps trying until it obtains a signature candidate that satisfies a number of tests. In FIPS-204, Table 1 lists the expected number of trials: {4.25, 5.1, 3.85} for ML-DSA-{44,65,87}, respectively. The publication also lists iteration bounds for a non-completion probability around 2^{-256}.

A.1.5. Subcategory N1.5: HBS Signing

This section considers NIST-standardized **h**ash-**b**ased **s**ignature (HBS) schemes. They are post-quantum, with security relying solely on NIST-standardized hash functions. The goal of this section is to identify the signature schemes in scope, distinguish between the **stateless** and the **stateful** approaches, and motivate the exploration of threshold schemes for (possibly alternative) HBS schemes. The section does not delve into the actual signing mechanisms. §A.1.5.1 discusses the (stateless) SLH-DSA, §A.1.5.2 considers the stateful HBS schemes, and §A.1.5.3 briefly comments on possible threshold HBS schemes.

A.1.5.1. Conventional SLH-DSA (stateless)

The **S**tate**l**ess **H**ash-Based Digital Signature Algorithm (SLH-DSA) is a post-quantum signature scheme standardized by FIPS-205. Its security does not rely on keeping track of which internal signing keys have been used. As components, SLH-DSA uses two other hash-based signature schemes: the forest of random subsets (FORS), and the eXtended Merkle Signature Scheme (XMSS). In turn, XMSS is based on the Winternitz One-Time Signature Plus (WOTS$^+$) scheme. The XMSS scheme is used within a *hypertree* signature scheme, which uses a tree of trees, rather than a single XMSS tree.

At a very high level, an SLH-DSA signature is produced as follows. the message is hashed, and a pseudorandom index is determined from the hash, to serve as an identifier of a FORS key, which is part of (i.e., derivable from) the SLH-DSA private key. Then, the message hash is signed by the FORS key. The final SLH-DSA signature is composed of the FORS signature, and an authenticator of the FORS public key, which is produced via a hypertree signature. The latter is composed of a sequence of XMSS signatures (one for each layer of the hypertree).

SLH-DSA specifies 12 parameter sets: SLH-DSA-{SHA2,SHAKE}-{128,192,256}{s,f}, corresponding to $2 \times 3 \times 2$ options (ee Table 12 of FIPS-205). More sets (e.g., allowing fewer but shorter signatures) may be allowed by the forthcoming SP800-230. The intermediate label {128,192,256} indicates a corresponding claimed PQC security category $\theta = \{1, 3, 5\}$. For each such level there are four possible parameter sets, resulting from a choice of hash/XOF family {SHA2,SHAKE}, and a mode from between “s” (relatively small signatures) and “f” (relatively fast signature generation). Furthermore, the signature has a pure version (SLH-DSA) and a pre-hash version (HashSLH-DSA) that — similar to EdDSA and ML-DSA— enables choosing whether to provide the entire message or just its hash to the core signing algorithm.

A.1.5.2. Conventional Stateful HBS

SP800-208 approves the use of certain **stateful h**ash-**b**ased **s**ignature (HBS) schemes. Being stateful, they require updating state across a sequence of signing operations, namely to prevent reusing a one-time-signature key, lest it would break unforgeability. In general, statefulness is undesired, as it poses difficult challenges for state management, and is thus unsuitable for general-purpose signatures. Their utility is for limited circumstances, if they bring some efficiency advantage compared to other stateless PQ signature schemes.

The approved stateful HBS schemes are Leighton-Micali Signature (LMS), specified by reference to RFC8554, and XMSS, specified by reference to RFC8391. They are both based on the Winternitz signature scheme. Additionally, their “multi-tree” variants — the Hierarchical Signature System (HSS) and the multi-tree XMSS (XMSS$^{\mathsf{MT}}$) — are also approved, respectively, . The multi-tree version allows for a more efficient way of determining the public key. SP800-208 also specifies a modified version of XMSS and XMSS$^{\mathsf{MT}}$, to distribute (not in a secret-shared sense) the implementation across multiple cryptographic modules.

In each scheme, the private key consists of a large set of one-time signature (OTS) keys, whereas the long-term public key is obtained as a succinct commitment (using a Merkle tree) of a large set of OTS public keys. The multi-tree variants allow the one-time keys to be organized into multiple trees, which eases the distribution of mutually exclusive subsets of private keys. For each new call to sign a message, an unused one-time private key is selected and used. Then, the signature also includes a proof of correct used key, to show that the corresponding one-time public key is committed by the long-term public key.

A.1.5.3. Threshold Hash-Based Signatures

Given the heavy use of threshold-unfriendly hash functions, a practical/efficient threshold scheme is not expected to be devised for these HBS schemes, i.e., for a general k-of-n case and with small state per party. It is still conceivable that a set of parties is configured with an initial secret-sharing of all one-time private keys, enabling them to later produce a signature for a given agreed message. This subcategory (N1.5) for HBS signing is intended to motivate exploration of the limits of HBS-thresholdization, and/or better HBS alternatives (see S1, in Section 10.1) (e.g., based on TF'ier/ signature-friendlier PRFs).

A.2. Category N2: NIST-Specified PKE Primitives

Category N2 is for submissions of threshold schemes for primitives of NIST-specified public-key encryption (PKE) schemes. See Section 9.2 with submission requirements.

NIST standards with PKE schemes. NIST specifies PKE schemes in the standards for (i) RSA-based pair-wise key establishment (2KE) [SP800-56B-Rev2], and (ii) Module-Lattice-based Key-Encapsulation Mechanism (ML-KEM) [FIPS-203]. These standards use PKE encryption (Enc) and decryption (Dec) primitives as building blocks to construct higher-level schemes, namely key-agreement schemes (KAS), key-transport schemes (KTS), and KEMs (see Appendices A.2.1 and A.2.2). In turn, all of these can be used to enable particular instantiations of 2KE, to allow two parties to agree on a secret key, without an eavesdropper learning it. In these applications, the core use of the PKE scheme is in allowing one party to encrypt a random contribution (i.e., a seed that will affect the derivation of an *agreed* key) and send it securely (i.e., with confidentiality) to a decryptor party.

In usual applications of a KAS, KTS, or KEM, it is important to prevent user access to the interface of the low-level PKE primitives, since their misuse poses a security risk (e.g., a dangerous decryption oracle). Thus, any actual proposal of a threshold scheme for replacing a party in a full-fledged PKE-based 2KE application needs to considers the security of the entire system.

Primitives of interest. Enc outputs a ciphertext $C = \text{Enc}(pubKey, M[, R])$, as an encryption of an input plaintext message M, using a public encryption key $pubKey$, and optionally (i.e., depending on the scheme) a random value R that directly enables probabilistic encryption. Correspondingly, Dec outputs the original plaintext $M = \text{Dec}(privKey, C)$, as decryption of an input ciphertext, using the private decryption key $privKey$.

A.2.1. Subcategory N2.1: RSA Encryption/Decryption

Appendix A.2.1.1 describes the conventional (non-threshold) "textbook" RSA encryption and decryption primitives. Appendix A.2.1.2 considers higher-level primitives useful for 2KE.

A.2.1.1. Conventional RSA-PKE

In SP800-56B-Rev2, the RSA cryptosystem enables various pair-wise key-establishment (2KE) protocols. The RSA KeyGen (see category N4.2) generates a public RSA modulus N (product of two primes), a public encryption key e, and a private decryption key d. The core encryption/decryption primitives of interest in the present subcategory (N2.1) are those from (the informally called) "textbook" RSA-PKE:

- **RSA Encryption Primitive (RSAEP):** Obtains a ciphertext $c = \text{RSAEP}(e, m) = m^e \bmod N$, using the public key e to encrypt the plaintext m. RSAEP is assumed to be a one way function. Being deterministic, it does not on its own provide the IND-CPA property of an encryption scheme. Providing semantic security and beyond requires a higher-level construction (see §A.2.1.2), including randomness.
- **RSA Decryption Primitive (RSADP):** Recovers the plaintext $m = c^d \bmod N$, by calculating $m = \text{RSADP}(privKey, c)$, where the private key privKey is used to decrypt the ciphertext c. The input privKey is acceptable in three possible formats [SP800-56B-Rev2, §6.2.2]: basic (N, d), prime-factor (p, q, d), and chinese-remainder theorem (CRT) $(N, e, d, p, q, dP, dQ, qInv)$, where [SP800-56B-Rev2, §3.2]: (p, q) is the pair of secret prime factors of N; dP is $d \bmod (p-1)$; dQ is $d \bmod (q-1)$; and $qInv$ is the inverse of $q \bmod p$.

A.2.1.2. Higher-Level Constructions (Based on RSAEP/ RSADP)

Conventional RSASVE and RSA-OAEP. The two low-level primitives (RSAEP, RSAEP) of RSA-PKE are used in the higher-level cryptosystems RSASVE and RSA-OAEP, yielding four higher-level primitives, in two pairs (each § is referenced from SP800-56B-Rev2):

1. RSASVE.{Generate, Recover}: RSA for **s**ecret-**v**alue **e**ncapsulation *generation* (of random value and corresponding ciphertext; §7.2.1.2) and *recovery* (§7.2.1.3).

2. RSA-OAEP.{Encrypt, Decrypt}: RSA with **O**ptimal **A**symmetric **E**ncryption **P**adding *encryption* (§7.2.2.3) and *decryption* (§7.2.2.4).

Both RSA-OAEP.Encrypt and RSASVE.Generate are probabilistic. Conversely, both RSA-OAEP.Decrypt and RSASVE.Recover are deterministic. At an even higher level, these primitives can be used for NIST-approved 2KE, which may further involve key-derivation/confirmation operations. Table 12 lists those RSA-based 2KE schemes.

Table 12. RSA-based primitives per RSA-2KE scheme, per party

Scheme	§ in SP 800 -56B-Rev2	Party	RSA-based primitive	KDM needed?
KAS1	§8.2	1st contributor	RSASVE.Generate	Yes
|	|	2nd contributor	RSASVE.Recover	|
KAS2	§8.3	Any	RSASVE.{Generate, Recover}	|
KTS-OAEP	§9.2	Sender	RSA-OAEP.Encrypt	No
|	|	Receiver	RSA-OAEP.Decrypt	|

Legend: §= section number. 2KE = Pair-Wise Key Establishment. KAS = Key Agreement Scheme. KDM = Key-Derivation Mechanism (not RSA-based). KTS = Key Transport Scheme. OAEP = Optimal Asymmetric Encryption Padding. RSA = Rivest-Shamir-Adleman. SVE = Secret Value Encapsulation. **Note:** Each scheme has a basic version, and another with key confirmation (unilateral or bilateral, not RSA-based).

A.2.2. Subcategory N2.2: K-PKE Encryption/Decryption

FIPS-203 is the first NIST standard to specify a PQ-PKE scheme (dubbed K-PKE), whose use is only approved to support ML-KEM. The present subcategory (N2.2) is interested in the core K-PKE encryption/decryption primitives: K-PKE.Encrypt (Enc) and K-PKE.Decrypt (Dec). The three approved parameter sets ML-KEM-{512,768,1024} determine corresponding parameters for K-PKE, respectively corresponding to PQC security categories $\theta = \{1, 3, 5\}$, provided that their internal RBG has security strength $\kappa = \{128, 192, 256\}$.

A.2.2.1. Conventional K-PKE

- **K-PKE.Enc**(ek$_{\text{PKE}}$, m, r): Deterministic algorithm, using a public encryption key ek$_{\text{PKE}}$ and a seed r to encrypt a plaintext m, and outputting a ciphertext c. It uses internal values $(y[i], e_1[i], e_2)$ obtained pseudorandomly from the input seed r.
- **K-PKE.Dec**(dk$_{\text{PKE}}$, c): Deterministic algorithm that uses the private decryption key dk$_{\text{PKE}}$ to decrypt the ciphertext c, thus recovering the original plaintext m.

For simplicity, the following text omits the prefix K-PKE, and abbreviates the primitives' names. For the purposes of this subcategory (N2.2), it is useful to conceptualize Enc' as a probabilistic variant of Enc, as follows: Enc' has the same input/output syntax, but randomly selects the internal values $(y[i], e_1[i], e_2)$, instead of computing them pseudorandomly from the input seed r. Effectively, Enc' ignores the input seed r. For threshold purposes (see §9.2), a key observation is that Enc' and Enc are interchangeable w.r.t. Dec. In particular, Dec(Enc'(...)) = Dec(Enc(...)) for any $\mathsf{ek}_{\mathsf{PKE}}$, m, and r.

A.2.2.2. Higher-Level Constructions (Based on K-PKE)

Conventional ML-KEM. Ignoring KeyGen, the ML-KEM specifies Encaps and Decaps algorithms, each of which relies on an auxiliary internal function (named with a suffix "_internal"), which in turn are based on primitives from the K-PKE scheme. Table 13 lists these relationships. Essentially, the ML-KEM.{Encaps, Decaps} primitives are based on K-PKE.{Enc, Dec} and a number of pseudorandom calculations. They are meant to ensure useful security properties (e.g., IND-CCA) and functionality (e.g., suitability for 2KE). The transformation is efficient in the conventional (i.e., non-threshold) setting, where the computation of NIST-standardized PRFs on secret material is cheap.

Table 13. Non-KeyGen Primitives in ML-KEM and K-PKE

Scheme	Primitive	Prob?	Inputs	Outputs	Alg. in FIPS-203	Internally calls
K-PKE	Encrypt	No	$(\mathsf{ek}_{\mathsf{PKE}}, m, r)$	c	14	—
|	Decrypt	No	$(\mathsf{dk}_{\mathsf{PKE}}, c)$	m	15	—
—	Encaps_internal	No	(ek, m)	(K, c)	17	Encrypt
—	Decaps_internal	No	(dk, c)	K	18	Encrypt, Decrypt
ML-KEM	Encaps	**Yes**	ek	(K, c)	20	Encaps_internal
|	Decaps	No	(dk, c)	K	21	Decaps_internal

Legend: Alg. = Algorithm. K is the "agreed key" in a 2KE; Prob? = Probabilistic?

Threshold ML-KEM: Going from threshold K-PKE.{Enc, Dec} to threshold ML-KEM.{Encaps, Decaps} is impractically expensive in the threshold setting, requiring distributed computations of threshold-unfriendly PRF functions. Submitters are welcome to explore the complexity of such implementations, and possibly propose threshold-friendlier (TF'ier) alternatives. This can be based on TF'ier symmetric primitives (see category S3 in Section 10.3), or TF'ier PKE-based KEM schemes (see category S2 in Section 10.2).

A.3. Category N3: NIST-Specified Symmetric Primitives

See Section 9.3 for submission requirements related to N3.

A.3.1. Subcategory N3.1: AES Enciphering/Deciphering

The Advanced Encryption Standard (AES) FIPS-197 includes an encryption (Enc) algorithm Enc$(K, M) = P$ and a decryption (Dec) algorithm Dec$(K, C) = P$, where K is the key (with 128, 192, or 256 bits), P is the 128-bit plaintext, and C is the 128-bit ciphertext.

Threshold AES enciphering/deciphering. In the threshold AES case of interest, the key-holder is distributed into multiple parties (i.e., a secret sharing of the key is distributed across the parties), and the key-holder computes the ciphertext without reconstructing the key. If implemented in an SSIO-threshold manner, then no party within the decentralized key-holder learns the plaintext or ciphertext.

Comparison with oblivious AES evaluation. Threshold AES enciphering with SSI-plaintext and SSO-ciphertext is similar to but different from two-party oblivious AES evaluation, which is a common secure 2-party computation (S2PC) benchmark in the MPC literature. In the latter, one party (the receiver) knows the plaintext and another party (the key-holder) knows the key. They both keep their inputs private, and yet enable the receiver to learn the corresponding ciphertext. Despite the differences, the building blocks presented for resolving threshold AES may also be useful for implementing oblivious AES evaluation.

A.3.2. Subcategory N3.2: Ascon-AEAD Encrypt/Decrypt

The recent Ascon-based draft NIST-standards describe an authenticated encryption with associated data (AEAD) scheme, named Ascon-AEAD128. It includes encryption (Enc) and decryption (Dec) algorithms, as follows:

- Enc$(K, N, A, P) = (C, T)$
- Dec$(K, N, A, C, T) = \{$output P if T is valid, and output `fail` otherwise$\}$,

where K is the 128-bit key, N is the 128-bit nonce, A is an optional associated data, P is an arbitrary-length plaintext, C is the ciphertext (with the same width as P), and T is a 128-bit authentication tag (truncatable).

One variant mode allows for a 256-bit key, where the extra 128 bits are used to XOR-mask the input nonce, in order to retain the 128 bits of security in a multi-key setting. In that case, Enc$(K||K', N, A, P) =$ Enc$(K, N \oplus K', A, P)$.

A.3.3. Subcategory N3.3: Hash and XOF

Hash functions. The syntax for hashing is Hash$(M) = H$, where M is the plaintext (i.e., message) and H is the output hash. The hash functions of interest for benchmarking in the threshold setting are those with non-truncated output and output length ≥ 224, from:

- SHA2 family [FIPS-180-4]: SHA-256, SHA-384, SHA-512
- SHA3 family [FIPS-202]: SHA3-256, SHA3-384, SHA3-512
- Ascon-Hash256 [SP800-232-ipd]

Extendable output functions (XOF). At a high level, the syntax for XOF'ing is XOF$(M, L) = H$, where M is the plaintext (i.e., message), L is the XOF output length, and H is the output. A XOF with a pre-defined length is essentially a hash function. Additionally, some XOFs are "customizable" via additional input parameters. The XOFs of interest are those based on Keccak and Ascon:

- SHAKE128, SHAKE256 [FIPS-202]
- cSHAKE128, cSHAKE256 [SP800-185]
- Ascon-XOF128 and Ascon-CXOF128 [SP800-232-ipd]

Customizable XOFs. cSHAKE128 and cSHAKE256 are customizable with a "function name" parameter N (e.g., "KMAC" when used as part of the "KMAC" algorithm) and a customization bit-string S. If N and S are empty, then cSHAKE matches the original SHAKE. Ascon-CXOF128 accepts an extra parameter "Z" (customization bit string). However, using Z as an empty string in Ascon-CXOF128 yields a function different from Ascon-XOF128.

A.3.4. Subcategory N3.4: MAC

The high-level syntax for MAC'ing is $T =$ MAC(K, M), where K is the key, M is a message of arbitrary length, and T is the output tag. Depending on the MAC scheme, there may be constraints on the length of the input key and of the output tag. Some MAC constructions allow additional parameters to specify the output length and even a customizable string to adjust the function (akin to domain separation). Depending on the application, a tag truncation (to λ bits) can also be considered, but is hereafter ignored.

MAC construction. In Class N, the NIST-approved MAC schemes are built from primitives (or building blocks therefrom) already considered in the other subcategories of the "symmetric" category (S3). The MAC families of interest for benchmarking are:

- CMAC [SP800-38B] and GMAC [SP800-38D], based on AES. In terms of key length, CMAC-AES-* and GMAC-AES-* use a *-bit key, where $* \in \{128, 256\}$.
- HMAC [SP800-224-ipd], based on a hash function (from the SHA2 or SHA3 families). The key is of arbitrary length, but keys larger than the block of the underlying hash function are first hashed and only then affect the message processing.
- KMAC [SP800-185], based on cSHAKE. KMAC*(K, X, L, S) uses cSHAKE*, where $* \in \{128, 256\}$, K is the key (of arbitrary length; possibly 0), X is the plaintext, L is the output tag length, and S is an optional customization string (possibly empty). A related construction KMACXOF*, with input/output syntax similar to its counterpart KMAC*, allows for specifying L after the algorithm starts computing.

Possibility of an Ascon-based MAC. The NIST draft standard for Ascon-based primitives [SP800-232-ipd] did not explicitly define an Ascon-based MAC. A conceivable construction based on Ascon-AEAD is to encrypt an empty plaintext, using non-empty associated data, which results in a tag that is essentially a probabilistic MAC of the associated data. More efficient specialized constructions are possible. If/when an Ascon-based MAC is defined by NIST, then it can be considered within this subcategory.

A.4. Category N4: NIST-Specified KeyGen Primitives

See Section 9.4 with submission requirements about distributed key generation.

Conventional KeyGen. A key generation (KeyGen) primitive determines a secret key (sometimes called private key) that is needed by subsequent primitives. Depending on the crypto-system, the KeyGen may also generate a related public key. For example, the KeyGen primitive of a digital signature scheme produces a private/public key-pair. The private key is use to sign messages, and the public key is used to verify signatures. A typical security requirement for the secret key is high entropy, which is obtained in case of uniformity in the corresponding key space. In practice, conventional KeyGen schemes often require randomness that is directly output by approved random-bit generators (RBG), which need to satisfy specific requirements [SP800-90A-R1; SP800-90B; SP800-90C-4PD]. Entropy is meant here as a measure of computational unpredictability, rather than in an information-theoretical sense. In fact, keys can even be pseudorandomly derived from other secrets. Also, secret keys can be persistent (e.g., for multiple-time uses, without planned erasure), or ephemeral (e.g., for single-time use, followed by erasure).

A.4.1. Subcategory N4.1: ECC KeyGen

A.4.1.1. Conventional ECC KeyGen

The EdDSA and ECDSA signature schemes [FIPS-186-5] and the ECC-2KE schemes [SP800-56A-Rev3] use particular elliptic curves and encodings. Yet, at a high level they have a similar KeyGen (i.e., their ECC component), determining a private/public key-pair, as follows:

1. Sample a random positive integer d (mod n, the order of the subgroup of interest).
2. Perform a scalar multiplication to obtain the corresponding public key $Q = d \cdot G$.

"Scalar multiplication" versus "exponentiation". Group operations over elliptic curves are usually described with additive notation. When a public key Q is determined by a repeated sum of the base-point G, a secret number d of times, their relationship is mathematically expressed as a scalar multiplication (i.e., $Q = d \cdot G$). However, the literature often refers to this as an "exponentiation", and correspondingly identifies the secret key as the "discrete log" of the public key. This a tolerated misnomer due to the prior popularization of schemes described with multiplicative notation (e.g., $q = g^d$).

KeyGen for the three ECC-based schemes:

- **EdDSA.** The scheme uses a precursor private key d to pseudorandomly derive a private signing key s and a nonce-derivation key ν, as $(s, \nu) = \text{Hash}(d)$. The private signing key is then used to derive the public key $Q = s \cdot G$. The generation of ν is optional in the threshold setting, since interchangeable signatures (w.r.t. verification) can be produced without using the nonce-derivation key (see Appendix A.1.1).
- **ECDSA.** The scheme requires establishing a private signing key (d in ECDSA), and a corresponding public key $Q = d \cdot G$. The private key is later used to produce signatures (see Appendix A.1.2).
- **2KA.** In a threshold 2KA scheme, each party may need a secret sharing of a static private key d_A (or $d_{s,A}$) and/or an ephemeral private key ($d_{e,A}$). After the private key(s) are generated, the side holding it in a secret-shared manner needs to use it (in a subsequent CDH or MQV operation of the 2KA protocol) as the scalar by which to multiply the public key of the other party, and let the result still be in SSO mode (See Appendix A.4.1.2).

A.4.1.2. Extension to CDH and MQV primitives for ECC-2KE

This Threshold Call also considers within the ECC KeyGen subcategory N4.1 the CDH and MQV primitives of NIST-specified ECC-based 2KE schemes [SP800-56A-Rev3]. The association with KeyGen is motivated by two similarities: (i) the essential operations are scalar multiplications; and (ii) the output is a secret key (to be used to derive another secret key). However, one difference is that the ECC-CDH and ECC-MQV primitives (in this section) for 2KE include a secret-shared input.

Conventional primitives. The setting of 2KE [SP800-56A-Rev3] considers two sides (i.e., parties) that want to agree on a fresh key. Let A denote one of the sides, and B the other, (d_i,Q_i) denote a private-public key pair of party $i \in \{A, B\}$, e and s denote *ephemeral* and *static*, and h be the cofactor. Depending on the scheme, the core ECC primitive is as follows, from the perspective of side A:

- ECC-CDH primitive: $P = (h \cdot d_A) \cdot Q_B$
- ECC-MQV primitive: $P = h \cdot impsig_A \cdot (avf(Q_{e,B}) \cdot Q_{S,b})$, where $impsig_A = (d_{e,a} + avf(Q_{e,A}) \cdot d_{s,A})$ mod n, and $avf(.)$ is the "Associate Value Function" [SP800-56A-Rev3, §5.7.2.2] that converts an EC point into an integer. Its **full form** is as described, when both static and ephemeral keys exist and are distinct. There is also a **one-pass form**, when exactly one party (A or B) does not have an ephemeral key, and so the algorithm replaces it with the corresponding static key.

These primitives are used in NIST-specified ECC-2KE to generate an intermediate agreed secret Z (i.e., agreed by both sides), which is then processed by key-derivation and/or key-confirmation primitives that are not ECC-based (and not discussed in this category). Tables 8 and 9 in §A.3 of NIST-IR8214C-ipd (2023) summarize the various approved modes.

Suggested additional curves. Submissions that implement ECC-based 2KE primitives are also welcome to compare the use of P-{256,384,521} versus Curve{25519,448}. The latter are specified in SP800-186, and suggested by RFC7748, but are not recommended by the older SP800-56A-Rev3.

A.4.2. Subcategory N4.2: RSA KeyGen

Conventional primitive. RSA KeyGen is needed for the RSADSA (signature) scheme (see Appendix A.1.1) and for the RSA PKE (encryption) scheme used for 2KE (Appendix A.2.1). In its *basic* format, RSA KeyGen is as follows (at a high level):

- Generate a pair of random secret primes (p, q), and output their product N; and
- Compute and output as private key d the inverse (mod LCM$(p-1, q-1)$) of a public exponent e (selected before the primes).

A.4.2.1. Size of RSA Modulus

The size of an RSA modulus determines an upper bound on the security strength of the RSA-related primitive (i.e., signing, encryption or decryption). The following specific cases are fixed: $|N| = 3072$ for $\kappa \approx 128$, $|N| = 7680$ for $\kappa \approx 192$, and $|N| = 15360$ for $\kappa \approx 256$. By standard, the RSA modulus length $|N|$ must be a multiple of 8. For benchmarking purposes, this Threshold Call further suggests that it be a multiple of 512. Implementations that aim for $\kappa \notin \{128, 192, 256\}$ can interpolate an RSA modulus size by rounding up to a multiple of 512 the solution (for $|N|$) of $\kappa \ln(2) = \sqrt[3]{(64/9) \cdot (|N| \ln(2)) \cdot \ln^2(|N| \ln(2))} - 4.69$, from "§7.5 Strength of Key Establishment Methods" of the "FIPS 140-2 Implementation Guidance" [IG-FIPS-140-2]. For example, one gets $|N| = 10,752$ for $\kappa \approx 224$.

A.4.2.2. Criteria for the Private Exponent and the Prime Factors

NIST specifies requirements for the prime factors of an RSA modulus, and their primality testing. These are described in FIPS-186-5 (§A.1 and §C), and SP800-56B-Rev2 (§6.2–§6.3), respectively for signing and PKE. The output private key can also be represented in a *prime-factor* format, or *CRT* format, as explained in Appendix A.2.1.1. The following paragraph list some of the requirements in the mentioned publications, though (as mentioned in 9.4.2) submissions of threshold schemes may judiciously depart from some of those requirements.

Criteria for the Private Exponent. The private exponent $d = e^{-1} \pmod{L}$, where $L = \text{LCM}(p-1, q-1)$, must be larger than $2^{|N|/2}$ and smaller than L, where the public exponent e, satisfies $2^{16} \leq e \leq 2^{256}$ and is selected before p and q are generated.

Criteria for the prime factors:

- p and q must be of the same bit length (i.e., half the length of the modulus N).
- p and q must be randomly generated (but the two most significant bits of each may be arbitrarily set), as "probable" or "provable" primes, satisfying at least one of the five options from Table 14.

To satisfy the "complex" type of key-generation, the auxiliary primes must exist with certain minimum lengths. If p and q are required to be provable primes, then their minimal required bit-length is roughly half of the minimal required length of probable primes.

A.4.3. Subcategory N4.3: ML KeyGen

Both ML-KEM [FIPS-203] and ML-DSA [FIPS-202] require the generation of a public key-pair, with elements related to lattices. The algorithms of interest are K-PKE.KeyGen and ML-DSA.KeyGen_internal. Both are pseudorandom per specification, using as input a 32-byte (256-bit) random seed. The following descriptions ignore encoding aspects.

Table 14. Criteria for the random primes of an RSA modulus

Type	Sub-type	Provable prime	Probable prime
Simple	provable	p, q	
|	probable		p, q
Complex	provable	p_1, p_2 q_1, q_2 p, q	
|	hybrid	p_1, p_2, q_1, q_2,	p, q
|	probable		p_1, p_2, q_1, q_2, p, q

Per §A.1.1 of FIPS-186-5: p_1, p_2, q_1, q_2 are called auxiliary primes and must be divisors of $p-1$, $p+1$, $q-1$ and $q+1$, respectively, i.e., $p_1|p-1$, $p_2|p+1$, $q_1|q-1$, $q_2|q+1$.

K-PKE.KeyGen. See Algorithm 13 in FIPS-203. Given a 32-byte input seed d, the algorithm starts by pseudorandomly generating a public seed ρ and a private seed σ. The public seed ρ is used to generate a matrix $\hat{\mathbf{A}}$. The private seed σ is used to generate a secret vector $\hat{\mathbf{s}}$ and a secret noise $\hat{\mathbf{e}}$. The pseudorandom samplings are based on the algorithm "SamplePolyCBD", and the hat ^ signifies an application of the Number Theoretic Transform (NTT). The other component of the public key is then obtained by the linear combination $\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$. The output encryption key is $(\hat{\mathbf{t}}, \rho)$ (where ρ is the seed of $\hat{\mathbf{A}}$). The output private key is $\hat{\mathbf{s}}$ (i.e., could be derived from σ).

ML-DSA.KeyGen. See Algorithms 6 and 1 in FIPS-204. ML-DSA.KeyGen selects a random 32-byte seed ξ and calls ML-DSA.KeyGen_internal with that seed as input. The latter is then deterministic, using various pseudorandom sampling routines, starting with generating a public seed ρ, a private seed ρ', and another private seed K. The public seed ρ is used to generate a matrix $\hat{\mathbf{A}}$. The private seed ρ' is used to generate secret vectors $\mathbf{s_1}$ and $\mathbf{s_2}$, and then compute the linear combination $\mathbf{t} = \text{NTT}^{-1}(\hat{\mathbf{A}} \circ \text{NTT}(\mathbf{s_1})) + \mathbf{s_2}$, which can then be parsed into $(\mathbf{t_1}, \mathbf{t_0})$, using the routine "'Power2Round". Finally, the public verification key is $(\rho, \mathbf{t_1})$, and the signing key is composed of $(K$, $\mathbf{s_1}$, $\mathbf{s_2}$, $\mathbf{t_0})$, besides (an encoding of) the public components ρ and $\mathbf{t_1}$, and what can be derived from them (e.g., tr).

A.4.4. Subcategory N4.4: HBS KeyGen.

See Section 9.4.4.

A.4.5. Subcategory N4.5: Secret RBG

Conventional generation. Various primitives require the random generation of a secret bit-string or an integer within an interval, without the need for a corresponding public component. For example, this is the case for: an AES key; a secret-key for encapsulation under an RSA PKE; a nonce for use in other schemes; a salt for a KDM or KC in the scope of a 2KA. Usually, these KeyGen are required to follow NIST-approved RBG methods [SP800-90A-R1; SP800-90B; SP800-90C-4PD], to obtain either truly randomness or pseudorandomness.

Appendix B. Notes on FHE and ZKPoK

This appendix includes informative notes about the FHE (S5) and ZKPoK (S6) categories. Corresponding schemes are welcome to be submitted and/or analyzed in connection with ongoing community efforts (e.g., HomomorphicEncryption.org, FHE.org, ZKProof.org), to promote: (i) fulfill community-based technical recommendations; (ii) align with existing reference material/specifications; and (iii) invite further public scrutiny of proposed schemes; and (iv) consider reference use cases for useful benchmarking.

B.1. Category S5: Fully-Homomorphic Encryption (FHE)

Category S5 (see submission requirements in Section 10.5) allows for the submission of FHE schemes. Informally speaking, an FHE scheme allows for arbitrary computations over the ciphertext space ($\mathcal{C}$), (i.e., over encrypted data). Therefore, in addition to key-generation ($\texttt{KeyGen}$), encryption ($\texttt{Enc}$) and decryption ($\texttt{Dec}$) algorithms, an FHE scheme also needs to specify algorithms for the efficient homomorphic evaluation of a "complete basis" of operations over the plaintext space ($\mathcal{P}$), The composition of operations allows for arbitrary computations (e.g., see Ref. [HES, §1.1.1]).

Traditionally, the complete basis is composed of addition and multiplication, such as, $\{\texttt{xor}, \texttt{and}\}$ over GF(2), or $\{+, \times\}$ over a field modulo a large prime. The FHE scheme supports corresponding homomorphic operations in the ciphertext space ($\mathcal{C}$).

More precisely (though for simplicity leaving the key implicit): supposed a function $f : \mathcal{P} \to \mathcal{P}$ or a binary operation $op : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$ over the plaintext space can be specified as a composition of operations in the supported basis; then, the homomorphism $\texttt{hom}$ allows for corresponding efficient operations in the ciphertext space, satisfying:

- If $c_1 = \texttt{Enc}(p_1)$, $F = \texttt{hom}(f)$, and $d_1 = F(c_1)$, then $\texttt{Dec}(d_1) = f(p_1)$,
- If $c_1 = \texttt{Enc}(p_1)$, $c_2 = \texttt{Enc}(p_2)$, $Op = \texttt{hom}(op)$, and $c_3 = Op(c_1, c_2)$, then $\texttt{Dec}(c_3) = op(p_1, p_2)$,

where p_1 and p_2 are plaintexts, and F and Op can be computed efficiently (i.e., with at most polynomial overhead) from f and op, respectively.

The application suitability of an FHE scheme may depend on variety of aspects, such as:

- The type of operations (and plaintext space) for which $\texttt{hom}$ is "friendly"
- The PQC security strength security categories.
- Whether it is of public-key or symmetric-key type (and how to convert it to the other).

- Threshold friendliness with respect to various primitives of the FHE scheme.

B.1.1. Use Case: FHE-Based AES Oblivious Enciphering

In an “oblivious” PRF evaluation, a client (A) holding a secret plaintext m, and a server (S) holding a secret key k, interact in a way that the client privately learns the output of the PRF evaluation over the plaintext, while both parties remain oblivious to the other party’s input. Replacing the PRF by the AES blockcipher (a PRP), the client learns the AES-enciphering of the plaintext m. Oblivious AES-enciphering is a typical benchmark case for secure 2-party computation, usually using techniques such as garbled circuits and/or oblivious transfer. Compared with an FHE-based solution, usual S2PC protocols lead to faster execution, but also larger communication complexity.

As an FHE use case, §B.1.1.1 considers a non-threshold use of FHE for AES oblivious enciphering. Then, §B.1.1.2 considers the corresponding threshold setting.

B.1.1.1. Non-Threshold FHE-Based AES Oblivious Enciphering

0a. **FHE setup (KeyGen):** An FHE scheme is initialized with encryption key e (for encryption operation FHE.Enc$_e$), and decryption key d (for decryption operation FHE.Dec$_d$), and allows homomorphic evaluation (over FHE ciphertexts) of any function f (within a certain range of functions) using operation FHE.Hom$[f]$.

0b. **AES setup (KeyGen):** An AES cipher is initialized with secret key k; AES.Enc$_k$ denotes the corresponding enciphering operation.

0c. **Parties setup (private inputs):** (i) Client A knows a secret plaintext m and the FHE encryption key e; (ii) Server S knows the AES secret-key k; (iii) and client B (possibly the same as client A) knows the FHE decryption key d.

1. **FHE-Encrypt.** The client A FHE-encrypts the secret plaintext m, obtains the FHE ciphertext $C =$ FHE.Enc$_e(m)$, and sends it to the server S.
2. **FHE-Homomorphic-Evaluate.** The server S homomorphically evaluates the AES enciphering, obtains $H =$ FHE.Hom[AES.Enc$_k](C)$ (which is a valid FHE encryption of the AES enciphering of secret plaintext m), and sends the result to client B.
3. **FHE-Decrypt.** The client B FHE decrypts the received ciphertext H, to obtain the AES-enciphering of the secret plaintext: AES.Enc$_k(m) =$ FHE.Dec$_d(H)$.

4a. **(Optional) Prove correctness.** The server S may also send to client B a ZKPoK string $\pi = \text{ZKPoK.Prove}[k; (H, C) : \text{FHE.Hom}[\text{AES.Enc}_k](C) = H]$, thus ZK-proving knowledge of a secret AES key (k) that is consistent with the homomorphic operation that transformed the initial FHE ciphertext C into the final FHE ciphertext H. A more sophisticated ZKPoK can also be used to prove consistency with some additional public commitment of the AES-key k.

4b. **Verify the proof.** Anyone with the FHE ciphertexts (C, H) can verify the correctness of the ZKPoK π, by checking $\texttt{true} \overset{?}{=} \text{ZKPoK.Verify}(\pi, (H, C), \text{AES.Enc})$.

B.1.1.2. Threshold FHE-Based AES Oblivious Enciphering

Considering the example in Appendix B.1.1.1, the following is a non-exhaustive list of conceivable decentralizations of one of the original participants (i.e., client A, server S, or client B) into a threshold entity composed of multiple parties.

1. **Threshold FHE.KeyGen.** If client B is thresholdized, then a DKG can distributively compute a secret sharing of an FHE decryption key d. If the FHE scheme is of asymmetric-key (i.e., public/private key pair), then the encryption key e might be simply learned by every party. (However, it is also conceivable the case of secret sharing the public key.)
2. **SSI threshold FHE.Enc (encrypt).** If client A is thresholdized and initialized with a secret-shared plaintext m, then a threshold scheme can compute $C = \text{FHE.Enc}_e(m)$ without anyone learning m.
3. **Threshold FHE.Hom (homomorphic evaluation, of a function with a secret parameter).** If the server S is thresholdized and initialized with a secret sharing of the AES key k, then the parties can distributively compute the homomorphic-evaluation, to obtain $H = \text{FHE.Hom}[\text{AES.Enc}_k](C)$, without anyone learning k.
    - In an NSS mode, all server parties learn H.
    - In an SSO mode, each server party learns a secret share of H.
4. **Threshold FHE.Dec (decrypt).** If client B is thresholdized and initialized with a secret sharing of the FHE-decryption key d, then a threshold scheme can decrypt the received value H to obtain $C = AES_k(m)$, without anyone learning d.
    - In an NSS mode, all clientB-parties learn C.
    - In an SSO mode, each clientB-party only learns a secret share of C.
5. **Threshold ZKPoK.** (See category S6 in Section 10.6 and Appendix B.2)

B.2. Category S6: Zero-Knowledge Proof of Knowledge (ZKPoK)

Category S5 allows for the submission of ZKPoKs, which are of great interest in the context of multi-party computation (and beyond). See submission requirements in Section 10.6.

In usual ZKP terminology [ZkpComRef], a ZKPoK is used to prove a **statement** of knowledge, such as knowledge of a secret **witness** (w) that satisfies a given **relation** (R) with a public **instance** (x) such that $R(x, w)$ is true. For example, a ZKPoK of a private RSA key can have as *instance* the public RSA modulus N, as secret *witness* the pair (p, q) of prime factors, and as *relation* the predicate that returns `true` if and only if the input *instance* N is a valid product of two secret primes. Additional refinements can be considered (e.g., proving the two primes have the same bit-length, and are both 3 mod 4).

B.2.1. Example Proofs of Interest

Table 15 lists various examples of ZKPoK of anticipated interest with regard to Class N primitives. Other examples can be conceived, including for primitives in Class S.

Table 15. Example ZKPoKs of interest related to Class N primitives

Related type	**Related sub-category: Primitive**	**Example ZKPoK (including consistency with a corresponding commitment, possibly secret shared)**
KeyGen	N4.1: ECC KeyGen	of discrete log (s or d, e.g., a signing key) of pub key Q
|	N4.2: RSA KeyGen	of factors (p, q), or group order ϕ (w.r.t. N)
Sign	N1.1: EdDSA sign	of nonce-derivation key ν (w.r.t. deterministic signature σ)
|	N1.2: ECDSA sign	of secret signature σ of public msg m, valid w.r.t. public key Q
PKE	N2.1: RSA Enc	of secret plaintext m (w.r.t. ciphertext c and public key N)
|	N2.2: K-PKE Enc	of secret plaintext m (w.r.t. ciphertext c and public key $\mathsf{ek}_{\mathsf{PKE}}$)
|	N2: RSA/K-PKE Dec	of secret-shared plaintext m (after SSO-threshold decryption)
Symmetric	N3.1: AES enciphering	of secret key K (w.r.t. a plaintext/ciphertext pair (P, C))
|	N3.2: Ascon-AEAD	of secret key K and plaintext P (w.r.t. ciphertext/tag pair (C, T))
|	N3.3: Hash/XOF'ing	of secret pre-image M (w.r.t. hash or XOF output H)
|	N3.4: MAC'ing	of secret key K and message M (w.r.t. macTag T)

Relationship between ZKPoK and signatures. Since a NIZKPoK allows for binding data to it (e.g., a message), there is a tight relationship to signature schemes. For example, an EdDSA/Schnorr signature (see Appendix A.1.1) is a message-bound (transferable) NIZKPoK of the discrete log of the public key. As another example, a ZKPoK of an AES key connecting a plaintext to a ciphertext can be the basis of a PQC signature scheme, as observed in submissions to the NIST-PQC standardization process.

B.2.2. Distinguishing Features and Types of "Proof"

1. **ZKP of knowledge (versus of correctness):** The proofs in scope are ZKPoKs, but can also serve the purpose of ZK-proving *correctness* of the secret data (whose knowledge is being proven) and the corresponding public data (e.g., that they are consistently related and satisfy some claimed property). In the literature, a ZKP of correctness is sometimes also known as a ZKP of "language membership".
2. **Interactiveness.** A ZKPoK can involve interaction between the prover and verifier, or be non-interactive (a single message sent from the prover to the verifier). The latter is known as a non-interactive ZKPoK (NIZKPoK). If it is succinct, then it is called a zk-SNARK (ZK succinct non-interactive ZK-argument of knowledge).
3. **Transferability versus deniability.** With a deniable (i.e., non-transferable) ZKPoK, a verifier convinced by the proof cannot transfer said confidence to a third party. This often stems from interactivity, and/or relies on local setup assumptions, such as a local common reference string or local random oracle. Other proofs, usually non-interactive, can be transferred and are publicly verifiable. Another option is a "designated-verifier" proof, which can be achieved by proving a disjunctive ("**or**") statement such as "this statement is true, **or** I know the verifier's private key".

B.2.3. Threshold Considerations

Threshold ZKPoK generation (distributed prover). When the witness is secret shared, a threshold ZKPoK protocol can generate a proof of distributed knowledge. For example, the set of parties that interacted in a DKG can distributively generate a ZKPoK (e.g., via MPC) of the corresponding secret/private key. The proof may relate to public commitments of the corresponding secret shares or/and to a corresponding public key. This ZKPoK generation can be embedded in the DKG protocol or performed afterward.

Threshold ZKPoK verification (distributed verifier). In a threshold ZKPoK verification, a distributed verifier interacts to verify a ZKPoK, without anyone learning the proof. This can make sense for some applications, such as when the ZKP itself or/and the instance is supposed to be private, or when a more efficient proof is possible in that setting and the ZK assurance requires non-collusion by a threshold number of verifier parties.

Conventional ZKPoK about distributed data. ZKPoK generation can be conventional (i.e., non-threshold) or distributed. For example, a dealer (single-party) of a secret sharing can produce a ZKPoK that enables each of the various parties of a threshold entity (i.e., the recipients of secret shares) to non-interactively verify that a given public key and a list of commitments of secret shares are consistent with an adequate secret sharing.

Threshold revealing/hiding. A ZKPoK related to a public *instance* produced by a threshold protocol (e.g., a signature, or an RSA modulus) or to a private (secret shared) *witness* may intentionally reveal or hide the distributed/threshold setting. For example, the threshold setting can be revealed if the proof relates to commitments of the secret shares and/or to public keys of the various parties. This can be achieved based on (i) publicly verifiable secret sharing (PVSS), or (ii) publicly-verifiable MPC. Conversely, the proof can be intentionally indistinguishable from a proof in a conventional setting. This category for ZKPoK submissions is open to the several options (i.e., revealing or hiding the existence of a secret-sharing setting).

B.2.4. On computational soundness from statistical soundness

In interactive protocols, soundness is often characterized as statistical. A transformation to a non-interactive protocol can convert the statistical soundness into computational soundness (i.e., $\kappa \leftarrow \sigma$), which would then be too low if $\sigma < 128$. However, it is sometimes possible to consider a tradeoff with the computational cost of the transformation.

For example, consider a proof generation that requires 2^{24} hashings before the last random value selection that affects the input of a Fiat-Shamir transformation. By intentionally making a Fiat-Shamir transformation based on a 2^{16}-iterated hashing, the computational soundness is increased by 16 bits of security, while only increasing the computational complexity by less than 0.5%. Examples of NIST-standardized iterative use of PRFs can be found in key-derivation functions [SP800-108-Rev1], and password-based key derivation [SP800-132].

B.2.5. Specialized versus generic ZKPoKs

Some ZKPoKs (e.g., of a discrete log, or of an RSA private key) may be based on specialized techniques that are somewhat similar to the operations (e.g., exponentiations) used to commit the secret. Conversely, other ZKPoKs (e.g., when proving knowledge of a pre-image of AES-enciphering or of a SHA-based hashing) may stem more easily from a generic ZKP system that "arithmetizes" the *statement* of knowledge, the *instance*, and the *witness* in some suitable representation (e.g., specifying a Boolean or arithmetic circuit, or some other constrains system, and instantiating its input variables).

For example, the NIST Circuit Complexity project [Proj-CC] collects a few Boolean circuit representations of various NIST-approved primitives/families, such as of AES and SHA. The project may, independently of the Threshold Call, eventually propose a specific representation (file format) for Boolean circuits, to facilitate an interchangeable specification of circuits of certain NIST-specified primitives.

Appendix C. Notes on the Threshold Setting

This section presents a baseline system model for threshold schemes (Appendix C.1), discusses the need for a security analysis (Appendix C.2), suggests various "threshold profiles" (Appendix C.3) and characterizes input/output interfaces (Appendix C.4).

C.1. System Model

The specification of each submitted threshold scheme will describe (in CSx.2; see Section 5.4) one system model (and may identify possible variants), including the set of participants, the communication model, and the adversarial model (including goals and capabilities). In addition to the actual "parties" that hold the secret-shared keys, the system may include coordinators, administrators, clients and other devices (e.g., routers, clocks, random-bit generators). The model needs to explain how the parties are activated (e.g., via an authorized/authenticated client request, or by an administrator) and describe the applicable input/output secret-sharing interfaces (see Appendix C.4). The description should strive for clarity about variable options across possible deployment scenarios (e.g., DKG versus secret sharing by a dealer).

The paragraphs ahead describe baseline assumptions and options with regard to participants (Appendix C.1.1), communication (Appendix C.1.2), and the adversary (Appendix C.1.3). These assumptions are intended to serve as a baseline reference, neither precluding submissions with sophisticated nuances, nor eliminating the utility of security evaluation across diverse deployment scenarios.

C.1.1. Participants

The parties in a threshold entity. There is a "threshold entity" composed on n "parties", which is collectively responsible for executing a cryptographic primitive. At the onset, all parties "know who" the n parties are and agree on n identifiers (e.g., public keys to support authenticated channels). The suitability of public keys may need to be verified (e.g., via zero-knowledge proofs) in the KeyGen phase or subsequently.

The assumption of initial agreement on n identifiers is a possibility, not a requirement. A threshold scheme ***may*** be bootstrapped without prior agreement about who the n parties/identifiers are (or even the value of n), case in which the protocol may an additional initial phase for setting up some agreement. However, that may be a distributed-systems problem outside the scope of exploring the essential cryptographic thresholdization of the primitive at stake. A submission that considers an additional preparatory phase for the agreement of n and who the n parties are ***should*** present that phase modularly separated from the remaining threshold scheme.

Beneficiaries. For some operations (e.g., threshold KeyGen), the *beneficiaries* of the computation are the parties that end with a new secret-sharing state (possibly requiring agreement in the sense of "security with **unanimous** abort"), and/or an administrator (e.g., who receives a new public key). For other operations (e.g., threshold signing), the beneficiary can be an external client who requested the computation (from a threshold entity), in order to obtain an output.

Client interface. The client may or may not be aware of or be able to interact distinctively based on the n-party composition. This ability can be affected by the input/output (I/O) interface (see Appendix C.4). For example, a secret sharing of the I/O can affect whether a client can separately send or receive input/output shares to or from each party.

Intermediaries. The possibility of **concurrent** execution requests ***should*** be considered. A baseline description ***may*** assume that there is a malicious **proxy** that can intermediate the communication between clients and the threshold entity, and authorize requested operations (e.g., the signing of a message).

C.1.2. Distributed Systems and Communication

When the interface and rules for composition are clear, the specification of a threshold scheme ***should*** decouple the description of (i) the building blocks (e.g., consensus, reliable broadcast) of classical distributed-systems, from that of (ii) the cryptographic operations needed to support the secure multiparty computation over (or of) a secret-shared key. See Pre4 and CSx.3 regarding recommendations for the modular specification of building blocks. The needed networking tools (e.g., broadcast) can be instantiated based on weaker resources (e.g., point-to-point channels) that are specified by referencing existing specifications available to the public for free. However, the reference implementation still needs to include code for them (see Imp1).

A baseline description ***may*** make strong assumptions about the communication network, including synchrony and reliability of transmission. However, different communication environments can have different optimal threshold schemes, depending on guarantees (or the lack thereof) of **synchrony**, **broadcast**, and **reliability** (of message delivery). A submission ***should*** discuss the pitfalls of deploying a threshold scheme in an environment with weaker guarantees (e.g., with asynchronous and unreliable channels), and possible mitigations (see CSx.4). Ideally, the security analysis (see CSx.4) explains which security guarantees break across these environments.

C.1.3. Adversary

The system model also needs to consider an adversary that corrupts some of the parties. As mentioned in Section 8.2.2, corruptions can be characterized in various ways, such as active, adaptive and mobile. The suggestion to consider a mobile adversary is intended to induce the characterization of various levels of insecurity (e.g., which properties break) when acceptable thresholds are surpassed. In practice, the adversary's capabilities ***may*** be modeled as part of the security idealization (see Appendix C.2.3)

C.2. Security in the Threshold Setting

C.2.1. Security Analysis (Based on the Specification)

In modern cryptography, security proofs are fundamental for a proper security assessment of multi-party threshold schemes. A "security proof" proves that a proposed threshold scheme satisfies a proposed security formulation in a suitable adversarial context (see Section 8.2.3). Such proof ***may*** be given by showing "emulation" of the ideal functionality, or by showing that a non-negligible adversarial advantage in each security game implies breaking an assumption. The security analysis, which ***may*** be based on assumptions different from those inherent to the underlying cryptographic primitive (being thresholdized), ***should*** assess security under various compositions, including concurrent executions.

Coverage of security properties. Some aspects of useful security analysis often overflow the scope of a proof/idealization. The security analysis ***should*** discuss which known useful properties are captured by the idealized security, and which ones are not. For example:

- **Security with abort.** Even though availability is a generically desirable property, a security formulation with emphasis on confidentiality and integrity ***may*** purposely specify that an adversary is allowed to abort protocol executions, so that the formulated security is more easily achievable.
- **Inadvertent malleability.** A sole requirement of hiding and binding for a commitment scheme would not suffice for a use (e.g., committing bids in an auction) that would also require a non-malleability property.

The security analysis ***should*** also discuss: (i) the security consequences (e.g., loss of some type of composability) of foreseen real instantiations of components or setups that were idealized in the security proof; and (ii) whether/how the cryptographic assumptions sustaining the threshold scheme are different from those sustaining the conventional primitive.

C.2.2. Practical Feasibility Versus Adaptive Security

Adaptive security (i.e., security against adaptive corruptions) may pose significant challenges in formal proofs of security, depending on the security formulation. For example, while deniability of execution may in some cases be required for indistinguishability between ideal and real executions, the use of non-committing encryption to achieve it could be excessive without a necessary practical benefit. However, a proposed protocol must not allow its critical safety properties to be trivially broken in case of adaptive corruptions, as in the classical example of a protocol that delegates all capabilities to a small quorum that is difficult to guess in advance, but which is announced during the protocol and whose overall corruption would be disastrous.

Certain security assurances (e.g., liveness and termination options) ***may*** vary across different adversaries. For example, a security analysis may prove security against static corruptions with respect to some formulation (e.g., simulation-based), and then in complement show which fundamental security properties or attributes (e.g., unforgeability) remain preserved against adaptive corruptions in another formulation (e.g., game-based), even if some other security properties (e.g., some aspect of composability) are not preserved. The set of security formulations across submissions of threshold schemes (some possibly proving adaptive security based on unrealizable assumptions, such as a programmable random oracle) will enrich the body of reference material for public analysis.

Feedback is welcome on security formulations and reference approaches that simultaneously enable both practical feasibility and security (for properties of interest) against adaptive corruptions, as well as acceptable tradeoffs.

C.2.3. Implementation and Deployment Security

The security analysis required in CSx.4 refers to the logical specification of the threshold scheme (CSx.2–CSx.3). Comments about implementation or deployment security are also welcome, including in CSx.6.

C.3. Threshold Profiles

For each primitive (see Sections 9 and 10) considered for thresholdization, a variety of solutions is possible across threshold parametrizations. Therefore, it is useful to consider the notion of “threshold profile”, defining a suitable threshold-parametrization range for secure operations. The threshold profile ***should*** characterize at least the total number (n) of parties, and the various corruption thresholds (f) and participation thresholds (k). Note the use of plural (“thresholds”), since they may vary depending on which security property is being evaluated. Table 16 proposes succinct labels for several default profiles obtained from a restriction in the number of parties and the corruption threshold.

Motivating adoption. There is value in identifying motivating applications for the adoption of threshold schemes in each threshold profile. Therefore, the submission ***should*** identify (in CSx.6) use-cases for which the proposed threshold ranges are adequate.

For convenience of discussion, the following nomenclature is defined to easily identify some default threshold profiles, based on the total number of parties and/or some corruption threshold (f) assumed clear in the context.

- **Number n of parties:** (2) “two” for $n = 2$; (3) “three” for $n = 3$; (S) “small” for $4 \leq n \leq 8$; (M) “medium” for $9 \leq n \leq 64$; (L) “large” for $65 \leq n \leq 1024$; and (E) “enormous” for $n > 1024$.
- **Corruption proportion f/n:** (D) “dishonest majority” for $f \geq n/2$; (h) “honest majority” for $f < n/2$; (H) “two-thirds honest majority” $f < n/3$.

Table 16. Labels for some template threshold profiles

Corruption proportion		**Number of parties (n)**					
f/n	**Majority type**	**Two (2)** $n = 2$	**Three (3)** $n = 3$	**Small (S)** $4 \leq n \leq 8$	**Medium (M)** $9 \leq n \leq 64$	**Large (L)** $65 \leq n \leq 1024$	**Enormous (E)** $n \geq 1025$
$\geq 1/2$	Dishonest (D)	$n2$	n3fD	nSfD	nMfD	nLfD	nEfD
$> 1/3$	Honest (h)	—	n3fh	nSfh	nMfh	nLfh	nEfh
$< 1/3$	2/3 Honest (H)	—	—	nSfH	nMfH	nLfH	nEfH

The default profiles exclude the cases $f = 0$ and $f = n$. Therefore, for the “two”-party profile (with $n = 2$) — the usual **s**ecure two-**p**arty **c**omputation (S2PC) setting — only the “dishonest majority” case matters (with $f = 1$). For the “three”-party profile, the 2/3 honest majority case does not apply.

Other threshold profiles. Other threshold profiles can be considered in concrete submissions. For example, some threshold schemes may have advantageous properties when considering an even stricter honest majority, such as more than 3/4 of honest parties. For other threshold schemes, the magnitude of the number of possible quorums may matter more. For example, if a quorum is valid if and only if it has $f+1$ parties out of n, then the number of such quorums is "n choose $f+1$" (i.e., $\binom{n}{f+1} = n!/((f+1)!(n-(f+1))!)$. A protocol may have a complexity proportional to this number $\binom{n}{f+1}$, in which case it will stop being practical for certain parameters.

The submission team is responsible for defining the threshold profile(s) with which their proposed threshold schemes are secure and practical. In some cases it may be useful to distinguish between corruption threshold and participation-minus-1 threshold.

Alternative monotonic access structures. The use of the traditional term "threshold" in this Threshold Call is not meant to suppress possible submissions of schemes suitable for other useful and properly justified access structures. Depending on which secret-sharing schemes and/or threshold schemes support the distributed computation, it is possible to consider monotone access structures (i.e., where any superset of a quorum is also a quorum) different from a simple threshold. In other words, a submission ***may*** consider a distributed system with a well-specified monotonic access structure different from a threshold one.

C.4. Secret-Shared Input/Output (I/O) Interfaces

Per §2.3 of NIST-IR8214A, there are various I/O interfaces of interest w.r.t. secret-sharing. The default case of secret-sharing the secret or private key is often left implicit. However, other input or output arguments can also be characterized. The baseline characterizations of interface (w.r.t. an identified input or output) are: non-secret-shared (NSS), secret-shared input (SSI), and secret-shared output (SSO). This section describes various cases of interest.

Implicit I/O secret-sharing modes: For keyed primitives, the secret sharing of the private/secret key is assumed by default, which is often left implicit:

- **[SSO] KeyGen.** By default, a threshold keygen scheme produces a secret-shared output (i.e., a secret-shared secret/private key) and (when applicable) a corresponding non-secret-shared public-key counterpart.
- **Subsequent [SSI] operation.** After the KeyGen (produced by a dealer or via a DKG), the subsequent threshold operation (e.g., signing) uses the private/secret key as a secret-shared input to retain its confidentiality.

Since the secret sharings in these modes are assumed by default, the modes can be referred to as non-secret-sharing (NSS) modes when no other input or output (i.e., besides the private key) are secret shared.

Other I/O secret-sharing modes: When other elements (i.e., besides the main secret/private key) are secret shared, to remain hidden from individual parties. For example:

- **SSO-decryption.** A threshold decryption can be in SSO mode w.r.t. the decrypted plaintext. This can be useful for pair-wise key-agreement (2KA) when one side (thresholdized) of the 2KA decapsulates the key-contribution sent by the other side.
- **SSI-encryption.** A threshold public-key encryption can be in SSI mode w.r.t. the plaintext (which ***may*** be a secret for use in another context).
- **SSO-KA.** In an ECC-2KA (a la Diffie-Hellman), the EC-primitive (e.g., CDH or MQV; see Appendix A.4.1.2) produces an output that will seed an agreed key. If one party is thresholdized, their output should be in SSO mode.
- **SSI signing.** A threshold signing can keep the message private by using an SSI mode w.r.t. the message. However, the needed threshold hashing then brings a significant overhead. Instead, one can consider the message hash being secret shared by a dealer (e.g., a client who is requesting the signature computation), such that each party in the threshold scheme directly receives a secret share of the message hash (or, as applicable, of the hash of a combination of various public elements and the message).
- **SSIO signing.** An SSIO mode (combining SSI and SSO) can be useful in privacy-enhancing contexts (e.g., for a time-stamping service). For example, a client can provide shares of a message hash to a threshold entity, and then receive signature shares. The result is similar to a blind-signing service, except that (i) the signing key is further thresholdized, and (ii) the knowledge of the message/signature is within the theoretical reach of a threshold coalition of signer-parties.

Provided that at least the default private/secret key is secret shared, a submission can choose to cover one or more secret-sharing interfaces (i.e., NSS, SSI, SSO, SSIO) for any of the other input or output elements. Some correctness challenges with SSI and SSO modes may be resolved with ZKPs or/and verifiable secret-sharing modes.

A threshold scheme receiving an SSI input (including a secret-shared key) ***may*** assume that the secret-shared input is correct. For example, this is the case if it is obtained from a previous secure threshold execution (e.g., a DKG), or from a trustworthy dealer, or has been verified as correct after having been dealt by an untrusted dealer. Alternatively, the threshold scheme ***may*** be devised to be inherently resilient against a malicious secret sharing, (e.g., by starting with a joint computation to confirm correctness of the secret-shared state).

Appendix D. Acronyms

- **2KA**: Pair-Wise **K**ey-**A**greement
- **2KE**: Pair-wise **K**ey-**E**stablishment
- **2PD**: Second **P**ublic **D**raft
- **ABE**: **A**ttribute-**B**ased **E**ncryption
- **AEAD**: **A**uthenticated **E**ncryption with **A**ssociated **D**ata
- **AES**: **A**dvanced **E**ncryption **S**tandard
- **API**: **A**pplication **P**rogramming **I**nterface
- **CDH**: **C**ofactor **D**iffie–**H**ellman
- **CMAC**: **C**ipher-based **MAC**
- **CPU**: **C**entral **P**rocessing **U**nit
- **CRT**: **C**hinese **R**emainder **T**heorem
- **DKG**: **D**istributed **K**ey **G**eneration
- **DOI**: **D**igital **O**bject **I**dentifier
- **CCA**: **C**hosen-**C**iphertext **A**ttack
- **CPA**: **C**hosen-**P**laintext **A**ttack
- **DSA**: **D**igital **S**ignature **A**lgorithm
- **ECC**: **E**lliptic **C**urve **C**ryptography
- **ECDSA**: **E**lliptic **C**urve **DSA**
- **EdDSA**: **E**dwards **C**urve **DSA**
- **EMSA**: **E**ncoding **M**ethod for **S**ignature with **A**ppendix
- **FFC**: **F**inite **F**ield **C**ryptography
- **FHE**: **F**ully-**H**omomorphic **E**ncryption
- **FIPS**: **F**ederal **I**nformation **P**rocessing **S**tandards
- **FORS**: Forest of Random Subsets
- **FR**: **F**ield **R**epresentation **I**ndicator
- **GB**: **G**iga**b**yte (1,000,000,000 bytes)
- **GC**: **G**arbled **C**ircuit
- **GF**: **G**alois **F**ield
- **HBS**: **H**ash-**B**ased **S**ignatures
- **HMAC**: **H**ash-based **MAC**
- **HQC**: **H**Hamming **Q**uasi-**C**yclic
- **HSS**: Hierarchical Signature Scheme
- **IBE**: **I**dentity-**B**ased **E**ncryption
- **IETF**: **I**nternet **E**ngineering **T**ask **F**orce
- **IND**: **Ind**istinguishability
- **I/O**: **I**nput/**O**utput
- **IPD**: **I**nitial **P**ublic **D**raft
- **IRTF**: **I**nternet **R**esearch **T**ask **F**orce
- **ITL**: **I**nformation **T**echnology **L**aboratory
- **KA**: **K**ey **A**greement
- **KAS1/2**: **K**ey **A**greement **S**cheme 1 or 2
- **KAT**: **K**nown-**A**nswer **T**est
- **KC**: **K**ey **C**onfirmation
- **KDM**: **K**ey-**D**erivation **M**echanism
- **KEM**: **K**ey-**E**ncapsulation **M**ethod
- **KMAC**: **K**eccak-based **MAC**
- **K-PKE**: ML-**K**EM-based **PKE**
- **KT**: **K**ey-**T**ransport
- **LCM**: **L**east **C**ommon **M**ultiplier
- **LMS**: Leighton-Micali signature
- **LTS**: **L**ong-**T**erm **S**upport

- **LWC**: **L**ight**W**eight **C**ryptography
- **MAC**: **M**essage **A**uthentication **C**ode
- **ML**: **M**odule **L**attice
- **MPC**: (Secure) **M**ulti**P**arty **C**omputation
- **MPTC**: **M**ulti-**P**arty **T**hreshold **C**ryptography
- **MQV**: **M**enezes-**Q**u-**V**anstone
- **NIST**: **N**ational **I**nstitute of **S**tandards and **T**echnology
- **NIZK**: **N**on-**I**nteractive **Z**ero-**K**nowledge
- **NISTIR**: **NIST** **I**nternal **R**eport
- **NSS**: **N**ot-**S**ecret-**S**hared (Input/Output)
- **OAEP** **O**ptimal **A**symmetric **E**ncryption **P**adding
- **OTS**: **O**ne **T**ime **S**ignature
- **PDF**: **P**ortable **D**ocument **F**ormat
- **PF**: **P**lat**f**orm
- **PEC**: **P**rivacy-**E**nhancing **C**ryptography
- **PQ**: **P**ost-**Q**uantum (i.e., quantum-resistant)
- **PQC**: **P**ost-**Q**uantum **C**ryptography
- **PKC**: **P**ublic-**K**ey **C**ryptography
- **PKCS**: **P**ublic-**K**ey **C**ryptography **S**tandards
- **PKE**: **P**ublic-**K**ey **E**ncryption
- **PRF**: **P**seudo**r**andom **F**unction Family
- **PRP**: **P**seudo**r**andom **P**ermutation Family
- **PSS**: **P**robabilistic **S**ignature **S**cheme
- **PVSS** **P**ublicly **V**erifiable **S**ecret **S**haring
- **RAM**: **R**andom **A**ccess **M**emory
- **RBG**: **R**andom-**B**it **G**enerator/**G**eneration
- **RFC**: **R**equest **f**or **C**omments
- **RSA**: **R**ivest–**S**hamir–**A**dleman
- **RSADP**: **RSA** **D**ecryption **P**rimitive
- **RSADSA**: **RSA** **D**igital **S**ignature **A**lgorithm
- **RSAEP**: **RSA** **E**ncryption **P**rimitive
- **RSASP**: **RSA** **S**ignature **P**rimitive
- **RSAVP**: **RSA** **V**erification **P**rimitive
- **RSASSA**: **RSA** **S**ignature **S**cheme with **A**ppendix
- **RSASVE**: **RSA** **S**ecret-**V**alue **E**ncapsulation
- **S2PC**: **S**ecure **Two**-**P**arty **C**omputation
- **SHA**: **S**ecure **H**ash **A**lgorithm
- **SHAKE**: **S**ecure **H**ash **A**lgorithm with **KE**CCAK
- **SLH**: **S**tate**l**ess **H**ash
- **SNARK**: **S**uccinct **N**on-Interactive **Ar**gument of **K**nowledge
- **SP 800**: **S**pecial **P**ublication in Computer Security
- **SSD**: **S**olid **S**tate **D**rive
- **SSI**: **S**ecret-**S**hared **I**nput
- **SSIO**: **S**ecret-**S**hared **I**nput-and-**O**utput
- **SSO**: **S**ecret-**S**hared **O**utput
- **SVE**: **S**ecret-**V**alue **E**ncapsulation
- **TagGen**: **Tag** **Gen**eration
- **TB**: **T**era**b**yte (1,000,000,000,000 bytes)
- **TF**: **T**hreshold-**F**riendly
- **URL**: **U**niform **R**esource **L**ocator
- **WOTS**$^{+}$: Winternitz One-Time Signature Plus
- **XMSS**: eXtended Merkle Signature Scheme
- **XMSS**$^{\mathbf{MT}}$: Multi-Tree XMSS
- **XOF**: E**x**tendable **O**utput **F**unction
- **ZKP**: **Z**ero **K**nowledge **P**roof
- **ZKPoK**: **Z**ero **K**nowledge **P**roof **o**f **K**nowledge

Appendix E. Changes Between the IPD and the 2PD

Several updates have been made between the initial public draft (ipd) [NIST-IR8214C-2pd] and the second public draft (2pd) [NIST-IR8214C-ipd-comms] of the NIST Threshold Call. The following list describes various updates:

Submission requirements:

1. **Submission subphases and deadlines.** Section 4 (old §4.1) refined the specification of phases and deadlines (see Table 4). The old "Ph1. (Optional) Early abstract" was refined into a "Ph1. Previews" phase with two opportunities for teams to share in advance their plans for package submission. The section also adds Ph3. Public analysis to clarify the intended period of public analysis after the submissions.
2. **Written specification.** Section 5 (old §4.2) defines a more refined structure for the "Specification" component. The main matter is now composed of a preliminaries part, and one or more crypto-systems parts. Distinct crypto-systems can be proposed by distinct subteams. The section "Pre3: Related work and design decisions" merges the old "S5" (Prior work) and "S11" (choices and comparisons).
3. **Reference implementation.** Section 6 (old §4.3) clarifies terminology, and requirements about scope, availability, open-source licensing, compilability, dependencies, clarity, scripts and instructions (the latter two were previously in another section). External dependencies (e.g., compiler, and third-party libraries) are allowed if they open source, well-identified, and automatically integrated in the compilation phase (X1).
4. **Implied agreement and patents disclosure.** Section 4.4 (old §4.6) defines expectations about submitted packages. Also, the section "Cv4 Patents disclosure" in the verso of the specification requires listing known related patents associated with the submitters.
5. **Notes specific to each (sub)category.** The content in the old §6, §7, and §A, with information about the primitives in Class N (old Cat1) and Class S (old Cat2), was revised and reorganized for better separation between (i) content with requirements and recommendations ("shall" statements) in Sections 9 and 10 and (ii) other informative content about the conventional schemes (in Appendices A and B).
6. **Security requirements.** The new Section 8.1 gathers new notes on security strength levels, including in relation to post-quantum security categories (θ). The new Section 8.2) adapts some content from the old §5, to list requirements for threshold security. The number of required parametrizations was reduced from two to one (aiming at $\kappa \gtrsim 128$, or $\theta \geq 1$). A second parametrization is encouraged (aiming at $\kappa \gtrsim 192$, or $\theta \geq 3$) but not required.

Organization of Categories:

1. **Succinct indexation.** The old categories Cat1 and Cat2 have been renamed to two Class N (for **N**IST-specified primitives) and Class S (for **S**pecial primitives, not specified by

NIST), respectively. Correspondingly, the old subcategories prefixed with “C1.” and “C2.”, have been renamed to categories prefixed with “N” and “S”.

2. **PQC primitives in Class N.** The categories for signing (new N1; old C1.1) and PKE (new N2; old C1.2) are now open to the recent NIST-PQC standardized primitives.
3. **KA primitives in Class N and Class S.** The categories (old C1.3 and C2.3) for non-PKE primitives for key-agreement (KA) (a la Diffie-Hellman) have been integrated into the categories of KeyGen primitives (new N4 and S4).
4. **Symmetric primitives.** In the “symmetric” categories (new N3 and S3; old C1.4 and C2.4), the primitives related to hash, XOF and MAC are presented more straightforwardly, rather than relying on the applications of key-derivation and key-confirmation used in pair-wise key-establishment.
5. **LWC primitives in Class N.** The “symmetric” category (new N3; old C1.4) is now also open to Ascon primitives (for encryption, hashing, and XOF’ing) selected by NIST-LWC.
6. **FHE category.** The old C2.6 for “advanced” primitives has been adapted into a narrower category S5 for fully-homomorphic encryption (FHE). The previously exemplified cases of IBE and ABE are left outside the scope. (They remain of interest to the NIST-PEC project.)
7. **ZKPoK.** The ZKPoK category (new S6; old C2.7) is better described, including a thorougher table of examples (in S6).

Appendices:

1. **Notes on Categories.** Appendices A (about Class N; old Cat1), and B (about Class S; old Cat2) include adapted informative notes from the old Appendix A (details for subcategories), after moving-with-revision related requirements to Sections 9 and 10.
2. **Notes on Threshold Setting.** The new Appendix C adapts informative content from the old §5, excluding requirements (“shall” statements, which moved with adaptation to Section 8.2). The old §6.3 on Input/Output interfaces was revised into new Appendix C.4.
3. **Checklists.** The old “B. Submission checklists” was removed. It may reappear in the future latex template for submissions.
4. **Acronyms.** The list of acronyms was moved from old §2 to Appendix D.
5. **List of changes.** The new Appendix E lists changes between the two public drafts (IPD and 2PD) of the Threshold Call.

Other edits:

1. **Front matter.** The front matter follows a new NISTIR template. It revised the Preface and Acknowledgments, and added a new Note to the Reviewers.
2. **PQ-QV combinations.** The new Section 3.2 discusses post-quantum (PQ) versus quantum vulnerable (QV) techniques, and expectations on PQC-migration.