NIST Internal Report
NIST IR 8214C

NIST First Call for Multi-Party Threshold Schemes

Luís T. A. N. Brandão
René Peralta

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

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

NIST Internal Report
NIST IR 8214C

NIST First Call for Multi-Party Threshold Schemes

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

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

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

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

January 2026

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.

Patent Disclosure Notice

NOTICE: ITL has requested that holders of patent claims whose use may be required for compliance with the guidance or requirements of this publication disclose such patent claims to ITL. However, holders of patents are not obligated to respond to ITL calls for patents and ITL has not undertaken a patent search in order to identify which, if any, patents may apply to this publication.

As of the date of publication and following call(s) for the identification of patent claims whose use may be required for compliance with the guidance or requirements of this publication, no such patent claims have been identified to ITL. No representation is made or implied by ITL that licenses are not required to avoid patent infringement in the use of this publication.

Publication History

Approved by the NIST Editorial Review Board on 2026-01-15.

Suggested Citation

Luís T. A. N. Brandão, René Peralta (2026). NIST First Call for Multi-Party Threshold Schemes. (National Institute of Standards and Technology, Gaithersburg, MD) NIST Internal Report (IR) 8214C. DOI:10.6028/NIST.IR.8214C

Author Names and ORCID Identifiers

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

Additional Information

Additional information about this publication, including related content, potential updates, and document history, is available at https://csrc.nist.gov/pubs/ir/8214/c/final

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

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

Abstract

This is the NIST Threshold Call, calling 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 materials on advanced cryptography. In a threshold scheme, a reference cryptographic primitive (e.g., signing, encryption, decryption, key generation) is computed in a distributed manner, while its private/secret key is or becomes secret-shared across various parties. The threshold schemes submitted in reply to this call will be *interchangeable* with a reference non-threshold primitive of interest, in the sense that their outputs can be used interchangeably in a subsequent operation. The primitives of interest are organized into various categories, across 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 friendly or have useful functional features. The scope of Class S also includes fully-homomorphic encryption, zero-knowledge proofs, and auxiliary gadgets. This document specifies submission phases, and the requirements for submitting a package, including a technical specification, a reference implementation, and a report on experimental evaluation. A subsequent phase of 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 future processes.

Keywords

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

Table of Contents

List of Tables

List of Requirements (*shall* statements)

Preface

From a standardization perspective, the exploration of “advanced” cryptography is a challenging endeavor. It may require “advanced” or custom processes, addressing challenging questions: *Focus on building blocks or protocols? Which security properties to require? How to cover a vast range of possibilities? What and when to standardize or recommend?*

This “NIST First Call for Multi-Party Threshold Schemes” (or, simply, “Threshold Call”) establishes a process for developing a high-quality body of reference materials, to be open for public analysis. The call stands at the threshold of advanced cryptography, proposing an exploration of multi-party computation (MPC) techniques, namely threshold schemes. These allow distribution of trust in the implementation of cryptographic primitives, including those standardized by NIST, and others with special useful properties. Fully-homomorphic encryption (FHE) and zero-knowledge proofs (ZKP) are also within the scope of interest.

The envisioned process relies on international community engagement by multiple teams of experts. A “Previews” phase will enable early presentation of plans of future package submissions, to foster public awareness and discussion, in time for adaptations. Each submission package can propose multiple crypto-systems, developed by collaborative teams.

For transparency and involvement, the Threshold Call was widely disseminated in advance, inviting public comments on an initial public draft (ipd) [NIST-IR8214C-ipd; PubComsIPD] and a second public draft (2pd) [NIST-IR8214C-2pd; PubComs2PD]. It was presented at international cryptography events, and promoted in various NIST workshops, such as the NIST Workshop on Multi-Party Threshold Schemes (MPTS) 2023, the Special Topics on Privacy and Public Auditability 6th event (STPPA6), and the NIST Workshop on Privacy-Enhancing Cryptography (WPEC) 2024. Also, (MPTS) 2026 is near simultaneous with this publication.

This document is intended for the following audience: cryptography experts interested in providing constructive technical feedback, or collaborating in the development of open reference materials; all those, in academia, industry, government, and the general public, who are interested in the development and use of threshold schemes and advanced cryptography, including corresponding best practices and recommendations.

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 many reviewers who submitted written feedback in the form of public comments [PubComsIPD; PubComs2PD] in response to the two public drafts (January 2023, and March 2025) of this publication, and the many researchers who demonstrated interest through conversations at events and invitations for related presentations.

Note to the Readers

The NIST First Call for Multi-Party Threshold Schemes asks the public to submit packages with the Technical Specification, Reference Implementation, and Report on Experimental Evaluation of crypto-systems. These are intended as proposals for consideration by the NIST Multi-Party Threshold Cryptography (MPTC) project [Proj-MPTC], to curate a public body of reference materials. The present document serves as the baseline reference for this process. There is also a supporting LaTeX-based template [Template] (live document). The preparation of packages should follow the described structure, and consider possible adaptations that may be announced in the MPTC-Forum and on the NIST-MPTC page. In particular, the deadlines for each phase will be announced via the MPTC-Forum. The envisioned discussion of "previews" may also result in related recommendations.

1. Introduction

Threshold schemes. Modern cryptography enables a multi-party implementation paradigm based on developments in the fields of threshold cryptography, secure multiparty computation (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 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*.

Applicable types of primitives. Threshold schemes can be applied to various types of cryptographic primitives, with regard to how the private or secret key is used: key-generation (KeyGen) outputs a private or secret key (possibly along with public-key material); key-based primitives (e.g., signing, decryption, and enciphering) require a private or secret key as input; and key-less primitives (e.g., hashing) can nonetheless be applied to keys with secrecy requirements. In a traditional (non-threshold) 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. Such party with access to the key is then a *system-failure point* for confidentiality, integrity and availability.

Threshold Call. This is the "NIST First Call for Multi-Party Threshold Schemes" (or, simply, "Threshold Call"). This public **call** asks for high-quality submissions of multi-party threshold schemes for selected primitives across two classes: **N**IST-specified primitives (Class N) and **S**pecial others (Class S). For example, Class S includes primitives from full-homomorphic encryption schemes. Beyond the threshold scope, this call also asks for the submission of auxiliary components, such as zero-knowledge proofs of knowledge (ZKPoKs) and threshold-useful *gadgets*, which do not need to be thresholdized.

In this Threshold Call, the term "**crypto-system**" is used in an encompassing sense to denote a main system submitted for analysis. It can be a threshold scheme, a ZKPoK, a gadget, a Class S conventional scheme (i.e., non-threshold), or a family thereof. The submission needs to follow the structure described in this Threshold Call.

Requirement 1.1. Four main components in a submission package

The submission of a crypto-system (or various at once) by a team ***shall*** be structured in a package with inter-related Technical Specification, Reference Implementation, Report on Experimental Evaluation, and Notes on Patent Claims.

The Technical Specification includes design rationale, technical description, and security characterization. The Reference Implementation is licensed as open-source and includes instructions and scripts for compilation, execution testing, and measurements. The Report on Experimental Evaluation describes the experimental setting and configurations, summarizes measurements of performance, and interprets the results. The Notes on Patent Claims lists known patents (or otherwise claims absent knowledge of patents) that, being associated with the team members, may cover techniques within the submitted crypto-systems.

Threshold friendliness. The term "threshold friendly" (TF) is informally used throughout, denoting some practical suitability in the threshold setting. The friendliness factor (e.g., performance) is sometimes implicit or considered in comparison with another scheme.

1.1. Cryptographic Schemes and Primitives

Traditional techniques. For several decades, the National Institute of Standards and Technology (NIST) in the United States has standardized important cryptographic primitives and schemes, in Federal Information Processing Standards (FIPS) publications, and Special Publications (SP) in Computer Security (the SP 800 series). For example, these documents specify digital signatures [FIPS-186-5], public-key encryption [SP800-56B-Rev2], pair-wise key-agreement [SP800-56A-Rev3], symmetric-key encryption [FIPS-197] message authentication [SP800-38B; SP800-185], and hash and extendable-output functions [FIPS-180-4; FIPS-202].

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 devise new standards for authenticated encryption with associated data and hashing-style primitives [SP800-232]. Further developments are expected for standards of PQC and other symmetric-cryptography primitives.

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, data privacy, and composability within broader systems. Other important techniques in scope are fully-homomorphic encryption (FHE) and zero-knowledge proofs (ZKP). The related state of the art continues to evolve based on important developments by the cryptography community, including in academia and industry.

1.2. Developing a Public Call

NIST develops standards, guidelines and recommendations about cryptographic primitives [NIST-IR7977]. Within that scope, the increasing relevance of advanced cryptography warrants a proactive exploration, including this public solicitation for structured inputs.

The NIST Threshold Call is expected to motivate broad community engagement, to result in a diverse set of high-quality submission packages that propose the consideration of pertinent crypto-systems. Packages that are accepted for posting by NIST-MPTC will form a public body of reference materials, open to subsequent public scrutiny by experts. This is intended to help identify sound approaches, best practices, and reusable building blocks, clarifying the feasibility (including threshold friendliness) and security of various techniques. The result of this process will inform the development of future NIST recommendations and processes. More concretely, the Threshold Call has the following goals:

1. **Reference materials:** 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. **New types of primitives:** Facilitate an initial assessment of the merits of promising types of cryptographic primitives that have not been previously considered by NIST.
4. **Quantum resistance and other features:** Examine the readiness of post-quantum and other advanced functional features for threshold schemes and related primitives.

Precursory NIST work. The exploration leading to the formulation of this Threshold Call included (i) holding the NTCW'19, MPTS 2020 and MPTS 2023 workshops, and (ii) publishing the NIST-IR8214 (2019) about challenges and opportunities, NIST-IR8214A (2020) with considerations toward criteria, the MPTC-Call2021a for feedback on criteria for multi-party threshold schemes (MPTS), and the NIST-IR8214B-ipd on threshold EdDSA/Schnorr signatures (2022). Related material is documented in the Multi-Party Threshold Cryptography (MPTC) and the Privacy-Enhancing Cryptography (PEC) project websites.

Public dissemination and feedback. This Threshold Call supersedes two public drafts that asked for public comments: an initial public draft (ipd, in 2023) [NIST-IR8214C-ipd; PubComsIPD] and a second public draft (2pd, in 2025) [NIST-IR8214C-2pd; PubComsIPD]. The development of the Threshold Call was also widely disseminated and socialized with the cryptography community, through talks and conversations at international conferences, including those organized by the International Association for Cryptologic Research [IACR].

1.3. Document Organization

Section 2 explains the scope of primitives organized into two classes. Section 3 describes a vision for the collaborative process, including the context of post-quantum migration, interchangeability, provable security, and a variety of options. Section 4 enumerates the phases and their expected timeline. Section 5 sets expectations that submitters need to take in consideration. Section 6 explains the structure of the Technical Specification. Section 7 discusses the required Reference Implementation. Section 8 asks for a Report on Experimental Evaluation. Section 9 presents requirements about classic and quantum security strength, and threshold security. Sections 10 and 11 specify submission requirements per category, in classes N and S. Appendices A and B give informative details about the primitives in scope, in classes N and S. Appendix C suggests a baseline system model for the threshold setting, and includes comments on threshold security, profiles, and input/output interfaces. Appendix D defines the acronyms used in the document.

2. Scope of the Call: Two Classes

To assess the viability of threshold schemes for cryptographic primitives, the scope of the Threshold Call is organized into various categories of primitives (as listed in Table 1), which are considered for thresholdization or as auxiliary. They are organized into two classes:

- **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 (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	Symmetric	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 primitives; Class S = **S**pecial others, not specified by NIST. FHE = Fully-Homomorphic Encryption. KeyGen = Key Generation. PKE = Public-Key Encryption. ZKPoK = Zero-Knowledge Proof of Knowledge.

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

2.1. Class N: NIST-Specified Primitives

Class N consists of selected NIST-specified cryptographic schemes/primitives. As a mnemonic, "N" is associated with "**N**IST". The class is organized into four categories: N1 for signing; N2 for PKE primitives; 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). Each category has more than one primitive of interest for thresholdization. For example, both threshold **decryption** (with a secret-shared private key) and threshold **encryption** (of a secret-shared message) are within the category N2 of PKE primitives.

The scope of Class N also includes primitives from the recent and emerging NIST-standards for (i) post-quantum (PQ) [Proj-PQC] signature schemes and key-encapsulation mechanisms (KEM), and (ii) "lightweight" (for constrained environments) [Proj-LWC] symmetric-key and hashing-related schemes.

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

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], Stateful HBS [SP800-208], and in upcoming NIST standards of PQ signature schemes (e.g., FN-DSA [News-FN-DSA]).
- N2 for encryption and decryption primitives used in PKE schemes based on RSA [SP800-56B-Rev2] and ML-KEM [FIPS-203], and on upcoming NIST standards of PQ-KEMs (e.g., HQC-KEM [News-HQC-KEM]).
- N3 for symmetric-key AES encipher/decipher [FIPS-197] and Ascon-AEAD encrypt/decrypt [SP800-232]; for hash functions and XOFs [FIPS-180-4; FIPS-202; SP800-232]; and for message authentication codes (MAC), including based on ciphers

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	10.1, A.1
	PQ	ML-DSA sign, HBS (SLH-DSA and stateful) sign	
N2: PKE	QV	RSA encrypt, decrypt	10.2, A.2
	PQ	ML-KEM encrypt, decrypt	
N3: Symmetric	Blockcipher	AES encipher, decipher	10.3, A.3
	AEAD	Ascon-AEAD encrypt, decrypt	
	Hash/XOF	SHA2, SHA3, [c]SHAKE, Ascon-{Hash,[C]XOF}	
	MAC	CMAC, GMAC, HMAC, KMAC	
N4: KeyGen	QV-PKC	ECC KeyGen (including for 2KA), RSA KeyGen	10.4, A.4
	PQ-PKC	ML KeyGen, HBS KeyGen	
	RBG	Random-bit generation (e.g., bitstring, integer)	

See more details in Section 10 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. [c] or [C] = Optional consideration of the Customized variant. DSA = "Digital Signature Algorithm" (the expression, not the standard deprecated in 2023). 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.

(CMAC [SP800-38B] and GMACSP:800-38D), based on hash functions (HMAC [SP800-224-ipd]), and based on the Keccak-permutation (KMAC [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].

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".

Categories. Class S has four regular categories (that match those in Class N), and three additional categories (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 the Gadgets categories do not have to consider threshold versions of their underlying primitives.

Class S is intended for primitives that, in comparison with those in Class N, have pertinent differentiating features, such as: (i) being "threshold friendlier"; (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.

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-PQ signatures	Signing	11.1
|	TF succinct & verifiably-determ. signatures	|	|
S2: PKE	TF-PQ public-key encryption (PKE)	Decrypt, Encrypt	11.2
S3: Symmetric	[Keyed] TF cipher/PRP	Encipher, Decipher	11.3
|	[Keyed] TF PRF/MAC	Tag Generate (Gen)	11.3
|	[Keyless] TF hashing and XOF'ing	Hash, XOF	|
S4: KeyGen	Any of the above	KeyGen	11.4
S5: FHE	Fully-homomorphic Encryption (FHE)	Decrypt, KeyGen	11.5, B.1
S6: ZKPoK	ZKPoK of private key, ZKPoK of signature	ZKPoK.Gen	11.6, B.2
S7: Gadgets	Garbled circuit (GC)	GC.Gen, GC.Evaluate	11.7

See more details in Section 11 and Appendix B. **Legend:** determ. = deterministic. MAC = Message Authentication Code. PRF/PRP= Pseudo**r**andom Function/Permutation [family]. PQ = Post quantum. TF = Threshold friendly. XOF= Extendable Output Function. ZKPoK = Zero-Knowledge Proof of Knowledge.

The following list enumerates the categories identified in Table 3:

- S1 for signing primitives (e.g., TF-PQ, or succinct and verifiably deterministic).
- S2 for public-key encryption primitives (e.g., TF-PQ decryption and encryption).
- S3 for "symmetric" primitives, including keyed (e.g., TF encipher/decipher and MAC) and keyless (e.g., hash and XOF).
- 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 Threshold Call.
- S7 for other auxiliary "gadgets" deemed useful to support the threshold setting, namely for the implementation of threshold schemes in scope.

3. Vision

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 scope of this Threshold Call is expected to motivate the Technical Specification, Reference Implementation, and Experimental Evaluation of a variety of crypto-systems, including threshold schemes for multiple primitives (both standardized and not standardized by NIST) and also encompassing ZKPoK and FHE techniques.

The envisioned process relies on public engagement and the collaboration of many experts (see Section 3.1). The scope-inclusion of post-quantum (PQ) and quantum-vulnerable (QV) techniques (see Section 3.2) will enable an observation of the feasibility "gap" between PQ and QV solutions, which is especially relevant in the setting of PQC-migration. While the scope of the call is wide, its exploration is meant to be judicious (see Section 3.3), guided by goals of feasibility, interoperability (namely *interchangeability*) and security.

3.1. Reliance on Contributions and Collaboration

The process envisioned in this Threshold Call depends on: **high-quality packages** submitted by teams with expertise in cryptography, including in areas of secure 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 combine various crypto-systems proposed by distinct subteams (see Section 6.3), who collaborate to enable good modularization, reduced redundancy and consistent terminology and notation in the specification, implementation and evaluation.

For collaborative purposes, it is important to set clear expectations about the reviewability and usability of submitted materials, the ability to produce derivative work and redistribute it, and the potential applicability of patent claims associated with the authors (see Section 5).

3.2. Post-Quantum and Quantum-Vulnerable Cryptography

This Threshold Call welcomes submissions of PQ solutions (i.e., plausibly secure against an adversary with a quantum computer, a.k.a. quantum resistant) and QV solutions (i.e., yet secure against 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 allowed 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 remain in use, their threshold implementation can be valuable. Furthermore, assessing the state of the art in QV threshold schemes and QV threshold-friendly primitives can be useful for setting references for future goals of 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. For example:

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. Selective exploration of a wide scope

The domain space of multi-party threshold schemes is considerably wider than that of the primitives (e.g., digital signatures) being thresholdized. In this context, the Threshold Call allows submitters to select from a variety of system models, threshold configurations, security formulations, technical approaches, and benchmarking focuses, and even to consider auxiliary crypto-systems like zero-knowledge proofs of knowledge and gadgets. Furthermore, the 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. However, the scope of the call needs to be considered with moderation.

Requirement 3.1. Pertinence of crypto-systems

The submission of a crypto-system ***shall*** be motivated by a serious intention of proposing for technical consideration a scheme that is believed to be secure and feasible, with high-potential for adoption (even if niche) in the real world.

3.3.1. Encouraged Pertinence

Requirement 3.1 is meant to discourage submissions of schemes that do not provide any meaningful advantage over (or any complement to) a well established practical scheme.

The team can (and should) make a case for the pertinence of their proposed crypto-systems, even if by explaining special use-cases. For example, the specification of a proposed threshold scheme can focus on a threshold profile that allows for making a good case of practicality. The team can argue for the pertinence of a crypto-system with regard to performance, functional properties, relied-upon assumptions (e.g., complementary in comparison with those of other competitive) schemes), and use cases (even if significantly specialized).

That said, the argued-for advantages are to be considered at the light of the state of the art, even if there is an intuition about subsequent improvements. If a scheme is impractical in the sense that it cannot even be implemented in practice for a best-effort aim of the minimum required security level, then it should not be submitted. For borderline cases of practicality, the suitability/pertinence of the submission may be assessed on a case-by-case basis.

3.3.2. 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).

3.3.3. Provable Security

The security of submitted threshold schemes (see Section 9.2 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. The submitted threshold schemes need to satisfy certain security requirements (see Section 9), including a proof of security against active corruptions in the threshold setting. 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.

4. Phases and Timeline

The specification of phases and timeline has taken into consideration the ample time of prior dissemination of the idea of this Threshold Call, which included the publication of two public drafts (in 2023 and 2025), compilations of public comments, and various presentations.

For informative purposes, Table 4 provides the intended high-level structure, listing three phases: (i) Previews (i.e., including a writeup and talk that present a plan for package submission), (ii) Packages (i.e., including Technical Specification, Reference Implementation Report on Experimental Evaluation, and Notes on Patent Claims), and (iii) Public Analysis of the submitted packages. See corresponding details in Sections 4.1, 4.2, and 4.3.

Table 4. Phases and timeline

Phase	**Subphase**	**Required?**	**Timeline**
1: Previews	1.1: Preview 1	No	Submit writeup by 2026-Jan-12
	1.2: Preview 2	No	Submit writeup by 2026-Apr-20
	1.3: Preview 3	**Yes***	Submit writeup by 2026-Jun-22
2: Packages	2.1: Preliminary package	No	Submit Jul-27–Sep-07, 2026
	2.2: Complete package	**Yes**	Submit by 2026-Oct-19
3: Analysis	3.1: Public presentations	**Yes**	2026 & 2027
	3.2: Package updates	—	—
	3.3: NIST-MPTC initial report	**Yes**	$\approx$ 2027

* The Preview 3 is only required for teams who did not participate in a previous preview.

Flexibility. The confirmation of deadlines will be communicated via the MPTC-Forum. Adjustments are possible, including adaptations to the timeline, opening a phase for updating packages, and calling for additional submissions within a restricted scope.

Requirement 4.1. LaTeX-based template

Submitted writeups ***shall*** follow the structure described in this call, and be compiled into portable document format (PDF) files, as outlined in the MPTC-provided LaTeX-based template [Template]. Teams can use editorial discretion to implement improvements.

The provided template outlines the intended structure of covers, front matter, chapters, and sections. It also includes certain required template statements, and facilitates certain accessibility features (e.g., PDF bookmarks for easy navigation, indices, hyper-references, tooltipped acronyms, and accessibility tags). The template may be improved across time, including based on suggestions, latex updates, and code adjustments.

Requirement 4.2. Documents in English

Submitted documents ***shall*** be written in English, aided where appropriate by mathematical notation, tables, figures and other possible elements.

4.1. Phase 1: Previews

The "Previews" phase provides the opportunity for each forming team to share their package-submission plan, in advance, with NIST-MPTC and the public. The phase is intended to: (i) provide an early public expectation of the coverage of crypto-systems to be submitted; (ii) enable early feedback, and possible recommendations for package adjustments; (iii) facilitate collaboration and identify opportunities for teams to strengthen their composition and/or the quality of their submission package.

Requirement 4.3. Submitting a Preview Writeup and giving a Preview Talk

Each package submission ***shall*** be preceded by the submission of a preview writeup and giving a preview talk (oral presentation).

Preview sub-phases. To promote adaptability, there are three "preview" opportunities:

- **Subphase 1.1: Preview 1.** Submitting a preview writeup (the plan) and then giving a preview talk at the MPTS 2026 workshop (January 26–29).
- **Subphase 1.2: Preview 2.** Similar to Preview 1. Can be used to present a new plan, or revise a previously presented plan. The preview talk is presented in a NIST-hosted public session sometime after the deadline for preview writeup.
- **Subphase 1.3: Preview 3.** Similar to Preview 2.

Participation in at least one preview subphase is required before a package submission.

Preview writeup. Using the available LaTeX template, each team compiles a PDF file with a genuine plan for a subsequent package submission, albeit remaining open to adjustments, e.g., regarding team composition, technical scope, used techniques, and achieved results. The resulting writeup is sent attached in an email to MPTC-submissions@

list.nist.gov. NIST-MPTC will decide on the acceptance of submitted preview writeups, possibly giving editorial suggestions, and will later post them online (publicly available).

The cover will identify the team, include a short abstract, and list the crypto-systems being proposed (and their categories). Then, a contacts page will identify the team members' affiliations and ORCID profiles, and main points of team contact. The main-matter follows, in at most six pages (letter size, 1 inch margins, 12-point font size), with a high-level description of the package planned for later submission (2). The description is encouraged to include: (i) an introduction about the scope and pertinence of the intended submission; (ii) notes about the specification (e.g., organization, system model, protocol approach, security properties); (iii) notes about the intended reference implementation (e.g., code structure, code availability, networking model, testing); (iv) notes about the experimental evaluation (e.g., used platform, performance metrics and expected results); (v) notes about the licensing, patent claims, and related funding. Finally, not counting toward the 6-page limit, the document lists the relevant bibliographic references, including links to freely accessible versions, preferably in a mainstream repository (e.g., at IACR ePrint Archive, or arXiv).

Preview talk. After each deadline for submitting preview writeups, NIST-MPTC will host a public session or workshop for the submitting teams to give a corresponding oral presentation (dubbed "preview talk"), explaining their plan for a package submission. The preview talk's slides and audiovisual recording will be posted online, for public access.

4.2. Phase 2: Packages

A submission "package" includes the following required components:

1. **Technical Specification:** A PDF document (see Section 6) with a detailed explanation of one or various crypto-systems, including a thorough security analysis.
2. **Reference Implementation: Open-source** code (see Section 7.4) that implements the crypto-systems, and includes comments, license(s), scripts and instructions. In practice, the code is distinguish between Team's Core Code, Bundled Dependencies, and External Dependencies, altogether forming the Reference Implementation.
3. **Report on Experimental Evaluation:** A PDF document (see Section 8) reporting performance measurements of an experimental execution of the crypto-systems.
4. **Notes on Patent Claims:** A PDF document (see Section 5.4) that lists the known issued or pending patents that are required to be disclosed.

Submission medium. Packages are submitted by email, as follows:

- sent to MPTC-submissions@list.nist.gov, cc'ing the team's mailing list, and obtaining subsequent email confirmation from all team members;
- attaching the required PDF documents: Technical Specification, Report on Experimental Evaluation, and Notes on Patent Claims;
- instructing how to obtain the Packaged Codebase from a publicly-accessible repo.

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

- **Subphase 2.1: (Optional) Preliminary package.** Preliminary packages received by NIST-MPTC in this subphase will be subject to a superficial review for completeness. Within approximately 30 days, the teams will be notified of identified deficiencies or other suggestions, to allow for amendments.
- **Subphase 2.2: Complete package.** NIST-MPTC will consider packages received by the submission deadline. Some editorial review might still be requested. After around 30 days, the accepted packages will be hosted on a NIST public-facing repository, accessible via the NIST-MPTC project website.

An optional complement: The team can adapt their previous **Preview Writeup** into an **Executive Summary** (aimed for non-experts), and bundle it in the package submission.

4.3. Phase 3: Public Analysis of Crypto-Systems

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

- **Subphase 3.1: Public presentations.** NIST-MPTC intends to host a seminar series for thorough presentations of the proposed crypto-systems, and to consider comments from the public. This may be complemented with occasional larger events, such as NIST Workshops on Multi-Party Threshold Schemes (MPTS).
- **Subphase 3.2: Future updates.** The NIST repository will enable public download of the accepted packages, and may also list links to the teams' repositories (external to NIST). Each team can use the MPTC-Forum to announce issues discovered after the submission, and possible updates incorporated into the team's repositories. As the process unfolds, NIST-MPTC may specify periods for optional submission of amendments or improvements, to update the body of NIST-hosted reference materials.
- **Subphase 3.3: 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.

5. Expectations About Submitted Material

The Threshold Call is devised to gather a body of reference materials, open for public review and evaluation. This is intended to serve as a basis for potential developments and improvements, to foster a widespread, secure use of cryptographic techniques. Accordingly, the submitting teams need to abide by baseline expectations about the submitted materials.

By submitting a package in reply to this Threshold Call, the submitting team (i.e., all of its members) **acknowledges and agrees** with the expectations expressed in this section.

5.1. Original Work

Attribution to prior work. The package content is produced by the identified team, except (where applicable) for properly **credited** copies or adaptations of:

(i) open-source code (e.g., compilers, libraries);

(ii) writeup material (e.g. text, tables, images), under fair-use or when known to not infringe on copyright (e.g., if permitted by the copyright holders, or by an applicable license, or if in the public domain).

The mentioned expectation does not restrict the submission of crypto-schemes that rely, build upon, or adapt mathematical techniques developed by other authors. In particular, a submitted package can cover long-standing techniques developed by other authors.

Potential use of Generative Artificial Intelligence (GenAI). The submitting team (composed only by humans) can use GenAI as a supporting tool for developing the Team's Core Code (in the Reference Implementation), and to obtain suggestions to improve the quality (e.g., grammar) of submitted writeups, provided that the team judiciously reviews all used GenAI outputs. In particular, the team must have an expert understanding of the technical choices made regarding the parameters and techniques used in the submitted crypto-systems (including in their specification, implementation, and evaluation).

The LaTeX template provided for the package writeups includes a template statement for the team to express that it has fully reviewed the submitted content, including possible outputs obtained with the assistance of GenAI. When applicable, teams should make clear which substantial portions of content may have been produced primarily by GenAI, with minimal intervention by team members (e.g., pictures produced by GenAI based on a text prompt), and provide credit to the used AI tool.

5.2. Security Guarantees

Technical effort is needed not only to ensure the security of crypto-systems in preparation for submission, but also afterward. Teams are expected to disclose undocumented vulnerabilities (in the Reference Implementation or Technical Specification) or errors (e.g., in the Report on Experimental Evaluation) that it becomes aware of, during at least three years after their submission. Reasonable delays can be justified for responsible disclosure.

5.3. Availability of the Submitted Materials

By submitting to NIST the preview writeup, the package documents — Technical Specification, Report on Experimental Evaluation, and Notes on Patent Claims — and any subsequent update within the unfolding process, the submitting team: (i) grants NIST the right to post online the submitted PDF files, for public access, becoming freely available worldwide for public review and evaluation purposes; and (ii) grants the public the right to use those PDF files posted by NIST, including for commercial purposes.

Teams are **encouraged** (but not required) to include in the cover of their documents an explicit license, such as the Creative Commons "CC BY 4.0 International" (Attribution 4.0 International) license [CC-BY-4.0] that allows further redistribution by other parties.

If a license different from CC BY is used in the submitted PDF files, then it must not attempt to restrict use for commercial purposes. For example, "share-alike" terms are possible (e.g., as in CC BY-SA 4.0 International).

Note: Regarding the Reference Implementation, the permission for NIST-MPTC to post the code follows from the open-source licensing requirement (Imp4) in Section 7.4.

NIST-MPTC reserves the right, in any phase, to approve or reject any submission, and decide which documentation to post or remove from the project website. For example, it may reject any submission deemed to be outside of scope, lack technical credibility, or be unreasonably non-conformant with main requirements. Conversely, given the collaborative and exploratory nature of the process, and the complex combination of requirements expressed in this Threshold Call, NIST-MPTC also reserves the right to accept, on a case-by-case basis, submissions that may deviate from some requirements in a justifiable way.

5.4. Notes on Patent Claims

Scope of required disclosure. Per Requirement 1.1, the Notes on Patent Claims (writeup) is one of the required components of each package submission. Its purpose is to disclose *any known issued or pending patent (foreign or domestic) that does or could have claims that may cover the contents of the Technical Specification, Reference Implementation, or Report on Experimental Evaluation, 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*. If no such patents are known by the team, then the document will include a template statement claiming such absent knowledge.

During a period of at least three years after the initial submission, it is expected that teams will voluntarily update (i.e., and resubmit) this document in case any team member becomes aware of a patent whose disclosure is required.

Additional notes. A LaTeX-based template [Template] is provided for listing the known patents whose disclosure is required. For each such patent, a note ***should*** indicate which submitted crypto-system, or underlying or complementary techniques, being specified, implemented or experimentally evaluated, may be covered by a patent claim. Teams are also welcome to comment on the known existence of (or known plans to develop) applicable licenses, and their Reasonable and Non-Discriminatory (RAND) nature [ITL-Patent-Policy].

Out of scope (third party claims). The scope of required patents disclosure does not include patent claims that are neither associated to any team member nor to their sponsors or direct affiliations (e.g., university, company, institute, or agency). Patent claims by third parties may be commented in public discussion along the process, in presentations and/or via the the MPTC-Forum.

Techniques in scope for disclosure of patents. The techniques within the scope of the requirement of disclosure of patent claims (i.e., when associated with a team member or their sponsor or affiliation) include not only the higher-level algorithms or protocols of the proposed crypto-systems (specified, implemented, or evaluated) but also their underlying techniques (e.g., including building blocks, and underlying conventional primitives being thresholdized), and the complementary primitives that are naturally used in conjunction with them. For example, if a threshold decryption protocol is within a crypto-system proposed for analysis, motivated as being useful for decentralization of trust in the use of a PKE scheme, then the encryption and KeyGen primitives are also within the scope of interest with regard to disclosure of patents associated with team members.

6. Technical Specification

The first main component of a submission package is the Technical Specification.

Requirement 6.1. Structure of the Technical Specification

The Technical Specification ***shall*** be submitted as a PDF file, following a chapter/section structure as outlined in this Threshold Call, including a front matter, a preliminaries chapter, one chapter for each main crypto-system (of family thereof), and a thorough list of accessible references.

In more detail, the content will be organized as follows:

- A ***front matter*** (see Fm1–Fm6 in Section 6.1), including cover with title, verso with contacts, abstract, preface, development context, and contents (including a table of contents up to subsection depth, and the lists of figures, and tables).
- One *unnumbered* chapter "**Preliminaries**" (see Pre1–Pre5 in Section 6.2), starting with a listing of local contents (table of contents, list of design decisions, list of examples, etc.), followed by numbered sections: introduction, notation, related work and design decisions, and conventional primitives and building blocks.
- One *main matter* chapter for each **crypto-system** (see CSX.1–CSX.7 in Section 6.3), each including a cover with abstract, followed by various sections: contents of this chapter, system model, algorithms and protocols, security analysis, complexity analysis, and deployment considerations. The chapter can include local appendices.
- A **References** section (chapter), listing all references corresponding to citation tags.
- An optional chapter of **Appendices**, with auxiliary material.

Nuances in the list of sections are possible, when allowing a better organization. Recommendations of possible adjustments to the mentioned structure can be communicated by NIST-MPTC via the MPTC-Forum, and/or by adjustment of the online template.

This baseline structure is intended to facilitate parsing across specifications. To aid navigation, authors are encouraged to organize content into deeper levels (e.g., subsections, subsubsections, and so on) and possibly index additional environments (e.g., theorems+lemmas+corollaries, design decisions, recommendations+requirements, definitions, examples).

The indices FmY (section within the front matter), PreY (sections within the Preliminaries chapter), and CSX.Y (sections within each crypto-system chapter) used in this organization description are not meant to appear in the actual specification.

6.1. Front Matter

The front matter starts with a cover and a verso, both in unnumbered pages. Then follow various unnumbered sections, in pages with roman numbering (i, ii, ...).

Fm1: Cover. A cover page with: (i) title (and optional subtitle) of the submission package; (ii) document type (technical specification), context (Threshold Call), version ("1.0" by default) and date (with possible footnote); (iii) short team name (and optional extension); (iv) list of team members (names only); (v) list of names of the proposed crypto-system, the applicable categories indices, and a corresponding legend; (vi) optional team logo; and (vii) optional document licensing (see Section 5.3).

Fm2: Verso with Contacts. A verso page with: (i) a template statement about the document purpose and its fit within a submitted package with other documents; (ii) URL of the team repository; (ii) lists of team members (names) cross-referenced to their ORCIDs, affiliations, and (optionally) associateship clarifications; (iii) a short list of main contacts, including the team's mailing list (i.e., a single email address that relays emails to all team members), a primary technical contact person (name and email address) and between one and two secondary contact persons (names and email addresses); (iv) a template statement about inexistent or limited use of GenAI.

Fm3: Abstract. A paragraph with 150 to 350 words, describing the technical scope of the submission, and hinting at the main features, cryptographic assumptions and performance highlights of the specified crypto-systems. The abstract is followed by a list of keywords, intended useful for indexation purposes.

Fm4: Preface. A short unnumbered section (up to two pages), preferably without mathematical notation, briefly describing the motivation for the submission, including: the envisioned relevance (e.g., utility, applicability, deployability) of the proposed crypto-systems (e.g., in industry, and for societal applications), considering the state of the art. The section can comment on the process of package development, and the connection between specification, implementation and experimentation.

Fm5: Development Context. A section with: (i) statement about author contributions and (if applicable) differentiating subteams; (ii) if applicable, acknowledgments to external contributors (i.e., non-authors); (iii) if applicable, further details about the use of GenAI; (iv) optional acknowledgments of funding (e.g., public research grants); and (v) template statement about prior work.

Fm6: Contents. A table of contents (TOC) with depth up to subsections, followed by lists of tables (LOT) and figures (LOF), with hyperlinks and page numbers. Note that each subsequent chapter (Preliminaries, Crypto-System, Appendices) will have its own local TOC (with higher depth) and other applicable lists.

6.2. Chapter Preliminaries

The main matter starts with a "Preliminaries" unnumbered *chapter*, initiating the arabic page-numbering style (1, 2, ...) in the document. After the chapter title comes a table of contents (and various lists), and then follows a a sequence of decimal-numbered sections.

Pre1: Contents. A table of local contents (up to subparagraph depth), and applicable lists of indexed contents (e.g., of design decisions, of examples).

Pre2: Introduction. A section that introduces the technical area of interest, identifies and motivates the proposed crypto-systems, and hints at their properties and the technical approaches. The section should also explain the pertinence of crypto-systems proposed within the scope of Class S. One subsection should describe the high-level structure of the document, to help the readers navigate across the content.

Pre3: Notation. A section that lists and explains the acronyms, math symbols, and terms used in the specification. It ***may*** contain subsections to explain relationships between symbols (e.g., sets, elements, operations).

Pre4: Related Work and Design Decisions. A section with rationale for the system model(s) and proposed crypto-system(s). The section will **identify** the techniques, building blocks, and other ideas that are known to have been developed or authored in prior or related work, and that have directly influenced this specification. This is an opportunity to gather numerous attributions/references (including citations; see 1) to prior works/authors (whether or not part of the submitting team), while informing related design decisions made in this specification. However, actual detailed **technical specification** of algorithms and protocols (e.g., using a latex algorithm environment) should be deferred to later sections (e.g., Pre5, CSX.3 or CSX.4).

Pre5: 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 the Crypto-System chapters (namely away from CSX.4), but that are nonetheless used in at least one proposed crypto-system. For example, this can recall the conventional (non-threshold) Class N primitive that will be thresholdized. The *explanation* of some building blocks here is meant to be more direct (and possibly thorougher) than a possible *identification* in Pre4, which is more concerned with providing proper attribution and rationale.

6.3. Chapters "Crypto-System X"

After the "Preliminaries" unnumbered chapter, the main matter continues with one or more "Crypto-System" chapters (I, II, ...). Each chapter will include one or a combination and/or family of threshold scheme, ZKPoK, gadget, or Class S conventional scheme proposed for standalone analysis. For example, a family ***may*** include 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).

Requirement 6.2. Requirements for crypto-systems

Each proposed crypto-system ***shall*** satisfy the applicable requirements indicated in Sections 9, 10, and 11.

Packages with multiple crypto-systems and multiple categories. A Technical Specification document ***may*** propose multiple crypto-systems, from multiple categories across the two classes. For example, a single submission package can propose:

- A threshold scheme for a non-KeyGen primitive in category N1, N2, or N3.
- A threshold scheme for another non-KeyGen primitive in category S1, S2, S3, S5.
- A DKG protocol (in N4 or S4) for distributed generation of a secret-shared key.
- A related ZKPoK (S6), such as to prove knowledge of an output (e.g., signature of ciphertext) correctly generated by one of the proposed primitives.
- Auxiliary gadgets (in S7).

Teams are encouraged to favor modularity and team collaboration. It is up to each team's editorial discretion to decide whether and how to organize the specification of the multiple crypto-systems across more than one crypto-system chapter. Also, a team can choose to associate differentiated subteams across the crypto-systems.

Chapter structure. Each crypto-system chapter includes the following sections:

CSX.1. Cover page: A cover page that indicates: the crypto-system name, the version and date of its last revision, a reference to the subteam (if different from the entire team identified in the main cover); 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. Contents: A table of local contents (up to subparagraph depth), and applicable lists of indexed contents. A complete list of the algorithms/protocols specified in this crypto-system part is required in this section. Authors choose which other lists to add (e.g., of theorems, of requirements). It can even include a list of needed algorithms / building blocks that have been specified/explained in other chapters.

CSX.3. System Model: A description of the system model (see Appendix C.1), including participants and their activation, communication network, and adversary. This section is intended to be straightforward about the chosen system model (in contrast with the previous explanation, in Pre4, of 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 (e.g., functionalities and security games) should be deferred to CSX.5.

CSX.4. Algorithms and Protocols: 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 (whichever applies): (i) interface and properties described in Pre5 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.4). The protocol can also be described with various phases (e.g., offline, online, secret resharing), which may have differentiated requirements.

CSX.5. Security: A security analysis of the proposed crypto-systems, addressing the requirements in Section 9. The section includes: a security formulation (e.g., ideal functionalities or games); an identification of assumed ideal components and other assumptions (cryptographic, and of trusted setup); a security proof sketch (small subsection) and a thorough security proof (which can go into appendix); a discussion of security consequences in case of (i) replacing ideal components by real ones, and/or (ii) deploying the system in environments that are (e.g., without synchrony) different from those assumed in the system models. Some of these components can be included in an appendix (within the chapter, or within the chapter of global appendices).

CSX.6. Complexity: An analytical estimation (as a function of well identified protocol parameters) of the (i) memory complexity (size), (ii) computational complexity (e.g., as a number of well identified high-level operations, such as encryptions, signatures, or group multiplications), (iii) communication complexity (unit similar to memory complexity), and (iv) round complexity (number of transmission rounds) of each proposed crypto-system. 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. See Section EvX.4 about the Report on Experimental Evaluation, meant to measure complexity of actual executions.

CSX.7. Deployment: A section that includes a list of known and proposed applications of the submitted crypto-systems, and a corresponding set of deployment requirements and/or recommendations. The section can discuss aspects of interoperability, deployment security, and other deployment-related insights. As deemed useful, the authors decide how to reduce redundancy with possible deployment information related to security (see CSX.5) and performance (see CSX.6).

CSX.8. Local Appendices: Each Crypto-system chapter can have local appendices, each being considered a section (e.g., I.A, I.B, ...). Each team decides what to include here vs. in the global appendices (see 2).

6.4. Back Matter

1. **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.
2. **Appendices.** A sequence of sections (numbered with upper-case english letters: A, B, ...) with auxiliary elements that the authors deem as too detailed or cumbersome for the main matter. For example, one appendix can be for cumbersome details of mathematical proofs (e.g., of certain lemmas or theorems) needed by the proof(s) of security in CSX.5). The last appendix should be (when applicable) a Change Log, with a descriptive listing of the main changes since previous versions.

7. Reference Implementation

The second main component of a submission package is the open-source Reference Implementation, which comprises the Packaged Codebase (composed of the Team's Core Code and the Bundled Dependencies), and the External Dependencies. At the time of submission, both are publicly available for copying: the Packaged Codebase from a repository controlled by the team; the External Dependencies via download (automated by installation scripts) from other repositories. The Reference Implementation needs to be compilable within a Baseline Platform (operating system and hardware), into a testable realization of the crypto-systems proposed in the Technical Specification.

Terminology. The following implementation-related components are distinguished:

- **Team's Core Code:** The code developed by the team to implement the specified crypto-systems, considering a software environment with access to "dependencies". For convenience, this core code also includes scripts and instructions to compile and test the Reference Implementation.
- **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 other repositories into the team's repository).
- **Packaged Codebase (or "submitted code"):** The combination of the Team's Core Code and the Bundled Dependencies, publicly available in a repository controlled by the team.
- **External Dependencies:** Externally developed **open-source** libraries that are neither part of the Bundled Dependencies nor the operating system, but are needed to produce, execute or test the Reference Implementation.
- **Reference Implementation (or Deployment Package):** The combination of the Packaged Codebase and the External Dependencies, sufficient (when combined with the baseline operating system) to compile a testable realization of the crypto-systems proposed in the Technical Specification.
- **Baseline Platform:** The computer (hardware, satisfying minimum requirements) together with the operating system, capable of sustaining the Reference Implementation's compilation into a testable realization of the proposed set of crypto-systems.

Summary of requirements. The Reference Implementation needs to fulfill the following:

- Imp1: Is compatible with a Baseline Platform (see Section 7.1)
- Imp2: Implements the proposed crypto-systems (see Section 7.2)
- Imp3: Is publicly available (see Section 7.3)
- Imp4: Is licensed as open-source (see Section 7.4)
- Imp5: Is clear, including inline comments (see Section 7.5)
- Imp6: Includes useful scripts and instructions (see Section 7.6)

7.1. Compatibility With a Baseline Platform

Requirement 7.1. [Imp1] Compatibility with the Baseline Platform

The Reference Implementation ***shall*** be compilable, executable and testable in the Baseline Platform described in this Threshold Call.

The Baseline Platform consists of a personal computer (possibly virtualized) equipped with:

1. **Central processing unit (CPU):** Sixteen x86-64 (64-bit) processing cores
2. **Fast primary memory:** 64 gigabytes (e.g., of random-access memory (RAM))
3. **Secondary memory (storage):** Two terabytes (e.g., in a solid state drive (SSD))
4. **Operating system:** Ubuntu Desktop [Ubuntu] from within one of the 24.04 (“Noble Numbat”) or 26.04 (“Resolute Raccoon”) Long-Term Support (LTS) series.

These features are meant to suffice for testing all submitted implementations, which will be automated based on scripts and instructions (see Requirement 7.6, Imp6, in Section 7.6). The identification of this baseline platform does not constitute an endorsement of the mentioned technologies, nor does it indicate a preference over other suitable platforms.

Teams are welcome, but not required, to also prepare complementary scripts and instructions (see Section 7.6) for compilation and testing in alternative platforms (e.g., lighter open-source operating systems, and different hardware ensembles). The Report on Experimental Evaluation can also refer to those additional platforms (see Section EvX.3).

7.2. Implementation of Crypto-Systems

Requirement 7.2. [Imp2] Implementation of Crypto-Systems

The Reference Implementation, when combined with the Baseline Platform, ***shall*** constitute a testable implementation of the crypto-systems (see Section 6.3) proposed in the Technical Specification, including the applicable building blocks.

Isolation across parties in a threshold scheme. The execution of a threshold-scheme implementation ***should*** run each “party” as a process (or set of processes) separate from other parties. Application containerization can be used to further isolate each party.

Networking versus cryptography. There can be significant challenges between (i) implementing networking between parties (see Appendix C.1.2) and (ii) implementing mathematical operations and cryptographic building blocks per party. Neglecting any of these implementation components can lead to serious vulnerabilities. Therefore, a **strong alignment** between the proposed system model (see CSX.3 and Appendix C.1) and the provided implementation is **strongly encouraged**, notwithstanding possible containerization and/or virtualization to enable testing on 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 trusted components (e.g., a hardware router), beyond the n main parties (from which the participation/corruption thresholds are measured), then the submission ***may*** include code for simulating those special components. Also, various software tools can be used to implement traffic control, in order to simulate diverse networking conditions.

7.3. Code Availability

Requirement 7.3. [Imp3] Code availability

The Packaged Codebase (with the Team's Core Code and Bundled Dependencies) ***shall*** be available via a publicly-available repository controlled by the team (preferably compatible with the "Git" distributed version-control system). All External Dependencies ***shall*** be publicly available at the time of submission.

Packaged Codebase. The email with the package submission will not include the code, but rather the instructions to obtain it from a repository (repo), as follows:

- **A complete Git-repo.** If the Packaged Codebase matches a complete Git repo, then the email will identify: (i) the Git repository URL, (ii) the **Git-commit hash** (with 40 hexadecimal characters) of the given Git-commit stage; (ii) the suggested command for cloning the repo at the given commit-stage; (iii) (if available) the URL to download an automatically generated archive file (e.g., .zip) containing the Git-repo content. For example, the URL might have a syntax similar to `https://<repo-host-server>/<team-name>/<repo-name>/archive/<Git-commit-hash>.zip`
- **Archive file within a repository.** If the team is unable (of finds inconvenient) to provide the Packaged Codebase as a full-fledged Git-repo version, then it can combine the Packaged Codebase into a single archive file (e.g., .zip, .tar.gz) and include it in

a publicly available repo (preferably with a version control system). Then, the email will include (i) a direct **URL** to the archive file, (ii) a SHA-based cryptographic hash (e.g., using SHA256 or SHA3-256) of the archive file, and (iii) its exact **byte size**.

External Dependencies. The reliance on External Dependencies, separate from the Packaged Codebase, may be motivated by differences in licenses (e.g., copyleft versus permissive), practical modularity (e.g., selectable depending on installation options), and packaging ease (e.g., avoiding an inconveniently large Packaged Codebase). In practice, the gathering (download) and integration of External Dependencies will be automated by installation scripts, to form the Reference Implementation (see Scripts #1).

Prepared virtual machine. Teams are welcome (but not required) to make available (or provide scripts to prepare) a virtual machine that includes a compiled Reference Implementation on top of the baseline operating system (Ubuntu 24.04 or 26.04), ready for experimental testing of the crypto-systems. In this case, the Ubuntu installation inside the virtual machine can be stripped away from unnecessary applications and tools, as a way of minimizing the virtual machine byte-size. Conversely, the virtual machine would already contain all needed external dependencies, and would not require Internet access to be tested.

7.4. Code Licensing and Posting

Allowed Licenses. The software licenses allowed for the submitted code are those from within the Open Source Initiative (OSI) list of "OSI Approved Licenses" [OSI-lic] (hereafter denoted **OSI-approved**), or within the Free Software Foundation (FSF) list of "GPL-Compatible Free Software Licenses" [FSF-lic] (hereafter denoted **FSF-libre**).

Requirement 7.4. [Imp4] Open-source code licensing

All code components of the Team's Core Code, Bundled Dependencies, and External Dependencies ***shall*** be licensed using OSI-approved or FSF-libre licenses.

Teams are encouraged to consider the OSI [OSI-def] and FSF [FSF-def] principles of open and free/libre software. When intending to use short identifiers to identify the applicable licenses, the "System-Package Data-Exchange" (SPDX) nomenclature is recommended [SPDX-lic].

The Packaged Codebase should include a top-level file (e.g., "LICENSE") that clarifies under which Allowed License(s) the Packaged Codebase can be redistributed. Also, if different components of the Packaged Codebase are subject to different Allowed Licenses, then the licenses-to-components relation ***should*** be made clear.

NIST posting of submitted code. NIST-MPTC intends to publicly post on a NIST-controlled repo the accepted Packaged Codebase, and (as applicable) a reference to the team's repo. If the licensing 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. After the submission, the team is welcome to continue updating/improving the code on their repo, while at the same time maintaining clear which version was submitted to NIST.

Redistributability of a compiled Reference Implementation. The combination of licenses in the Packaged Codebase and the External Dependencies ***should*** allow for redistribution of the Reference Implementation software, upon compilation, under at least one of the Allowed Licenses, which may be different from the license(s) of the Team's Core Code. The submitted code ***may*** include a method for generating a Software Bill of Materials (SBOM) applicable to the Reference Implementation software resulting from that compilation.

Patents related to used dependencies. The scope of implemented techniques (in the Reference Implementation) that are subject to disclosure of patents associated with team members covers not only the Team's Core Code but also the used techniques implemented in the Bundled Dependencies and External Dependencies.

7.5. Clear Code

Requirement 7.5. [Imp5] Clarity of the Team's Core Code

The Team's Core Code ***shall*** favor clarity with regard to how the proposed crypto-systems are implemented, even if at detriment of some performance. Additionally, one or more files ***shall*** explain which components of the proposed crypto-systems are implemented by which libraries/modules of the Bundled Dependencies and the External Dependencies, and how they are meant to interact with the Team's Core Code.

Optionally, additional code that is **optimized for performance** but less clear can be included to showcase better experimental performance. In any case, most functions/modules of the Team's Core Code ***should*** be accompanied by auxiliary explanatory comments.

Language, compiler, API. This Threshold Call intentionally does not specify (see Section 3.3) a concrete programming language, compiler, or application-programming interface (API). However, the Packaged Codebase ***should*** include rationale for the choices made.

Validation and verification. This Threshold Call does not require formal verification or validation of implementations, although it suggests some operational testing (see Section 7.6.3) and asks for test vectors that can support some input/output testing (see Section 7.6.4). Additionally, the public scrutiny of submitted implementations is expected to contribute to clarifying suitable testing mechanisms, which can promote the production of high-assurance cryptographic software. The webpage of the NIST Cryptographic Algorithm Validation Program (CAVP) [CAVP] includes information about validation testing for various NIST-approved cryptographic algorithms.

7.6. Useful Scripts and Instructions

Requirement 7.6. [Imp6] Scripts and instructions (build, test, benchmark)

The Team's Core Code ***shall*** incorporate a set of useful scripts, and corresponding English-written instructions, for (i) building the reference implementation, (ii) configuring parameters, (ii) obtaining performance measurements useful for benchmarking and performing operational testing, and (iii) performing input/output testing.

A README file ***should*** provide basic orientation about the repository. Each team decides how to organize the scripts and instructions mentioned ahead.

7.6.1. Software Build

Scripts #1 (Build): A script, executable in the Baseline Platform, with a command to automatically download the needed External Dependencies (if applicable), and to compile the software that will later be used to execute/test the proposed cryptosystems. Teams are encouraged to strive for a script that can obtain the External Dependencies with a specific version, in order to favor a reproducible compilation.

Instructions #1 (Build): An explanation on:

(a) How to download (possibly via Git-clone) the Packaged Codebase from the team's repository, as available at the time of package submission.

(b) How to execute the build script (Scripts #1) to download the External Dependencies (specific or latest versions) and compile the Reference Implementation.

Precise version identification. In the interest of future reproducibility, the External Dependencies ***should*** be precisely identified (e.g., their version) and publicly available (preferably in a repo with a version control system). 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) then it can be relaxed in favor of the team's focus on achieving a workable Reference Implementation, assuming reliance on the most updated version of the External Dependencies. However, the inadvertent combination of future External Dependencies (i.e., with versions that are subsequent to those tested by the team) may result in (i) failure to compile the reference implementation, or (ii) to a compiled implementation that does not work correctly. Further recommendations about reproducibility may emerge from public feedback and the experience analyzing the submitted implementations.

Testing with secure isolation. With regard to testing, the reliance on external code poses a security risk (e.g., in case a source repository becomes compromised after the submission date). As a mitigation, it is recommended that any testing be performed within a virtualized platform that is properly isolated. The baseline testing should also be possible without any access to the Internet, other than for downloading the external dependencies.

Automatic update. The team ***may*** include an additional script designed to check for and download existing updates to the Packaged Codebase (from the teams' repository) a

7.6.2. Configuration

Instructions #2 (Parametrization): An explanation on how to configure the parameters for execution and testing. Preferably, the parameters should be easily configurable via a human-editable text file, and/or command line arguments. Example parameters: security strength, number of parties, corruption threshold, type of communication channels, adversarial choices, client choices (e.g., input to the cryptographic primitive, such as message to be signed), parallelization level (e.g., max number of processors).

Instructions #3 (Execution): An explanation on how to run the benchmark and test script(s) (see Scripts #2), to test various phases/modules/primitives of the crypto-systems, and how to run the test-vector and related scripts (see Scripts #3 and Scripts #4) to perform basic input/output testing.

Instructions #4 (API): A description (e.g., in `API.md`) of the application-programming interface (API), to facilitate (i) interoperability, and use in higher-level applications, (ii) performance comparison with other implementations with similar API.

7.6.3. Performance Measurements and Operational Testing

Scripts #2 (Benchmark and test): A script (or set thereof) to automatically benchmark and test the stability of the crypto-systems operations in the Baseline Platform, automatically producing performance measurements (similar to those in the Report on Experimental Evaluation) for various selected configurations. In the case of threshold schemes, at least one testing case ***should*** consider at least one corrupted party (see EvX.3 in Section 8.4). The script can also stress-test (even if without benchmarking the performance) the execution of the crypto-systems in adverse conditions, such as with many corrupted parties, or with low computational resources.

If, besides the reference code (favoring clarity), the Packaged Codebase includes additional code optimized for performance and whose performance results are also reported in the Report on Experimental Evaluation, then the corresponding scripts ***should*** also be provided.

7.6.4. Input/Output Testing

With regard to input/output testing, the depth and thoroughness of the provided scripts and instructions is at the discretion of each team. If a testing obstacle is insurmountable, it would be useful to describe the challenges found, and what is left as open engineering problems.

The input/output testing can be considered globally, per party, and/or per algorithm/phase. Some challenges of non-determinism in multi-party systems might be overlooked, by focusing on single-party testing. For example, even if message ordering is unpredictable, one may focus on the behavior of each party, at each protocol phase, if it is known how a party in a given state is supposed to react to a received message [PubComs2PD, Comment Subset F2d].

Scripts #3 (Test-vector): A script (or set thereof) to automatically execute the crypto-systems, possibly in various configurations, in a way that produces a set of test vectors that can be used for known-answer tests (KAT).

Instructions #5 (Baseline test-vectors): A list of test vectors, produced with the corresponding script (Scripts #3), to facilitate some baseline testing and correctness verification.

Scripts #4 (KAT): A script (or set thereof) to perform known-answer tests (KATs), based on a provided set of test vectors (and/or additional test parameters). When this script is parametrized to run with the provided baseline test vectors (Instructions #5), then it will use their inputs to run the implementation and then compare the obtained outputs against the outputs in the test vectors. Teams are expected to confirm that their submitted code passes their proposed baseline tests.

On the input/output of a KAT. A KAT is meant to test an operation with static inputs and outputs provided by a test vector. The inputs to the operation are obtained from the test vector. In particular, any randomness that would typically emerge from the process is, instead, also provided to the test as input (directly or derived from an input seed). Then, the KAT checks equality between (i) the outputs obtained by the cryptographic operation being tested, and (ii) the outputs present in the test vector. For example, (msg, seckey, seed-nonce, sig) is a test vector that allows testing (i.e., allows the KAT) that TestSign(seckey, msg, seed-nonce) outputs sig, where TestSign is a version of the signature algorithm that uses seed-nonce as a proxy for randomness. Conversely, the test that a static tuple (msg, seckey) will result in a signature that is accepted as correct by the verification algorithm of the signature scheme is not on its own a KAT, since it abstracts from the signature value.

On testing combinations of parameter sets. The use of a single parameter-set per desired security-level may help reduce the complexity of testing combinations. For example, even if the specification of an algorithm allows a building block to be any cryptographic hash function with a given security strength, the choice of a specific hash function will alleviate the testing complexity, e.g., in comparison with the testing with all NIST-approved hash functions. This testing-related argument is not in detriment of the utility that the specification of a proposed crypto-system explains how/that certain building blocks can indeed be replaced by a variety of instantiations with certain properties and interfaces.

8. Report on Experimental Evaluation

The third main component of a submission package is the Report on Experimental Evaluation (a PDF file), which describes results from executing the Reference Implementation in various experimental configurations. This is meant to provide a realistic assessment of the feasibility of the proposed crypto-systems, instantiated with appropriate security strength.

8.1. Parametrizations and Computational Resources

The following requirements establish the dual need to (i) allow executing the crypto-systems with realistic parameters, and (i) allow testing in a platform similar to the Baseline Platform.

Requirement 8.1. Experimental evaluation with required security strength

Each crypto-system ***shall*** be experimentally evaluated with at least one parametrization fulfilling the applicable security-strength requirements {9.2, 9.3, 9.4} (see Section 9.1).

Requirement 8.2. Experimental evaluation with toy parameters

If the resources indicated for the Baseline Platform are insufficient to fulfill the Requirement 8.1, then at least one "toy" parametrization (not achieving the required security strength) ***shall*** be experimentally evaluated (possibly in a virtual platform) with resources not exceeding those of the Baseline Platform.

Toy parametrization. If the conditions of Requirement 8.2 are in place, then a toy parametrization can, for example, aim for $\kappa \approx 16$ bits of "security strength" and then extrapolate what are the expected performance results for a **secure** parametrization with $\kappa \approx 128$. For example, suppose that the memory complexity is quadratic ($\mathcal{O}(\kappa^2)$), in the security parameter κ. If, simultaneously, the needed memory for evaluating a threshold scheme is linear in the number of parties, then the toy parametrization allows (for this metric) evaluating 64 times more parties than the configuration with realistic parameters ($\kappa \approx 128$).

Example 1 (Insufficient Memory). If each party in a threshold scheme requires $2/3$ of the memory available in the Baseline Platform, then even a 2-party instantiation would exhaust the available memory (assuming symmetric parties, with required simultaneous use of memory). In such case, Requirement 8.1 can be met using a platform with much more memory (possibly distributed, e.g., with one computer per party). Correspondingly, Requirement 8.2 can be met by using a toy parametrization with weaker memory requirements.

Example 2 (Insufficient Parallelization). Consider a threshold scheme where each party requires the full parallelization allowed by the multi-core processor of the Baseline Platform in order to perform in a practical amount of time. Then a multi-party implementation may require a prohibitive amount of time in the Baseline Platform. Then the two requirements of experimental evaluation (8.1, 8.2) can be met with two executions in two different platforms. The toy case can be used to test a threshold profile with many more parties.

Toy Configuration for Didactic Purposes. Even when the Baseline Platform can execute and test with realistic parameters, the specification and evaluation of a toy parametrization is also welcome for didactic or verification purposes. For example, such toy configuration could be akin to testing RSA encryption and decryption when the modulus is $N = 43 \times 67$, instead of having at $\geq$ 3,072 bits. This may ease the understanding, or even enable a more extensive coverage of KAT vector testing (e.g., if otherwise the vectors would be too large).

8.2. Structure of the Report on Experimental Evaluation

Requirement 8.3. Structure of the Report on Experimental Evaluation

The Report on Experimental Evaluation ***shall*** follow a chapter/section structure as outlined in this Threshold Call, with a front matter, an Overview chapter, one chapter for each main crypto-system (of family thereof), and a list of accessible references.

In more detail, the report will be organized as follows:

- A **frontmatter**, like the one of the Technical Specification, including cover with title, verso with contacts, abstract, preface, development context, and contents listing.
- An unnumbered chapter called "**Overview**" (see Ov1–Ov3 in Section 8.3), with various numbered sections: (i) overview of the experimental evaluation; (ii) notes on the Reference Implementation; (iii) experimental setting.
- A chapter per crypto-system (see EvX.1–EvX.5 in Section 8.4), with: (i) a list of experimental configurations, (ii) measurements, and (iii) interpretative analysis.
- A list of applicable **references**, including the companion Technical Specification.
- The authors decide whether to include **appendices** (per chapter or in the backmatter).

The occasional repetition of topics between the "Overview" chapter (e.g., in the "Experimental Setting" section) and the "Crypto-System" chapters (e.g., in the "Experimental Configurations" section) are not meant to imply redundancy. Instead, the content in the Overview chapter doesn not have to be repeated across the sections in the "Crypto-System" chapters. The authors decide how to best organize their content across this structure.

8.3. Chapter "Overview"

An "Overview" unnumbered chapter, including the following sections:

Ov1. Summary of Experimental Evaluation: A reasonably short section summarizing the experimental evaluation (high-level parameters, measurements and conclusions). This can be similar to an executive summary of the entire document. It gives a glimpse of the results across the evaluated crypto-systems, and defers to the corresponding chapters the detailed notes.

Ov2. Notes on the Reference Implementation: A short section enabling a baseline understanding of how the Reference Implementation and the experimental execution/evaluation depend on the Team's Core Code, the Bundled Dependencies, and the External Dependencies. The section, which is not meant to be a comprehensive analysis, should gently aid/motivate the reader to initiate a review of the Reference Implementation. It is useful to allude to the existence of scripts that enable the automated experimental execution and corresponding measurements of the crypto-systems.

Ov3. Experimental Setting: A section introducing the relevant characteristics of the used implementation platform, namely the (possibly emulated) hardware, including the processing units, communication network, and memory. Preferably, the used platform ***should*** be similar to the Baseline Platform (see Imp1 in Section 7.1). If applicable, the report identifies noteworthy differences, and explains whether/how they are expected to affect performance. The experimentation ***may*** use additional platforms, case in which it is useful to index them (e.g., PF1, PF2, ...) for easier reference. This is meant to alleviate the description needed in the subsequent chapters.

8.4. Chapters "Crypto-System X"

For each crypto-system chapter in the Technical Specification, there is a corresponding chapter in the Report on Experimental Evaluation. As mentioned in Section 6.3, the team decides the scope of each crypto-system chapter. Therefore, a "crypto-system" chapter may in fact cover a family of crypto-systems, with various protocols (threshold scheme, ZKPoK, gadget, conventional scheme) and variants thereof. The chapter includes:

EvX.1. Cover page: Similar to the cover of crypto-system chapters in the Technical Specification. It marks the chapter beginning, allows local versioning and (if applicable) identifies the authors, and includes a short abstract specific to the chapter.

EvX.2. Local contents: A table of local contents (up to subparagraph depth), and applicable lists (e.g., of tables, of figures, and of experimental configurations).

EvX.3. Experimental Configurations: A characterization of the diverse experimental configurations. It should clarify the platform's capabilities (e.g., number of processors, and network characteristics), and the protocol parameters (e.g., variant of the crypto-system, number of parties, number of faulty parties, security parameters, input size). Where applicable, this section can refer to details already explained in the "Experimental Setting" section (Ov3) in the Overview Chapter.

In the case of evaluating a threshold scheme, there ***should*** be one configuration with all honest parties, and at least one configuration with at least one corrupted party.

The configurations can be indexed (e.g., Cfg_1, Cfg_2, ...) in order to allow easy referencing in the subsequent sections with measurements and analysis. Also, the indexations can be parametric, e.g., using $\mathrm{Cfg}_1(n, f)$ where n is the total number of parties and f is the number of parties corrupted in some specified manner.

EvX.4. Measurements: A section reporting measurements of various complexities:

- **Perf1. Memory**: Number of bytes simultaneously stored.
- **Perf2. Processing (work)**: Time or number of operations of specified types.
- **Perf3. Communication**: Number of communicated bytes.
- **Perf4. Messages (and rounds)**: Number of transmissions (phases).
- **Perf5. Time**: Duration in seconds.

The batch of measurements ***should*** be obtainable automatically by running a simple command for executing the benchmark script (see Scripts #2). Preferably, the measurements ***should*** be reported: (i) per main phase of the protocol, and in total across an execution; (ii) per party and collectively.

Each metric ***should*** be evaluated across a representative set of configurations supported by each proposed crypto-system. For example, this can include various numbers of parties, various security strengths, and even different platforms. In the case of evaluating a threshold scheme, there ***should*** be at least one comparison between a run with all honest parties, and a run with at least one corrupted party.

The team decides the depth of the measurements report, while striving to provide clarity about the complexity of the crypto-systems. For example, the total time duration of a protocol execution may include differentiated periods of either-or-both-or-none of processing (work) and/or communication for a given party.

EvX.5. Interpretative Analysis: An explanation of the experimental results, interpreting the expected and unexpected observations, namely in comparison with the analytic complexity described in the Technical Specification (see CSX.6 in Section 6.3). 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, or confirm scalability and tradeoffs. It is useful to identify cases where a metric (e.g., processing, communication) can be pipelined or parallelized to reduce latency, and where amortization is possible across executions. The analysis may also include comparisons with the known performance of other relatable schemes.

9. Security Requirements

Requirement 9.1. Parametrization of crypto-systems

The specification, implementation, and experimental evaluation of each crypto-system ***shall*** instantiate at least one concrete parametrization.

Section 9.1 discusses the general goals of security strength, with regard to computational and statistical complexity. Section 9.2 specifies requirements about the threshold setting.

Critical safety properties. The security requirements in this section apply only to critical safety properties, to be identified in CSX.3 (see Section 6.3). Key secrecy is always assumed to be critical, whereas the criticality of other properties depends on the crypto-system. For example, unforgeability is a critical safety property for signature schemes. Properties that are not deemed critical ***may*** be sacrificed (e.g., a "security with abort" notion sacrifices availability in the presence of a malicious adversary). The criticality of some properties may depend on the use case, and can be argued in each submission.

9.1. Security Strength Levels

Table 5 lists three parameters of security strength: classic computational, quantum computational, and statistical. For each parameter, there is a required (if applicable) lower bound of security strength (i.e., with regard to crypto-systems submitted in reply to the present call), and (for comparison) a suggested lower bound for an optional second instantiation.

Table 5. Security-strength parameters

Security parameter	**Domain type/unit**	**Required** (1st case)	**Suggested** (2nd case)
κ (Classic computational)	Number of bits	$\gtrsim$128	$\gtrsim$192
θ (Quantum computational)	Category	$\geq$1	$\geq$3
σ (Statistical)	Number of bits	$\gtrsim$40	$\gtrsim$64

Per SP800-57-P1-R6-IPD (see its §4.6.1 and §C), the classic computational security strength notion is measured in number of bits, whereas the computational security strength against adversaries with quantum capabilities is measured in categories. The requirement for $\theta \geq 1$, and suggestion for $\theta \geq 3$ is only applicable if the analysis claims plausible post-quantum security. The statistical security parameter does not appear in NIST standards, but is customary in advanced cryptographic protocols (e.g., related to MPC, FHE, and ZKP).

The symbol $\gtrsim$ denotes "greater than or approximately equal to", in order to allow a small slack. For example, it can be acceptable to consider a protocol that uses standard "128-bit" parameters (e.g., AES-128) but then the security estimate only guarantees at least 125 bits of security. This slack is also assumed implicit in the PQ categories when using the symbol $\geq$ ("greater than or equal to"), since there the parameter θ is discrete. The strength is also dependent on the computational complexity (e.g., a Boolean gate or a full-fledged AES evaluation) equated to one bit of strength, which is often left implicit.

9.1.1. Computational Security

Classic security levels. Security strengths are measured in number of bits, with usual levels being powers of two (e.g., 128, 256) or simple sums thereof (e.g., 192, 224), which typically match the security strengths of standardized primitives (e.g., AES-{128,192,256}).

Requirement 9.2. Classic security $\gtrsim 128$ bits

Each submitted crypto-system ***shall*** include 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 [SP800-57-P1-R6-IPD, §4.6.1 & §C], 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.

Requirement 9.3. Post-quantum security $\theta \geq 1$

Each submitted crypto-system that is claimed to be PQ ***shall*** include at least one instantiation with PQ security $\theta \geq 1$.

Preferably (when applicable, but not required), the submission presents two instantiations: one with $\theta \leq 2$ (yet satisfying requirements 9.2 and 9.3), and another with $\theta \geq 3$.

Parameter sets. For each category in Class N (see Section 10), the parameter sets in scope already satisfy $\kappa \gtrsim 128$ or $\theta \geq 1$. For Class S (see Section 11), the submission needs to specify at least one such parameter set. This applies to submitted threshold schemes and their conventional primitives, as well as to ZKPoKs and gadgets.

Considering the exploratory/research interest of this Threshold Call, the computational security strength of a submitted threshold scheme does not need to be as high as that of the primitive being thresholdized. Also, the PQ/QV property of the threshold scheme does not have to match that of the conventional primitive (see Section 3.2). 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*** propose a QV threshold scheme with classical security $\kappa \approx 128$ bits.
- ***May*** propose a PQ threshold scheme with quantum security $\theta = 5$.

9.1.2. Statistical Security

The security strength of a protocol (e.g., some threshold schemes and interactive ZKPoKs) can also depend on a statistical parameter σ, which is the additive inverse of the binary logarithm of the probability that a security property is broken during a protocol execution.

Requirement 9.4. Statistical security $\gtrsim 40$ bits

Each submitted crypto-system ***shall*** include at least one instantiation with statistical security strength $\sigma \gtrsim 40$, provided that the computational security is satisfied as required.

Preferably, there should also be a described instantiation with $\sigma \gtrsim 64$. Depending on the protocol, it may be possible to carefully transform it (e.g. from interactive to non-interactive) so that it stops relying on a statistical security parameter (see Appendix B.2.4).

9.2. Security of Threshold Schemes

The following applies to threshold schemes submitted as crypto-systems (see Section 6.3).

9.2.1. Security Formulation/Idealization and Security Analysis

Requirement 9.5. Security formulation/idealization

The security analysis in the Technical Specification ***shall*** specify at least one security formulation or idealization, as reference to assess the security of each crypto-system.

The security formulation/idealization entails formulating (see CSX.5 in Section 6.3) an **ideal functionality** (e.g., in the ideal-real simulation paradigm / universal composability framework) or/and an idealized **game** (or set of games) that define an idealized input/output interaction (e.g., based on an oracle) and the capabilities and goals of an adversary.

The security analysis can include more than one security formulation or idealization, to convey complementary perspectives. For example, a threshold scheme may be:

- proven secure in a game-based setting against active-**adaptive** corruptions, and simultaneously secure in a simulatable setting against active-**static** corruptions, up to a certain threshold or thresholds; or
- proven secure (without abort) against active-**static** corruptions up to one threshold, and proven (or "technically argued" as) secure-with-abort against active-**adaptive** corruptions up to another threshold.

9.2.2. Adversarial Corruption Capabilities to Consider

Requirement 9.6. Adversarial corruption capabilities

The security analysis of proposed threshold schemes ***shall*** take in consideration the **corruption capabilities** (with regard to corruption thresholds) of a modeled adversary.

In particular, the following corruption capabilities need to be considered:

1. **Active (malicious):** Can choose which parties to corrupt, accessing their internal state and controlling their behavior (possibly deviating from the prescribed protocol).
2. **Adaptive:** Can decide, before and throughout the protocol, which parties to corrupt.
3. **Mobile:** Can progressively corrupt honest parties across multiple executions of a protocol, including those that may be progressively recovered from previous corruptions.

Section 9.2.4 discusses requirements of threshold security (proven or/and technically argued for), with regard to some of these capabilities.

The above requirement per se means that the security analysis will discuss the consequences of combinations of these adversarial capabilities, regardless of whether security is protected against them, or broken by them. For example, it is useful to know what security properties can be broken when the number of corruptions of a certain type go beyond what is acceptable by threshold profile with respect to which security is proven. For example, a mobile adversary that steadily corrupts additional parties, progressively increasing the number of corrupted parties (possibly even in spite of some proactive recovery mechanism), may break security goals when the number of corruptions exceeds a given threshold.

9.2.3. Threshold Profiles

Requirement 9.7. Threshold profile

The system model (see CSX.3 in Section CSX.3) ***shall*** define at least one threshold profile for which the threshold scheme satisfies the security requirements expressed ahead.

See also the informative notes in Appendix C.3. The term "threshold" is used for convenience, but the actual access structure ***may*** be different.

Requirement 9.8. Differentiated thresholds

The security analysis ***shall*** differentiate thresholds across security properties, clarifying which thresholds apply to which main security properties. The analysis ***should*** also characterize the breakdown that occurs when threshold-profile assumptions are broken.

For at least one security formulation/idealization of a proposed threshold scheme, and considering at least one threshold profile, with regard to safety properties deemed critical (e.g., key secrecy for any cryptographic primitive, unforgeability for signatures, semantic hiding for encryption), the security analysis:

9.2.4. Threshold Security (Safety Goals)

The following security characteristics in the threshold setting are meant with regard to critical safety properties, rather than every conceivable security property. For instance, "security-with-abort" is acceptable, when availability is not considered a critical property.

Requirement 9.9. Provable security against active corruptions

The security analysis ***shall*** prove active security, i.e., against **active** (malicious) corruptions (possibly active-static), as described in §9.2.4.1.

Requirement 9.10. Technically-argued security against adaptive corruptions

The security analysis ***shall*** "technically argue" about active-adaptive security, i.e., against **adaptive** corruptions, as described in §9.2.4.2.

Requirement 9.11. Discussion of mechanisms for proactive or reactive security

The security analysis ***shall*** discuss envisioned mechanisms for proactive and/or reactive security, i.e., against **mobile** and/or identifiable corruptions, as described in §9.2.4.3.

9.2.4.1. Active Security (Against Active Corruptions)

Regarding Requirement 9.9: The submitted threshold schemes ***shall*** be proven secure with regard to a security formulation (e.g., game-based or simulation based) where some of the parties can maliciously deviate from the prescribed protocol. While active security is required with regard to critical safety properties (e.g., key-secrecy, unforgeability), other properties can be compromised. For example, *security with abort* (where an active adversary is still able to abort protocol executions) is acceptable if availability is deemed non-critical.

The requirement of provable active security can be limited to a setting with some well-identified trusted setup and/or assumptions, including about the actual input given to the cryptographic operation. For example, a threshold FHE-decryption scheme can be submitted as actively secure for a restricted use case where the input ciphertext is assumed to have been honestly produced. Naturally, it is relevant to be aware of the insecurity consequences of not meeting the prescribed trusted setup.

The security proof of the submitted threshold scheme can make security assumptions about building blocks (e.g., broadcast). The proof in the specification document still has to list and explain those assumptions, and include appropriate references to publicly-accessible-for-free security analysis of the building blocks. Also, the implementation will have to include code for the building blocks.

9.2.4.2. Adaptive Security (Against Adaptive Corruptions)

Regarding Requirement 9.10: With respect to critical safety properties, there is a **strong preference** for achieving provable security against adaptive corruptions, even if some other security properties are only satisfied against a static adversary. However, considering technical difficulties in the field, the requirement is relaxed as follows:

There ***shall*** a best-effort technical discussion about security against an adversary capable of adaptive corruptions. In particular, the Threshold Call wants to avoid pathological protocols that, despite being proven secure against static-active corruptions, would leak the secret key (or break some other critical safety property) in case of adaptive corruptions. A proof of security against adaptive corruptions is not required, but it would add value.

Appendix C.2.2 acknowledges the challenge of proving security in case of adaptive corruptions. 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.

9.2.4.3. Recovery Mechanisms (Against Mobile Attacks)

Regarding Requirement 9.11: A submitted threshold scheme is **not required** to include recovery mechanisms (proactive or reactive) whereby corrupted parties are identified, removed or replaced (or recovered back to a honest state). However, the submission ***shall*** discuss how it envisions corresponding augmentations, which are important for handling a **mobile** adversary that persistently attempts to increase the number of corrupted parties. For example, such discussion can tackle whether it is trivial or there are challenges in integrating some specific recovery mechanism (e.g., secret resharing; see S7) to handle the case of past-leaked shares. In such case, it is also useful to discuss the needed conditions (e.g., requirement of some initial/final agreement by a qualified quorum) for its integration. For the reactive-recovery case, the property of identifiable abort is of particular interest. The submission of such recovery augmentations (i.e., specification, implementation and evaluation) is also welcome.

On the combination of required-and-recommended security properties. The following is an example of a possible combination in a submission of a threshold scheme:

- static-active security-with-abort is proven in a simulation-based or game-based setting;
- adaptive-active security is technically argued for, less formally than in a proof;
- proactive security is argued for as achievable with an ad-hoc subprotocol/phase that is not specified/implemented, but is referenced in the submission.

The security analysis can go beyond the required minimum. For example, this could be the case of including (i) a proof of security against active-adaptive corruptions, (ii) a subprotocol for proactive recovery, or (iii) a process for safely handling maliciously produced inputs. A threshold scheme can understandably be less efficient in order to provide more sophisticated security guarantees.

9.2.5. Security Strength Across Formulations

Section 9.1 considers the requirements on security strength (e.g., 128-bit level). However, a threshold scheme may have different security strengths across the static-active and **adaptive-active** corruption scenarios. The specified requirement of minimal security strength (for specification, implementation, and evaluation) can be assumed to relate to the active-static case. Submissions are encouraged to also explain which parameters are needed to achieve 128 bits of security against **active-adaptive** corruptions.

10. Requirements for Class N Schemes

Class N has four categories (types of primitive), as already listed in Section 2.1 (Table 2). The present section specifies requirements for the submission of threshold schemes for primitives in each of those categories: signing (N1; see Section 10.1), PKE (N2; see Section 10.2), symmetric (N3; see Section 10.3), and KeyGen (N4; see Section 10.4).

10.1. Category N1: Signing

Primitives in scope. The third column of Table 6 lists various signing primitives of interest. Appendix A.1 has additional details. Signing primitives from other PQ signature schemes (e.g., FN-DSA) may be considered in scope once they are NIST-standardized.

Table 6. Primitives in the Signing 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-SIGN	$\|N\| = 3,072$	$\|N\| \geq 7,680$	A.1.3
|	|	RSASSA-PKCS-v1.5.Sign	|	|	|
N1.4: ML-DSA	[FIPS-204]	[Hash]ML-DSA.Sign	ML-DSA-44	ML-DSA-{65,87}	A.1.4
N1.5: SLH-DSA	[FIPS-205]	[hash_]slh_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. {...} = set of options. [Det-] = Optional **Det**erministic variant. [Hash] or [hash_] = Optional pre-hash variant. 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.

Requirement 10.1. [N1] Interchangeability with regard to verification

Submitted threshold schemes for signing operations in N1 ***shall*** be interchangeable (see Section 3.3.2) with regard to the standardized verification mechanism of the NIST-standardized signature scheme.

Requirement 10.2. [N1] Characterization of safety properties

The security analysis (CSX.5) ***shall*** characterize the critical safety properties protected by the threshold scheme, including the achieved type(s) of **unforgeability** (which requires definition), and whether a non-aborting adversary can bias the signature value.

Note: The publication NIST-IR8214B-ipd, with notes on threshold EdDSA/Schnorr signatures, includes technical reflections that are useful to consider within category N1.

Security Strength: Per Section 9.1, the submission will consider a parametrization that aims to achieve, for the critical safety properties, a security strength consistent with the parameters described in Table 6.

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 submitted **threshold scheme** (interchangeable with regard to verification) ***may*** opt between probabilistic and pseudorandom (PR) modes, including Prob, Q-PR, and F-PR (as described ahead).

Threshold modes with regard to signature (non-)determinism. The mechanism by which the secret randomness (or pseudorandomness) of the signature is obtained, combining contributions from 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 standardized (non-threshold) signature scheme specifies the signing operation as deterministic (possibly pseudorandom), but the resulting signatures are not verifiably deterministic, then the F-PR mode (i.e., deterministic even if the quorum changes) can 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)

10.2. 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-establishment (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. PKE primitives from other PQ-KEMs (e.g., HQC) may be considered in scope once they are NIST-standardized.

Table 7. Primitives in the PKE 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-Establishment. PKE = Public-Key Encryption. K-PKE = ML-KEM-related Public-Key Encryption. RSADP = RSA Decryption Primitive. RSAEP = RSA Encryption Primitive.

10.2.1. Threshold Higher-Level Primitives

A submission with a threshold scheme for a primitive in N2 (see Table 7) ***may*** (i.e., optionally) 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 key agreement/transport/encapsulation schemes/mechanisms (KAS/KTS/KEM), although likely impractical, considering that an interchangeable

threshold implementation of one side of a full-fledged NIST-standardized 2KE protocol requires thresholdizing threshold-unfriendly symmetric primitives (N3) used for key-derivation and/or key-confirmation steps.

10.2.2. Threshold PKE-Decryption

Appendix C.4 recalls possible distributed interfaces (as described in NIST-IR8214A) with regard to (w.r.t.) an identified input/output: not secret-shared (NSS), secret-shared input (SSI), and secret-shared output (SSO).

Requirement 10.3. [N2] I/O interface of threshold scheme for PKE-`Encrypt`

Within category N2, a submitted threshold scheme for a PKE-`Encrypt` primitive ***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.

Requirement 10.4. [N2] Functionally equivalent decryption

A threshold scheme for decryption (core primitive or higher-level operation) ***shall*** be functionally equivalent to a NIST-specified one (e.g., RSADP, RSASVE.Recover, RSA-OAEP.Decrypt, K-PKE.Decrypt, ML-KEM.Decaps), barring negligible differences.

10.2.3. Threshold PKE-Encryption

Requirement 10.5. [N2] I/O interface of threshold scheme for PKE-`Decrypt`

Within category N2, a threshold scheme for a PKE-`Decrypt` primitive ***shall*** be SSI w.r.t. the private decryption key (i.e., as required by the default).

The scheme ***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.

Requirement 10.6. [N2] Interchangeable encryption, with regard to decryption

A threshold scheme for public-key encryption ***shall*** be interchangeable w.r.t. standardized decryption (and preserve the security notions of interest).

The encryption ***may*** follow a probability distribution that is different from the standardized one (RSASVE.Generate, RSA-OAEP.Encrypt, K-PKE.Encrypt, ML-KEM.Encaps). 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 Requirement 10.2 for threshold schemes for non-verifiably deterministic schemes).

10.2.4. Threshold Security

Requirement 10.7. [N2] Characterization of safety properties

The security analysis (CSX.5) ***shall*** characterize the critical safety properties protected by the threshold scheme, including the hiding of the secret material, and whether a non-aborting adversary can bias the output.

Security Strength: Per Section 9.1, the submission will consider a parametrization that aims to achieve for the critical safety properties a security strength consistent with the parameters described in Table 7. It is worth noting that the N2-category primitives enumerated in Table 7, intended for thresholdization, are not themselves the main algorithms of a KEM, and therefore have different security properties. For example, RSAEP provides one-wayness but is not even a probabilistic encryption procedure.

10.3. Category N3: Symmetric Primitives

On the scope of the "symmetric"category: The "symmetric" category includes not only symmetric-key crypto (e.g., cipher, and MAC), but also *keyless* hash and XOF primitives. This semantic extension of "symmetric" borrows from the traditional association of "symmetric-key" crypto and the mentioned keyless primitives, which are often based on traditional designs related to constructions of symmetric-key crypto. For example, hash functions can be build based on block-ciphers, and can be used to create MAC's.

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 Tag generation (TagGen, usually just called MAC) (see §A.3.4), based on above-mentioned primitives/schemes (cipher, hash function, Keccak permutation).

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

Table 8. Primitives in the "Symmetric" category N3

Subcategory: Type	**NIST reference**	**Spec or family**	**Primitive type**	**Cryptographic parameters** $\kappa \approx 128$ or $\theta = 1$	**Cryptographic parameters** $\kappa \gtrsim 192$ or $\theta \geq 3$	**§ in this call**
N3.1: Blockcipher	[FIPS-197]	AES	{Enc,Dec}	AES-128	AES-{192,256}	A.3.1
N3.2: AEAD	[SP800-232]	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]	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	|

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.

Requirement 10.8. [N3] Threshold schemes for "symmetric" primitives

A submission within category N3 ***shall*** specify, implement and evaluate at least one threshold scheme for one the primitives listed in Table 8.

If the package proposes a threshold-MAC (subcategory N3.4), then it ***should*** first specify and implement the threshold scheme for the primitive (e.g., cipher, hash, XOF) underlying that MAC, and then use it to implement the MAC.

Security Strength. Per Section 9.1, the submission will consider a parametrization that aims to achieve for the critical safety properties a security strength consistent with the parameters described in Table 8.

Input lengths. The experimental evaluation ***should*** benchmark at least the case of an input that fits in a single primitive evaluation. For example: for AES enciphering, a 128-bit plaintext (i.e., one block); for SHA-256 hashing, a 447-bit message, so that the required padding of 65 bits leads it to the block size of 512 bits. The experimental evaluation ***may*** also benchmark the threshold execution of many (e.g., 256) operations. This may help clarify possible complexity amortizations, as well as the feasibility of the threshold approach for modes of operation that repeat the evaluation of many building blocks.

Input/Output (I/O) interface: The thresholdization of keyless primitives makes sense when its input or/and output are considered secret.

Requirement 10.9. [N3] I/O of threshold schemes for symmetric primitives

For hashing and XOF'ing (keyless primitives), the threshold scheme ***shall*** consider an SSI mode w.r.t. the message. For the symmetric-key'ed primitives (i.e., AES, Ascon-AEAD, MAC), the threshold scheme ***shall*** consider the key is secret-shared.

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 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; 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. These can be useful when using MAC for key-derivation and/or confirmation, within a 2KE protocol.

10.4. 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.

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, usually not secret-shared, obtained from the product of two secret primes),

Table 9. Primitives in the KeyGen category N4

Subcategory # **Type of KeyGen**	**Related operations**	**Sections in this Call**
N4.1: ECC KeyGen	Scalar multiplication (in additive notation)	10.4.1, A.4.1
N4.2: RSA KeyGen	Generate modulus and/or key-pair	10.4.2, A.4.2
N4.3: ML KeyGen	ML-DSA.KeyGen, K-PKE.KeyGen	10.4.3, A.4.3
N4.4: HBS KeyGen	Generate hash trees (for SLH-DSA, LMS, XMSS)	10.4.4, A.4.3
N4.5: Secret RBG	RBG for secret bit-strings or integers	10.4.5, 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.

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)	10.4.1
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 $N = p \cdot q$	10.4.2
|	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 $\hat{\mathbf{s}}, \hat{\mathbf{e}}$	$\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$	10.4.3
ML for ML-DSA Sign	Secret vectors $\mathbf{s_1}, \mathbf{s_2}$	$\mathbf{t} = \text{NTT}^{-1}(\hat{\mathbf{A}} \circ \hat{\mathbf{s_1}}) + \mathbf{s_2}$	|
AES Encipher/Decipher	Bit-string k	—	10.4.5
Key derivation	$k = \text{KDM}(Z, \ldots)$	—	—
Key confirmation (KC)	Bit-string k	MacTag $T = \text{KC}(\ldots, k, \ldots)$	—

Legend: Enc/Dec = Encrypt/Decrypt. EC = Elliptic Curve. KDM = Key-Derivation Mechanism. NTT = Number Theoretic Transform. RSAEP = RSA Encryption Primitive. See legends of Tables 2, 6, and 7.

and/or commitments of everyone's private keys. Some domain parameters are agreed upon before the DKG (e.g., elliptic curve, security strength κ, RSA public encryption key e).

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.

Requirement 10.10. [N4] DKG interchangeable with KeyGen

A submitted DKG ***shall*** be interchangeable with a conventional KeyGen, with regard to a subsequent operation of interest (e.g., signing or encryption). The Technical Specification ***shall*** explain the appropriateness of the obtained, from a security perspective, considering the conventional (non-threshold) KeyGen.

Security Strength: Per Section 9.1, the submission will consider a parametrization that aims to achieve for the critical safety properties a security strength consistent with the parameters described in Table 8.

10.4.1. 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).

Requirement 10.11. [N4.1] Elliptic curves

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.

10.4.2. Subcategory N4.2: RSA KeyGen

Requirement 10.12. [N4.2] RSA modulus size

A submission of RSA DKG ***shall*** obtain a modulus of size $|N| \geq 3072$, for $\kappa \gtrsim 128$.

There are applications of an RSA modulus without known factorization. Therefore, an RSA-DKG scheme ***may*** be submitted standalone (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 the one described in §A.4.2.2. However, any such differences need to 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) that is different from what is described in the NIST publications need to present a rationale to convey adequacy (e.g., adequate number of rounds).

10.4.3. Subcategory N4.3: ML KeyGen

Requirement 10.13. [N4.2] Modulo-lattice parameters

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 (i.e., 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 if it retains the functionally and security of the final key.

- **Different encodings/representations.** The optimization of threshold schemes may suggest different encodings or representations that are mathematically equivalent (e.g., whether/when to use the Number Theoretic Transform (NTT) and its inverse NTT^{-1}). This ***may*** be done, provided that the keys are interchangeable with regard to 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 $\mathbf{t} = \mathsf{NTT}^{-1}(\hat{\mathbf{A}} \circ \mathsf{NTT}(\mathbf{s_1})) + \mathbf{s_2}$, 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.

10.4.4. 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.

10.4.5. 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 that 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 à 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.

11. Requirements for Class S Schemes

Class S has categories for the submission of threshold schemes for primitives that are not part of a NIST standardized crypto-scheme, and others for the submission of zero-knowledge proofs of knowledge and other auxiliary gadgets. A prior established credibility of the proposed crypto-systems (conventional and/or threshold) is welcome, though not strictly required. In particular, Class S welcomes the submission of crypto-systems that have been previously peer-reviewed, deployed, and/or standardized or proposed as standards.

Class S categories matching with Class N. 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 match the Class N categories, in the sense that they refer to the same types of primitives, though meant to have some distinctive factor that motivates their submission.

Requirement 11.1. Distinctive factors in Class S primitives

The Technical Specification of a threshold scheme for a primitive within categories S1, S2, S3 and S4 ***shall*** include a motivating comparison between that primitive (or the scheme in which it is used) and the NIST-specified primitives of the same type (or its corresponding NIST standardized schemes), identifying distinctive factors.

For example, Class S is suitable for the submission of a threshold scheme for a signing primitive that produces signatures that are succincter than signatures obtainable from NIST standardized signatures schemes. In such case, the motivating comparison ***should*** highlight the succinctness feature. More generally, a proposed primitive (or corresponding scheme) in categories S1–S4 ***should*** have a distinctive feature, such as being threshold friendlier, relying on different cryptographic assumptions, or having better efficiency (the conventional scheme or its threshold version) in some useful metric (e.g., output length, 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 considers primitives from some types of schemes that are not standardized by NIST. Correspondingly, Class S has the additional categories S5 for FHE (see Sections 11.5 and B.1), S6 for ZKPoKs (see Sections 11.6 and B.2), and S7 for gadgets (see Section 11.7). In submitted proposals of ZKPoKs (S6) or gadgets (S7), a conventional scheme (i.e., non-threshold) suffices, i.e., the specification of a corresponding threshold scheme is optional. FHE and ZKPoK are also of specific interest to the NIST Privacy-Enhancing Cryptography (PEC) project [Proj-PEC].

Specification of conventional primitives. Submission packages proposing threshold schemes for primitives in Class S need to provide proper context about the corresponding primitives, since they are not part of NIST standards.

Requirement 11.2. Explanation of conventional primitive in Class S

In a submission of a threshold scheme for a primitive in Class S, the Technical Specification ***shall*** also explain the conventional (i.e., non-threshold) scheme (and at least one concrete set of parameters for implementation), 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 that is not specified by NIST needs to explain the verification and KeyGen primitives, in addition to 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 6.3); or (ii) at a higher-level (in Pre5, CSX.4) but with sufficient detail to understand the notation, interface, and security properties, and including a reference to an authoritative specification that is freely available to the public.

11.1. Category S1: Signing

Category S1 is for submissions of threshold schemes for the signing primitive of digital signature schemes that are not standardized by NIST. Featured examples in scope:

1. Threshold friendly and quantum resistant.
2. Succinct and verifiably deterministic (i.e., a function of the message and the public-key), even if quantum vulnerable (e.g., based on pairings).
3. ZKP friendly (e.g., easier to generate unlinkable ZKPoKs of a signature).
4. Blinding friendly (i.e., with a structure that enables an efficient protocol for blind signing, e.g., 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).

A signature is essentially a transferable 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., of the signing key).

11.2. 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. Similar to category N2 (see Section 10.2), submitted threshold schemes in S2 need to apply to the Decrypt primitive (when the decryption key is secret-shared), or to the Encrypt primitive (SSI with regard to the plaintext, i.e., when the plaintext is secret-shared). For example, PKE-encryption can be considered for the purpose of key encapsulation.

11.3. Category S3: Symmetric

Typical "symmetric" primitives (such as the ciphers, hash functions, XOFs, and MACs in category 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. Category S3 enables proposals of threshold-friendlier primitives of this type. It is also of interest to consider friendliness with regard to 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 subcategories/families: S3.1 for pseudorandom permutations (PRP, e.g., for enciphering); S3.2 for pseudorandom functions (PRF, e.g., for MAC'ing); S3.3 for fash functions or XOFs. If a design principle allows primitives to be built 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 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 (TF) (despite the use of TF-PKE primitives), because of their use of threshold-unfriendly key-derivation/confirmation primitives. The present category S3 can, for example, be used to propose TF ciphers, hashing, or MAC 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 (hiding and binding, and either non-malleable of homomorphic), with either succinct commitment or opening, and possibly ZKP friendly with regard to selective disclosure. Such additional specification in this category ***should*** only be considered if it is based on building blocks that are used to first specify one of the above-mentioned families of primitives of interest. Alternatively, it can be submitted in the category of gadgets (S7).

11.4. 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 (possibly PQ), or 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).

11.5. Category S5: FHE

Category S5 relates to fully-homomorphic encryption (FHE), which is a special type of encryption scheme that allows for arbitrary computation over encrypted data. Besides the usual operations of `KeyGen`, encryption (`Enc`), and decryption (`Dec`), it also allows for homomorphic evaluation (`hom`). Given one or more ciphertexts, produced via `Enc`, the `hom` operation can be used to produce a new ciphertext that encrypts the result of an intended operation over the original plaintexts. This operation is performed without the original plaintext and the decryption key. Appendix B.1 has additional details.

11.5.1. Threshold Schemes for FHE

The use of threshold schemes as a way of decentralizing trust regarding secret material (e.g., key, or confidential plaintext) also applies to FHE schemes.

Requirement 11.3. [S5] FHE operations for thresholdization

A submission in S5 ***shall*** specify (in CSX.4), implement and evaluate a scheme for (i) threshold decryption (i.e., with a secret-shared key), **or** (ii) threshold encryption of a secret-shared value, **or** (iii), in case of a symmetric-key FHE encryption scheme, threshold encryption of clear plaintext when the secret key is secret-shared.

The thresholdization of other primitives (such as homomorphic evaluation) is optional. The thresholdization of KeyGen (assigned to S4) is also optional.

11.5.2. Conventional FHE

Since FHE is not part of NIST-standards, a submission in S5 also needs to provide context about the conventional (i.e., non-threshold) FHE scheme (symmetric-key of public-key).

Requirement 11.4. [S5] Operations in a conventional FHE scheme

The Technical Specification ***shall*** explain at least the interface and properties of the four main algorithms: `KeyGen`, `Enc`, `Dec`, `hom`.

Explanation of conventional FHE. The explanation of conventional FHE can be a high-level description (i) in the Preliminaries chapter (see Pre5), or (ii) in the Crypto-System chapter (see CSX.4) that also explains the threshold scheme. If the description is at a high-level, then it needs to refer to an external reference with a thorough specification. Alternatively, the conventional scheme can itself be proposed as a full-fledged crypto-system (i.e., specified, implemented and evaluated), provided that Requirement 11.3 is also met.

The specification of homomorphic evaluation (`hom`) can include a set of related 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 (i.e., a *bootstrapping* to reduce the internal 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, even if their specification has been provided only as a high-level explanation. This recommendation is in addition to the required benchmarking of the threshold schemes.

Benchmarking use-cases. The selection of benchmarking use-cases to evaluate performance (in the Report on Experimental Evaluation) 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, or motivated with practical scenarios.

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) benchmark one use case in which it performs well, and (ii) explain the anticipated real-world adoptability of that application (see CSX.7). For comparison, the benchmarking is encouraged to also measure performance for some operation that is anticipated to be performed better by a different FHE scheme.

11.6. 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 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. This call uses some ZKPoK-related terms in a broad sense:

- the "proof" in ZKPoK encompasses not only traditional *proofs* with perfect or statistical soundness, but also *arguments* (with computational soundness).
- the "zero-knowledge" (ZK) in ZKPoK encompasses both the cases of deniable (i.e., non-transferable) and transferable (i.e., publicly verifiable) proofs.

For example, a zk-SNARK (succinct non-interactive ZK argument of knowledge) is within the scope of this category. However, any submission of a ZKPoK needs to technically clarify its properties detail, including composability.

Requirement 11.5. [S6] Type of soundness, ZK and composability

The submission of a ZKPoK ***shall*** clarify its security properties, including its soundness type (e.g., "proof" vs. "argument"), its zero-knowledge properties (e.g., transferable or not, computational or statistic), and its composability.

Conventional versus threshold. Category S6 does not require submitting a threshold version of a ZKPoK, but it needs (to specify, implement, and evaluate) a conventional one.

Requirement 11.6. [S6] Conventional ZKPoK

A submission of ZKPoK ***shall*** specify at least a conventional (i.e., non-threshold) ZKPoK. If a threshold ZKPoK generation is proposed (which is optional), with the secret input (the witness) being secret-shared across a distributed prover, then it ***shall*** be interchangeable with regard to verification of the explained conventional ZKPoK.

ZKPoK applicability. Category S6 is for ZKPoKs that can be related to the scope of other categories. Appendix B.2.1 lists several examples in Table 16.

Requirement 11.7. [S6] ZKPoK of interest to another category

A submission proposing a ZKPoK ***shall*** showcase an instantiation of interest related to a primitive in the scope of another category (e.g., a ZKPoK of a signature that is valid with regard to a public key, or of a private-key that corresponds to a public key).

The proposed ZKPoK system can be (i) specialized, i.e., tailored to a knowledge statement related to a specific type of primitive (e.g., knowledge of a type of signature); or (ii) general, i.e., applicable to a broad range of problems/languages (e.g., for proving any statements expressed in terms of a non-deterministic polynomial (NP) complete relation). In the general case, the instantiation depends on a representation, such as based on circuits (see Appendix B.2.5), or some other constraint system. This instantiation ***should*** be achieved by modularly specifying in a standalone file (e.g., a circuit file) a concrete system of constraints. Both the prover and the verifier parse this file, respectively while generating and verifying the proof. The ZKPoK statement can then be changed by simply changing that file.

ZKPoK security strength. Per Section 9.1, the submission of a ZKPoK will indicate a parameter set for achieving at least one profile of security strength for soundness and zero-knowledge, satisfying $(\kappa, \sigma) \gtrsim (128, 40)$. The proposal of a second parameter set achieving $(\kappa, \sigma) \gtrsim (192, 64)$ is also encouraged. See also 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.

11.7. Category S7: Gadgets

"Gadget" in a broad sense. In this Threshold Call, the term "gadget" can be used in a broad sense to denote a module (primitive or scheme; conventional or distributed/thresholdized) that can be used as a building block of another crypto-system. For example, a secret-sharing scheme is a typical gadget used to support threshold schemes. Most crypto-schemes submitted in other categories will therefore make internal use of gadgets in this broad sense. However, not every such gadget should be framed as a full-fledged crypto-system proposed for standalone analysis in that same submission package.

"Gadget" in the strict sense of submitted crypto-system. Category S7 allows for the submission of high-value *gadgets* that are deemed useful for the threshold setting, and simultaneously warrant a standalone analysis as a full-fledged crypto-system.

Requirement 11.8. [S7] Gadget as a full-fledged crypto-system

In a submission that proposes the standalone analysis of a gadget as a full-fledged crypto-system, the gadget ***shall*** be (i) thoroughly specified in the Technical Specification, along with a strong case for why the gadget can be used to support crypto-systems in other categories; (ii) implemented and accompanied with scripts and instructions in the Team's Core Code of the Reference Implementation, and (iii) reported in the Report on Experimental Evaluation.

Example Gadgets: Sophisticated variants of secret-sharing scheme, such as verifiable or publicly-verifiable; garbled circuits; oblivious transfer; correlated-randomness generation; 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. See related notes in §5.3.1 of NIST-IR8214A, and §5.5.2 of NIST-IR8214B-ipd.

Framing of gadgets within a submission package. Table 11 differentiates three alternative framings of gadgets in a submission package. Only the third case corresponds to proposing the gadget for standalone analysis as a *full-fledged crypto-system*. The other two correspond to a lighter inclusion, either as an *external module* (case 1, still requiring some explanation, and an implementation (possibly by externally-developed code)); or as a contributed module (case 2, needing a more thorough explanation or specification, and implementation in the Team's Core Code).

Table 11. Framing a gadget within a package

#	"Framing"	Indep-endent interest	Explana-tion level	Where in the Technical Specification	Where in the Reference Implementation	Where in the Experimental Evaluation Report
1	External **module**	No	High-level	Pre5 or auxiliary part of CSX.4	Any (core code or dependency)	Optional
2	Contributed **module**	Yes	Thorough	Auxiliary part of CSX.4	Core code	**Optional**, within EvX.4–EvX.5
3	Full-fledged **crypto-system**	Yes	Thorough	Main part of CSX.4–CSX.7	Core code	Main part of EvX.4–EvX.5

1. **External module.** The gadget is explained modularly in Pre5 or CSX.4 of the Technical Specification, including at least notation, syntax, properties, and a reference to a detailed specification publicly available for free. The team, favoring clarify, decides whether to include further details. For example, a submission that uses Shamir secret-sharing and/or Lagrange interpolation may explain them succinctly in an indexed paragraph inside Pre5. Within the Reference Implementation, the gadget can be implemented as a code dependency (i.e., possibly not part of the Team's Core Code). A standalone performance evaluation of the gadget does not have to be reported.
2. **Contributed complex module not framed as crypto-system.** The gadget is thoroughly specified as a module in CSX.4 of the Technical Specification. In the Reference Implementation, the gadget is implemented in the Team's Core Code. However, the Report on Experimental Evaluation does not have to include a standalone performance evaluation (report) of the gadget, Instead, it can (optionally) be evaluated simply as a component of another crypto-system that uses it.
3. **Gadget as a full-fledged crypto-system proposed for standalone analysis.** The Technical Specification identifies the gadget as a full-fledged crypto-system (either standalone or as part of a family of crypto-systems) in a chapter CSX, and includes a security formulation and proof of security for the gadget. The gadget is implemented as part of the Team's Core Code of the submitted Reference Implementation, including scripts and instructions for standalone testing the gadget. The Report on Experimental Evaluation includes a modular performance evaluation of the gadget.

A crypto-system specification that uses a gadget that is framed in any of the above-mentioned manners can explain/suggest whether and/or how the crypto-system is open to replacing the gadget by another one (interoperable) with distinct performance/properties.

A gadget proposed as a crypto-system may be simultaneously related to another category. For example, a threshold-friendly hash function also fits in category S3.3 (symmetric), within Class S, if the submission includes a corresponding threshold scheme.

References

[**CAVP**] National Institute of Standards and Technology. *Cryptographic Algorithm Validation Program (CAVP)*. 2025. URL: https://csrc.nist.gov/projects/cavp.

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

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

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

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

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

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

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

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

[**FSF-def**] Free Software Foundation. *What is Free Software?* 2025. URL: https://www.gnu.org/philosophy/free-sw.html.

[**FSF-lic**] Free Software Foundation. *Various Licenses and Comments about Them*. 2025. URL: https://www.gnu.org/licenses/license-list.html#GPLCompatibleLicenses.

[**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. *Homo-*

morphic Encryption Standard. Published by HomomorphicEncryption.org. November 2018. URL: https://homomorphicencryption.org/standard.

[**IACR**] International Association for Cryptologic Research. URL: https://iacr.org.

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

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

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

[**MPTC-Forum**] MPTC-Forum. *Multi-Party Threshold Cryptography (MPTC) Forum*. 2026. URL: https://groups.google.com/a/list.nist.gov/g/mptc-forum.

[**MPTS-2020**] National Institute of Standards and Technology. *NIST Workshop on Multi-Party Threshold Schemes (MPTS) 2020*. Virtual conference. November 2020. URL: https://csrc.nist.gov/events/2020/mpts2020.

[**MPTS-2023**] National Institute of Standards and Technology. *NIST Workshop on Multi-Party Threshold Schemes (MPTS) 2023*. Virtual conference. September 2023. URL: https://csrc.nist.gov/events/2023/mpts2023.

[**MPTS-2026**] National Institute of Standards and Technology. *NIST Workshop on Multi-Party Threshold Schemes (MPTS) 2026*. Virtual conference. January 2026. URL: https://csrc.nist.gov/events/2026/mpts2026.

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

[**News-FN-DSA**] National Institute of Standards and Technology. *NIST Releases First 3 Finalized Post-Quantum Encryption Standards*. August 2024. URL: https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards. Note: A future FIPS 206 is expected to standardize FN-DSA (Falcon-based).

[**News-HQC-KEM**] National Institute of Standards and Technology. *NIST Selects HQC as Fifth Algorithm for Post-Quantum Encryption*. March 2025. URL: https://www.nist.gov/news-events/news/2025/03/nist-selects-hqc-fifth-algorithm-post-quantum-encryption. Note: A future FIPS 207 is expected to standardize HQC.

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

[**NIST-IR8214**] Luís T. A. N. Brandão, Nicky Mouha, and Apostol Vassilev. *Threshold Schemes for Cryptographic Primitives: Challenges and Opportunities in Standardization and Validation of Threshold Cryptography*. (National Institute of Standards and Technology) NIST Internal Report (NISTIR) 8214. March 2019. DOI: 10.6028/NIST.IR.8214.

[**NIST-IR8214A**] Luís T. A. N. Brandão, Michael Davidson, and Apostol Vassilev. *NIST Roadmap Toward Criteria for Threshold Schemes for Cryptographic Primitives*. (National Institute of Standards and Technology (NIST) Internal Report (NISTIR) 8214A. July 2020. 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. *Notes on Threshold EdDSA/Schnorr Signatures*. (National Institute of Standards and Technology) NIST Internal Report (NISTIR) 8214B ipd (initial public draft). August 2022. DOI: 10.6028/NIST.IR.8214B.ipd.

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

[**PubComsIPD**] NIST-MPTC. *Compilation of Public Comments on NIST IR 8214C ipd*. National Institute of Standards and Technology, Multi-Party Threshold Cryptography. Includes copied comments received from 32 reviewers. April 2023. URL: https://csrc.nist.gov/files/pubs/ir/8214/c/ipd/docs/nistir-8214c-ipd-public-feedback.pdf.

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

[**PubComs2PD**] NIST-MPTC. *Compilation of Public Comments on NIST IR 8214C 2pd*. National Institute of Standards and Technology, Multi-Party Threshold Cryptography. Includes copied comments received from 16 reviewers. June 2025. URL: https://csrc.nist.gov/files/pubs/ir/8214/c/2pd/docs/nistir-8214c-2pd-public-feedback.pdf.

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

[**NTCW-2019**] National Institute of Standards and Technology. *NIST Threshold Cryptography Workshop (NTCW) 2019*. Virtual conference. March 2019. URL: https://csrc.nist.gov/events/2019/ntcw19.

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

[**OSI-lic**] Open Source Initiative. *OSI Approved Licenses*. 2025. URL: https://opensource.org/licenses.

[**Proj-CC**] National Institute of Standards and Technology. *NIST Project on Circuit Complexity (CC)*. List of circuits at https://csrc.nist.gov/projects/circuit-complexity/list-of-circuits, and GitHub:usnistgov/Circuits. 2024. URL: https://csrc.nist.gov/projects/circuit-complexity.

[**Proj-LWC**] National Institute of Standards and Technology. *NIST Project on Lightweight Cryptography (LWC)*. 2026. URL: https://csrc.nist.gov/projects/lightweight-cryptography.

[**Proj-MPTC**] National Institute of Standards and Technology. *NIST Project on Multi-Party Threshold Cryptography (MPTC)*. 2026. URL: https://csrc.nist.gov/projects/threshold-cryptography.

[**Proj-PEC**] National Institute of Standards and Technology. *NIST Project on Privacy-Enhancing Cryptography (PQC)*. 2026. URL: https://csrc.nist.gov/projects/pec.

[**Proj-PQC**] National Institute of Standards and Technology. *NIST Project on Post-quantum Cryptography (PQC)*. 2026. URL: https://csrc.nist.gov/projects/post-quantum-cryptography.

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

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

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

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

[**SP800-38B**] Morris Dworkin. *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. May 2005. DOI: 10.6028/NIST.SP.800-38B.

[**SP800-38D**] Morris Dworkin. *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. November 2007. DOI: 10.6028/NIST.SP.800-38D.

[**SP800-56A-Rev3**] Elaine Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, and Richard Davis. *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. April 2018. DOI: 10.6028/NIST.SP.800-56Ar3.

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

[**SP800-57-P1-R6-IPD**] Elaine Barker and William Barker. *Recommendation for Key Management: Part 1 — General.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-57 Part 1 Rev. 6 IPD (Initial Public Draft). December 2025. DOI: 10.6028/NIST.SP.800-57pt1r6.ipd.

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

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

[**SP800-90C**] Elaine Barker, John Kelsey, Kerry McKay, Allen Roginsky, and Meltem Sönmez Turan. *Recommendation for Random Bit Generator (RBG) Constructions.* (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-90C. September 2025. DOI: 10.6028/NIST.SP.800-90C.

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

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

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

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

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

[**SP800-224-ipd**] Meltem Sönmez Turan and Luís T.A.N. Brandão. *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. June 2024. DOI: 10.6028/NIST.SP.800-224.ipd.

[**SP800-232**] Meltem Sönmez Turan, Kerry McKay, Donghoon Chang, Jinkeon Kang, and John Kelsey. *Ascon-Based Lightweight Cryptography Standards for Constrained Devices: Authenticated Encryption, Hash, and Extendable Output Functions*. (National Institute of Standards and Technology) NIST Special Publication (SP) 800-232. August 2025. DOI: 10.6028/NIST.SP.800-232.

[**SPDX-lic**] System Package Data Exchange. *SPDX License List*. 2025. URL: https://spdx.org/licenses.

[**STPPA6**] National Institute of Standards and Technology. *NIST Special Topics on Privacy and Public Auditability 6th Event (STPPA6)*. Virtual workshop. July 2023. URL: https://csrc.nist.gov/events/2023/stppa6.

[**Template**] Luís T. A. N. Brandão. *Template for NIST Threshold-Call Submissions: Organizing Various Crypto-Systems*. Available via NIST-MPTC. 2026. URL: https://csrc.nist.gov/projects/threshold-cryptography.

[**Ubuntu**] Canonical Ubuntu. *Download Ubuntu Desktop*. (Older versions at https://old-releases.ubuntu.com/releases/). 2025. URL: https://ubuntu.com/download/desktop.

[**WPEC-2024**] National Institute of Standards and Technology. *NIST Workshop on Privacy-Enhancing Cryptography (WPEC) 2024*. Virtual conference. September 2024. URL: https://csrc.nist.gov/events/2024/wpec2024.

[**ZkpComRef**] ZKProof. *ZKProof Community Reference. Version 0.3*. July 2022. URL: https://docs.zkproof.org/reference.pdf.

Appendix A Notes on Categories in Class N

This section includes informative notes about some of the categories, subcategories, and primitives within Class N. Related requirements for submissions are discussed in Section 10.

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 10.1). 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). The verification of signatures does not require the use a private key, analogously to the case of PKE-encryption. Yet, a threshold scheme can still be considered in an SSI setting with regard to another input. However, PKE-encryption usually relates to protecting confidentiality, whereas signatures usually relate to protecting integrity. Therefore, threshold schemes for signature verification are considered of secondary interest in this Threshold Call. They can still be proposed, in SSI mode with regard to the signature and/or the message, if included in a package that also proposes a corresponding threshold scheme for signing with a secret-shared key.

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. The publication NIST-IR8214B-ipd has additional notes of interest regarding EdDSA/Schnorr signatures (conventional and threshold). Threshold schemes can be proposed for randomized variants (like Schnorr) with interchangeable verification.

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 12 compares the conventional notations of EdDSA and ECDSA.

Table 12. Notation of EdDSA versus ECDSA (in FIPS 186-5)

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

The ECDSA signing $\texttt{Sign}_n[d](M)$ 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 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; $R = k \cdot G$ 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

The "RSA Digital Signature Algorithm" (RSADSA), defined in §5.4 of FIPS-186-5 by reference to IETF RFC8017, and with some constraints on the KeyGen and possible hash functions, specifies 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 that 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 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. Its construction is inspired by Schnorr signatures, but it is based on lattices in order to achieve quantum resistance, and includes 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 generations of random bits 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].

Pure, 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 context (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), where the input M' already integrates the context and the plaintext message or its hash.

Probabilistic and deterministic modes in the conventional algorithm. Similar to ECDSA (in N1.2), the ML-DSA scheme allows 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 $\rho'' \leftarrow H(K||rnd||\mu, 64)$, 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. Their security relies solely on NIST-standardized hash functions, which are post-quantum secure. The goal of this section is to identify the signature schemes in scope, distinguish between **stateless** and **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 Stateless Hash-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. The message hash is then 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 that correspond to $2 \times 3 \times 2$ options (see Table 12 of FIPS-205): SLH-DSA-{SHA2,SHAKE}-{128,192,256}{s,f}, 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 standard defines a "pure" version (SLH-DSA) and a "pre-hash" version (HashSLH-DSA). These options enable 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 the state across a sequence of signings. However, state management is a difficult challenge, and the reuse of a a one-time signing key can be catastrophic, e.g. break unforgeability. Therefore, their use (and utility) is for limited circumstances [SP800-208, §1.1], while not superseded by 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$^{\mathsf{MT}}$), respectively — are also approved. 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. 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, it is not expected that an efficient scheme will be devised for these HBS schemes (i.e., for a general k-of-n case and with small state per party). Still, it is conceivable that a set of parties can be 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 11.1) (e.g., based on threshold-friendlier or 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 submission requirements in Section 10.2). In practice, PKE is often used as a building block in other standardized schemes, which in turn are used to enable pair-wise key-establishment (2KE) protocols. Future schemes may be added to this scope, once NIST-standardized.

NIST standards with PKE schemes. NIST specifies PKE schemes in the standards for (i) RSA-based 2KE [SP800-56B-Rev2], and (ii) the 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 typical 2KE application, based on a KAS, KTS, or KEM, it is important avoid misuse of the low-level PKE primitives, which poses a security risk (e.g., a dangerous decryption oracle). Any actual proposal of a threshold scheme for replacing a party in a full-fledged PKE-based 2KE application needs to consider 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 the (informally called) "textbook" RSA-PKE, the KeyGen (see category N4.2) generates a public RSA modulus N (product of two secret primes), a public encryption key e, and a private decryption key d. In subcategory N2.1, the core encryption/decryption primitives are:

- **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 provide on its own 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 $m = c^d \text{modN}$, 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 to yield 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).

RSA-OAEP.Encrypt and RSASVE.Generate are probabilistic. Conversely, 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 13 lists those RSA-based 2KE schemes.

Table 13. 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	Both	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 (i.e., 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, which correspond to PQC security categories $\theta = \{1, 3, 5\}$, if their internal RBG has security strength $\kappa = \{128, 192, 256\}$.

A.2.2.1 Conventional K-PKE

- **K-PKE.Enc**($\mathsf{ek}_{\mathsf{PKE}}, m, r$): Deterministic algorithm, using a public encryption key $\mathsf{ek}_{\mathsf{PKE}}$ and a seed r to encrypt a plaintext m, and outputting a ciphertext c. It uses internal values $(y[i], e_1[i], e_2)$ that are obtained pseudorandomly from the input seed r.
- **K-PKE.Dec**($\mathsf{dk}_{\mathsf{PKE}}, c$): Deterministic algorithm that uses the private decryption key $\mathsf{dk}_{\mathsf{PKE}}$ 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 §10.6), a key observation is that Enc' and Enc are interchangeable with regard to 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 is based on primitives from the K-PKE scheme. Table 14 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 14. 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 pair-wise key-establishment. 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, due to requiring distributed computation of threshold-unfriendly PRFs. Teams can still explore the complexity of such implementations, and possibly propose threshold-friendlier alternatives, e.g., based on threshold-friendlier symmetric primitives (see category S3 in Section 11.3), or threshold-friendlier PKE-based KEM schemes (see category S2 in Section 11.2).

A.3 Category N3: NIST-Specified Symmetric Primitives

Category N3 is for submissions of threshold schemes for NIST-specified symmetric-key and hashing-related primitives (see submission requirements in Section 10.3).

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 party is distributed into multiple parties (i.e., a secret sharing of the key is distributed across the parties), who then compute 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, 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

NIST recently standardized an authenticated encryption with associated data (AEAD) scheme called Ascon-AEAD128 [SP800-232]. 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 uses a 256-bit key, from which the extra 128 bits are used to XOR-mask the input nonce, in order to retain 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]

Extendable output functions (XOF). The basic 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. 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]

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 different function (i.e., not 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 lengths of the input key and 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] 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

Category N4 is for distributed key-generation (DKG) protocols for (i.e., threshold schemes for the generation of) keys used by NIST-specified primitives (see submission requirements in Section 10.4). These protocols produce a secret-sharing of a key across multiple parties.

Conventional KeyGen. A KeyGen determines a secret key (sometimes called a 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 used 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 RBGs, which need to satisfy specific requirements [SP800-90A-R1; SP800-90B; SP800-90C]. 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:

1. **EdDSA.** The scheme uses a precursor private key d to pseudorandomly derive a private signing key s and a nonce-derivation key ν, as $(s, \nu) = \mathsf{Hash}(d)$. 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 (with regard to verification) can be produced without using the nonce-derivation key (see Appendix A.1.1).
2. **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).
3. **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 keys are generated, the side holding them in a secret-shared manner needs to use them (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

The ECC KeyGen subcategory N4.1 also includes the CDH and MQV primitives of NIST-specified ECC-based 2KE schemes [SP800-56A-Rev3], namely since: (i) the essential operations are ECC-based 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, B denote the other side, (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 $impsig_A = (d_{e,a} + avf(Q_{e,A}) \cdot d_{s,A}) \bmod n$, 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):

1. Generate a pair of random secret primes (p, q), and output their product N; and
2. 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 an 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, $\kappa \approx 224$ the result is $|N| = 10,752$.

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) for signing, and SP800-56B-Rev2 (§6.2–§6.3) for 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 paragraphs list some of the aforementioned requirements, though submissions of threshold schemes may judiciously depart from some of them (see Section 10.4.2).

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, as "probable" or "provable" primes, satisfying one row of Table 15. The two most significant bits of each prime may be arbitrarily set.

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 minimum required bit-length is roughly half of the minimum 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/private 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 a 32-byte (256-bit) random seed as input. The following descriptions ignore encoding aspects.

Table 15. 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 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 $(K, \mathbf{s_1}, \mathbf{s_2}, \mathbf{t_0})$, and (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 10.4.4.

A.4.5 Subcategory N4.5: Secret RBG

Conventional generation. Various primitives require a randomly generated secret bit-string or secret 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, and a salt for a KDM or KC in the scope of a 2KA. Usually, these KeyGens are required to follow NIST-approved RBG methods [SP800-90A-R1; SP800-90B; SP800-90C], to obtain either pseudorandomness or true randomness.

Appendix B Notes on FHE and ZKPoK in Class S

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: (i) create or improve community-based technical recommendations; (ii) promote development of reference material/specifications; (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 11.5) allows for the submission of FHE schemes. Informally speaking, an FHE scheme allows for arbitrary computations over encrypted data. Being an encryption scheme, in includes `KeyGen`, encryption (`Enc`) and decryption (`Dec`) algorithms Being fully homomorphic it also includes algorithms for the efficient homomorphic evaluation, over the ciphertext space ($\mathcal{C}$), of a "complete basis" of operations from the plaintext space ($\mathcal{P}$) (e.g., see Ref. [HES, §1.1.1]). The composition of these operations allows for arbitrary computations.

Traditionally, the complete basis over $\mathcal{P}$ is composed of addition and multiplication, such as $\{\mathtt{xor}, \mathtt{and}\}$ over GF(2), or $\{+, \times\}$ over a field modulo a large prime. More precisely (though for simplicity leaving the key implicit), suppose that 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 `hom` allows for corresponding efficient operations F and Op in the ciphertext space, as follows:

- If $c_1 = \mathtt{Enc}(p_1)$, $F = \mathtt{Hom}(f)$, and $d_1 = F(c_1)$, then $\mathtt{Dec}(d_1) = f(p_1)$;
- If $c_1 = \mathtt{Enc}(p_1)$, $c_2 = \mathtt{Enc}(p_2)$, $Op = \mathsf{Hom}(op)$, and $c_3 = Op(c_1, c_2)$, then $\mathtt{Dec}(c_3) = op(p_1, p_2)$.

Here, 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 a variety of aspects, such as:

- The type of operations (and plaintext space) for which 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 there is a client (A) holding a secret plaintext m, and a server (S) holding a secret key k. They interact in a way that allows the client to privately learn the output of the PRF evaluation over the plaintext, while both parties remain oblivious to the other party's input. By replacing the PRF with 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, typically 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 the 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 a 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 $\pi =$ ZKPoK.Prove$[k; (H, C)$: FHE.Hom[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 `true` $=^?$ ZKPoK.Verify$(\pi, (H, C),$ 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 type (i.e., with a public/private key pair), then the encryption key e might be simply learned by every party. The case of secret sharing the public key is also conceivable.
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 $C =$ 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 $H =$ FHE.Hom[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 11.6 and Appendix B.2)

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

Category S6 (see submission requirements in Section 11.6) allows for the submission of ZKPoKs, which are of great interest in the context of multi-party computation (and beyond).

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 16 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 16. 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)

From a ZKPoK to a signature. Since a non-interactive ZKPoK (NIZKPoK) allows for binding data (e.g., a message) to a proof, there is a tight relationship to signatures. 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. Also, a ZKPoK of an AES key that connects a plaintext to a ciphertext can be the basis of a signature scheme, e.g., as observed in some schemes submitted to the NIST-PQC process [Proj-PQC].

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 the *correctness* of the secret data (whose knowledge is being proven) and of 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 (i.e., a NIZKPoK consisting of a single message/transcript that is sent from the prover to the verifier). If it is succinct, then it is called a zk-SNARK (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.

Revealing/hiding of threshold setting. A ZKPoK related to a public *instance* produced by a threshold protocol (e.g., a signature, or an RSA modulus) or to a secret-shared private *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 that in a conventional setting. This category for ZKPoK submissions is open to both options (i.e., revealing and hiding the existence of a secret-sharing setting).

B.2.4 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 uses 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, 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 constraint 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 AES and SHA. Independently of the Threshold Call, the project may eventually propose a file format for representing 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.3; see Section 6.3) 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, RBGs). 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).

This subsection describes baseline assumptions and options about participants (Appendix C.1.1), communication (Appendix C.1.2), and the adversary (Appendix C.1.3). The setting of this baseline reference neither precludes submissions with sophisticated nuances, nor eliminates 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", 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) during or after the KeyGen phase.

The assumption of an 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), in which case the protocol may have an additional initial phase for setting up some agreement. However, that may be a distributed-systems problem outside of 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 Pre5 and CSX.4 for recommendations about 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 specifications that are free and publicly available. However, the reference implementation still needs to include code for them (see Imp2).

A baseline description ***may*** make strong assumptions about the communication network, including synchrony and transmission reliability. 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.5). Ideally, the security analysis (see CSX.5) 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 9.2.2, 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 9.2.1). Such proof ***may*** be given by showing "emulation" of the ideal functionality, or that a non-negligible adversarial advantage in each security game implies breaking an assumption. The security analysis, which ***may*** be based on assumptions that are 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 are not. For example:

- **Security with abort.** Even though availability is a generically desirable property, a security formulation with an 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 also requires 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.5 refers to the logical specification of the threshold scheme (CSX.3–CSX.4). Comments about implementation or deployment security are also welcome, including in CSX.7.

C.3 Threshold Profiles

For each primitive (see Sections 10 and 11) 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). The plural is used in "thresholds" since the thresholds can vary depending on which security property is being evaluated. Table 17 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.7) use-cases for which the proposed threshold ranges are adequate.

The following nomenclature is defined to conveniently 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 $4 \le n \le 8$; (M) "medium" for $9 \le n \le 64$; (L) "large" for $65 \le n \le 1024$; and (E) "enormous" for $n > 1024$.
- **Corruption proportion f/n:** (D) "dishonest majority" for $f \ge n/2$; (h) "honest majority" for $f < n/2$; (H) "two-thirds honest majority" $f < n/3$.

Table 17. 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) $4 \le n \le 8$	Medium (M) $9 \le n \le 64$	Large (L) $65 \le n \le 1024$	Enormous (E) $n \ge 1025$
$\ge 1/2$	Dishonest (D)	$n2$	$n3f$D	nSfD	nMfD	nLfD	nEfD
$> 1/3$	Honest (h)	—	$n3f$h	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 that are suitable for other useful and properly justified access structures. A submission ***may*** consider secret-sharing schemes and/or threshold schemes with well-specified monotonic access structures (i.e., where any superset of a quorum is also a quorum), including those that are different from a simple threshold.

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

Per §2.3 of NIST-IR8214A, there are various I/O interfaces of interest with regard to 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 (with regard to an identified input or output) are: not-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 with regard to the decrypted plaintext. This can be useful for pair-wise key-agreement (2KA) when one side (thresholdized) decapsulates the key-contribution sent by the other side.

- **SSI-encryption.** A threshold public-key encryption can be in SSI mode with regard to 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 with regard to the message. However, the needed threshold hashing then brings a significant overhead. If, instead, the message hash is secret-shared by a dealer (e.g., a client who is requesting the signature computation), then 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 proposal of threshold scheme that receives an SSI input (including a secret-shared key) ***may*** be based on an explicit assumption that the secret-shared input is correct. For example, this is the case if the input 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
- **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 (the suffix; not the standard deprecated in 2023)
- **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
- **FHE**: **F**ully-**H**omomorphic **E**ncryption
- **FIPS**: **F**ederal **I**nformation **P**rocessing **S**tandards
- **FORS**: **F**orest **o**f **R**andom **S**ubsets
- **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**amming **Q**uasi-**C**yclic
- **HSS**: Hierarchical Signature Scheme
- **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
- **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
- **NIST IR**: **NIST I**nternal **R**eport
- **NSS**: **N**ot-**S**ecret-**S**hared (Input/Output)
- **NTT**: **N**umber **T**heoretic **T**ransform
- **OAEP O**ptimal **A**symmetric **E**ncryption **P**adding
- **OTS**: **O**ne **T**ime **S**ignature
- **PDF**: **P**ortable **D**ocument **F**ormat
- **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
- **QV**: **Q**uantum-**V**ulnerable
- **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$^+$**: **W**internitz **O**ne-**T**ime **S**ignature **P**lus
- **XMSS**: e**X**tended **M**erkle **S**ignature **S**cheme
- **XMSS$^{\mathbf{MT}}$**: **M**ulti-**T**ree 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