NIST Internal Report
NIST IR 8214C ipd

NIST First Call for Multi-Party Threshold Schemes
(Initial Public Draft)

Luís T. A. N. Brandão
René Peralta

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

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

NIST Internal Report
NIST IR 8214C ipd

NIST First Call for Multi-Party Threshold Schemes
(Initial Public Draft)

Luís T. A. N. Brandão*
Strativia

René Peralta
Computer Security Division
Information Technology Laboratory

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

January 2023

U.S. Department of Commerce
Gina M. Raimondo, Secretary

National Institute of Standards and Technology
Laurie E. Locascio, NIST Director and Under Secretary of Commerce for Standards and Technology

Publication History

This version is the initial public draft (ipd).

How to cite this NIST Technical Series Publication

Luís T. A. N. Brandão, René Peralta (2023). NIST First Call for Multi-Party Threshold Schemes (Initial Public Draft). (National Institute of Standards and Technology, Gaithersburg, MD) NIST IR 8214C ipd.
https://doi.org/10.6028/NIST.IR.8214C.ipd

NIST Author ORCID identifiers

Luís T. A. N. Brandão: 0000-0002-4501-089X

René Peralta: 0000-0002-2318-7563

Contact Information

nistir-8214C-comments@nist.gov

Public Comment Period

January 25, 2023 – April 10, 2023

Submit Comments

Only via email: nistir-8214C-comments@nist.gov

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

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.

Abstract

This document calls for public submissions of multi-party threshold schemes, to support the National Institute of Standards and Technology (NIST) in developing future recommendations and guidelines. In a threshold scheme, an underlying key-based cryptographic primitive is executed while a private/secret key is or becomes secret-shared across various parties. Submissions in response to this call should include security characterization, technical description, open-source implementation, and performance evaluation. Submitted threshold schemes should produce outputs that are "interchangeable" with a key-based cryptographic primitive of interest. There are two **cat**egories of primitives for the submission of threshold schemes: Cat1, for selected NIST-specified primitives; and Cat2, for primitives not specified by NIST, but which are *friendlier* (more amenable to) to the threshold paradigm, have enhanced functional features, or/and are based on different cryptographic assumptions. The analysis of Cat1-submissions will help develop future recommendations and guidelines for threshold implementations of the corresponding NIST-specified primitives. The analysis of Cat2-submissions will help assess new interests on primitives not standardized by NIST.

Keywords

Cryptography; distributed systems; provable security; secure multi-party computation; standards; threshold cryptography; threshold schemes.

Preface

Please do not yet submit any threshold scheme.

The present **draft** is published for the purpose of obtaining public feedback. The final version of the "NIST First Call for Multi-Party Threshold Schemes" will consider received feedback about this document and will integrate other formal components. Please submit feedback comments to nistir-8214C-comments@nist.gov by April 10, 2023.

This document is intended for: technicians engaged in the development of recommendations for threshold schemes; cryptography experts interested in providing constructive technical feedback, or in collaborating in the development of open reference material; and all those, including from academia, industry, government and the public in general, interested in future recommendations about threshold schemes. Relevant preliminary context about this call can be found in the NIST-IR8214A (2020), the MPTC-Call2021a for feedback on criteria for threshold schemes (2021), and the NIST-IR8214B-ipd (2022).

Acknowledgments

The first author performed this work as a Foreign Guest Researcher (non-employee) at NIST, while under a contract with (employed by) Strativia. The authors thank their NIST colleagues Lily Chen, Michael Davidson, Dustin Moody, Ray Perlner, 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.

Call for Patent Claims

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

ITL may require from the patent holder, or a party authorized to make assurances on its behalf, in written or electronic form, either:

a) assurance in the form of a general disclaimer to the effect that such party does not hold and does not currently intend holding any essential patent claim(s); or

b) assurance that a license to such essential patent claim(s) will be made available to applicants desiring to utilize the license for the purpose of complying with the guidance or requirements in this ITL draft publication either:

   i) under reasonable terms and conditions that are demonstrably free of any unfair discrimination; or

   ii) without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

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

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

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

Table of Contents

List of Tables

1. Introduction

Over several decades, the **N**ational **I**nstitute of **S**tandards and **T**echnology (NIST) has standardized important key-based cryptographic schemes, in various **F**ederal **I**nformation **P**rocessing **S**tandards (FIPS) publications, and in **S**pecial **P**ublications in Computer Security (the SP800 series). For example, they provide specifications for digital signatures [FIPS-186-5-Draft], public-key encryption [SP800-56B-Rev2], pair-wise key-agreement (including key-derivation primitives) [SP800-56A-Rev3], and symmetric-key enciphering [FIPS-197].

In a traditional description or implementation of a key-based cryptographic primitive, the operation is performed by an individual party that has access to the private/secret key, when said key is created (in key-generation) or/and used as input (e.g., for signing, enciphering, or decryption) in the underlying basic primitives. In a corresponding conventional implementation, said party is a *single-point of failure* for confidentiality, integrity and availability.

Modern cryptography enables a multi-party implementation paradigm, based on developments in the fields of threshold cryptography, secure **m**ulti-**p**arty **c**omputation (MPC) and distributed systems. In a (multi-party) *threshold* scheme, multiple parties perform a distributed computation, emulating the operation of a key-based cryptographic algorithm, without combining the private/secret key in any single place, and ensuring security as long as the number of corrupted parties does not exceed a certain *threshold*. This enables decentralization of trust regarding the creation, storage and use of the private/secret keys. This threshold paradigm can be applied to NIST-specified primitives and beyond.

The development of recommendations and guidelines for threshold schemes, tapping into the domain of advanced cryptography, is an important step in addressing various challenges in cybersecurity and privacy. As part of such development, it is expected that the present "Call for Multi-Party Threshold Schemes" will motivate broad community engagement for a diverse set of submissions, followed by expert public scrutiny by stakeholders.

Recent context leading to the formulation of this call can be found in the Multi-Party Threshold Cryptography (MPTC) project webpage, the NIST-IR8214A (2020) with considerations toward criteria, the MPTC-Call2021a for feedback on criteria for multi-party threshold schemes (MPTS), the 2020 MPTS workshop webpage, and the NIST-IR8214B-ipd on threshold EdDSA/Schnorr signatures (2022). The present call has the following goals:

1. **[Reference material]** Create a basis of properly motivated, specified, implemented and analyzed threshold schemes, to support future recommendations and guidelines.
2. **[Threshold feasibility]** Assess the viability of threshold implementations of various primitives of interest, including of selected NIST-specified primitives.
3. **[Pertinence of other primitives]** In the threshold context, facilitate an initial assessment of the merits of other cryptographic primitives that may be mature for adoption.

4. **[Quantum resistance and other features]** Help explore the space of threshold readiness in terms of quantum-resistance versus other advanced functional features.

The process of collecting high-quality security formulations, technical descriptions, open implementations, and performance evaluations is intended to compose a body of reference material. This will support a phase of analysis to identify sound approaches, best practices, and reusable building blocks. The results will help shape recommendations and guidelines.

Two categories for submissions. To assess the viability of threshold schemes for cryptographic primitives, the present call is organized into two categories of submissions, with regard to the primitives in consideration for thresholdization:

- **Cat1:** Selected NIST-specified primitives used in digital signature schemes in FIPS-186-5-Draft, public-key encryption and respective decryption in SP800-56B-Rev2, elliptic-curve based pair-wise key-agreement in SP800-56A-Rev3, symmetric enciphering/deciphering in FIPS-197, key-derivation and key-confirmation mechanisms in the SP 800-56 series (parts A, B, and C); and the corresponding key-generations.
- **Cat2:** Primitives not specified by NIST, including primitives for "regular" schemes of type similar to those in Cat1 (signing, public-key encryption, key-agreement, enciphering/deciphering, key-derivation and key-confirmation, and their keygen), primitives for "advanced" functionalities (e.g., fully-homomorphic, identity-based or attribute-based encryption), zero-knowledge proofs/arguments of knowledge (e.g., of a secret-shared private key that is consistent with a public key); and other threshold-auxiliary gadgets. Primitives submitted in Cat2 should aim for threshold-friendliness and may be based on cryptographic assumptions different from those in Cat1. There is a particular interest in combined threshold-friendliness and quantum resistance.

The analysis in Cat1 will help assess threshold friendliness and develop future recommendations and guidelines for threshold schemes of NIST-specified primitives. The analysis in Cat2 will help assess new interests on primitives not currently standardized by NIST, and help characterize the possible alignment between (i) threshold-friendliness, (ii) quantum resistance, and (iii) additional useful features. This may also serve as relevant input to assess the ability to deploy secure multi-party applications with advanced privacy features.

Organization. Section 2 explains the acronyms used in the document. Section 3 calls for submissions and explains the partition into two categories. Section 4 enumerates logistic and formatting requirements for the submission of packages. Section 5 defines technical requirements for threshold schemes. Section 6 lists primitives and threshold modes of interest for each subcategory of Cat1 (NIST-specified primitives), mentioning possible I/O interfaces and recommending cryptographic parameters. Section 7 describes the subcategories of interest in Cat2 (primitives not specified by NIST). Appendix A provides further details about subcategories. Appendix B displays a checklist of the elements of a submission.

2. Acronyms

Acronym	Extended form
2KA	Pair-wise **k**ey-**a**greement
2KE	Pair-wise **k**ey-**e**stablishment
ABE	**A**ttribute-**b**ased **E**ncryption
AEAD	**A**uthenticated **e**ncryption with **a**ssociated **d**ata
AES	**A**dvanced **E**ncryption **S**tandard
API	**A**pplication **p**rogramming **i**nterface
CDH	**C**ofactor **D**iffie–**H**ellman
CMAC	**C**ipher-based **MAC**
CPU	**C**entral **p**rocessing **u**nit
CRS	**C**ommon **r**eference **s**tring
CRT	**C**hinese **r**emainder **t**heorem
DKG	**D**istributed **k**ey **g**eneration
DOI	**D**igital **o**bject **i**dentifier
ECC	**E**lliptic **c**urve **c**ryptography
ECDSA	**E**lliptic **C**urve **D**igital **S**ignature **A**lgorithm
EdDSA	**E**dwards **C**urve **D**igital **S**ignature **A**lgorithm
FFC	**F**inite **f**ield **c**ryptography
FHE	**F**ully-**h**omomorphic **e**ncryption
FIPS	**F**ederal **I**nformation **P**rocessing **S**tandards
FR	**F**ield **r**epresentation indicator
GB	**G**iga**b**yte (1,000,000,000 bytes)
GC	**G**arbled **c**ircuit
HMAC	**H**ash-based **MAC**
IBE	**I**dentity-**b**ased **e**ncryption
IETF	**I**nternet **E**ngineering **T**ask **F**orce
I/O	**I**nput/**o**utput
IRTF	**I**nternet **R**esearch **T**ask **F**orce
ITL	**I**nformation **T**echnology **L**aboratory

Acronym	**Extended form**
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
KT	**K**ey-**t**ransport
KMAC	**K**eccak-based **MAC**
LCM	**L**east **c**ommon **m**ultiplier
LTS	**L**ong **t**erm **s**upport
LWC	**L**ight**w**eight **C**ryptography
MAC	**M**essage **a**uthentication **c**ode
MPC	(Secure) **multi****p**arty **c**omputation
MPTC	**M**ulti-**P**arty **T**hreshold **C**ryptography
MPKA	**M**ulti**p**arty **k**ey **a**greement
MQV	**M**enezes-**Q**u-**V**anstone
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
NIZK	**N**on-**i**nteractive **z**ero-**k**nowledge
NISTIR	**NIST** **I**nternal **R**eport
NSS	**n**ot-**s**ecret-**s**hared (input/output)
OAEP	**O**ptimal **A**symmetric **E**ncryption **P**adding
PC	**P**ersonal **c**omputer
PDF	**P**ortable **d**ocument **f**ormat
PF	**P**lat**f**orm
PEC	**P**rivacy-**E**nhancing **C**ryptography
PQC	**P**ost-**Q**uantum **C**ryptography
PKC, PKCS	**P**ublic-**K**ey **C**ryptography, PKC **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

Acronym	**Extended form**
PSS	**P**robabilistic **s**ignature **s**cheme
PVSS	**P**ublicly **v**erifiable **s**ecret **s**haring
QR	**Q**uantum-**r**esistant or **q**uantum **r**esistance
RAM	**R**andom **a**ccess **m**emory
RBG	**R**andom-**b**it **g**enerator/**g**eneration
RFC	**R**equest **f**or **C**omments
RO	**R**andom **o**racle
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
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
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, SSIO	**S**ecret-**s**hared **i**nput, secret-**s**hared **i**nput-and-**o**utput
SSO	**S**ecret-**s**hared **o**utput
SVE	**S**ecret-**v**alue **e**ncapsulation
TB	**T**era**b**yte (1,000,000,000,000 bytes)
TF	**T**hreshold-**f**riendly
URL	**U**niform **r**esource **l**ocator
VSS	**V**erifiable **s**ecret **s**haring
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

3. Call and Scope for Submissions

This document is a **call** for multi-party threshold schemes. It solicits high-quality specifications of threshold schemes for primitives across two **cat**egories: Cat1 (selected NIST-specified primitives) and Cat2 (primitives not specified by NIST). Each submission should include a security characterization, a technical description, an open-source reference implementation, and a performance evaluation. Submitted schemes will benefit from exposure to public analysis, and will be considered in a future report. This is a preliminary phase for collection of reference material, and assessment of threshold schemes. The results of this phase will inform future development of recommendations, and may be considered in possible future efforts for development of guidelines or standards.

3.1. Category 1 (Cat1)

Cat1 consists of selected, stateless, NIST-specified cryptographic primitives, organized in Table 1 across five subcategories:

- C1.1, for EdDSA, ECDSA and RSADSA signing [FIPS-186-5-Draft];
- C1.2, for RSA encryption (for key-encapsulation) and decryption [SP800-56B-Rev2];
- C1.3, for ECC-based pair-wise **k**ey-**a**greement (2KA) [SP800-56A-Rev3] via CDH or MQV;
- C1.4, for AES-enciphering/deciphering [FIPS-197], and **k**ey-**d**erivation (KD) and **k**ey-**c**onfirmation (KC) for 2KE [SP800-56C-Rev2; SP800-135-Rev1; SP800-108-Rev1];
- C1.5, for ECC keygen [FIPS-186-5-Draft; SP800-56A-Rev3; SP800-186-Draft], RSA keygen [FIPS-186-5-Draft; SP800-56B-Rev2], and bitstring (or integer) keygen.

Table 1. Subcategories of interest in Cat1

Subcategory: Type	Families of specifications	Section in this call
C1.1: Signing	EdDSA sign, ECDSA sign, RSADSA sign	A.1
C1.2: PKE	RSA encryption, RSA decryption	A.2
C1.3: 2KA	ECC-CDH, ECC-MQV	A.3
C1.4: Symmetric	AES encipher/decipher, KDM/KC (to support 2KE)	A.4
C1.5: Keygen	ECC keygen, RSA keygen, bitstring keygen	A.5

Note: In the second column, each item within a subcategory is itself called a family of specifications, since it may include diverse primitives or modes/variants, some of which are mentioned in Table 4 (in Section 6).

Section 6 presents more details about versions and modes of primitives in Cat1, including options for input/output interfaces (Section 6.1) and cryptographic parameters recommended for evaluation (Section 6.2). The analysis of Cat1 submissions will facilitate the development of recommendations and guidelines on threshold schemes for the corresponding NIST-specified primitives, highlighting reference approaches, techniques, building blocks, and best practices. The results will be reported in a NISTpublication.

3.2. Category 2 (Cat2)

The goal of Cat2 is to enable submissions that make a strong case for certain threshold-feasible primitives that are not standardized by NIST. While the scope is wide, Cat2-submissions should be justified on the basis of the primitives being thresholdized having/enabling useful differentiating features, such as having/being: (i) **t**hreshold-**f**riendly(ier) (TF); (ii) based on alternative cryptographic assumptions (e.g., pairings), possibly **q**uantum-**r**esistant (QR) (e.g., lattice-based); (iii) useful probabilistic properties (e.g., determinism versus non-determinism), (iv) more efficient in a relevant metric, or/and (v) advanced functional features (e.g., allowing homomorphic computation over encrypted data).

Cat2 has eight subcategories, including five “regular” (somewhat matching the subcategories of Cat1), and three others (“advanced”, “ZKPoK” and “gadgets”), as listed in Table 2:

- **“Regular”:**
  - C2.1, for signing (e.g., verifiably-deterministic succinct signatures, and/or TF-QR);
  - C2.2, for PKE (e.g., TF-QR decryption and key-encryption);
  - C2.3, for key agreement (e.g., TF primitives that are QR and/or that facilitate low-round key-agreement for more than two parties);
  - C2.4, for symmetric-key primitives (e.g., TF enciphering/deciphering), and hashing-related primitives for key derivation and key confirmation;
  - C2.5, for keygen for primitives in other subcategories.
- **“Others”:**
  - C2.6, for primitives for cryptographic schemes with **advanced** functional features, e.g., **f**ully-**h**omomorphic, **i**dentity-**b**ased, and **a**ttribute-**b**ased **e**ncryption schemes.
  - C2.7, for **z**ero-**k**nowledge **p**roofs of **k**nowledge (ZKPoK) that are deemed useful to support the threshold setting, such as for proving knowledge of private/secret information consistent with a correct secret-sharing setup.
  - C2.8, for other auxiliary “gadgets” deemed useful to support the threshold setting, namely to support the implementation of other threshold schemes in scope.

Table 2. Examples of primitives in subcategories of Cat2

Subcategory: Type	Example scheme	Example primitive
C2.1: Signing	Succinct & verifiably-deterministic signatures	Signing
C2.2: PKE	TF-QR **p**ublic-**k**ey **e**ncryption (PKE)	Decryption/encryption
C2.3: KA	Low-round multi-party **k**ey-**a**greement (KA)	Single-party primitives
C2.4: Symmetric	TF-QR blockcipher/PRP	Encipher/decipher
|	TF-QR key-derivation / key-confirmation	PRF and hash function
C2.5:Keygen	Any of the above	Keygen
C2.6: Advanced	QR **f**ully-**h**omomorphic **e**ncryption	Decryption; Keygen
|	Identity-based and attribute-based encryption	Decryption; Keygens
C2.7: ZKPoK	ZKPoK of private key	ZKPoK.Generate
C2.8: Gadgets	Garbled circuit (GC)	GC.generate; GC.evaluate

Legend: PRF = **p**seudo**r**andom **f**unction [family]. PRP = **p**seudo**r**andom **p**ermutation [family]. QR = **q**uantum **r**esistant. TF = **t**hreshold-**f**riendly. ZKPoK = **z**ero **k**nowledge **p**roof **o**f **k**nowledge.

Section 7 contains more details and examples on Cat2. Some Cat2-submissions may be evaluated within the scope of the NIST Privacy-Enhancing Cryptography (PEC) project [Proj-PEC]. It is expected that the results of this exercise will be reported in a NIST publication.

3.3. Vision

Quantum-resistant versus quantum-breakable primitives. There is a strong interest in receiving submissions of threshold schemes for **t**hreshold-**f**riendly **q**uantum-**r**esistant (TF-QR) primitives. As there is currently a gap between some known useful cryptographic features and quantum-resistance, there is also interest in submissions that have enhanced functional features even if they are only secure with respect to non-quantum adversaries.

Interchangeability. This call is scoped on threshold schemes whose output can be used in subsequent operations (e.g., signature verification) that were specified to use the output of the corresponding conventional (non-threshold) primitive (e.g., signing). The intended notion is that of *interchangeability*, from §2.4 of NIST-IR8214A. EdDSA signing provides a notable example: the threshold setting favors a consideration not only of pseudorandom signatures, but also of probabilistic ones that are *interchangeable* in the sense of being verifiable by the standardized EdDSA verification (see NIST-IR8214B-ipd). In Cat1, the primitives of interest are already fixed. In Cat2-submissions, the primitives of interest need to be specified along with the corresponding threshold schemes.

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

Diversity. The domain space of multi-party threshold schemes is considerably wider than that of the primitives (e.g., digital signatures) being thresholdized. Acknowledging this, the present call allows leeway for the submitters to select from a variety of system models, threshold configurations, security formulations, technical approaches, and benchmarking focuses. Thus, the usual criteria for "apples-to-apples" comparison (e.g., number of parties, common programming language, application programming interface, etc.) will not be required in the initial phase. Nonetheless, the submissions are expected to adhere to certain criteria, with respect to both technical documentation (see Section 4) and technical characteristics of the proposed threshold schemes (e.g., needs to include a security formulation against active corruptions — see Section 5). After a review of the system models proposed in the initial set of submissions, a request may be made for submitters to provide new performance evaluation results (e.g., with a particular number of parties and threshold values) based on adjusted parameters to facilitate a comparison across submissions.

Initial phase. The initial phase of analysis is expected to take about one year after the submission deadline, and will consider comments from the public. It will also include a workshop for presentation of the submitted threshold schemes. A NIST report will follow. For Cat1, the results will help determine how the development of future recommendations and guidelines may be differentiated per primitive, and whether it will focus on full-fledged threshold schemes, on identifying building blocks and composition techniques, or a hybrid of these. For Cat2, the results will include an initial characterization of the space of submissions to help assess possible interest in a subsequent more-focused analysis.

Reliance on contributions. The success of the process will depend on:

- **high-quality submissions** by teams with appropriate expertise, including in the areas of secure multiparty computation and distributed systems;
- **expert public scrutiny**, including assessments of security;
- **comments on pertinence**, by stakeholders of applications of threshold schemes.

4. Components of a Submission

4.1. Phases Until Full Submission

The submission process is organized with a deadline for package submissions, while also considering a possible early abstract and preliminary submission, as follows:

Ph1. (Optional) Early abstract: No later than **about 90 days** (exact date to be determined) after the final version of this call is published, a short document (with no more than three pages) can be submitted with a title, a list of team members, and a preliminary abstract of a planned full package to be submitted later (Ph3). The abstract should identify the primitives to be thresholdized and their corresponding category and subcategory(ies)/type(s), give an outline of the threshold approach (including system model, the protocol approach, and main security properties), and list the most relevant bibliographic references. This phase for optional submission (not mandatory and non-committing) is intended to facilitate early discussion of the expected coverage of each category/subcategory, and may help determine useful merges, differentiations, or alternative submissions.

Ph2. (Optional) Preliminary package: Submission packages received by NIST at least **45 days before the deadline** for full packages will be early reviewed for completeness. The submitters will be notified of identified deficiencies, tentatively within 25 days, to allow amendments before the deadline.

Ph3. Full package: Full submission packages must be received by NIST no later than **about 150 days** (exact date to be determined) after the final version of this call is published. Despite possible adjustments to be made in this call, submitters are encouraged to prepare early for future submissions, using the present draft as a baseline. A complete and proper package must contain the following **m**ain components:

- **M1. Written specification:** A technical specification (including security analysis) of the threshold scheme and primitives (see Section 4.2).
- **M2. Reference implementation:** An open-source implementation (software), including code, license, comments, and explaining an API (see Section 4.3).
- **M3. Execution instructions:** Instructions to enable the execution of the threshold scheme and reproduction of experimental results (see Section 4.4).
- **M4. Experimental evaluation:** A report describing an experimental setting, measuring performance, and interpreting the results (see Section 4.5).
- **M5. Additional statements:** Various statements (see Section 4.6).

Submissions medium. The submission of any documentation — early abstract (Ph1), preliminary package (Ph2), full package (Ph3), or any amendment — must be at least confirmed by sending an email to MPTS-submissions@nist.gov. The final version of this call may specify a complementary platform to help manage the process of submission and review. More-specific instructions will be provided in the final version of this call.

Public posting. after the SUBMISSION deadlines, approved submissions of early abstracts (Ph1) and full packages (Ph3) will be posted online, and hyperlinked from the MPTC project website [Proj-MPTC], for public review.

Note on LaTeX templates. To facilitate some common document structure across submissions, the final version of the call will provide LaTeX-based templates applicable to some of the submission documents, for compilation into portable document format (PDF) files.

Note on multiple threshold schemes per package. A submission package may include a family of distinguished threshold schemes based on common building blocks, and whose implementations may make use of common portions of open-source code. Even if a submission package proposes more than one threshold scheme, each of the above-mentioned five components should appear only once, possibly using subsections (when applicable) to distinguish which primitives/schemes the comments relate to.

4.2. Main component M1: Written specification

Submitted specifications of threshold schemes must be compiled in a PDF document, written in English and aided with mathematical notation, containing various (numbered or unnumbered) sections, as described ahead across a frontmatter (see Section 4.2.1), a main matter (see Section 4.2.2), and backmatter (see Section 4.2.3).

4.2.1. Frontmatter

S1. Title pages: Two title-pages, as follows:

- A first title-page (cover page) with: a title for the proposed submission, the names and affiliations of the submitters; and the submission date.
- A second title-page, with all content of the first title-page, and additionally including: contact email-addresses for all the submitters; applicable disclaimers related to affiliations and funding; and, if applicable, other pertinent information about the team and the submission.

S2. Abstract: A text with up to 500 words, identifying the primitives being thresholdized, their corresponding category and subcategory/type in the scope of this call, and the types of threshold schemes being proposed (i.e., their main features, cryptographic assumptions and performance highlights).

S3. Executive summary: An abridged explanation (up to four pages) of the content of the submission, highlighting relevant properties of the proposed threshold schemes, their applicability, their performance, and some of the challenges (e.g., in proving security). It should also briefly mention the submitted components beyond the specification, including the open-source software with reference implementation.

S4. Index: A table of contents (i.e., index of sections, subsections, etc.); and (however applicable) lists of figures, tables, pseudo-code, and other relevant enumerated components. Each referenced element in the index should be hyperlinked to the respective position in the document, and also indicate the corresponding page number.

4.2.2. Main matter

S5. Clarification of prior work: An enumeration of the building blocks, techniques and ideas known to have been developed or authored in prior work and that are used in the specification of the primitives and threshold schemes of the present submission. With regard to the building blocks, techniques and ideas in the submission (preferably including hyper-references to the related portions of the submitted specification), this section should aim to clarify and distinguish between (i) those that may have been designed by authors that are not part of the submitters' team, (ii) those that may have been previously developed/authored by members of the submitters' team, and (iii) those that may be original in the present submission. Appropriate bibliographic references should be given where applicable, preferably including (when possible) a hyperlink to online-accessible documentation. If applicable, this section can also include known information pertinent to the "call for patent claims".

S6. Conventional primitives/scheme: A review of the conventional (non-threshold) primitives/scheme that constitute the objects of thresholdization and determine the interchangeability requirements. For example, if a submitted package proposes a threshold scheme for ECDSA signing, then this section will provide a brief review of the conventional ECDSA signing algorithm, and the requirements related to the corresponding keygen and verification algorithms. The notation used in this description should be consistent with the one later used to describe the threshold scheme. Cat2-submissions are expected to be more thorough in this description.

S7. System model: A thorough description of the system model, including participants, communication network, and adversary (see T2).

S8. Protocol description: A detailed description of the multi-party threshold scheme, modularizing the description of primitives/gadgets where appropriate.

S9. Security analysis: A detailed security analysis, including security formulation (e.g., ideal functionalities and/or games), proof(s) of security, and discussion of security properties and ideal components (see T3 and T4).

S10. Analytic complexity: An analytical estimation of (i) memory complexity, (ii) computational complexity, (ii) communication complexity, and (iii) round complexity. The estimates should: include a breakdown across the various possible phases of the protocol; clarify the complexity per party versus the aggregate in the entire system; clarify its dependence on various configurable parameters, such as for example the security strength, the number of parties and the thresholds.

S11. Choices and comparisons: A rationale for design decisions and the chosen system model, as well as an explanation of known advantages and limitations compared to other options and approaches.

S12. Technical criteria: An evaluation of various items of technical criteria (see Section 5 and Section B.7).

S13. Deployment recommendations: A set of deployment requirements and recommendations, including those related to security. This section should also include a list of known and proposed applications of the submitted threshold scheme(s).

4.2.3. Backmatter

S14. Notation: A section explaining the notation, including:

- a list of the used acronyms, and their extended expressions;
- a list of the used abbreviations, and their complete words;
- a list of the used mathematical symbols, and their brief explanations;
- (optional) a glossary of selected important terms, with succinct explanations.

S15. References: A list of external references cited throughout the document, ideally including persistent identifiers (e.g., DOI, and ia.cr) and a link to a corresponding publicly and (when possible) freely accessible version of the referenced document.

S16. Appendices: Auxiliary elements deemed too detailed or cumbersome for a first read may be deferred to appendices, at the end of the document, as long as properly referenced and hyperlinked in the corresponding above-mentioned sections.

4.3. Main component M2: Reference Implementation

Required clear implementation. The submissions packages must contain **open-source code** (software), including explanatory inline comments, constituting a "clear" reference implementation of the proposed threshold scheme(s). The code and comments should strive for clarity and understanding, even if at some detriment to efficiency. Optionally, some modules may include additional code optimized for some efficiency metric(s), to enable demonstration of better experimental performance.

The implementation(s) must support all main features of the threshold scheme and be suitable to run each "party" in a modern **p**ersonal **c**omputer (PC). To facilitate testing, the implementation should enable "running" the set of all parties in a *baseline* **pl**at**f**orm (PF1) consisting of a single PC (possibly virtualized), equipped with:

1. **Processor:** Central processing unit (CPU) with up to eight 64-bit processing cores.
2. **Fast primary memory**: Up to 32 gigabytes (e.g., of **r**andom-**a**ccess **m**emory [RAM])
3. **Secondary memory:** Up to 4 terabytes (e.g., in a **s**olid **s**tate **d**rive [SSD])

The code (and its instructions) should be designed to allow for a compilation and execution of the submitted implementation on top of a Linux Ubuntu Desktop 22.04.1 long-term support (LTS) operating system running installed in platform PF1, without requiring software download from external sources. Each party should be executed as one (or more) process(es), or within a software virtual container, separate from the other parties.

The submitted open-source software (and documentation) should satisfy the following:

Src1. Is self-contained: The code was tested to compile and execute properly within the baseline platform (PF1) with a Linux Ubuntu Desktop v22.04.1 operating system.

Src2. Is licensed as open-source: The code is explicitly licensed as open-source (e.g., possibly based on a license listed in https://opensource.org/licenses).

Src3. Contains inline comments: The code is explained with auxiliary comments.

Src4. Has a clear API: It explains the **a**pplication **p**rogramming **i**nterface (API), aimed at facilitating (i) testing, (ii) use in higher-level applications, and (iii) comparison of performance with other implementations that may follow the same API.

On programming choices. As explained in Section 3.3, it is intentional that this call does not specify a concrete programming language, compiler, or API to be used across submissions. That said, it would be useful that the provided open-source reference implementation comes accompanied with explained rationale for choices made. This may include recommendations on the API that future implementations should follow to be easily comparable with the provided reference implementation.

On validation and verification. The validation of implementations and formal verification are not included as technical requirements for this call. However, it is expected that the public scrutiny of submitted schemes (namely their specifications and implementations) will facilitate the production of high-assurance software. The analysis of the submissions may clarify what software testing may be proposed across various types of threshold schemes.

4.4. Main component M3: Execution Instructions

A submission package must include execution instructions, as follows:

1. **User manual:** A "user manual" with instructions (and examples) on:
   - **X1. Compilation:** How to compile the open-source code.
   - **X2. Parametrization:** How to configure execution parameters, such as the number of parties, the corruption threshold, the type of communication channels, some adversarial choices, and some client choices (e.g., input to the cryptographic primitive). Preferably the configuration of each parameter can be done via the editing of a human-readable text file, and/or command line arguments.
   - **X3. Execution:** How to test and execute the various phases of the proposed threshold schemes and underlying primitives.
   - **X4. KAT set:** A set of "known answer-test" (KAT) values, to aid in correctness verification of the execution of the protocol.
2. **Set of scripts:**
   - **X5. KAT-script:** A script to automatically execute the threshold schemes in a way that reproduces the set of KAT values (X4) provided in the user manual.
   - **X6. Benchmark-script:** A script to automatically benchmark the threshold scheme in platform PF1, using the "clear" reference implementation, to produce a table recording various performance measurements (similar to that required in Section 4.5) for various configurations. If the submitted implementation

includes additional code optimized for performance, and whose performance results are reported in M4, then corresponding scripts shoudl also be provided, to enable reproducibility of results.

X7. Other scripts (optional): Optionally, other scripts to provide better insights into the workings of the underlying primitives and threshold scheme.

4.5. Main component M4: Experimental evaluation

The package must include a report on experimental performance, obtained by executing the provided code in the baseline platform (PF1), evaluating a representative set of configurations supported by the proposed threshold scheme(s). The report must describe:

1. the experimental setting (see Section 4.5.1);
2. the measured performance (see Section 4.5.2); and
3. an analysis/interpretation of the results (see Section 4.5.3).

4.5.1. Experimental setting

The report must describe the expected performance characteristics of the experimental setting (namely of the underlying hardware) supporting the baseline implementation platform PF1. The description must describe at least the relevant expected characteristics of the (possibly emulated) processor (e.g., instruction set, and clock frequency), communication network (e.g., bandwidth, and latency), and memory (e.g., read and write speed).

The benchmarking can also include experimentation with different platforms (PF2, ...) of the submitter's choice (motivated by real or conceivable applications). The performance results obtained with these alternative platforms (to also be described) may be better or worst than with PF1. For example, if there are more than eight parties and all require intensive computing, then the testing in a platform with more than eight cores may provide better results than with the baseline PF1.

4.5.2. Measurements

The evaluation of experimental performance should report, at least for platform PF1, at least the following metrics:

- **Perf1. Memory complexity** (in # bytes required to be simultaneously stored).
- **Perf2. Processing time** (in seconds) and/or processing (e.g., # of processing cycles).

- **Perf3. Communication complexity** (in # communicated bytes).
- **Perf4. Networking time** (in seconds).
- **Perf5. Round complexity** (in # alternations of the direction of communicated messages).

The mentioned metrics should be evaluated and reported in (i) total per execution, (ii) per identifiable phase of the protocol, and (iii) per party. The results can be reported across various configurations, e.g., with distinct numbers of parties, and across two distinct security strengths (e.g., 128 and 224–256 bits).

The reported measurements should include results obtained with the submitted "clear" reference implementation (see Section 4.3). If the submission includes additional code optimized for performance, then the corresponding results can be added to the measurements' report. As prescribed in X7, all these benchmarking should be reproducible by a simple execution of the submission-required scripts.

4.5.3. Analysis

The performance analysis should include a written explanation/interpretations of the experimental results, indicating expected or unexpected observations (e.g., some observed correlation between some complexity metric and the number of parties). The comparison of results across different configurations and/or experimental settings may be useful to understand, test of verify tradeoffs and scalability of the system across different metrics.

4.6. Main component M5: Additional Statements

The packages must include certain statements (on intellectual property, agreements or disclosures) to ensure free worldwide availability of the submitted packages for public review and evaluation purposes, and allowing derivative work and use, in particular for the possible future elaboration and publication of recommendations, guidelines and standards. The concrete statements (to be included or referenced in the final version of this call) will be aligned with the NIST ITL Patent policy, and are likely to be similar to those used by the NIST Post-Quantum Cryptography (PQC) project [Proj-PQC].

5. Technical Requirements (T) for Submission of Threshold Schemes

In addition to the structural requirements for submission packages, the specification of threshold schemes is subject to certain technical requirements (T1–T6) at a logical level. The following are based on a previous call for feedback on criteria [MPTC-Call2021a].

5.1. T1: Primitives

A submitted specification must explain in S6 the conventional (non-threshold) primitives (e.g., decryption) that are the object of thresholdization. Each such primitive must be framed within the subcategories structure established for Cat1 (see Sections 3.1 and 6) and Cat2 (see Sections 3.2 and 7). The primitive must also be explained within the scope of an underlying conventional scheme, composed of various primitives. For example, a decryption primitive of a **p**ublic-**k**ey **e**ncryption (PKE) scheme relates to corresponding encryption and key-generation primitives. The explanation of the primitive must define the corresponding scope of *interchangeability*, to be considered by the proposed threshold scheme.

Notwithstanding the advantage of referenceability to NIST specifications, a submission in Cat1 still needs to include a technical description of the primitives being thresholdized. The description should try to follow the notation and and operations specified in the corresponding NIST documentation. Some Cat2-submissions may require a more thorough description, since their underlying non-threshold primitive is not part of a NIST specification. The explanation should also include references to authoritative descriptions in publicly free documentation (e.g., papers and standards).

5.2. T2: System Model

A proposal of threshold schemes must strive for a clear description that facilitates understanding various options across possible deployment scenarios. Therefore, the specification of each submitted threshold scheme must describe (in S7) one system model (and may identify possible variants), including the set of participants, the communication model and the adversarial model (goals and capabilities). In addition to the actual "parties" that hold the secret-shared keys, the system may include coordinators, administrators, clients and other devices (e.g., routers, clocks, random-bit generators), etc. The model must also explain how the parties are activated (e.g., via an authorized/authenticated client request, or by an administrator). See also §2.3 of NIST-IR8214A.

Some of the paragraphs ahead describe baseline assumptions and options for a system model, with regard to participants (Section 5.2.1), communication (Section 5.2.2), and adversary (Section 5.2.3). These assumptions are intended as a baseline, neither precluding submissions with sophisticated nuances, nor eliminating the utility of security evaluation across diverse deployment scenarios.

5.2.1. T2.1: Participants

The parties in a threshold entity. There is a "threshold entity" composed on n "parties", responsible for executing a cryptographic primitive. At the onset, all parties "know who" the n parties are, agreeing on n identifiers (e.g., possibly public keys to support authenticated channels). The suitability of public keys may need to be verified, locally or interactively, possibly via zero-knowledge proofs, in the keygen phase or in subsequent proposed phases.

It is conceivable that a threshold scheme is bootstrapped without prior agreement of who the n parties/identifiers are (or even what is value of n). However, said agreement problem may, in some system models, be a distributed-systems problem outside the scope of exploring the essential cryptographic thresholdization of the primitive at stake. Therefore, the assumption of initial agreement on n identifiers is a possibility, not a requirement. A submission that considers an additional preparatory phase for agreement of n and who the n parties are should try to present said phase modularly separated from the remaining threshold scheme.

Beneficiaries. For some operations, such as threshold keygen, the *beneficiaries* of the computation are the parties, who 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, such as threshold signing, the beneficiary can be an external client who requested the computation, to obtain an output.

Client interface. The client may or may not be aware of (and be able to interact distinctively based on) the n-party threshold composition. This can be affected by the input/output (I/O) interface (see §2.3 of NIST-IR8214A). For example, a secret-sharing of the I/O can affect whether or not a client can separately send/receive input/output shares to/from each party.

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

5.2.2. T2.2: Distributed Systems and Communication

As long as the interface and rules for composition are clear, the specification of a threshold scheme can (and is recommended to) decouple the description of (i) the building blocks (e.g., consensus, reliable broadcast) of classical distributed-systems, from (ii) the description of cryptographic operations needed to support the secure multiparty computation over (or of) a secret-shared key.

The specification of instantiations of building blocks that make use of weaker resources (e.g., enabling broadcast based on point-to-point channels) can be provided by referencing existing specifications, while evaluating the impact of those replacements. Then, the provided open-source implementation (see Section 4.3) of the overall threshold scheme can include (with proper attribution) open-source code from the referenced existing implementation of the applicable building blocks. The protocol can also be described with various phases (e.g., offline, online, secret resharing), which may have differentiated requirements.

A baseline description can make strong assumptions about the communication network, including synchrony and reliability of transmission. However, the proposal must discuss the pitfalls of deployment in environments with weaker guarantees (e.g., with asynchronous and unreliable channels), and possible mitigations.

Different threshold schemes may be better suited to different communication environments, with dependence on guarantees (or lack thereof) of **synchrony**, **broadcast**, and **reliability**. It is important to understand how security guarantees break across these environments.

5.2.3. T2.3: Adversary

The security analysis in S9 must consider a well-specified adversary, namely their goals and capabilities. In particular, the specification must consider an adversary that:

1. **[active]** is able to corrupt parties (up to one or various specified corruption thresholds), them controlling them to arbitrarily deviate from the prescribed multi-party protocol;
2. **[adaptive]** is able to decide which parties to corrupt after observing some of the protocol execution; and
3. **[mobile]** persistently continues (attempting to) corrupt parties across multiple executions of the main protocol, possibly corrupting parties after they have been recovered from a previous corruption.

The concrete ways in which the adversary performs corruptions may be related to other system-model options (e.g., communication network). In practice, some of the adversary's capabilities will be modeled as part of the idealization required in T3. The characterization of threshold security may vary across various ranges of acceptable corruption thresholds mentioned in item 1. Furthermore, the case of item 3 is intended to induce characterization of various levels of insecurity (e.g., which properties break and which ones do not) when acceptable thresholds are surpassed. The latter characterization may in particular be affected by the use of proactive recovery mechanisms (see Section T4.3).

5.3. T3: Security Idealization

As mentioned in Section 3.3, provable security is a fundamental component of how modern cryptography analyzes the security of proposed multi-party threshold schemes. Therefore, the present call includes a requirement to include a security idealization that supports a proof of security. Such idealization will encompass the security goals of the threshold scheme. That said, there are aspects of security analysis that overflow the scope of a proof/idealization and that should also be discussed.

A proposal of threshold scheme must be supported on a **simulation**-based and/or a **game**-based security formulation. This entails defining an ideal **functionality** (e.g., in the ideal-real simulation paradigm, within the universal composability framework) or/and an idealized adversarial **game** (or set of games). Since security analysis is a multi-dimensional exercise, it may include more than one form of idealization, and possibly even diverse proofs across different nuanced security properties or formulations.

A submission must include, in S9, a "security proof" that the proposed threshold scheme satisfies the proposed security formulation in a suitable adversarial context (see T4). Such proof can be given by showing "emulation" of the ideal functionality, or by showing that a non-negligible adversarial advantage in each security game implies breaking an assumption.

The security analysis must discuss which known useful properties are captured, and which ones are not, by the idealized security formulation. For example, even though availability is a desirable property, generically speaking, a security formulation with stronger emphasis on confidentiality and integrity may purposely specify that an adversary is allowed to abort protocol executions, so that the formulated security notion is achievable. As another example (now of an unsuitable formulation), a sole requirement of hiding and binding for a commitment scheme would not suffice for a use (e.g., committing bids in an auction) that would also require a non-malleability property.

In both cases (simulation and game-based), the security analysis should also discuss the security consequences of real implementation of idealized components. In particular, it must:

- identify the required cryptographic assumptions, and any possibly-idealized trusted components in the setup or operations;
- discuss the (in)security consequences of foreseen real instantiations of the setup and ideal components.

The "security analysis" (S9) asked in this call relates to the logical specification of the threshold scheme (S6–S8), and not to the submitted reference implementation (M2). Nonetheless, comments about implementation security are also welcome in the security analysis. Further details about implementation security can be included in S13.

5.4. T4: Security Versus Adversaries

The security analysis in S9 must consider a well-specified adversary (see T2.3), namely their goals and capabilities. In consideration of the modeled adversary (see T2.3), a proposed threshold scheme must aim for certain security goals, particularly with regard to how the adversary corrupts up to a corruption threshold number f of parties.

5.4.1. T4.1: Active Security (Against Active Corruptions)

Proposed threshold schemes **must** achieve **active security** (i.e., against active corruptions, which enable corrupted parties to "maliciously" deviate from the protocol), as opposed to *passive* only.

5.4.2. T4.2: Adaptive Security (Against Adaptive Corruptions)

There is a strong preference for considering threshold schemes that achieve **adaptive security** (i.e., security against adaptively chosen corruptions), as opposed to *static* only, with respect to critical safety properties (e.g., unforgeability [NIST-IR8214B-ipd, §5.2.3] and key-secrecy). Therefore, submitted schemes should also aim for security against adaptive corruptions for the major safety properties of interest.

Adaptive security 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. On the other extreme, a proposed protocol must not allow the major safety properties of interest 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 whose overall corruption (by an adaptive adversary) would be disastrous.

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) is expected to serve as reference material for public discussion. It is acceptable that certain security assurances (e.g., liveness and termination options) 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.

Practical feasibility is also needed. Feedback is welcome on security formulations and reference approaches that simultaneously enable both practical feasibility and security against adaptive corruptions, as well as possible acceptable tradeoffs.

5.4.3. T4.3: Proactive Security (Against Mobile Attacks)

The proposed threshold schemes schould be compatible with modular subprotocols / mechanisms for **proactive** (and reactive) recovery, which attempt to recover possibly corrupted parties back to an uncorrupted state. This is especially important to better handle a persistent **mobile** adversary that continuously attempts to corrupt more parties. With respect to refreshing secret shares, the solutions can be based on a modularized phase of secret-resharing (see T6), while also specifying the needed conditions (e.g., requirement of some initial/final agreement by a qualified quorum) for its integration.

5.5. T5: Threshold Profiles

For each primitive (to be identified in S6, within the scope established in Sections 6 and 7) considered for thresholdization, it may be useful to consider differentiated solutions across possible threshold parametrizations. Therefore, it is useful to consider a “threshold profile” that defines, for certain threshold-related parameters, which parametrization ranges are suitable for secure operation. The threshold profile should characterize at least the total number (n) of parties and the various thresholds (f) of corruption and (k) of participation. Table 3 proposes succinct labels for each default profile obtained from a restriction in the number of parties and the corruption threshold.

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

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

Table 3. Labels for some template threshold profiles

Corruption proportion		Number of parties (n)					
f/n	Majority type	Two (2): $n = 2$	Three (3): $n = 3$	Small (S): $4 \leq n \leq 8$	Medium (M): $9 \leq n \leq 64$	Large (L): $65 \leq n \leq 1024$	Enormous (E): $n \geq 1025$
$\geq 1/2$	Dishonest (D)	$n2$	$n3f$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

Note: 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 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.

A submission can focus on a single or on various threshold profiles. In particular, a protocol may be designed for *full threshold*, i.e., to ensure (for some range of number n of parties) some specific useful security notion regardless of the corruption threshold value f (with $f < n$) that it is instantiated with. In some of such cases it may be especially relevant to distinguish between corruption threshold and participation-minus-1 threshold. For each submitted threshold scheme, the system model (S7) and the security analysis (S9) must:

- characterize its proposed threshold profile(s), including discussing the diversity of thresholds associated with various security properties; and
- characterize the breakdown that occurs when threshold-profile assumptions are broken.

Note on alternatives access structures. Depending on which secret-sharing schemes support the distributed computation, it is possible to consider monotone access structures (i.e., where the superset of a valid quorum is also a quorum) different from a simple threshold. The use of the traditional term "threshold" in this call is not meant to suppress possible submissions for other useful and properly-justified access structures.

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 S13) use-cases for which the proposed threshold ranges are adequate.

5.6. T6: Building Blocks

A submission should identify and modularize the description of building blocks (gadgets) that can be securely replaced by other instantiations with similar interface. These may be useful across various threshold schemes across various submissions. While some future guidelines and recommendations documents may focus on gadgets, the decision to do so is likely to be subordinate to their utility for concrete threshold schemes.

Example building blocks. A notable building block is Shamir **secret sharing** (and Lagrange interpolation), either in the clear or homomorphically (e.g., "in the exponent"). Other secret sharing variants may also be useful, such as verifiable or publicly-verifiable secret-sharing. Other examples of gadgets include **garbled circuits**, **oblivious transfer**, **generation of correlated randomness**, **commitments**, **secret resharing** (possibly for new values f and n), **multiplicative-to-additive share conversion**, **additively homomorphic encryption**, MPC or ZKP friendly hashing, some **zero-knowledge proofs**, **consensus** and **broadcast**.

Modularized description. To the extent possible, proposals of threshold schemes should modularize the description of gadgets. This means that a high-level description of the threshold scheme uses references to the interface and security properties of the gadgets, but not necessarily to low-level details. A lower level description can then be made for one (or more) possible instantiation of each needed gadget.

Modularized code. The submitted open-source code (see Section 4.3) must include code for at least one instantiation of each used building block. If the proposed system model depends on special hardware components (e.g., a router) beyond the threshold "parties", the submission should also include code for emulating the special component.

The challenges faced in (i) implementing networking between parties can be significantly different from those in (ii) implementing certain mathematical operations (cryptographic building blocks) per party. Also, neglecting any of these can lead to serious vulnerabilities. Therefore, it is strongly encouraged that there is a strong alignment between the proposed system model (see T2 in Section 5.2) and the provided implementation (see Section 4.3), notwithstanding possible virtualizations to enable execution in a personal computer. For example, if a system model relies on broadcast, then the provided implementation should instantiate it in alignment with the assumptions of the proposed system model.

6. Cat1 primitives — Specified by NIST

Table 4 lists various Cat1 primitive-families of interest for thresholdization, organized in various "types" (subcategories): Signing (Section A.1); PKE (Section A.2); ECC-2KA (Section A.3); Symmetric (Section A.4); and Keygen (Section A.5). Within each type, each listed "primitive family" (itself identified with a more detailed subcategory index) may include several primitive variants (including ones not listed) and/or threshold modes, some of which are listed (non-exhaustively) in the third column of Table 4. A submission of threshold schemes fitting within a primitive family is not required to cover all indicated variants or modes, and may instead focus on a single one.

Table 4. Primitives of interest in subcategories of Cat1

Subcategory: Type	**(Sub)subcategory #: Family of primitives**	**Some [Primitives] and/or {Threshold Modes}**	**Section in this call**
C1.1: Signing	C1.1.1: EdDSA sign	[EdDSA, HashEdDSA] {Prob; Q-PR; F-PR (not FE); FE}	A.1.1
	C1.1.2: ECDSA sign	{Prob-FE; Q-PR; F-PR not-FE; PR-FE to Det-ECDSA)}	A.1.2
	C1.1.3: RSADSA sign	[RSASSA-PSS; RSASSA-PKCS-v1.5]	A.1.3
C1.2: PKE	C1.2.1: RSA encryption	[RSASVE.Generate, RSA-OAEP.Encrypt] {SSI}	A.2.1
	C1.2.2: RSA decryption	[RSASVE.Recover, RSA-OAEP.Decrypt] {NSS, SSO}	A.2.2
C1.3: ECC-2KA	C1.3.1: ECC-CDH	{NSS; SSO}	A.3.1
	C1.3.2: ECC-MQV	[Full; One-pass] {NSS; SSO}	A.3.2
C1.4: Symmetric	C1.4.1: AES (en/de)cipher	[encipher, decipher]	A.4.1
	C1.4.2: KDM/KC (for 2KE)	[Hash, CMAC, HMAC, KMAC]	A.4.2
C1.5: Keygen	C1.5.1: ECC keygen	[For ECC-signing and ECC-2KA]	A.5.1
	C1.5.2: RSA keygen	[Just the modulus (mod); mod & keypair]	A.5.2
	C1.5.3: Bitstring keygen	[RBG for AES keygen, RSA-SVE, and nonces] {SSO}	A.5.3

Legend: 2KE = pair-wise **k**ey-**e**stablishment. Det = **det**erministic . FE = **f**unctionally **e**quivalent. F-PR = **f**ully PR (i.e., deterministic even if the quorum changes). KD/KC = **k**ey **d**erivation and **k**ey **c**onfirmation mechanisms; NSS = input/output is **n**ot **s**ecret-**s**hared (i.e., apart from the key); PKE = **p**ublic-**k**ey **e**ncryption. PR = **p**seudo**r**andom. Prob = **prob**abilistic. RBG = **r**andom-**b**it **g**eneration. Q-PR = PR per **q**uorum. SSI/SSO = **s**ecret-**s**hared **i**nput/**o**utput (see §2.3 of NIST-IR8214A). SVE = **s**ecret-**v**alue **e**ncapsulation.

There are significant differences in threshold-friendliness and usefulness across the Cat1-primitives. For example, some symmetric-key primitives, such as HMAC and KMAC used for key-confirmation, are much less threshold-friendly than primitives based on public-key cryptography for signing and encryption/decryption. These differences are expected to affect the interest of stakeholders in submitting corresponding threshold schemes. Threshold-friendlier primitives can be considered in Cat2, as already conveyed in Table 2 in Section 3.2.

6.1. Input/Output (I/O) Interfaces

As discussed in §2.3 of NIST-IR8214A, threshold schemes can be considered in various modes with regard to the I/O interface. By default, a threshold keygen scheme produces a secret-shared output (SSO), i.e., a secret-shared secret/private key, and (when applicable) a corresponding not-secret-shared (NSS) public-key counterpart. Then, a subsequent threshold operation (e.g., signing) uses the private/secret key in a secret-shared input (SSI) manner. The mentioned secret-sharings (SSO and SSI) of the private/secret key are often left implicit. However, the secret-sharing of other input/output (that may itself be subject to confidentiality requirements) is relevant in some use cases, to hide said input/output from the threshold entity. Some of these SSI/SSO modes are explicit in Table 4. For example:

- a threshold decryption scheme can be in SSO mode to hide the decrypted plaintext;
- a threshold public-key encryption (exceptional case where there is no private key) can be in SSI mode to hide some secret key being encapsulated;
- a threshold CDH or MQV ECC key-agreement primitive may produce a SSO to hide the agreed key before it is subject to a final key-derivation (KD) transformation;
- a threshold signature scheme can be in SSI mode to hide the message being signed (not shown in Table 4).

A submitted specification of a threshold scheme must unequivocally identify which I/O parameters need to be in secret-shared form and which ones need not.

6.2. Cryptographic Parameters

Submitted threshold schemes should be implemented and evaluated with one set of parameters for security strength $\kappa \approx 128$, and another one for some security strength $\kappa \in \approx [224, 256]$). Table 5 lists recommended options for cryptographic parameters.

6.2.1. Elliptic Curves, for ECC-related Primitives

NIST-approved curves for elliptic-curve cryptography are specified in SP800-186-Draft. There are various representations and curves over prime fields, including

- Weierstrass: P-256, P-384, P-521, W-25519, W-448
- Montgomery: Curve25519, Curve448
- Twisted Edwards: Edwards25519, Edwards448, E448

Table 5. Recommended implementation parameters for Cat1 primitives

Parameter type	**Primitives using said parameters**	**For $\kappa \approx 128$**	**For $\kappa \gtrsim 224$**
Elliptic curve	EdDSA signing and keygen	Edwards25519	Edwards448
|	ECDSA signing and keygen	P-256	P-521
|	ECC CDH/MQVfor 2KA, and keygen	{Curve25519, P-256}	{Curve448, P-521}
RSA modulus size	RSADSA, RSA PKE, and their keygen	$\lvert N \rvert = 3,072$	$\lvert N \rvert \geq 11,264$ *
RSA enc./ver. key	RSA-related	$2^{16} < e < 2^{256}$	$2^{16} < e < 2^{256}$
Hash function	EdDSA signing	SHA-512	SHAKE256 (len 512, 912)
|	ECDSA/RSADSA; HMAC for KDM/KC	SHA-256, SHA3-256,	SHA-512, SHA3-512
|	|	SHA-512/256	
|	|	SHAKE128 (len 256)	SHAKE256 (len 512)
KMAC	for KDM and KC	KMAC128	KMAC256
Cipher	KC (for RSA or ECC), encipher/decipher	AES-128	AES-256
AES key-size	AES encipher/decipher/keygen/CMAC	$\lvert k \rvert = 128$	$\lvert k \rvert = 256$

Legend: κ = standardized "security strength" (in bits). enc./ver. = encryption/verification. len = length.

* The RSA modulus length $|N|$ must be a multiple of 8; this call further suggests that it be a multiple of 512. Approved hash functions or XOFs are specified in FIPS-180-4, FIPS-202, and SP800-185, but only a subset of them are suggested in this call. A XOF with predetermined length (len) can also be called a hash function.

A submission of threshold scheme for an ECC-based primitive should include an implementation based on at least one curve for security level for $\kappa \approx 128$, and another for $\kappa \gtrsim 224$, from the subsets detailed in Table 5. The curves W-x (for some x) and E448 do not appear in Table 5, as they are only intended for possible intermediate representations.

Note that SP800-186-Draft also specifies curves over binary fields (in short-Weierstrass form, namely Koblitz curves (K-163, K-233, K-283, K-409, K-571) and some pseudorandom curves (B-163, B-233, B-283, B-409, B-571). However, these are for legacy-only applications, and have been deprecated due to their limited adoption. Therefore, these are not recommended for submissions of threshold schemes.

Additive notation. In elliptic-curve cryptography, it is customary to use additive group notation. There, a public key Q can be determined by a repeated sum of the base-point G, a secret number d of times. The repeated-sum operation is (in additive notation) usually expressed as a multiplication by an integer. Thus, the private key d is the integer (not an elliptic curve element) needed to be multiplied with G to obtain $Q = d \cdot G$.

On the set of suggested curves for 2KA. SP800-56A-Rev3 (from 2018) considers (in its Table 24 in Appendix D) various curves for ECC key-agreement. Apart from Koblitz

(K-x) and pseudorandom (B-x) curves that have been deprecated by SP800-186-Draft, the Weierstrass curves (P-x) remian valid. From the latter, P-256 and P-521 cover the cases for security levels $\kappa \approx 128$ and $\kappa \gtrsim 224$. The recent SP800-186-Draft also specifies new Montgomery curves Curve25519 and Curve448, and references the IRTF RFC7748 where those curves are suggested for use in 2KA. Despite their current potential for adoption, the older SP800-56A-Rev3 does not include the new Montgomery curves (from the more recent SP800-186-Draft) in the list of approved curves for 2KA. Therefore, for Cat1-submissions of threshold schemes for ECC-2KA (subcategory C1.3): (i) the reference implementation should use at least the approved Weierstrass curves (P-256, P-521); (ii) a complementary suggestion is that Montgomery curves (Curve25519, Curve448) also be implemented to allow for a comparison across the uses of the two types of curves.

6.2.2. RSA Modulus, for RSA-related Primitives

A submission of threshold schemes for RSA-related primitives (for signing, key-encapsulation or decryption): should provide implementations with moduli of size $|N| = 3072$ for $\kappa \approx 128$, and $|N| \geq 11,264$ (or greater) for $\kappa \approx 224$ (or greater, respectively). Note: SP800-56B-Rev2 uses the symbol s, instead of κ, to denote the "security strength" (in bits).

The recommended RSA-modulus length $|N|$ for security parameter $\kappa \gtrsim 224$ was obtained, from exponential interpolation between the cases (specified in SP800-57-P1-R5) using $|N_1| = 7680$ for $\kappa_1 = 192$, and N_2=15,360 for $\kappa_2 = 256$, and rounding up to the nearest multiple of 512. The used formula is $|N| = 512 \cdot \lceil |N_1| \cdot (\kappa/\kappa_1)^a/512 \rceil$, where $a = \log_{(\kappa_2/\kappa_1)}(N_2/N_1)$. This is also the value that would be obtained by rounding up the result provided by the FIPS 140-2 implementation guidance [IG-FIPS-140-2, §7.5, page 125].

NIST-specified requirements for the prime factors of an RSA modulus, and their primality testing, are described in Appendices A.1 and C of FIPS-186-5-Draft, for single-party generation. For threshold schemes that warrant different methods (e.g., direct biprimality testing), a rationale must be presented to convey why the used test (including the number of rounds) is appropriate. In particular, it is acceptable that the RSA modulus be biased toward being a Blum integer, i.e., with both primes being 3 mod 4.

7. Cat2 Primitives — Not Specified by NIST

Cat2 allows for submissions of threshold schemes for primitives that are not specified by NIST. This category is aimed to allow for the consideration of primitives that are threshold-friendlier than those in Cat1, and/or that have distinctive features, such as being based on distinct cryptographic assumptions (possibly being quantum-resistant), or having advanced functional features. Section 3.2 already enumerated the subcategories and listed some examples (see Table 2). A submission in Cat2 must provide a thorough description of the corresponding conventional (non-threshold) scheme that the primitive (being thresholdized) is part of. For example: a submission of threshold scheme for a signing primitive not specified by NIST must include a description of not only the conventional signing primitive but also its corresponding verification and keygen primitives.

7.1. "Regular" Primitives (Subcategories C2.1–C2.5)

As already enumerated in Section 3.2 (including listed in Table 2), Cat2 covers five regular types of primitives across subcategories C2.1 (for signing), C2.2 (for PKE), C2.3 (for key-agreement), C2.4 (for symmetric-key and hashing primitives) and C2.5 (for keygen).

Since selected candidates from the NIST PQC and Lightweight Cryptography (LWC) projects [Proj-PQC; Proj-LWC] are not yet standardized, possible threshold schemes for their primitives can be presented in the scope of Cat2, specifically in their matching subcategories: C2.1 (signatures) and C2.2 (public-key encryption) for PQC; C2.4 (symmetric-key and hashing primitives) for LWC. However, the present call is also intended to elicit submissions for threshold schemes for primitives that are threshold-friendlier. Submissions of threshold schemes for quantum-resistant primitives should include a comparison with the security levels (1–5) defined by the NIST PQC project [Proj-PQC].

Subcategory C2.3, for single-party primitives for use in multi-party key-agreement, also expects possible submissions of TF-QR type. Such submissions should demonstrate the use of the thresholdized primitives in the scope of an actual key-agreement application. Compared to NIST-standardized KA protocols, submissions in this sub-category may enable improved KA schemes, justified based on different assumptions.

Note on PKE versus KA. Primitives within subcategory C2.2 for PKE can be used for multi-party key-establishment protocols, by allowing the confidential transmission of a contribution to a key. The subcategory C2.3 for KA (within Cat2) is intended for complementary primitives, such as those that may enable key-exchange protocols *a la*

Diffie-Hellman, though possibly based on different assumptions (e.g., to be QR) or for more than two parties. Therefore, the subcategory C2.3 for KA excludes the key-transport-only mechanisms (whose main cryptographic primitive is already scoped by PKE).

7.2. "Other" Primitives/Schemes (Subcategories C2.6–C2.8)

Beyond the "regular" type of primitives (covered by Cat1 and Cat2), there are "other" types of primitives covered by Cat2, namely "advanced" primitives (C2.6; see Sections 7.2.1 and A.6), "ZKPoKs" (C2.7; see Sections 7.2.2 and A.7) and "auxiliary gadgets" (C2.8; see Sections 7.2.3 and A.8). The subcategories for ZKPoK (C2.7) and gadgets (C2.8) are meant to allow for the submission of primitives that can support the threshold setting. Such a submission requires the specification of a conventional (non-threshold) primitive (see S6), but (in contrast with other subcategories) the specification of a threshold scheme is optional.

7.2.1. Cat2 subcategory C2.6: "Advanced"

Subcategory C2.6 (see more details in Section A.6) is suited for primitives with *advanced functional features* that are not covered by current NIST standards. For example, an encryption scheme may allow (i) homomorphically performing operations over encrypted data (possible with fully-homomorphic encryption), or (ii) selectively restricting the ability for decryption to designated sets of recipients (possible with identity-based and attribute-based encryption). A submission in subcategory C2.6 should present a strong rationale for the utility of the enhanced features, compared to what is possible with primitives in the other subcategories. Since quantum resistance is a strongly desirable feature, a submission without such a property is encouraged to specifically present rationale about the lack of good TF-QR alternatives.

7.2.2. Cat2 subcategory C2.7: ZKPoK

Subcategory C2.7 (see more details in Section A.7) allows for the submission of **z**ero-**k**nowledge **p**roof**s** **o**f **k**nowledge (ZKPoKs) that can support the threshold environment. For example, they may be useful to prove knowledge of a secret/private key or input that is consistent with:

- a public-key and/or with the public commitments of secret-shares;
- the output of a cryptographic operation (e.g., public-key encryption, AES enciphering, or KDM hashing), when the input was secret-shared and committed.

The generation of a ZKPoK can be considered both in conventional (non-threshold) and in threshold forms. For example:

- **[Conventional generation]** A dealer (single-party) of a **s**ecret-**s**haring (SS) can produce a ZKPoK that enables the various parties of a threshold entity (recipients of secret-shares) to non-interactively verify that the SS is adequate;
- **[Threshold generation]** The set of parties that interacted in a DKG to obtain a secret-sharing of a secret/private-key, and when applicable also obtain a corresponding public-key, can interact in an MPC to distributively generate a ZKPoK string that proves access to (i.e., knowledge of, albeit in a threshold manner and despite the secret-sharing aspect possibly remaining hidden from the proof) an adequate secret/private key consistent with a corresponding public commitment (possibly the public key) of the given threshold scheme.

  (Note that the latter example is dissociated from a conceivable proof of distributed generation of a key, which can be considered if tied to public keys of the intervening parties, believed to not reveal their private keys.)

The above two examples have similarities with, respectively, (i) verifiable secret sharing (VSS), which can also be extended to publicly verifiable secret-sharing (PVSS), and (ii) publicly verifiable MPC. Said verifiable features are welcome in submitted threshold schemes, and may (preferably) be included as part of a submission more focused on one of the other subcategories, while identifying the applicability of the ZKPoK to the present subcategory. A submission that simply focuses in subcategory C2.7 must specify at least a conventional ZKPoK, and may (optionally) specify a corresponding threshold version thereof.

7.2.3. Cat2 subcategory C2.8: Auxiliary Gadgets

Subcategory C2.8 (see more details in Section A.8) allows for the submission of specifications of other auxiliary primitives, here called *gadgets*. They may be auxiliary in their conventional (non-threshold) form and/or in a threshold form. Gadgets can be modularized in the submission of a higher-level threshold scheme associated with another subcategory within Cat1 or C2.1–C2.7. Such modularization is already recommended by criterion T6 (in Section 5.6) for various gadgets (e.g., those enumerated in §4.5.2 of NIST-IR8214B-ipd and §5.3.1 of NIST-IR8214A) whose underlying primitives (e.g., garbled-circuit generation, garbled circuit evaluation, commit, decommit) are not themselves thresholdized.

A. Details for Subcategories and Primitives of Interest

A.1. Subcategory C1.1: Cat1 Signing

The three Cat1-signing primitives of interest are from EdDSA, ECDSA, and RSADSA. Submissions in this subcategory should take in consideration the aspects of unforgeability and threshold security mentioned in NIST-IR8214B-ipd (while some aspects are specific to EdDSA, others are applicable to generic signature schemes). For example, it is useful to differentiate between regular unforgeability and strong unforgeability.

A.1.1. Subcategory C1.1.1: EdDSA Signing

EdDSA is specified in §7 of FIPS-186-5-Draft. The default signing mode is pseudorandom, determining the secret nonce r as a hash output whose pre-image includes a nonce-derivation key v. Ignoring some encoding details, the algorithm for EdDSA signing $\texttt{Sign}_n[s,v](M)$ of a message M outputs a signature $\sigma = (R,S)$, where $R = r \cdot G$, G is the conventioned base-point of the elliptic curve, $r = H(v,M)$, H represents a cryptographic hash function, $S = r + \chi \cdot s$, $\chi = H(R,Q,M)$ is the "challenge", and s is the private signing key (integer) needed to be multiplied with G to obtain the public-key Q.

A submission of threshold scheme for EdDSA signing: can choose to implement just one of or both HashEdDSA and EdDSA types (defining whether or not the message is "pre-hashed"); should provide implementations with curves Edwards25519 (for $\kappa \approx 128$) and Edwards448 (for $\kappa \approx 224$), which are specified in SP800-186-Draft; and must include only schemes that are interchangeable with regard to EdDSA verification (see related notes in NIST-IR8214B-ipd). With respect to nonce generation, submissions are expected to include one or more of the following modes:

1. **Probabilistic** (via a random or hybrid contribution per party)
2. **Pseudo-random per quorum** (via a ZKP of pseudorandom contribution per party)
3. **Pseudo-random** (based on a threshold-friendly PRF)
4. **Functionally equivalent to HashEdDSA** (via MPC hashing)

Note. An SSI mode for threshold signing is costly because it requires a distributed computation of a threshold-non-friendly hash of the message. However, if the regular NSS mode already requires such type of difficult computation (which is the case in functionally-equivalent EdDSA threshold signing), then the SSI mode may be achieved with a simple extension, using the gadgets already required for the NSS mode.

A.1.2. Subcategory C1.1.2: ECDSA Signing

ECDSA is specified in §6 of FIPS-186-5-Draft. The default signing mode is probabilistic (§6.3.1), but there is also a deterministic ECDSA mode (§6.3.2). Table 6 shows how the meanings of some symbols change significantly between EdDSA and ECDSA.

Table 6. Notation of EdDSA versus ECDSA (in Draft FIPS 186-5)

Element's role	**In EdDSA**	**In ECDSA**
Signature	(R, S)	(r, s)
Private† key	s	d
Secret nonce	r	k
[Final]‡ nonce commitment	R	r
Challenge	χ	e

† EdDSA also uses d, but for the precursor private-key from which the signing key s and another nonce-derivation key are obtained. ‡ The use of [final] is to convey that it is the actual value output in the signature. It is an encoding of other intermediate computed values that are themselves also commitments to the nonce. In particular, in ECDSA one of the intermediate values is denoted with symbol R.

Ignoring some encoding details, the algorithm for ECDSA signing $\mathtt{Sign}_n[d](M)$ of a message M outputs a signature $\sigma = (r, s)$, where d is the private signing key (the integer needed to be multiplied with the base-point G to obtain the public-key Q); the "challenge" $e = Encode_n^{(1)}(\text{Hash}(M))$ is an encoding (mod n) of the hash of the message being signed; $k \leftarrow^{\$} [1, \ldots, n-1]$ is (in the probabilistic version) a uniformly selected nonce that needs to remain secret; $R = k \bullet G$ is the "nonce commitment" and $r = Encode_n^{(2)}(R)$ is a corresponding encoding (mod n); and $s = k^{-1} \cdot (e + r \cdot d) \pmod{n}$.

A submitted threshold scheme for ECDSA signing should provide an implementation with at least one parametrization for $\kappa \approx 128$ and another for $\kappa \gtrsim 224$, with parameters recommended in Table 5. With respect to nonce generation, submissions are expected to include at least one of the following modes:

1. **Probabilistic** (via random or hybrid contributions per party)
2. **Pseudo-random per quorum** (via a ZKP of pseudorandom contribution per party)
3. **Pseudo-random** (based on a threshold-friendly PRF)
4. **Pseudo-random functionally equivalent to Deterministic ECDSA** (via MPC hashing)

Note on SSI-signing: In the case of SSI-signing for Deterministic ECDSA, the client can directly provide a secret-shared challenge (the hash e of the message), whereas in (Deterministic) EdDSA the pseudorandom challenge χ requires knowledge of a nonce commitment that depends on a private element not known by the client. Note that signature verification still requires the ability to hash the message.

A.1.3. Subcategory C1.1.3: RSADSA Signing

RSA signature modes are specified in §5.4 of FIPS-186-5-Draft, by reference to IETF RFC8017. A submission for the RSADSA signing family is expected to implement a threshold signature scheme that is interchangeable with at least one of the following modes:

1. RSASSA-PSS (**p**robabilistic **s**ignature **s**cheme), using an approved hash function or XOF
2. RSASSA-PKCS-v1.5 (deterministic), using an approved hash function

A.1.4. Signing in Secret-Shared-Input (SSI) Mode

In an SSI-signing mode, no single-party (nor any collusion up to a certain number of parties) of the threshold entity will learn the hash of the message. This is akin, though not the same as, what is achieved with blind signatures. The difference is that in the threshold setting it is possible that a large enough collusion of parties is able to reconstruct the input message.

The SSI mode may be of use, for example, for private-preserving time-stamping, producing a certificate interchangeable with those produced by the conventional protocol where the authority learns the hash of the document being timestamped.

The threshold-generation of signatures in SSI mode may pose challenges with regard to unforgeability. For example, a protocol must prevent that a malicious party that maliciously changes their secret-share would affect the overall message being signed, i.e., must prevent the signing of a message whose signature has bot been requested. Such challenges may be resolved based on various techniques, including zero-knowledge proofs, or based on verifiability or error correction properties of the secret-sharing. For example, each party can prove that their interaction in the distributed computation is consistent with a secret-share that has been certified by the client, with regard to the ongoing signing session.

A.2. Subcategory C1.2: Cat1 Public-Key Encryption (PKE)

The PKE cryptosystem of interest is RSA. The main use case considered for RSA encryption/decryption is pair-wise key-establishment (2KE), as specified in SP800-56B-Rev2. 2KE can take the form of a key-agreement (KA) type of protocol (with contributions from both parties) or be more simply based on key-transport (KT) type of protocol (with contribution from a single party). For RSA-based instantiations, both types of protocol rely on secret-value encapsulation (SVE), where RSA encryption is used to encapsulate a secret value

k (also denoted as a plaintext m) into a ciphertext c, which is then sent to another party for decryption. Ignoring some encoding details, the low-level RSA-based cryptographic primitives of interest are:

- **RSA encryption primitive (RSAEP):** Encryption $c = m^e \bmod N$ (transforming a plaintext m into a ciphertext c). A threshold version of it uses a secret-shared input m (SSI) and a not-secret-shared public encryption key.
- **RSA decryption primitive (RSADP):** Decryption $m = c^d \bmod N$. A threshold version of it uses a secret-shared private-key d (which is never reconstructed); the threshold operation produces an output that is either secret-shared (SSO) or not (NSS).

Additional relevant primitives include:

- Generation of an RSA modulus and/or key-pair (see Section A.5.2).
- Generation of a random bit-string (see Section A.5.3).

The values generated in SSO mode are for subsequent consumption in SSI mode.

A.2.1. Subcategory C1.2.1: RSA Encryption (of a Secret-Value)

Threshold schemes in this call are intended to operate over secret-shared material. Therefore, in the case of public-key encryption the secret-sharing does not usually apply to the public key. However, the application of key-encapsulation for key-transport/agreement uses the plaintext itself (being encrypted) as a value whose confidentiality requirement may warrant threshold protection. By default, a threshold scheme for such encryption will be in "secret-shared input" (SSI) mode (see [NIST-IR8214A]) with regard to the value being encrypted, but will not secret-share the public key (to be known by every party).

The basic **RSA** **e**ncryption **p**rimitive (RSAEP) computes a ciphertext $c = m^e \pmod{N}$, where m is a secret plaintext, e is the public encryption key, and N is the public modulus. The goal is to compute c from a secret sharing $[m]$ of m. For interchangeability with regard to a subsequent decryption, an actual full-fledged threshold scheme for RSA key encapsulation should consider all of the appropriate encoding and padding details. In SP800-56B-Rev2, the primitive RSAEP (§7.1.1) is specified for use within two higher-level primitives:

1. RSASVE.Generate (§7.2.1.2): RSA for **S**ecret-**V**alue **E**ncapsulation (which also includes the generation of the random key to encapsulate)
2. RSA-OAEP.Encrypt (§7.2.2.3): RSA with **O**ptimal **A**symmetric **E**ncryption **P**adding

A.2.2. Subcategory C1.2.2: RSA Decryption

SP800-56B-Rev2 specifies the use of RSA decryption in two higher-level primitives:

1. RSASVE.Recover (§7.2.1.3): Secret-Value Encapsulation recovery
2. RSA-OAEP.Decrypt (§7.2.2.4): Optimal Asymmetric Encryption Padding decryption

The RSA decryption primitive, RSADP(privKey, c), used to decrypt a ciphertext c, accepts the private decryption key privKey [SP800-56B-Rev2, §6.2.2] in three possible formats:

1. Basic format: (n, d)
2. Prime-factor format: (p, q, d)
3. Chinese-remainder theorem (CRT) format: $(n, e, d, p, q, dP, dQ, qInv)$

The notation [SP800-56B-Rev2, §3.2] is as follows: n is the public modulus; (p, q) is the pair of secret prime factors of n; d is the private decryption key; e is the public encryption key; dP is $d \bmod (p-1)$; dQ is $d \bmod (q-1)$; and $qInv$ is the inverse of $q \bmod p$.

A.2.3. Implementation Recommendations and Options

A submitted threshold scheme for RSA encryption or decryption primitives should include an implementation in the scope of an RSA-based 2KE protocol, as follows:

- With an instantiation for $\kappa \approx 128$ and another for $\kappa \gtrsim 224$ (see Table 5).
- Showcasing at least one of the key-establishment protocols listed in Table 7, with at least one of the parties (U, or V) being threshold-decentralized;
- If implementing threshold RSADP:
  - secret-sharing the decryption key, for at least one of the three approved formats (Section A.2.2); the public elements (n and e) do not need to be secret shared;
  - outputting the plaintext (the key that was encapsulated) in one of two forms: secret-shared, or not secret-shared.
- If implementing threshold RSAEP: using an SSI mode for the plaintext.

The various RSA-2KE schemes. SP800-56B-Rev2 specifies various RSA-2KE schemes. Two are of the *key agreement* (KA) type (obtaining contributions from both parties), whereas another one is based on *key transport* (KT) using a contribution from a single party. Table 7 lists, across these three schemes, the corresponding RSA-based operations (excluding needed RSA key-pair generation). Each of the listed schemes allows for a basic version, and a version with key confirmation (unilateral or bilateral, not based on RSA). The KDM operation specified for KA schemes is not RSA based.

Table 7. RSA-based primitives per party per RSA-2KE scheme

Type	Scheme	§ in SP 800 -56B-Rev2	Party	RSA-based primitive	KDM needed?
KA	KTS1	§8.2	1st contributor (U)	RSASVE.Generate	Yes
|	|	|	2nd contributor (V)	RSASVE.Recover	|
|	KTS2	§8.3	Any	RSASVE.{Generate & Recover}	|
KT	KTS-OAEP	§9.2	Sender (U)	RSA-OAEP.Encrypt	No
|	|	|	Receiver (V)	RSA-OAEP.Decrypt	|

In KTS1, one party (U) uses RSASVE.Generate to generate and encrypt a secret value Z, and the other party (V) uses RSASVE.Recover to decrypt Z. The latter party then contributes a non-encrypted nonce N_V. (Per §5.4 of SP800-56B-Rev2, the nonce used in KTS1 should be random.) Both the secret value and the nonce are then used as input to a KDM, which produces a final agreed key k (not to be confused with the nonce k of ECDSA). In KTS2, the clear-text nonce from party V is replaced with an encapsulated key, therefore requiring both parties to implement both RSASVE.Generate and RSASVE.Recover. Both KTS1 and KTS2 include a subsequent KDM, either in a one-step version or a two-step version, which transforms the pair of contributions (Z and N_V) into a final derived key k. A threshold keygen can consider the generation of Z and/or N_V in SSO mode Section A.5.3, if they are to then be consumed in SSI mode by the subsequent KDM.

The KTS-OAEP scheme does not use a KDM. Instead, the output key is decided by one of the parties, who then sends it encrypted to the other party. The threshold modes of interest for KTS-OAEP depend on the primitive, as follows:

- RSA-OAEP.Encrypt with the plaintext (a key to be encapsulated) in SSI mode.
- RSA-OAEP.Decrypt with the plaintext (the key that was encapsulated) in SSO mode.

Each 2KE scheme can be implemented in either a basic form (without key confirmation), or with KC in either a unilateral or bilateral manner. Both KDM and KC primitives rely on hash-functions of symmetric-key cryptography (see Section A.4.2).

SP800-56B-Rev2 also specifies that any of the mentioned RSA-2KE schemes (KTS1, KTS2, and KTS-OAEP) can be followed by a key transport where the established key is wrapped with an approved (symmetric-key based) key-wrapping algorithm [SP800-38F]. However, threshold-wise said key-wrapping algorithms are more-unfriendly than KTS-OAEP.

On the ability to bias the key in a 2KE protocol. The various mentioned NIST-specified protocols allow one of the parties to significantly bias the result. Specifically, the second contributor party in the KTS1 and KTS2 protocols can brute-force its contribution to bias several bits (e.g., 40 bits, at a parallelizable computational cost of approximately 2^{40} KDM operations). In KTS-OAEP the sender fully determines the key being transported. This is is contrast with Blum-style coin-flipping protocols, where the contribution from each party is only revealed once the contribution from the other party is committed to, thus implying that an honest party can guarantee that the output is not biased (up to abort by the other party).

A.3. Subcategory C1.3: Cat1 ECC Primitives for Pair-Wise Key-Agreement (2KA)

Pair-wise key-agreement (2KA). SP800-56A-Rev3 specifies various pair-wise (i.e., two-party) key-establishment (2KE) schemes of the KA-type (where the final key depends on contributions from the two parties), based on discrete logarithm cryptography. In a 2KA scheme, each party uses their own private key(s) and the public key(s) from the other party, to first obtain an intermediate common secret Z, and then applies a transformation to obtain a final key (called *DerivedKeyingMaterial*) k that is equal to the one obtained by the other party (not to be confused with the nonce k of ECDSA).

In some NIST publications the intermediate secret Z is referred to as a "shared" secret, meaning it is known by both parties of the 2KA. This should not be confused with the case of a "secret-shared" Z when "thresholdizing" (i.e., decentralizing) one of the original parties.

Each 2KA protocol specified in SP800-56A-Rev3 can be described with up to three phases:

1. **A public-key cryptography (PKC) phase**, where the parties interact to determine an intermediate common secret Z.
2. **An asymmetric-key cryptography phase**, where each individual party uses a ***key-derivation mechanism*** (KDM) to derive a final key k.
3. **An optional *key confirmation* (KC) phase**, based on comparison of **m**essage **a**uthentication **c**ode (MAC) tags, which allows at least one of the parties to confirm that their obtained key is equal to the key of the other party.

The subcategory C1.3 (2KA) of Cat1 in this call is only focused on the PKC primitives used in the initial phase, namely the Cofactor Diffie-Hellman (CDH) or Menezes-Qu-Vanstone (MQV) primitives. However, a submission of a threshold scheme for such a primitive should be demonstrated in an implementation of a full-fledged 2KA protocol. Therefore, this section also provides some context about the KDM and (the optional) KC operations, whose possible thresholdization is considered in Section A.4.2.

ECC scope. From the schemes in SP800-56A-Rev3, Cat1 only includes those based on ECC, which are implementable with elliptic curves specified in SP800-186-Draft. Table 5 in Section 6.2 lists the curves of interest. 2KA based on finite field cryptography (FFC) is left out of scope, following the trend of deprecating FFC in favor of more succinct ECC, as done in FIPS-186-5-Draft (which deprecated DSA in favor of ECDSA). The seven 2KA schemes in scope are listed in Table 8 and can be classified based on three factors:

- the underlying ECC primitive: CDH or MQV.
- the number of ephemeral (e) keys (2, 1 or 0),
- the number of static (s) keys (2, 1 or 0); and

Table 8. Seven ECC-2KA schemes

Primitive (f)	e	s	**Scheme**	Intermediate secret Z (“agreed” by U and V)	**§ in SP 800-56A-Rev3**
ECC **CDH**	2	2	(Cofactor) Full Unified Model	$f(e_U, E_V) \| \| f(s_U, S_V)$	§6.1.1.2
|	2	0	(Cofactor) Ephemeral Unified model	$f(e_U, E_V)$	§6.1.2.2
|	1	2	(Cofactor) One-Pass Unified Model	$f(e_U, E_V) \| \| f(e_U, S_V)$	§6.2.1.2
|	1	1	(Cofactor) One-Pass Diffie-Hellman	$f(e_U, S_V)$	§6.2.2.2
|	0	2	(Cofactor) Static Unified Model	$f(s_U, S_V)$	§6.3.2
ECC **MQV**	2	2	Full MQV	$f(s_U, S_V, e_U, E_U, E_V)$	§6.1.1.4
|	1	2	One-Pass MQV	$f(s_U, S_V, e_U, E_U, S_V)$	§6.2.1.4

Legend: $\|$ = concatenation. § = section in another document. e = number of generated *ephemeral* key pairs. f = symbol representing the ECC primitive (CDH or MQV). s = number of generated *static* key pairs; U and V = the two parties in the 2KA protocol. Let A represent one of the parties (U or V). **Abbreviated notation for keys:** e_A $(= d_{e,A})$ and E_A $(= Q_{e,A})$ are the *ephemeral* private and public keys of party A; s_A $(= d_{s,A})$ and S_A $(= Q_{s,A})$ are the *static* private and public keys of party A. The primitive f makes use of additional parameters not shown here.

Interchangeability scope. Regardless of the decentralization of any party, a 2KA scheme is already a protocol between two parties that intend to obtain a commonly agreed secret. Therefore, when considering a threshold scheme for a Cat1-primitive of a 2KA protocol, the interchangeability requirement is narrowed to “functional equivalence”. This ensures that the output secret (albeit possibly in secret-shared format) on one decentralized side will be equal to the one obtained by the other (possibly legacy) party in the 2KA interaction. Cat2 (see Section 7) allows for interchangeability in a broader sense, assuming that both parties interacting in the 2KA can agree on the new subsequent (KD/KC) mechanisms.

Single-party primitives. The objects of thresholdization are the primitives (see Table 9) computed by each individual party in the 2KA protocol. Each of these primitives has private/secret key-material in the input or/and output. The threshold protection provided to the keys handled by one side of the ECC-2KA depends on which primitives are thresholdized.

Table 9. ECC-2KA primitives of interest for thresholdization

Primitive	Secret input?	Secret ouptut?	Threshold friendly?	Section in SP800-56A-Rev3	Section in this call
ECC keygen: get key-pair (d, Q)	—	Yes	**Yes**	§5.6.1.2	A.5.1
ECC CDH/MQV: $Z = f(d_A, Q_B, \ldots)$	Yes	Yes	**Yes**	§5.7	A.3.1/2
Key derivation: $k = \mathrm{KDM}(Z, \ldots)$	Yes	Yes	**No**	§5.8	A.4.2
Key confirmation: $\mathrm{KC}(Z, \ldots)$	Yes	—	**No**	§5.9	A.4.2

Legend: d = private key. f = CDH or MQV transformation (primitive). k = final secret established by both parties. KC = "**k**ey **c**onfirmation" pseudorandom function, to allow comparison between A and B. KDM = "**k**ey **d**erivation **m**echanism" function. Q = public key. Z = intermediate secret (before KDM) computed by both parties.

A threshold scheme for an ECC CDH/MQV primitive allows for confidentiality of the private key d. This can be useful even if the intermediate secret Z is reconstructed due to a subsequent non-thresholdized KDM. Conversely, in a full-fledged thresholdization of the sequence of 2KA primitives, the output Z of the ECC CDH/MQV primitive would be secret-shared (i.e., SSO mode), to serve as input to the subsequent threshold KDM phase.

The ECC-2KA"type" includes only the ECC primitives that produce the intermediate secret Z, from secret-shared ECC private keys (static or ephemeral). There are two such primitives: ECC-CDH (Section A.3.1) and ECC-MQV (Section A.3.2). The ECC key-gen and KDM/KC primitives are respectively considered in Sections A.5.1 and A.4.2.

Submissions. A submitted threshold scheme for an ECC CDH or MQV primitive should:

- Evaluate it for at least one curve for $\kappa \approx 128$, and another for $\kappa \in \approx[224, 256]$ — see Table 5 in Section 6.2.
- Showcase the execution of at least one of the seven 2KA ECC-based schemes (see Table 8), with at least one decentralized party (A, B, or both) using secret-shared private keys in the threshold ECC CDH/MQV computation. The implementation should also include the KDM (and optionally the) KC procedures, either threshold (see

Section A.4.2, if the threshold ECC CDH/MQV is in SSO mode) or non-threshold. In other words, the ECC CDH/MQV output may or not be secret-shared, depending on whether or not the subsequent KDM/KC primitive is thresholdized.

A.3.1. Subcategory C1.3.1: ECC-CDH Primitive

With a decentralized party A (which can be U or V), the ECC-CDH primitive is as follows:

- **Secret-shared input:**
  - $[d_A]$ (secret sharing of private key of party A)
- **Public input:** (known to every party of the decentralized entity representing A)
  - Q_B (the public key of party B);
- **Secret-shared output:** Secret sharing $[Z]$ of a secret $Z = Encode(P)$, where:
  - $P = (h \cdot d_A) \cdot Q_B$ (where h is the cofactor)
  - $Encode$ is an encoding that does a field-element-to-byte string conversion of the x-coordinate of the input.

The output is distributively computed in a way that Z remains threshold confidential.

A.3.2. Subcategory C1.3.2: ECC-MQV Primitive

With a decentralized party A (which can be U or V), the ECC-MQV primitive is as follows:

- **Secret-shared input:**
  - $[d_{s,A}]$, $[d_{e,A}]$ (secret sharings of the static and ephemeral private keys of party A)
- **Public input:** (known to every party of the decentralized entity representing A)
  - $Q_{e,A}$ (the ephemeral public key of party A);
  - $Q_{s,B}$ and $Q_{e,B}$ (the static and ephemeral public keys of party B)
- **Secret-shared output:** Secret sharing $[Z]$ of a secret $Z = Encode(P)$, where:
  - $P = h \cdot impsig_A \cdot (avf(Q_{e,B}) \cdot Q_{S,b})$;
  - $impsig_A = (d_{e,a} + avf(Q_{e,A}) \cdot d_{s,A}) \bmod n$;
  - $avf(Q)$ is an integer associated to a public key Q, computed via an "Associate Value Function" ([SP800-56A-Rev3, §5.7.2.2]);

– *Encode* is the same encoding as defined for ECC CDH.

There are two possible implementation forms for the ECC MQV primitive:

1. The **full form** ([SP800-56A-Rev3, §5.7.2.3.1]), implemented as described above, where both static and ephemeral keys exist and are distinct.
2. The **one-pass form** ([SP800-56A-Rev3, §5.7.2.3.2]), where exactly one other party (A or B) does not have an ephemeral key, and so the above algorithm uses instead the corresponding static key:
   - If party A does not have an ephemeral key, then $d_{e,A}$ and $Q_{e,A}$ are respectively instantiated by $d_{s,A}$ and $Q_{s,A}$.
   - If party B does not have an ephemeral key, then $Q_{e,B}$ is instantiated by $Q_{s,B}$.

A.4. Subcategory C1.4: Cat1 "Symmetric"

The "symmetric" subcategory includes primitives for the NIST-approved symmetric-key enciphering scheme (the advanced encryption standard [AES]), as well as for other NIST-approved primitives used for KDM/KC. Some primitives in scope (e.g., hashing) are technically defined as keyless, but in practice they can be considered in settings (e.g., for KDM/KC) where their "plaintext" input is a key (symmetrically) known by two parties.

While "symmetric" primitives are often used in standardized "modes of operation" for large inputs, the thresholdization focus of this call is on the basic primitives, where the complexity of specifying a threshold scheme lies. For example, once a threshold scheme for AES enciphering/deciphering is defined, then it is straightforward to apply it to some mode of operation based on AES, including for the purpose of computing a **c**ipher-based **m**essage **a**uthentication **c**ode (CMAC), or a ciphertext based on a mode for **a**uthentication **e**ncryption with **a**ssociated **d**ata (AEAD). Similarly, a threshold scheme for an approved hash function could then also be applied to calculate an HMAC. Some threshold schemes may nonetheless allow a cost amortization when repeatedly executed.

A.4.1. Subcategory C1.4.1: AES Enciphering/Deciphering

With respect to threshold enciphering/deciphering in Cat1, there is only one symmetric-key block-cipher of interest: AES, specified in FIPS-197. A submission of threshold scheme for AES enciphering/deciphering must assume a secret-sharing of the secret key, and should provide implementations for at least the key-sizes 128 and 256. A submission can choose to implement any (or various) types of input/output interface from {NSS, SSI, SSO and SSIO}. In applications where the high-sensitivity of the plaintext warrants a distribution of trust over its knowledge, then it can make sense to consider: an SSI mode for enciphering, and/or an SSO mode for deciphering, so that the plaintext is not reconstructed within the decentralized AES-evaluator. For benchmarking purposes, a submission should evaluate performance at least in the single evaluation case, i.e., for a single AES enciphering and/or deciphering. However, to help clarify possible amortization gains and/or clarify the feasibility of the threshold approach for AES modes of operation (in the SP800-38-series), the benchmarking can also measure performance for the threshold execution of 2^6 and/or 2^{10} AES encipherings/decipherings in some specific mode of operation.

Threshold AES enciphering versus oblivious AES evaluation. Oblivious AES evaluation is a common **s**ecure **2**-**p**arty **c**omputation (S2PC) benchmark in the literature. There, a single party holding the plaintext does not share it with a single party holding the key, and yet receives the corresponding ciphertext. The application of threshold AES in scope in this call is different, in that the threshold entity is responsible for computing the output, when the key has been secret-shared. The plaintext is either (i) directly shared with the threshold-decentralized entity responsible for the enciphering or deciphering, or (ii) is secret-shared in the input/output. A secret-shared-I/O threshold AES enciphering may also be useful for the computation of a CMAC, which can in turn be useful for 2KE KDM/KC. That said, techniques developed for threshold AES are likely to also be useful for oblivious AES evaluation.

A.4.2. Subcategory C1.4.2: KDM and KC for 2KE

The protocols for pair-wise key-establishment (2KE), in both the ECC-based [SP800-56A-Rev3] and RSA-based [SP800-56B-Rev2] cases, are finalized with the use of a **k**ey-**d**erivation **m**echanism (KDM) [SP800-56C-Rev2; SP800-108-Rev1] and optional **k**ey-**c**onfirmation (KC). These operations follow after the generation of a precursor intermediate secret M, obtained/produced via a key-agreement of key-transport type of 2KE protocol.

Threshold unfriendliness. The current NIST-specified KDM and KC primitives are possible to thresholdize based on complex MPC protocols, but are based on threshold-unfriendly hash-or-XOF functions ([FIPS-180-4; FIPS-202]) or MAC/PRFs (of the type CMAC [SP800-38B], HMAC [FIPS-198-1] or KMAC [SP800-185]).

Considering the “pair-wise” nature of key-establishment protocols (i.e., involving two sides), some use cases (namely when party A has to be thresholdized, but party B has to use a legacy implementation) may require the use of a KDM and/or KC that is functionally-equivalent to a currently NIST-specified one. However, the costs and benefits of implementing a potentially costly MPC in such a case should be carefully considered.

Threshold schemes for AES enciphering/deciphering may be easy to adapt to threshold schemes for CMAC primitives. Techniques used to enable threshold schemes for the hashing that is useful for KDM or KC may also be reusable for (pseudorandom) EdDSA and Deterministic ECDSA, which require a secret-nonce computed as a hash whose pre-image contains a private nonce-derivation key.

Cat2 of this call enables proposals of threshold-friendlier KDM and KC primitives that would still retain the desired properties of the final generated key, namely indistinguishability from uniform selection, and one-wayness with respect to the intermediate key Z used as input.

A.4.2.1. Key Derivation Mechanism (KDM)

A threshold KDM scheme makes sense if the corresponding party (in the pair-wise key--establishment) is supposed to not learn the final secret k. The threshold KDM scheme produces a secret-shared output (SSO) (similar to a threshold keygen scheme), so that the final secret k (to be consumed by another primitive) is secret-shared. There are one-step (extraction) and two-step (extract-then-expand) KDMs (see SP800-108-Rev1 for the second step). Additionally, there are variants (see SP800-135-Rev1) approved for specific applications.

Since the final key k can be easily derived from the intermediate key M, it follows that it only makes sense to thresholdize a KDM if the input (intermediate) key M is also secret-shared. Conversely, if a KDM is not thresholdized but Z has itself been produced in a threshold manner, (i.e., based on a secret-shared private key d), then the reconstruction of Z does not break the confidentiality of the private key d.

A.4.2.2. Key Confirmation (KC)

A threshold **key-confirmation** primitive computes a PRF image of the intermediate secret Z, without Z ever being reconstructed. This can make sense if the KDM is also thresholdized in SSI mode, to directly use a secret-shared Z as input, withouth needing to reconstruct it. Key-confirmation is defined, in various possible modes (unilateral or bilateral), for ECC-based key-agreement in SP800-56A-Rev3 (§5.9, Table 5) and RSA-based key-establishment in SP800-56B-Rev2 (§5.6, Table 1).

A.5. Subcategory C1.5: key-Generation (keygen) for Cat1 Schemes

A key-generation (keygen) primitive determines a private/secret "key" that is needed by subsequent primitives. The threshold scheme may also compute other public parameters. For example, the keygen primitive of a digital signature scheme produces a private/public keypair, whose private element is then required to produce signatures, and whose public element is used to verify the correctness of signatures. Typical requirements for private keys include unbiasing and confidentiality. These requirements can also apply to the generation of other secret material, such as a random secret nonce. Secrets generated via a keygen primitive may be persistent (e.g., for multiple-times use, without planned erasure), or ephemeral (e.g., for single-time use, followed by erasuse). Table 10 provides a non-exhaustive list of parameters that may be generated via a keygen operation (some variations are possible).

Table 10. Examples of keygen purposes

Keygen purpose (subsequent operation)	**Private/secret key**	**Other public elements**
ECC-signing; ECC-2KA primitives	exponent d (integer mod n)	$Q = d \cdot G$ (elliptic curve point)
RSA signing and decryption	primes (p, q)	modulus $N = p \cdot q$
|	exponent $d = e^{-1} \text{mod } \phi_N$	exponent e
RSA encryption for 2KE	random bit-string Z	$c = \text{RSAEP}((n,e), Z)$
Key-derivation / key-confirmation	|	$\text{KC}(Z, \ldots)$
AES enciphering/deciphering	random bit-string k	—

Terminology and scope for threshold schemes for keygen. Threshold schemes for keygen are often called **d**istributed **k**ey **g**eneration (DKG) protocols. In this call, the focus on DKG is only on the generation of the private/secret keys and (when applicable) the public parameters that depend on them (e.g., an RSA modulus obtained from the product of two secret primes, or the elliptic curve public point obtained from integer-multiplying a base point by the secret key). Other "domain parameters", such as the security strength κ, the parameters of an elliptic curve, or an RSA encryption key, which may be determined before the computation of the private key (but which in conventional specifications may sometimes be included within the keygen primitive) can be assumed to be fixed or pre-agreed upon.

Interchangeability of random values. In a DKG protocol, the random private/secret key to be output in secret-shared form, and possibly other intermediate random elements, is obtained by combining random contributions from several parties. This call does not pose specific requirements on these random values, i.e., beyond the requirement of interchangeability with regard to some subsequent operation of interest, However, a submitted DKG protocol should be accompanied by an explanation of why the proposed randomness generation mechanism provides appropriate security assurances, namely compared to the assurances provided by the conventional random-bit generation (RBG) [SP800-90A-R1; SP800-90B; SP800-90C-3PD] that may be required in the corresponding conventional (non-threshold) keygen specification. Some original RBG-related requirements associated with random values in the conventional specification may still be considered for the individual contributions of each party in a corresponding DKG.

A.5.1. Subcategory C1.5.1: ECC Keygen (for ECDSA, EdDSA, and ECC-2KA)

The ECC keygen of a private/public key-pair is similar across various schemes, including for ECDSA and EdDSA signature schemes [FIPS-186-5-Draft], and for ECC-2KA primitives, such as CDH and MQV [SP800-56A-Rev3]. In a threshold **ECC keygen** (i.e., DKG for an ECC scheme), the usual goal is to produce a secret-sharing $[d]$ of a private key d (usually a positive integer mod n, the order of the subgroup of interest), along with a corresponding (not-secret-shared) public key $Q = d \cdot G$. In a threshold 2KA scheme, each party may need this decentralization (secret-sharing) for their static private key d_A (or $d_{s,A}$) and/or an ephemeral private key ($d_{e,A}$).

Some schemes, such as EdDSA, may include additional private/secret elements (e.g., a nonce-derivation key for pseudorandom generation of nonces) that do not require a subsequent verifiable relation with the public key. The generation of said components in the threshold setting may be considered differently (or may even not be necessary), provided that an appropriate interchangeability property is satisfied with regard to the subsequent operations that use the ECC private/public keypair.

Submissions of threshold schemes for ECC signing and ECC-2KA primitives are expected (though not required) to include a corresponding proposal of a compatible ECC-DKG protocol. Implementation recommendations for a submitted DKG (e.g., which elliptic curves and security parameters) should apply to at least one subsequent threshold scheme of interest.

A.5.2. Subcategory C1.5.2: RSA Keygen

RSA keygen is needed for the RSADSA scheme (Section A.1.1) and the RSA PKE scheme used for 2KE (Section A.2). In its *basic* format, RSA keygen consists of:

- generating a pair of random secret primes (p,q), and outputting their product N; and
- computing and outputting as private key d the inverse (mod $\mathrm{LCM}(p-1, q-1)$) of a public exponent e, where e is selected (randomly or as an input parameter) before the selection of the primes.

DKG schemes for RSA can be submitted separately from subsequent threshold operations, such as threshold RSA signing, threshold RSA decryption, or threshold RSA SSI-encryption. Still, a submission of RSA DKG should be compatible with said subsequent schemes, and should include evaluation for at least two security parameters consistent with the recommendations from Table 5.

FIPS-186-5-Draft (§A.1) and SP800-56B-Rev2 (§6.2–§6.3) specify various requirements for the RSA keygen, respectively for signing and PKE. Possible variations of the format of the output key include the *prime-factor* format and the *CRT* format, as explained in Section A.2.2. The following paragraph list some of the requirements.

A.5.2.1. Criteria for the RSA Modulus and Primes

- p and q must be of the same bit length (i.e., half the length of the RSA modulus N).
- p and q must be randomly generated (but the two most significant bits of each may be arbitrarily set), as "probable" or "provable" primes, satisfying at least one of the five options from Table 11.

Table 11. Criteria for the random primes of an RSA modulus

Type	Sub-type	Provable prime	Probable prime
Simple	pro**v**able	p, q	
$\vert$	pro**b**able		p, q
Complex	pro**v**able	p_1, p_2 q_1, q_2 p, q	
$\vert$	hybrid	p_1, p_2, q_1, q_2,	p, q
$\vert$	pro**b**able		p_1, p_2, q_1, q_2, p, q

Per §A.1.1 of FIPS-186-5-Draft: 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., $p_1|p-1$, $p_2|p+1$, $q_1|q-1$, $q_2|q+1$.

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

In a submitted RSA DKG, the threshold computation of the primes and modulus may be modularized from the subsequent calculation of the private decryption/signing exponent d. Interestingly, there are conceivable applications (beyond signatures, encryption, and decryption) where RSA moduli are useful and a private exponent is not necessary.

A.5.2.2. Criteria for the Private Exponent

The private exponent $d = e^{-1} \pmod{L}$, where $L = \mathrm{LCM}(p-1, q-1)$, must be larger than $2^{nlen/2}$ and smaller than L, where the public exponent e is an integer between 2^{16} and 2^{256} selected before the generation of p and q.

A.5.3. Subcategory C1.5.3: Bitstring Keygen

Various primitives require the random generation of a secret bit-string (or integer within a defined interval), without the need for a corresponding public component. For example, this is the case with generating: an AES key; a secret-key for encapsulation under an RSA PKE; a nonce for use in other schemes; a salt for a KDM or KC in the scope of a 2KA.

A DKG based on verifiable secret-sharing may require public commitments of the shares of each party, even if the original primitive did not require any public key. A submission should explain how/whether the cryptographic assumptions sustaining the security of the threshold scheme change in comparison with those required for the security of the original primitive. For example, AES-256 is considered to be post-quantum secure, whereas ECC-based commitments used in typical MPC protocols might not be.

A.6. Subcategory C2.6: Advanced

As mentioned in Section 7.2.1, subcategory C2.6 allows for the submission of threshold schemes for primitives that support cryptographic schemes with advanced functional features that are different from those in current NIST standards. For example, in the case of a **f**ully-**h**omomorphic **e**ncryption (FHE) scheme, the supported operations go beyond the usual keygen, encryption and decryption from a regular encryption scheme. There is also a set of homomorphic operations (e.g., addition and multiplication) over ciphertexts (see, e.g., [HES, §1.1.1]). As another example, an **i**dentity-**b**ased **e**ncryption (IBE) scheme has not just one key-generation primitive, but rather two: one for generating a public key and a master private key, and another one (requiring the master key as input) for generating a decryption key for each possible "identity" (e.g., email addresses). A generalization of IBE is **a**ttribute-**b**ased **e**ncryption (ABE), where the private key of each user is created based on a set of attributes.

In this subcategory, the selection of the use-cases used to benchmark performance is left to the discretion of the submitters. For example, different FHE schemes may require different benchmarking operations to highlight their best features. One FHE scheme may be better suited to homomorphic Boolean operations (operations over bits), while another one may be better suited for homomorphic modular operations over large integers.

A.6.1. Use-Case Example: Non-Threshold FHE-Based AES Oblivious Enciphering

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

0b. **Setup AES (keygen):** An AES cipher is initialized with secret key k, with AES.Enc$_k$ denoting the corresponding enciphering operation.

0c. **Setup parties (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 $C = \text{FHE.Enc}_e(m)$, and sends it to the server S.

2. **FHE-Homomorphic-Evaluate.** The server S homomorphically evaluates the AES-enciphering, obtains $H = \text{FHE.Hom}[\text{AES.Enc}_k](C)$ (which is a valid FHE-encryption of the AES-enciphering of secret plaintext m), and sends the result to client B.

3. **FHE-Decrypt.** The client B FHE-decrypts the received ciphertext H, and thus obtains the AES-enciphering of the secret plaintext: $\text{AES.Enc}_k(m) = \text{FHE.Dec}_d(H)$.

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

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

External engagement. Proposals of FHE schemes (and their threshold schemes) are welcome to be submitted and/or analyzed in connection with other related ongoing public efforts, such as HomomorphicEncryption.org and FHE.org, as a way of promoting: (i) fulfillment of community-based technical recommendations; (ii) alignment with existing reference material/specifications; and (iii) further public scrutiny of proposed schemes. Such engagements may also help clarify reference use-cases for useful benchmarking.

A.6.2. Threshold Schemes for FHE-based AES Oblivious Enciphering

Once a conventional (non-threshold) scheme is specified (S6) in scope of the "advanced" subcategory C2.6, there may be multiple types of decentralization to consider. For the above-described example of FHE application (Section A.6.1), the following is a non-exhaustive list of possible decentralizations of one of the original participants (client A, server S, or client B) into a threshold entity composed of multiple parties.

1. **Threshold FHE.Keygen.** In a setup phase with a thresholdized client B, a DKG can distributively compute a secret-sharing of an FHE decryption key d. Whether or not the encryption key e is secret-shared can depend on whether the FHE scheme is of, respectively, symmetric-key or asymmetric-key (i.e., public/private key pair) type.
2. **SSI threshold FHE-Encryption.** If client A is thresholdized, and set up with a secret-shared plaintext m, a threshold scheme can compute $C = \text{FHE.Enc}_e(m)$ without anyone learning m.
3. **Threshold Homomorphoic evaluation (of function with secret parameter).** If the server S is thresholdized, and setup with a secret-sharing of the AES key k, then the parties can distributively compute the homomorphic-evaluation operation, to obtain $H = \text{FHE.Hom}[\text{AES.Enc}_k](C))$, without anyone learning k.
    - In an NSS mode, all server-parties learn H.
    - In an SSO mode, each server learns a secret-share of H.
4. **Threshold FHE decryption.** If client B is thresholdized, and setup 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 a NSS mode, all clientB-parties learn C.
    - In a SSO mode, each clientB-party learns only a secret-share of C.
5. **Threshold ZKPoK.** (See subcategory C2.7 in Section A.7)

On the use case of oblivious AES enciphering. The use case is called oblivious AES-enciphering because the client B obtained an AES-enciphering of the secret plaintext m even though the AES-key holder (the server S) remained oblivious to the secret plaintext. Interestingly, oblivious AES-enciphering is also a typical benchmark case for secure 2-party computation (S2PC; consider the case where clients A and B are the same), usually using different techniques, such as garbled circuits and/or oblivious transfer. Compared with an FHE-based solution, usual S2PC protocols (expectably) lead to much faster execution, but also much larger communication complexity.

A.7. Subcategory C2.7: ZKPoKs

Besides (secure) **m**ulti-**p**arty **c**omputation (MPC), a broad type of primitive of great interest in the threshold context is the **z**ero-**k**nowledge **p**roof **o**f **k**nowledge (ZKPoK), which is covered by subcategory C2.7. As mentioned in Section 7.2.2, a submission of ZKPoK in this subcategory must specify a conventional ZKPoK, and possibly also specify a threshold version (when the prover is distributed and there is a secret-sharing of the secret input).

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, in a ZKPoK of a private RSA key, the *instance* can be the RSA modulus N, the secret *witness* can be the corresponding pair (p,q) of prime factors, and the *relation* can be the predicate that returns `true` if and only if the input witness is indeed a pair of primes and their product is the public modulus.

Type of "proofs" of interest:

- **Proofs and arguments:** The use of "proof" in this call is meant to also include the case of *arguments* with computational soundness. Any submission of ZKPoK should clarify its soundness type (to allow for differentiation between "proof" and argument).
- **ZKP of knowledge (versus of correctness):** The proofs in scope are ZKPoKs, but can also serve the purpose of ZK-proving *correctness* of the secret data (whose knowledge is being proven) as well as of the corresponding public data. In the literature, a ZKP of correctness is also known as a ZKP of "language membership".
- **Transferable and non-interactive.** Traditionally, ZKPs and ZKPoKs are defined as two-party protocols with a requirement of deniability (also known as non-transferability), implying that a verifier convinced by a proof cannot later transfer said confidence to a third party. This property often stems from interactivity between prover and verifier, and/or relies on local setup assumptions, such as a local **c**ommon **r**eference **s**tring (CRS) or local **r**andom **o**racle (RO). Conversely, the present call is by default interested on transferable non-interactive zero-knowledge (NIZK) proofs that can be publicly verified non-interactively. A submission of ZKPoK can deviate from this default (non-interactiveness and transferability) as long as justified on the basis of utility to the threshold setting.

The instantiation of some of the above-listed attributes (e.g., transferability, and computational soundness) may affect some aspects of composability. These effects should be discussed in any submission that proposes a ZKPoK.

Distributed prover (not verifier). In this call, the default setting of interest for thresholdization of a ZKPoK is the secret-sharing, across multiple parties, of the secret key (traditionally held by a single prover) whose knowledge is being proven. While a ZKPoK variant can also be conceived for the case of distributed verification (with the ZK property requiring that a threshold number of verifier parties do not collude), such setting is not the default. A deviation from the mentioned default in a submission of ZKPoK is possible but its auxiliary utility for the threshold setting then needs to be thoroughly argued for.

Examples. Table 12 lists various examples of ZKPoK of anticipated interest with regard to Cat1 primitives. Other examples can be conceived for primitives in Cat2.

Table 12. Example ZKPoKs of interest related to Cat1 primitives

Related type	Related (sub)sub-category: Primitive	Example ZKPoK (including consistency with public commitments of secret-shares, when applicable)
Keygen	C1.5.1: ECC keygen	of discrete-log (s or d) of pub key Q
|	C1.5.2: RSA keygen	of factors (p, q), or group order ϕ, or decryption key d
|	C1.5.3: AES keygen	of secret key k (with regard to secret-sharing commitments)
PKE	C1.2.1: RSA encryption	of secret plaintext m (encrypted)
|	C1.2.2: RSA decryption	of secret-shared plaintext m (after SSO-threshold decryption)
Symmetric	C1.4.1: AES enciphering	of secret key k (with regard to plaintext/ciphertext pair)
|	C1.4.2: Hashing in KDM	of secret pre-image Z

Some observations:

- A ZKPoK of a secret AES key that transforms a given plaintext into a given ciphertext corresponds to a signature primitive submitted to the PQC process.
- No ZKPoK example was provided in association with the signing operation, since their public verification operation already inherently verifies the signature correctness. In fact, a digital signature often constitutes a transferable NIZKPoK of the private signing key corresponding to the public key, with said proof being additionally bound to a message (the element being signed). For example, an EdDSA/Schnorr signature (Section A.1.1) is itself a NIZKPoK of discrete-log.
- The cases of ZKPoK related to a private **signing** key, but possibly without producing a signature, are associated with keygen (subcategories C1.5 and C2.5).

If a submission of threshold scheme uses a ZKP/ZKPoK that may be of interest to support other threshold schemes, then it should modularize the specification of said ZKP/ZKPoKand indicate it as useful also for consideration in subcategory C2.7.

Submission of a ZKPoK as auxiliary to other threshold scheme(s):

- **Specification of a non-threshold version.** A submission in the ZKPoK subcategory must specify a conventional (non-threshold) ZKPoK. This may be submitted without a corresponding distributed/threshold version, as long as the documentation clarifies how the conventional ZKPoK can be useful for the threshold setting (perhaps some other concrete threshold scheme). For example, a conventional ZKPoK can be justified for use by a dealer to prove correctness of an established secret-sharing setup. There may nonetheless be an additional value in also specifying a threshold version of the ZKPoK (i.e., when the secret input is distributed).
- **Standalone versus embedded proposal of a ZKPoK.** A package that proposes an auxiliary ZKPoK (and possibly a distributed version thereof) can be submitted within the standalone ZKPoK subcategory, or within a submission of a threshold scheme(s) for other primitives in Cat1 or Cat2. In the standalone case, the proposal must clarify how the secret and public knowledge matches the setting of (e.g., a particular secret-sharing useful for) a threshold scheme for some primitive of interest.
- **External engagement.** Proposals of ZKPoK schemes (and their threshold schemes) are welcome to be submitted and/or analyzed in connection with other related on-going public efforts, such as ZKProof.org, as a way of promoting: (i) fulfillment of community-based technical recommendations; (ii) alignment with existing reference material/specifications; and (iii) further public scrutiny of proposed schemes. Such engagements may also help clarify reference use-cases for useful benchmarking.

Notes on features.

- **Succinctness:** For practicality, **succinctness** is a useful feature of a ZKPoK. When focusing on succinct and non-interactive ZKPoKs, it is also common to refer to them as SNARKs (**s**uccinct **n**on-interactive **ar**guments of **k**nowledge).
- **Transferability:** As mentioned above, non-interactive public verifiability / transferability are default desired features
- **Security assumptions:** While the assessment of security of a ZKPoK may be based on assumptions different from those inherent to the underlying cryptographic primitive, or to a related proposed threshold scheme, said implications should be distinguished across various security properties. In particular, it is relevant to characterize the properties of ZK, soundness and non-malleability, and how they may vary upon various types of protocol composition (e.g., concurrent executions).

Specialized versus generic ZKPoKs. Some ZKPoKs (e.g., of a discrete-log, or of an RSA private key) may be based on specialized techniques somewhat similar to the operations (e.g., exponentiations) used to commit the secret pre-image. Conversely, other ZKPoKs (e.g., when proving knowledge of a pre-image of AES-enciphering, or of SHA-based hashing) may stem more easily from a generic ZKP system that simply requires "arithmetizing" the *statement* of knowledge, the *instance* and the *witness* in some suitable representation (e.g., specifying a Boolean or arithmetic circuit, and instantiating its input variables). In the latter case, a submitted ZKPoK can be explained generically, and then a simple explanation be given on how to apply it to a circuit (or other applicable representation). For example, the NIST Circuit Complexity project [Proj-CC] collects Boolean circuit representations of various NIST-approved primitives, such as from AES and SHA. The final version of this call may reference a specific representation for Boolean circuits, to facilitate an interchangeable specification of circuits of certain NIST-specified primitives (e.g., of certain block-ciphers and hash-functions) whose proof of knowledge of pre-image may be useful.

A.8. Subcategory C2.8: (Auxiliary) Gadgets

As mentioned in Section 7.2.3, subcategory C2.8 allows for the consideration of gadgets, such as garbled circuits, oblivious transfer, generation of correlated randomness, commitments, secret resharing (possibly for a new threshold value and a new total number of parties), multiplicative-to-additive share conversion, additively homomorphic encryption (AHE), MPC or ZKP friendly hashing, consensus, and broadcast. The specification of some gadgets may also fit other subcategories. For example, an AHE scheme allows for an *advanced* feature (homomorphic addition over ciphertexts), and thus can fit in "advanced" subcategory C2.6 (if accompanied by a corresponding threshold scheme), and at the same time can also be useful to support multiple other threshold schemes, and thus fit in subcategory C2.8. In such type of cases, a submission should identify (e.g., including in S2 and S3) the fit in various subcategories.

Gadgets can be proposed in a standalone manner in a submission, or as a module in a more encompassing submission in the scope of other subcategories. A standalone submission of an auxiliary gadget (and possible threshold version thereof) should make a strong case for its utility in supporting the threshold environment, and/or in directly supporting various concrete threshold schemes in scope of other subcategories in this call.

B. Submission Checklists

The following are draft templates of checklists to help keep track of the fulfillment of the various requirements for a complete submission:

B.1. Checklist for Submission Phases (Ph) (see Section 4)

Check	#	Item	Comments
☐	Ph1	(Optional) Early abstract	
☐	Ph2	(Optional) Preliminary package	
☐	Ph3	Full package (M1–M5)	

B.2. Checklist for Package Main Components (M) (see Section 4)

Check	#	Item	Comments
☐	M1	Written specification (S1–S16)	
☐	M2	Reference implementation (Src1–Src4)	
☐	M3	Execution instructions (X1–X7)	
☐	M4	Experimental evaluation (Perf1–Perf5)	
☐	M5	Additional statements	

B.3. Checklist for M1: Written Specification Sections (S) (see Section 4.2)

Check	#	Item	Comments
☐	S1	Title pages	
☐	S2	Abstract	
☐	S3	Executive summary	
☐	S4	Index	
☐	S5	Clarification of prior work	
☐	S6	Conventional primitives/scheme	
☐	S7	System model	
☐	S8	Protocol description	
☐	S9	Security analysis	
☐	S10	Analytic complexity	
☐	S11	Choices and comparisons	
☐	S12	Technical criteria	
☐	S13	Deployment recommendations	
☐	S14	Notation	
☐	S15	References	
☐	S16	Appendices (optional)	

B.4. Checklist for M2: Open source (Src) Reference Implementation (see Section 4.3)

Check	#	Item	Comments
☐	Src1	Is self-contained	
☐	Src2	Is licensed as open-source	
☐	Src3	Contains inline comments	
☐	Src4	Has a clear API	

B.5. Checklist for M3: Execution Instructions (X) (see Section 4.4)

Check	#	Item	Comments
☐	X1	User manual: compilation	
☐	X2	User manual: parametrization	
☐	X3	User manual: execution	
☐	X4	User manual: KAT set	
☐	X5	Script: KAT	
☐	X6	Script: benchmark	
☐	X7	Script: others (optional)	

B.6. Checklist for M4: Performance Analysis (Perf) (see Section 4.5)

Check	#	Item	Comments
☐	Perf1	Memory complexity	
☐	Perf2	Processing time	
☐	Perf4	Networking time	
☐	Perf3	Communication complexity	
☐	Perf5	Round complexity	

B.7. Checklist for Technical Requirements (T) (see Section 5)

Check	#	Item	Comments
☐	T1	Primitives	
☐	T2	System model	
☐	T2.1	Participants	
☐	T2.2	Distributed systems and communication	
☐	T2.3	Adversary	
☐	T3	Security idealization	
☐	T4	Security versus adversaries	
☐	T4.1	Active	
☐	T4.2	Adaptive	
☐	T4.3	Pro-active	
☐	T5	Threshold profiles	
☐	T6	Building blocks	

References

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

[FIPS-186-5-Draft] National Institute of Standards and Technology (NIST). *Digital Signature Standard (DSS).* (U.S. Department of Commerce) Draft Federal Information Processing Standards Publication (FIPS PUBS) 186-5. October 2019. DOI: 10.6028/NIST.FIPS.186-5-Draft.

[FIPS-197] National Institute of Standards and Technology (NIST). *Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197*. November 2001. DOI: 10.6028/NIST.FIPS.197.

[FIPS-198-1] National Institute of Standards and Technology (NIST). *The Keyed-Hash Message Authentication Code (HMAC). Federal Information Processing Standards Publication 198-1*. July 2008. DOI: 10.6028/NIST.FIPS.198-1.

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

[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. *Homomorphic Encryption Standard*. Published by HomomorphicEncryption.org. November 2018. Updated versions at https://homomorphicencryption.org/standard/.

[IG-FIPS-140-2] National Institute of Standards and Technology (NIST) and Canadian Centre for Cyber Security (CCCS). *Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program.* Version updated by the Cryptographic Module Validation Program on 2022-October-07. https://csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/fips140-2/FIPS1402IG.pdf. October 2022.

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

[MPTC-Call2021a] Luís Brandão. (National Institute of Standards and Technology (NIST) Multi-Party Threshold-Cryptography (MPTC)) Call 2021a for Feedback on Criteria for Threshold Schemes. July 2021. URL: https://csrc.nist.gov/csrc/media/projects/threshold-cryptography/do cuments/MPTC-call2021a-feedback.pdf. Public comments: https://csrc.nist.gov/csrc/media/projects/threshold-cryptography/documents/MPTC-Call2021a-Feedback-compilation.pdf. Accessed via the NIST Computer Security Resource Center (CSRC) in January 2023.

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

[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.

[Proj-CC] National Institute of Standards and Technology. *NIST Project on Circuit Complexity (CC)*. https://csrc.nist.gov/projects/circuit-complexity. See list of circuits at https://csrc.nist.gov/projects/circuit-complexity/list-of-circuits, and repository of circuits at GitHub:usnistgov/Circuits. Accessed in January 2023.

[Proj-LWC] National Institute of Standards and Technology. *NIST Project on Lightweight Cryptography (LWC)*. https://csrc.nist.gov/projects/lightweight-cryptography. Accessed in January 2023.

[Proj-MPTC] National Institute of Standards and Technology. *NIST Project on Multi-Party Threshold Cryptography (MPTC)*. https://csrc.nist.gov/projects/threshold-cryptography. Accessed in January 2023.

[Proj-PEC] National Institute of Standards and Technology. *NIST Project on Privacy-Enhancing Cryptography (PQC)*. https://csrc.nist.gov/projects/pec. Accessed in January 2023.

[Proj-PQC] National Institute of Standards and Technology. *NIST Project on Post-quantum Cryptography (PQC)*. https://csrc.nist.gov/projects/post-quantum-cryptography. Accessed in January 2023.

[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.

[SP800-38-series] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation — Parts A through G.* (National Institute of Standards and Technology) NIST Special Publications (SP): 800-38A (*Methods and Techniques*, 2021); 800-38B (*the CMAC Mode for Authentication* 2016); 800-38C (*the CCM Mode for Authentication and Confidentiality*, 2007); 800-38D (*Galois/Counter Mode (GCM) and GMAC*, 2007); 800-38E (*the XTS-AES Mode for Confidentiality on Storage Devices*, 2010); 800-38F (*Methods for Key Wrapping*, 2012); 800-38G (*Methods for Format-Preserving Encryption*, 2016). October 2016.

[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. May 2005. DOI: 10.6028/NIST.SP.800-38B.

[SP800-38F] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-38F. December 2012. DOI: 10.6028/NIST.SP.800-38F.

[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.

[SP800-56C-Rev2] Elaine Barker, Lily Chen, and Richard Davis. *Recommendation for Key-Derivation Methods in Key-Establishment Schemes.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-56C Rev. 2. August 2020. DOI: 10.6028/NIST.SP.800-56Cr2.

[SP800-57-P1-R5] Elaine Barker. *Recommendation for Key Management: Part 1 — General.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-57 Part 1 Rev. 5. May 2020. DOI: 10.6028/NIST.SP.800-57pt1r5.

[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-3PD] Elaine Barker, John Kelsey, Kerry McKay, Allen Roginsky, and Meltem Sönmez Turan. *Recommendation for Random Bit Generator (RBG) Constructions. Third Public Draft.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-90C 3PD. September 2022. DOI: 10.6028/NIST.SP.800-90C.3pd.

[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. August 2022. DOI: 10.6028/NIST.SP.800-108r1.

[SP800-135-Rev1] Quynh Dang. *Recommendation for Existing Application-Specific Key Derivation Functions.* (National Institute of Standards and Technology) NIST Special Publication (SP) 800-135 Rev. 1. December 2011. DOI: https://doi.org/10.6028/NIST.SP.800-135r1.

[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 (Draft). December 2016. DOI: 10.6028/NIST.SP.800-185.

[SP800-186-Draft] Lily Chen, Dustin Moody, Andrew Regenscheid, 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 (Draft). October 2019. DOI: 10.6028/NIST.SP.800-186-draft.

[ZkpComRef] ZKProof. *ZKProof Community Reference. Version 0.3*. Published by ZKProof.org. Editors: D Benarroch, L Brandão, M Maller, and E Tromer. Contributors since version 0: S Agrawal, T Arcieri, D Benarroch, V Bharathan, N Bitansky, S Bowe, L Brandão, B Bünz, R Canetti, A Caro, K Chalkias, J Cincinnati, H Corrigan-Gibbs, J Daniel, M Dixon, M Dubovitskaya, B Falk, D Genkin, N George, S Goldwasser, A Gupta, J Grigg, J Groth, K Gurkan, Y Hang, D Hopwood, Y Ishai, C Jutla, Y Kalai, H Krawczyk, J Law, A Lysyanskaya, M Maller, E Morais, Z Manian, A Miller, E Morais, N Narula, R Ostrovsky, G Pacini, O Paneth, R Peralta, A Poelstra, T Rabin, M Raykova, A Robinson, R Rothblum, J Rouach, A Scafuro, a shelat, K Solipuram, J Thaler, E Tromer, M Varia, M Venkitasubramaniam, M Virza, I Visconti, R Wahby, D Wikstrom, P Wuille, Y Zohar, and A Zhang. July 2022. Updated versions at https://docs.zkproof.org/reference.