**"Preview Writeup": In anticipation of a package submission to the NIST Threshold Call**

# **Title:** tBLS: Threshold BLS Signature Scheme

## **Subtitle:** Threshold BLS Signature

**Version:** 0.1 (2026-01-19)[1]

**Team name:** BBDL

**Team members:** Renas Bacho, Alexandra Boldyreva, Sourav Das, Julian Loss

**Abstract:** This document proposes a threshold variant of the Boneh–Lynn–Shacham (BLS) signature scheme for use in high-assurance distributed systems. The construction distributes signing capability among $n$ parties such that any subset of at least $t+1$ participants can jointly produce a single compact signature, while preserving the verification interface, encodings, and compatibility of standard single-party BLS signatures. Partial signatures are generated non-interactively, are publicly verifiable, and are combined deterministically using Lagrange interpolation in the exponent, yielding a unique final signature indistinguishable from a standard BLS signature.

The scheme supports both trusted-dealer and distributed key generation settings and is designed to remain secure against adaptive adversaries that may corrupt participants over time. Key generation is treated as an external and modular component, and this document does not specify or normatively adopt any concrete distributed key generation protocol. Security is established in the algebraic group model and the random oracle model under the one-more discrete logarithm assumption, and the construction can be instantiated with practical distributed key generation protocols satisfying algebraic simulatability. Recommended ciphersuites target approximately 128-bit security using pairing-friendly elliptic curves and standardized hash-to-curve mechanisms.

Owing to its compact and unique signatures, non-interactive and deterministic signing and combination, public verification of partial signature shares, compatibility with standard BLS verification, and provable adaptive security, the proposed threshold BLS scheme is well suited for deployment in applications such as certificate authorities, distributed key-management systems, blockchain validator infrastructures, quorum-based hardware security modules, and Byzantine fault-tolerant consensus protocols, aligning with the goals of NIST's Threshold Cryptography initiative.

**Proposed crypto-systems:** tBLS: Threshold BLS Signature (Category S1)

**Keywords:** Threshold Cryptography; NIST Threshold Call; Threshold Signatures; BLS Signatures

[1] Preliminary version submitted to NIST-MPTC for review

**Preview writeup.** This document is provided to NIST for online publication, to foster public awareness and support public discussion within the scope of the NIST First Call for Multi-Party Threshold Schemes [NIST-IR8214C]. This "preview writeup" represents a good-faith plan for a subsequent "package submission". However, until the deadline for package submission, the team may still modify its own composition and the submission plan, including possible changes to the technical scope, and/or the used techniques or achieved results.

**Team members:** Renas Bacho [i1,a1], Alexandra Boldyreva [i2,a2], Sourav Das [i3,a3], Julian Loss [i4,a4]

## Open Researcher and Contributor Identifiers (ORCID):

i1 (0009-0007-7037-2458); i2 (0009-0000-7019-884X); i3 (0000-0001-5298-621X); i4 (0000-0002-7979-3810)

## Affiliations:

[a1] CISPA Helmholtz Center for Information Security, Saarbrücken, Saarland, Germany

[a2] Georgia Institute of Technology, Atlanta, GA, USA

[a3] Category Labs, New York, NY, USA

[a4] Ruhr University Bochum, Bochum, North Rhine-Westphalia, Germany

## Main contacts:

- **Team mailing list:** team-bbdl@googlegroups.com
- **Primary technical contact person:** Julian Loss, lossjulian@gmail.com
- **Secondary contact person 1:** Alexandra Boldyreva, sasha@gatech.edu

**Produced by humans.** The team hereby confirms that the content in this preview writeup: (i) was produced by the team members, and (ii) was not produced by generative artificial intelligence (AI), with the possible exception of AI-proposed grammar improvements, minor integrated suggestions, or some well-identified and short localized portions of auxiliary content (e.g., some illustration); and (iii) was proofread by the team members.

# 1. Introduction

This document proposes a standardized Threshold BLS Signature Scheme (tBLS) in response to NIST's Call for Multi-Party Threshold Schemes. The submission specifies, implements, and evaluates a threshold signature construction that distributes signing capability among $n$ participants such that any quorum of $t+1$ parties can jointly produce a valid signature, while any smaller coalition cannot produce a valid signature. The proposed cryptosystem preserves full compatibility with the Boneh–Lynn–Shacham (BLS) signature scheme [BLS01], yielding signatures that are indistinguishable from standard single-party BLS signatures and verifiable using existing BLS verification code.

The proposed scheme falls within Category S1: [Special] Signing and specifies a complete threshold signing interface, including setup, partial signing, share verification, deterministic combination, and final signature verification. While the scheme can be instantiated using either a trusted-dealer setup or a distributed key generation (DKG) protocol, key generation is treated as an external primitive and is not specified as part of the scheme. When instantiated with a DKG, the protocol is only required to satisfy oracle-aided algebraic simulatability [BL22], which is a weaker assumption than full simulatability, the latter being the property usually required of DKG protocols in the literature. The scheme's security is established under strong unforgeability against adaptive adversaries, with a reduction to the one-more discrete logarithm assumption in the algebraic group and random oracle models.

From a practical perspective, threshold BLS signatures address a critical need in modern distributed systems where single-key signing creates unacceptable single points of failure. The proposed construction supports applications such as distributed certificate authorities, blockchain validator committees, custodial key-management systems, quorum-based hardware security modules, and Byzantine fault-tolerant consensus protocols. Its compact, unique signatures, non-interactive partial signing, and public share verification make it especially suitable for large-scale and performance-sensitive deployments.

By providing a complete and security-analyzed specification of an adaptively secure threshold signature scheme, this submission contributes to the Threshold Call's objective of building a public body of reference material for threshold cryptography. The proposed construction is intended to serve as a reference design for implementers and evaluators, supporting consistent implementation and comparative analysis of threshold signature designs across a range of real-world distributed systems, in line with the goals of NIST's Threshold Cryptography initiative.

# 2. Specification

The specification is organized to separate the cryptosystem definition, system model, and security formulation. The core threshold BLS signature cryptosystem is specified in terms of its algorithms and parameters, with concrete scheme instantiations described within this framework. The design emphasizes a modular separation between the signing layer and the key-generation layer, supporting interchangeable key-generation mechanisms, including trusted-dealer and distributed key generation setups, without specifying any concrete key-generation protocol.

## 2.1. System Model and Security Formulation

The system consists of $n$ parties denoted $P_1, \ldots, P_n$ and a threshold parameter $t < n/2$. The parties communicate asynchronously over pairwise authenticated channels. All operations are defined over cyclic groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of prime order $p$ with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, and a bilinear, non-degenerate, efficiently computable pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. A standardized hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{G}_1$, modeled as a random oracle, maps messages to $\mathbb{G}_1$ parameterized by a domain-separation tag DST. All scalar arithmetic is modulo $p$. Concatenation of byte strings is denoted by $\|$.

**Security model.** Security is defined via an unforgeability experiment against an *adaptive adversary* that may corrupt up to $t$ parties over time. Upon adaptively corrupting a party $P_i$, the adversary obtains its long-term key share $\mathsf{sk}_i$ and may thereafter control its behavior arbitrarily (e.g., issue malformed partial signatures, refuse to sign, etc.). The adversary may also query signing oracles for partial signatures on messages of its choice.

The scheme is *strongly unforgeable under adaptive corruptions* if no probabilistic polynomial-time adversary can produce a valid signature $\sigma^\star$ on any message $m^\star$ such that the tuple $(m^\star, \sigma^\star)$ was not obtained from any previous signing session, except with negligible probability in the security parameter. In particular, the adversary is allowed to query $m^\star$ for partial signatures, as long as the total number of partial signatures obtained from honest parties does not exceed $t - |\mathsf{Cor}|$, where $|\mathsf{Cor}| \subseteq [n]$ denotes the index set of corrupted parties at the end of the game. Otherwise, the adversary could reconstruct (or obtain) the signature $\sigma^\star$ from the available information, that is, from its own secret key shares and the partial signatures of the honest parties.

## 2.2. Proposed Cryptosystem

The threshold BLS signature scheme is defined as

$$\mathsf{tBLS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{ShareSign}, \mathsf{ShareVerify}, \mathsf{Combine}, \mathsf{Verify}).$$

- **Setup** $\mathsf{Setup}(1^\lambda)$**.** On input the cryptographic security parameter $\lambda$, select public parameters par by choosing a pairing-friendly elliptic curve at the desired security level (e.g., BLS12-381 for $\lambda \approx 128$-bit security), fixing generators $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, and instantiating the bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Define a standardized hash-to-curve function $\mathsf{H} : \{0,1\}^* \to \mathbb{G}_1$ per RFC 9380 [FSSWW23], parameterized by a domain-separation tag DST. Output

$$\mathsf{par} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e, \mathsf{H}, \mathsf{DST}).$$

- **Key Generation** $\mathsf{KeyGen}(\mathsf{par}, n, t)$**.** Two interchangeable modes are supported:
  - *Trusted dealer:* Sample a uniformly random degree-$t$ polynomial $f(X) \in \mathbb{Z}_p[X]$. Give each party $P_i$ its secret key share $\mathsf{sk}_i = f(i)$ and publish its verification key $\mathsf{pk}_i = g_2^{\mathsf{sk}_i}$. The global public key is $\mathsf{pk} = g_2^x$ where $x = f(0)$.
  - *Distributed key generation:* The parties execute a DKG to obtain public-secret key shares $(\mathsf{sk}_i, \mathsf{pk}_i)$ and the global public key pk, with $\mathsf{sk}_i$ known only to party $P_i$. The

adaptive security proof in [BL22] applies to any DKG satisfying *oracle-aided algebraic simulatability*, including efficient constructions such as Pedersen's DKG [Ped91].

- **Partial Signing** ShareSign$(\mathsf{par}, \mathsf{sk}_i, m)$**.** Compute $H \leftarrow \mathsf{H}(\mathsf{DST}\|m) \in \mathbb{G}_1$ and output the deterministic, non-interactive partial signature $\sigma_i = H^{\mathsf{sk}_i} \in \mathbb{G}_1$.
- **Share Verification** ShareVerify$(\mathsf{par}, \mathsf{pk}_i, m, \sigma_i)$**.** Recompute $H \leftarrow \mathsf{H}(\mathsf{DST}\|m)$ and accept if and only if $e(\sigma_i, g_2) = e(H, \mathsf{pk}_i)$. Invalid shares are rejected and may be logged with their indices.
- **Combining** Combine$(\mathsf{par}, \{(i, \sigma_i)\}_{i\in S})$**.** Let $S \subseteq [n]$ be a subset with $|S| \geq t{+}1$ and distinct indices. Compute Lagrange coefficients at zero: $\lambda_{i,S} = \prod_{j\neq i\in S} \frac{j}{j-i} \pmod p$. Output the combined signature $\sigma = \prod_{i\in S} \sigma_i^{\lambda_{i,S}} \in \mathbb{G}_1$.
- **Verification** Verify$(\mathsf{par}, \mathsf{pk}, m, \sigma)$**.** Recompute $H \leftarrow \mathsf{H}(\mathsf{DST}\|m)$ and accept if and only if $e(\sigma, g_2) = e(H, \mathsf{pk})$. This verification equation is identical to that of standard BLS.

### 2.2.1. Optional Extensions

The following optional extensions and optimizations may be supported for enhanced interoperability, robustness, or performance. These do not modify the core tBLS interface or its underlying security assumptions.

- **Proof of correctness (NIZK).** Each signer may attach a non-interactive zero-knowledge (NIZK) proof $\pi_i$ demonstrating the correctness of its partial signature $\sigma_i$ with respect to the verification key $\mathsf{pk}_i$. Specifically, $\pi_i$ proves knowledge of $\mathsf{sk}_i$ such that
$$\sigma_i = H^{\mathsf{sk}_i} \quad \text{and} \quad \mathsf{pk}_i = g_2^{\mathsf{sk}_i},$$
without revealing $\mathsf{sk}_i$. In this variant, the verification step replaces the pairing check $e(\sigma_i, g_2) = e(H, \mathsf{pk}_i)$ with a NIZK proof verification of $\pi_i$. This eliminates the need for pairings in share verification.
- **Batch share verification.** To improve efficiency when verifying multiple partial signatures $\{\sigma_i\}_{i\in S}$ on the same message $m$, the verifier may combine pairings and check a single equation:
$$e\left(\prod_{i\in S} \sigma_i^{r_i}, g_2\right) = e\left(H, \prod_{i\in S} \mathsf{pk}_i^{r_i}\right),$$
for random non-zero scalars $\{r_i\}_{i\in S}$. This optimization preserves soundness in the random oracle model. Batch signature share verification reduces the number of pairing equations from $|S|$ to 1 per batch of $|S|$ partial signatures, at the cost of two multi-exponentiations of degree $|S|$ over group elements.
- **Batch final verification.** Multiple final signatures $\{\sigma_j\}_{j\in[k]}$ on distinct messages $\{m_j\}_{j\in[k]}$ may be verified together by checking:
$$e\left(\prod_{j\in[k]} \sigma_j^{r_j}, g_2\right) = e\left(\prod_{j\in[k]} H_j^{r_j}, \mathsf{pk}\right),$$

where $H_j = \mathsf{H}(\mathsf{DST} \| m_j)$ and $\{r_j\}_{j \in [k]}$ are random non-zero scalars. Batch final signature verification reduces the number of pairing equations from $k$ to 1 per batch of $k$ signatures, at the cost of two multi-exponentiations of degree $k$ over group elements.

## 2.3. Security Guarantees

**Correctness.** For any qualified subset $S \subseteq [n]$ with $|S| \geq t+1$, we have

$$\sum_{i \in S} \lambda_{i,S} \cdot \mathsf{sk}_i = x \pmod{p}, \quad \sigma = \prod_{i \in S} \sigma_i^{\lambda_{i,S}} = H^x \in \mathbb{G}_1.$$

Hence $e(\sigma, g_2) = e(H, g_2^x) = e(H, \mathsf{pk})$, confirming correctness and compatibility with standard BLS verification.

**Underlying assumptions.** Following [BL22], security is analyzed in the algebraic group model [FKL18] and reduces to the one-more discrete logarithm (OMDL) assumption [BNPS03]. The hash-to-curve function is modeled as a random oracle instantiated according to RFC 9380. The key-generation phase may involve either a trusted dealer or any DKG protocol satisfying oracle-aided algebraic simulatability, as formalized in [BL22]; this class includes Pedersen's DKG [Ped91]. We note separately that the scheme achieves static security in the random oracle model under the computational Diffie-Hellman (CDH) assumption [Bol03]. Further, authenticated channels (and secure broadcast, when required) are assumed during the DKG phase.

**Security bounds.** For any probabilistic polynomial-time algebraic adversary $\mathcal{A}$, issuing at most $Q_{\mathsf{h}}$ random oracle queries and $Q_{\mathsf{sign}}$ partial-signing queries, adaptively corrupting at most $t$ parties, there is a reduction $\mathcal{B}$ to OMDL with

$$\mathsf{Adv}^{\text{uf-cma}}_{\mathcal{A}}(\lambda) \;\leq\; 4 \cdot \mathsf{Adv}^{\text{omdl}}_{\mathcal{B}}(\lambda) \;+\; 4\left(Q_{\mathsf{h}} + Q_{\mathsf{sign}}\right)^2 / p,$$

where $\mathcal{B}$'s running time is polynomially related to that of $\mathcal{A}$. For details, we refer to [BL22].

## 2.4. Analytic Complexity

**Memory complexity.** Each signer maintains a single scalar $\mathsf{sk}_i \in \mathbb{Z}_p$ and its public key $\mathsf{pk}_i \in \mathbb{G}_2$. The combiner holds the global public key pk, the set of participant keys $\{\mathsf{pk}_i\}_{i=1}^n$, and transient values such as the message hash $H \in \mathbb{G}_1$ and up to $t+1$ partial signatures per signing session. No long-term state beyond these elements is required.

**Computational complexity.** The computational cost of each algorithm in tBLS is dominated by scalar multiplications and pairings in the underlying groups. All operations are independent and parallelizable across participants. The final verification cost is independent of $t$ and $n$, and identical to that of a single-party BLS signature.

- **Share signing:** Each party $P_i$ performs a single exponentiation in $\mathbb{G}_1$ to compute its partial signature $\sigma_i = H^{\mathsf{sk}_i}$. This is a deterministic and non-interactive operation.

- **Share verification:** Any verifier (or the combiner) checks a partial signature via one pairing equation $e(\sigma_i, g_2) \stackrel{?}{=} e(H, \mathsf{pk}_i)$. Verification costs one pairing computation per share.
- **Combining:** The designated combiner combines $t{+}1$ valid shares by computing $t{+}1$ exponentiations and $t$ multiplications in $\mathbb{G}_1$ weighted by Lagrange coefficients $\lambda_{i,S}$. This step is deterministic and involves no pairings.
- **Final verification:** The combined signature is verified by one pairing check $e(\sigma, g_2) \stackrel{?}{=} e(H, \mathsf{pk})$. The cost matches standard BLS verification and is independent of $t$ and $n$.

**Round complexity.** After KeyGen, partial signing is fully non-interactive. Combining requires only receipt of $t{+}1$ valid shares and no further communication. This enables asynchronous and offline combination and verification.

# 3. Open-Source Implementation

**Code structure.** The core implementation is written in Go and focuses on the threshold BLS signing and verification logic, including partial signing, partial signature verification, and deterministic aggregation. The implementation relies on the gnark-crypto library [BPHTGK25] as an external dependency for finite-field arithmetic, elliptic-curve operations, and pairings over the BLS12-381 curve. Efficient multi-exponentiation in $\mathbb{G}$ is implemented using Pippenger's method [BDLO12]. The code is compiled using the standard Go toolchain, with default compiler optimizations enabled. Build and benchmarking are performed using Go's native build and test infrastructure, with dedicated benchmarking routines used to measure signing, partial signature verification, and aggregation performance.

**Code progress and availability.** An initial implementation of the proposed threshold signature scheme is complete and publicly available in a Git-compatible repository at https://github.com/sourav1547/adaptive-bls under the name "boldyreva 1".

**Implementation of the networking model.** The current implementation focuses on the cryptographic core and does not implement a full networking layer. Communication between parties and the aggregator is modeled by direct invocation of signing and verification routines. This abstraction isolates cryptographic costs from networking effects and allows reproducible benchmarking. In practical deployments, authenticated point-to-point channels (and broadcast functionality for a DKG, if used) are assumed to be provided by the underlying system or protocol framework and are orthogonal to the threshold signature construction itself.

**Testing.** Testing focuses on correctness, performance, and reproducibility of the cryptographic operations. Benchmarking is conducted on an Amazon Web Service (AWS) `t3.2xlarge` instance with 32 GB RAM, 8 virtual cores, and 2.50GHz CPU. Testing under malicious behavior is partially supported through validation of malformed or invalid partial signatures, which are rejected during partial signature verification. Adverse or non-optimal networking conditions are not explicitly simulated in the current implementation, as the signing and aggregation procedures are non-interactive after key generation; networking effects are therefore limited to message delivery and are expected to be handled at a higher protocol layer.

# 4. Experimental Performance Evaluation

**Performance.** The per-signer signing time is $0.81$ ms, corresponding to a single $\mathbb{G}_1$ exponentiation. The partial signature verification time is $1.12$ ms, corresponding to a single pairing check. Signatures and signature shares consist of single elements in $\mathbb{G}_1$ (48 bytes), while the public key and public key shares consist of single elements in $\mathbb{G}_2$ (96 bytes). The total aggregation time for a threshold parameter $t = 64$ is $74.01$ ms.

**Optimistic aggregation.** Aggregation can be optimized by first combining partial signatures without individual verification and then verifying only the aggregated signature. If the final verification succeeds, aggregation completes immediately; otherwise, the aggregator falls back to individual verification to identify and discard invalid shares before recomputing the aggregate. For our scheme, optimistic aggregation takes $7.7$ ms. Owing to the public verifiability of partial signatures, each fallback execution identifies at least one malicious party and never excludes an honest signer, so the fallback path is invoked at most $t$ times over the lifetime of the system.

**Platform.** Compared to the AWS t3.2xlarge instance used in our evaluation (8 virtual cores, 32 GB RAM), a baseline platform consisting of a single machine with 16 cores and 64 GB RAM primarily changes the available parallelism and memory capacity, rather than the underlying execution model. Increased core count and memory may allow higher concurrency when benchmarking multiple signers or aggregators. At the same time, as both platforms execute all parties on a single host, neither setup captures network latency, asynchronous execution, or adversarial scheduling effects inherent to distributed deployments; these aspects remain abstracted and are expected to be handled by higher protocol layers.

# 5. Licensing, Patent Claims, and Funding

**Open-source licenses.** The core implementation of the proposed threshold signature scheme is released under the MIT License. External dependencies used by the implementation are open-source and distributed under OSI-approved licenses, including the Apache License 2.0 for the gnark-crypto library. No proprietary or non-OSI-approved components are required.

**Patents.** To the best of our knowledge, there are no known patents with claims that cover the contents of this submission and that involve any team member as an inventor, applicant, or assignee, or that are held by an entity employing or sponsoring a team member.

**External funding.** Alexandra Boldyreva is supported by the National Science Foundation under Grant No. 2453434. Julian Loss is supported by the European Union, ERC-2023-StG-101116713. Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

# References


[BDLO12] Daniel J. Bernstein, Jeroen Doumen, Tanja Lange, and Jan-Jaap Oosterwijk. "Faster Batch Forgery Identification". In: *Progress in Cryptology - INDOCRYPT 2012*. Ed. by Steven Galbraith and Mridul Nandi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 454–473. DOI: 10.1007/978-3-642-34931-7_26. Also at ia.cr/2012/549.

[BL22] Renas Bacho and Julian Loss. "On the Adaptive Security of the Threshold BLS Signature Scheme". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2022, pp. 2895–2909. DOI: 10.1145/3548606.3560656. Also at ia.cr/2022/534.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *ASIACRYPT 2001*. Vol. 2248. LNCS. 2001, pp. 514–532. DOI: 10.1007/3-540-45682-1_30. URL: https://www.iacr.org/cryptodb/data/paper.php?pubkey=150.

[BNPS03] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. "The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme". In: *Journal of Cryptology* 16.2 (2003), pp. 185–215. DOI: 10.1007/s00145-002-0120-1. Also at ia.cr/2001/002.

[Bol03] Alexandra Boldyreva. "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie–Hellman-Group Signature Scheme". In: *PKC 2003*. Vol. 2567. LNCS. 2003, pp. 31–46. DOI: 10.1007/3-540-36288-6_3. URL: https://faculty.cc.gatech.edu/~aboldyre/papers/b.pdf.

[BPHTGK25] Gautam Botrel, Thomas Piellard, Youssef El Housni, Arya Tabaie, Gus Gutoski, and Ivo Kubjas. *Consensys/gnark-crypto: v0.15.0*. Version v0.15.0. January 2025. DOI: 10.5281/zenodo.5815453.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. "The Algebraic Group Model and its Applications". In: *Advances in Cryptology – CRYPTO 2018*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10992. Lecture Notes in Computer Science. Springer, 2018, pp. 33–62. DOI: 10.1007/978-3-319-96881-0_2. Also at ia.cr/2017/620.

[FSSWW23] Armando Faz-Hernández, Sam Scott, Nick Sullivan, Riad S. Wahby, and Christopher A. Wood. *RFC 9380: Hashing to Elliptic Curves*. RFC 9380, Internet Engineering Task Force (IETF). Crypto Forum Research Group (CFRG) Informational RFC. August 2023. DOI: 10.17487/RFC9380. URL: https://www.rfc-editor.org/rfc/rfc9380.html.

[Ped91] Torben P. Pedersen. "A Threshold Cryptosystem without a Trusted Party". In: *EUROCRYPT 1991*. Vol. 547. LNCS. 1991, pp. 522–526. DOI: 10.1007/3-540-46416-6_47.

[NIST-IR8214C] Luís T. A. N. Brandão and René Peralta. *NIST First Call for Multi-Party Threshold Schemes*. (National Institute of Standards and Technology) NIST Internal Report (NISTIR) 8214C. 2026. DOI: 10.6028/NIST.IR.8214C.