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

Title: Lemur: Scalable Post-Quantum Synchronized Multi-Signatures

Subtitle: Aggregatable, Batch-Verifiable, and Threshold-Friendly Post-Quantum Signatures

Version: Preview Writeup 0.2 (2026-05-18)[1]

Team name: Lemur: Lemur Multi-Signature Team

Team members: Muhammed F. Esgin, Yini Lin, Markku-Juhani O. Saarinen, Amin Sakzad, Ron Steinfeld

Abstract: We present Lemur, a lattice-based synchronized multi-signature scheme with non-interactive aggregation that is particularly well suited for high-throughput blockchain consensus and other settings requiring efficient signature compression from large signer sets. Lemur follows the Chipmunk (ACM CCS'23) blueprint but fundamentally redesigns the security foundations of its building blocks: a compact key-homomorphic one-time signature based on the new Dual Hint-MLWE assumption (with a formal reduction from the standard Module-LWE), and a generalized homomorphic vector commitment from Module-SIS. At 128-bit post-quantum security, the current Rust implementation profile yields an aggregate of about 384 KB for one million individual signatures (versus Chipmunk's $\sim$831 KB), while keeping stateful signing millisecond-scale and aggregated verification in the tens of milliseconds for 1024 signers. In the context of the NIST Threshold Call, Lemur targets Category S1 (Special Signing), offering threshold-friendly, aggregatable, and batch-verifiable signatures from lattice assumptions. Any subset of signers can produce valid individual signatures, and a verifier can enforce a threshold policy at verification time without the signers coordinating interactively.

Proposed crypto-systems: (I) Lemur: Scalable Post-Quantum Synchronized Multi-Signatures (Categories S1).

Keywords: Threshold Cryptography; NIST Threshold Call; Multi-Signatures; Lattice-Based Cryptography; Post-Quantum Cryptography; Signature Aggregation

Document License: CC BY 4.0 International

[1] Updated version submitted to NIST-MPTC during the extended preview-writeup submission period

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 [BP26]. 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: Muhammed F. Esgin [i1,a1], Yini Lin [i2,a1], Markku-Juhani O. Saarinen [i3,a2], Amin Sakzad [i4,a1], Ron Steinfeld [i5,a1]

Open Researcher and Contributor Identifiers (ORCID):

i1 (0000-0003-1650-3748); i2 (0009-0006-8837-5158); i3 (0000-0002-2555-235X); i4 (0000-0003-4569-3384); i5 (0000-0003-1745-4183)

Affiliations: [a1] Faculty of Information Technology, Monash University @ Melbourne, Australia

[a2] Faculty of Information Technology and Communication Sciences, Tampere University @ Tampere, Finland

Main contacts:

- **Team mailing list:** lemur-sig@googlegroups.com
- **Primary technical contact person:** Yini Lin, yini.lin@monash.edu
- **Secondary contact person:** Markku-Juhani O. Saarinen, markku-juhani.saarinen@tuni.fi

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

Lemur is a lattice-based synchronized multi-signature scheme designed for settings where a large number of signers independently sign the same message within a synchronized time step, and their signatures are non-interactively aggregated into a single compact result. The original motivating application for Lemur is proof-of-stake blockchain consensus, where thousands of validators attest to each block; Lemur compresses these attestations into a single aggregate signature that is sublinear in the number of signers.

Lemur targets **Category S1** (Special Signing) of the NIST Threshold Call [BP26], satisfying several of the featured properties listed in Section 11.1 of the call:

1. **Threshold friendly and quantum resistant.** Lemur is based on the Dual Hint-MLWE and Module-SIS lattice assumptions, which are believed to be quantum resistant. We provide a formal reduction from the standard Module-LWE problem to Dual Hint-MLWE, basing Lemur's security on Module-LWE overall. Each signer holds an independent key pair and can produce an individual signature without coordinating with other signers. A verifier enforces the threshold policy at verification time: given an aggregated signature from any subset of signers, the verifier checks whether the subset satisfies a desired threshold condition.
2. **Aggregatable** (feature 5 in the call). Multiple signatures aggregate into a sublinear-size result. At 128-bit security, aggregating one million signatures yields an aggregate of about 378 KB under the current implementation profile, less than half the size of Chipmunk's ∼831 KB. The aggregation of the key-homomorphic one-time signature (KOTS), one of the two main building blocks of Lemur, is extremely compact, requiring less than 8 KB for even one million signers.
3. **Batch verifiable** (feature 6 in the call). A verifier can efficiently check an aggregated signature against a set of public keys in a single pass, without processing each individual signature separately.

Compared to NIST-standardized signatures (ML-DSA, SLH-DSA), the distinctive factor (per Requirement 11.1) is the non-interactive aggregation property: no NIST signature scheme supports sublinear-size compression of multiple signatures on the same message. Lemur fills this gap with post-quantum security from lattices.

The conventional (non-threshold) multi-signature primitive is fully specified in the Lemur technical paper [LESSS26], including algorithms for Setup, KeyGen, Sign, Aggregate, and Verify, with concrete parameter sets and security proofs in the random oracle model.

Blockchain Context. In proof-of-stake blockchains such as Ethereum and DFinity, thousands of validators independently attest to each block. Storing all attestations individually is impractical — a full Ethereum block averages around 180 KB.[2] Squirrel [FSZ22] aggregates 4096 signatures into 771 KB at 112-bit security, which alone exceeds four full blocks. Chipmunk [FHSZ23] reduced this to 136 KB for 8192 signers at 112-bit security, but its concrete parameters rely on Ring-SIS estimates that, when re-evaluated under the state-of-the-art Lattice Estimator[3], only deliver

[2] https://etherscan.io/chart/blocksize
[3] https://github.com/malb/lattice-estimator

$\sim$30-bit security rather than the claimed 112-bit. Re-instantiating Chipmunk at the standard 128-bit level (root Hermite factor $\delta \approx 1.0045$) requires a ring dimension $d = 1024$ and pushes its aggregated signature beyond 800 KB at one million signers. Lemur overcomes these bottlenecks by shifting to computational security (Dual Hint-MLWE) and to Module-SIS with a smaller ring dimension ($d = 256$), achieving less than half the aggregate size of Chipmunk at one million signers at 128-bit post-quantum security under the current implementation profile.

2. Specification

2.1. System Model and Threshold Profile

Lemur operates in the *synchronized* multi-signature setting: signers are indexed by a signer set $\mathcal{S}$ with $|\mathcal{S}| = N$, and time is divided into discrete steps $t \in \{0, \ldots, 2^\tau - 1\}$. Each signer independently generates a key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ during setup, producing 2^τ one-time key pairs internally. Signing at time t consumes one-time key $(\mathsf{sk}_t, \mathsf{pk}_t)$ and outputs an individual signature.

The *threshold profile* is verifier-chosen: given an aggregated signature over a subset $C \subseteq \mathcal{S}$, the verifier checks whether $|C| \geq T$ for the desired threshold T. This model is strictly more flexible than a fixed T-of-n threshold set at key generation time, since the same public keys and signatures support any threshold policy without re-keying.

No trusted setup is required beyond public parameters (a random matrix sampled from a hash function). No interaction between signers is needed during signing. The aggregator is untrusted for unforgeability (it cannot produce forgeries) but is trusted for liveness. The role of the aggregator can be played by anyone and can change over time.

2.2. Building Blocks

The Lemur construction is built from two novel lattice-based primitives:

1. **Key-Homomorphic One-Time Signature (KOTS).** Each signer samples a discrete-Gaussian secret matrix $\mathbf{S} \in \mathcal{R}^{k \times m}$ and publishes $\mathbf{T} = \mathbf{SA} \bmod q'$ as the public key, for a public matrix $\mathbf{A} \in \mathcal{R}_{q'}^{m \times n}$. A signature on a message μ is $\mathbf{z}^\top = \mathbf{h}^\top \mathbf{S}$, where $\mathbf{h}$ is a short vector derived from μ via a random oracle. The key-homomorphic property enables aggregation: given signatures $\sigma_1, \ldots, \sigma_N$ under keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$, one can compute an aggregate signature $\sum w_i \cdot \sigma_i$ that verifies against $\sum w_i \cdot \mathsf{pk}_i$ for hash-derived weights w_i. Security relies on the new *Dual Hint-MLWE* assumption, which we formally prove as hard as the standard Module-LWE.

2. **Homomorphic Vector Commitment (HVC).** A Merkle-tree commitment based on Ajtai hashing over Module-SIS, where each signer commits to their vector of 2^τ one-time public keys. The homomorphic property allows openings to aggregate: given N openings for position t, a weighted sum produces a valid aggregated opening for the aggregated commitment. We generalize Chipmunk's HVC from Ring-SIS to Module-SIS and from

vector to matrix message spaces, enabling the use of a smaller ring dimension $d = 256$ (versus $d = 1024$ for Chipmunk re-instantiated at 128-bit security).

2.3. Security

Lemur achieves existential unforgeability (EUF) against rogue-key attacks in the random oracle model. The security reduction proceeds in two cases: either the adversary breaks HVC position binding (reduced to Module-SIS hardness) or it breaks KOTS multi-user unforgeability under rerandomized keys (reduced to Dual Hint-MLWE, itself reduced to Module-LWE, and Module-SIS).

The KOTS component relies on the Dual Hint-MLWE assumption, a new problem introduced in the Lemur paper [LESSS26]. We prove a formal reduction from the standard Module-LWE (MLWE) problem to Dual Hint-MLWE, establishing its hardness under well-studied lattice assumptions. The HVC component relies directly on Module-SIS. Concrete parameters target ≥ 128-bit post-quantum security, estimated via the Lattice Estimator with root Hermite factor $\delta \leq 1.0045$.

Byzantine signers cannot produce forgeries: (i) producing a valid individual signature is computationally infeasible, and (ii) producing a valid aggregate signature even with full control of all-but-one keys is also computationally infeasible. The aggregator verifies each individual signature before combining, so an invalid individual signature is rejected in aggregation.

3. Open-Source Implementation

1. **Code structure.** The implementation [Lem26] consists of two independent but fully interoperable codebases:

   - A **Rust** implementation (`lemur-rs`), serving as the primary reference implementation with release-optimized builds. Ring arithmetic uses a Montgomery-domain NTT for the HVC modulus and a CRT-via-auxiliary-primes backend for the KOTS modulus $q' \equiv 17 \pmod{32}$, which is not natively NTT-friendly.
   - A **Python** implementation (`lemur-py`), serving as a readable prototype and byte-level test oracle.

   Both implementations ship the fixed parameter cell $(d, k, m, n) = (256, 4, 9, 4)$ at $\tau = 20$ and cover the full Lemur pipeline: ring arithmetic, KOTS (keygen, sign, verify), HVC (commit, open, aggregate), the top-level multi-signature (setup, keygen, sign, aggregate, verify), and bit-packed serialization (fixed-width plus Golomb–Rice for components whose folded-Gaussian distribution makes Rice strictly shorter).

2. **Signer state.** The default signer uses a BDS08 auth-path traversal cache of roughly 134 KB at $\tau = 20$, giving amortized $O(\tau)$ signing from a 32-byte master seed. An optional in-memory materialized HVC tree ($\sim$8 GiB at $\tau = 20$ under the current profile, storing one projected node label as `i64` coefficients per tree node) provides millisecond-scale signing for callers that can afford the RAM; this tree is rebuilt from the master seed on demand and never persisted to disk. Chipmunk, by contrast, only supports the materialized-tree path.

3. **Side-channel considerations.** Lemur's signing procedure is structurally well-suited to side-channel-resistant implementation: the online signing step reduces to the linear map $\mathbf{Z} = \mathbf{h}^\top \mathbf{S}$, which admits straightforward arithmetic masking when needed. The current Rust implementation already takes several basic precautions: secret material is zeroized on drop via the `zeroize` crate, modular reductions in the hot inner loops use branchless signed-mask idioms (data-independent under LLVM auto-vectorization), and the discrete-Gaussian secret sampler uses a CDT-based design with a fixed XOF byte budget per coefficient and fixed precision and tail-cut, pinned by a Rényi-divergence analysis at $\lambda = 128$. The CDT lookup itself is currently a data-dependent binary search and is not yet constant-time; hardening it (e.g. to a fully-traversed table) is a planned implementation refinement. These measures address timing leakage in the most obvious hot paths, but the present codebase is a research artifact: full coverage against power, EM, fault, and microarchitectural side-channels has not been audited and is left for a future hardened implementation.

4. **Code progress and availability.** Both implementations exist, are functional, and are byte-for-byte interoperable through shared JSON test vectors. The Rust crate includes integration tests for serialization, stateful signing, and robustness against malformed inputs, plus benchmarking binaries for sizes, signing, aggregation, and large-N batch verification. The public Git repository [Lem26] is available at https://github.com/lemur-sig/lemur.

5. **Networking model.** Lemur's signing phase requires no interaction between signers. The aggregator collects partial signatures via point-to-point channels (or a bulletin board) and computes the aggregate locally. No broadcast channel is needed. In practice, the aggregator is a blockchain node that receives attestations over the peer-to-peer network.

6. **Testing.** The Python implementation provides cross-validation against the Rust implementation by generating common JSON vector output and comparing the two implementations byte-for-byte. Malicious-signer testing is straightforward: invalid partial signatures fail individual verification and are excluded before aggregation. No special adversarial testing framework is needed since the aggregator performs only public operations.

4. Experimental Performance Evaluation

1. **Performance.** Table 1 summarizes aggregated signature sizes at 128-bit post-quantum security, comparing Lemur ($d = 256$) with Chipmunk ($d = 1024$).[4] Lemur uses $\tau = 20$ (approximately one million time slots). The Lemur entries are predicted encoded sizes: the $N = 2^{10}$ cell is computed by the Rust implementation's sizing tool from the encoder's per-component formulas (the realized Rice-coded length of any single aggregate fluctuates by a few percent around this prediction because Rice output length is data-dependent), and the larger N entries are the corresponding Rice-encoded sizes derived from the parameter estimator.

[4] Re-evaluating the 128-bit Chipmunk parameter set ($d = 512$) proposed in [FHSZ23] under the Dilithium MSIS estimation framework yields only ∼40-bit security. We therefore use the updated $d = 1024$ Chipmunk parameter set required to actually reach 128-bit security at root Hermite factor $\delta = 1.0045$.

Table 1: Aggregated signature sizes at 128-bit security ($\tau = 20$).

N (signers)	Chipmunk (KB)		Lemur (KB)	
	Sig	Total	Sig	Total
2^{10} (1K)	26	237	5.6	190.3
2^{15} (32K)	>90	>418	7.2	268.9
2^{20} (1M)	>110	>831	7.8	378.4

Key observations:

- The KOTS aggregate component grows slowly with N, from 5.6 KB at 2^{10} signers to 7.8 KB at 2^{20} signers, versus 26 to >110 KB for Chipmunk.
- At $N = 2^{20}$, Lemur's total aggregate (378.4 KB) remains less than half of Chipmunk's (>831 KB).
- Chipmunk's numbers for large N are lower bounds because feasible parameters at $d = 1024$ require relaxing the KOTS root Hermite factor above the 128-bit target (δ_{KOTS} up to 1.00501 at $N = 2^{20}$). Lemur maintains $\delta \leq 1.0045$ throughout.

Representative Rust timings at $\tau = 20$, measured with 24 hardware threads (AMD Ryzen AI 9 HX 370) and release builds without target-specific compiler flags, are about 4.6 ms per stateful signature using the BDS08 traversal cache and about 1.5 ms when the full HVC tree is materialized in memory. Batch verification of one aggregated signature takes about 14.9 ms for $N = 2^{10}$ signers, about 0.47 s for $N = 2^{15}$, and about 14.0 s for $N = 2^{20}$ (linear extrapolations anchored on the $N = 2^{13}$ measurement). Secure aggregation (which preverifies each partial signature before combining) takes 1.12 s at $N = 2^{10}$ and scales roughly linearly with N.

2. **Platform.** All operations are single-machine computations (no networking required for individual signing or aggregation). Signing is per-party and parallelizes trivially via `rayon`; aggregation is a weighted linear combination whose memory footprint scales linearly with N. Most measurements use a 24-thread server-class machine; the implementation also runs on commodity laptops, where stateful signing remains in the millisecond range.

5. Licensing, Patent Claims, and Funding

1. The Rust and Python implementations will be released under a permissive open-source license (Apache-2.0 or MIT), and depend only on standard permissively-licensed packages.
2. The authors are not aware of any patent claims covering the techniques used in Lemur.
3. This work is partially funded by the Ethereum Foundation. Additional funding acknowledgments will be included in the package submission.

6. Comparison with Other Submissions

Among the first-round preview writeups (January 2026), the following submissions target Category S1:

- **Tanuki** [Tan26]: Threshold Raccoon signatures from MLWE/MSIS, t-of-n with $n \leq 1024$, two signing rounds.
- **Hermine** [Her26]: Threshold Raccoon with identifiable aborts via Vandermonde sharing, $N \leq 64$, two rounds.
- **Gargos** [BDLR26]: Threshold Schnorr from DDH, three rounds, not post-quantum.
- **tBLS** [BBDL26]: Threshold BLS, non-interactive but pairing-based, not post-quantum.
- **LEAST** [LEA26]: Code-based threshold LESS signatures, Round Robin sequential protocol.
- **Vinaigrette** [Vin26]: Threshold UOV/MAYO from multivariate assumptions, MPC-based.

Lemur is *complementary* rather than competing with these submissions. The key distinctions are:

Table 2: Comparison of S1 submissions on features from Section 11.1.

Feature	Lemur	Tanuki	Hermine	LEAST	Vinaigrette	tBLS
Post-quantum	Yes	Yes	Yes	Yes	Yes	No
Aggregatable (5)	**Yes**	No	No	No	No	No
Batch verif. (6)	**Yes**	No	No	No	No	No
Non-interactive	Yes	No	No	No	No	Yes
Scalability (N)	$\geq 10^6$	$\leq 10^3$	≤ 64	small	flexible	flexible

No other first-round submission covers the aggregation (feature 5) and batch verification (feature 6) properties listed in Section 11.1 of the call. Lemur's multi-signature model, where each signer holds an independent key and the verifier enforces the threshold, is distinct from the shared-key threshold model used by all other S1 submissions. The two models serve different deployment scenarios and are not interchangeable, making Lemur a unique addition to the S1 category coverage.

References

[BBDL26] Renas Bacho, Alexandra Boldyreva, Sourav Das, and Julian Loss. *tBLS: Threshold BLS Signature Scheme*. NIST Threshold Call Preview Writeup. 2026.

[BDLR26] Renas Bacho, Sourav Das, Julian Loss, and Ling Ren. *Gargos: Threshold Schnorr Signature Scheme*. NIST Threshold Call Preview Writeup. 2026.

[BP26] Luís T. A. N. Brandão and René Peralta. *NIST First Call for Multi-Party Threshold Schemes*. NIST Internal Report 8214C. National Institute of Standards and Technology, 2026. DOI: 10.6028/NIST.IR.8214C.

[FHSZ23] Nils Fleischhacker, Gottfried Herold, Mark Simkin, and Zhenfei Zhang. "Chipmunk: Better Synchronized Multi-Signatures from Lattices". In: *ACM CCS*. ACM, 2023, pp. 386–400. DOI: 10.1145/3576915.3623219. URL: https://eprint.iacr.org/2023/1820.

[FSZ22] Nils Fleischhacker, Mark Simkin, and Zhenfei Zhang. "Squirrel: Efficient Synchronized Multi-Signatures from Lattices". In: *ACM CCS*. ACM, 2022, pp. 1109–1123. DOI: 10.1145/3548606.3560655. URL: https://eprint.iacr.org/2022/694.

[Her26] Hermine Team. *Hermine: An Efficient Raccoon-Style Non-Interactive Threshold Signature*. NIST Threshold Call Preview Writeup. 2026.

[LEA26] LEAST-project. *LEAST: Linear Equivalence Action Threshold Signature*. NIST Threshold Call Preview Writeup. 2026.

[Lem26] Lemur Team. *Lemur: Scalable Post-Quantum Synchronized Multi-Signatures*. Public Reference Source Code. 2026. URL: https://github.com/lemur-sig/lemur.

[LESSS26] Yini Lin, Muhammed F. Esgin, Markku-Juhani O. Saarinen, Amin Sakzad, and Ron Steinfeld. *Lemur: Scalable Post-Quantum Synchronized Multi-Signatures*. Full version in preparation. 2026.

[Tan26] Tanuki Team. *Tanuki: Two-round Threshold Signatures from Lattices*. NIST Threshold Call Preview Writeup. 2026.

[Vin26] Vinaigrette Team. *Vinaigrette: An MPC Approach for Threshold UOV and MAYO*. NIST Threshold Call Preview Writeup. 2026.