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

Title: KU

Subtitle: Honest-Majority Threshold ECDSA for Key Management Networks

Version: Preview Writeup 1.0 (2026-07-27)[1]

Team name: Dfns-KU

Team members: Jonathan Katz, Antoine Urban, Denis Varlakov, Nikita Sorokovikov

Abstract: This document outlines KU, a submission to the NIST Threshold Call for ECDSA Signing (Category N1.2). KU is an honest-majority threshold ECDSA protocol featuring batch generation of key-independent presignatures, a capability specifically designed for Key Management Networks (KMNs). By pre-generating a global pool of presignatures that can be instantly and non-interactively used with any shared key, KU eliminates per-key management overhead. This allows KMNs to efficiently secure millions of keys with ultra-low latency for generating standard ECDSA signatures.

Proposed crypto-systems:

- (I) KU (Categories N1.2)

Keywords: Threshold Cryptography; NIST Threshold Call; ECDSA Signature

[1]Version submitted to NIST-MPTC for publication

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: Jonathan Katz [i1,a2*], Antoine Urban [i2,a1], Denis Varlakov [i3,a1], Nikita Sorokovikov [i4,a1]

Open Researcher and Contributor Identifiers (ORCID):

i1 (0000-0001-6084-9303); i2 (0009-0000-3670-0146); i3 (0009-0000-8497-3536); i4 (0009-0003-6315-4269)

Affiliations:

[a1] Dfns Labs, France

[a2] Google, United States

Associateship clarifications:

[*] Developed the protocol while previously employed at Dfns.

Main contacts:

- **Team mailing list:** nist-submission-ku@dfns.co
- **Primary technical contact person:** Antoine Urban, antoine@dfns.co
- **Secondary contact person:** Jonathan Katz, jkcrypto@google.com

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

The planned submission package will specify a threshold ECDSA signing protocol based on the KU scheme [KU25]. The upcoming submission of this threshold scheme fits within Category N1.2 (ECDSA Signing) of the NIST Threshold Call [NIST-IR8214C].

Threshold ECDSA has received significant attention in recent years, primarily driven by applications to secure custody of cryptocurrency assets and, more broadly, to *key-management networks* (KMNs). In a KMN a fixed set of n servers jointly holds a large number of ECDSA keys—potentially millions—on behalf of a large number of users, with each key secret-shared so that no single server holds it in the clear. To keep signing latency low at this scale, modern threshold ECDSA protocols rely on *presignatures* to divide the workload into two distinct stages. During the **offline phase** (presignature generation), the parties interact to produce shared state before any message to be signed is known. Later, during the **online phase** (when a signing request arrives), each party uses its presignature share to locally compute a signature share that it then sends to a coordinator. Crucially, no inter-server communication is needed during the online signing process.

Most recent protocols supporting presigning, notably Lindell–Nof [LN17], Doerner et al. [DKLS19; DKLS23], and CGGMP [CGGMP20], target the dishonest-majority setting ($t < n$). In those protocols, each presignature is inherently tied to a specific key, and batch presignature generation at amortized rates significantly below the single-instance cost is not supported. For a KMN holding millions of keys, this means either pre-generating at least one presignature per key or accepting high latency when a signature is requested for a key that lacks an associated presignature.

In KMNs, however, the server set is fixed, curated, and typically operated across independent trust domains, making an honest-majority assumption ($t < n/2$) plausible. The KU protocol [KU25] takes advantage of this assumption to unlock two properties that are difficult to achieve otherwise:

1. **Key-independent presignatures.** Each presignature can be used with *any* key the servers share, eliminating the need to pre-generate or store one or more presignatures per key (which may number in the millions), or to accept high latency when a signature is requested for a key lacking a presignature.
2. **Batch presignature generation.** The protocol can generate $m \gg 1$ presignatures at an aggregate cost far below m independent presignature generations, yielding dramatic amortized speedups.

The output is a standard ECDSA signature verifiable by the usual single-party verification algorithm.

Robustness. The protocol achieves security with abort: unforgeability and key secrecy hold against a malicious adversary corrupting up to t servers, but output delivery is not guaranteed. A corrupted party can force an execution to abort, and the abort is not identifiable. Honest parties detect cheating but cannot attribute it.

Comparison with other submissions. To our knowledge, two other teams plan submissions of threshold-ECDSA protocols for arbitrary n: Fireblocks–3MI Labs (CGGMP [CGGMP20]) and BiCyCCLiST (THE CLASH [CCLST20]). Both target the dishonest-majority setting ($t < n$), require broadcast, and produce key-dependent presignatures; they differ in their underlying

cryptographic machinery (Paillier for CGGMP, class groups for THE CLASH). CGGMP additionally provides identifiable abort. KU is complementary: it trades the dishonest-majority guarantee for an honest-majority assumption ($t < n/2$) which in turn enables key-independent and batch-generated presignatures over point-to-point channels without broadcast. The three submissions share no building blocks.

2. Specification

2.1. System Model

We assume the following when describing KU:

- **Parties and honest majority.** There are n servers $P_1, \ldots, P_n$, of which an adversary may statically corrupt up to $t < n/2$. The adversary fixes its set of corruptions before protocol execution begins. For clarity, the existing security analysis [KU25] assumes $n = 2t + 1$; the protocol remains secure for any $n > 2t$, with straightforward adjustments to the ideal functionalities. Since KU leverages Pseudo-Random Secret Sharing (PRSS) to generate random sharings without interactions, which has exponential complexity in n, the protocol explicitly targets deployments where the number of signing nodes is relatively small (e.g., $n < 20$), even as the number of managed keys scales into the millions.
- **Idealized key generation.** We model key generation as an idealized functionality that outputs $(t + 1)$-out-of-n Shamir secret shares of one or more ECDSA private keys corresponding to some public keys, where n total parties are initialized with their respective secret signing key share, public keys, and the threshold t required to perform signing.
- **Coordinator.** A semi-honest coordinator orchestrates communication among the parties. It is important to note that the semi-honest assumption is required *only for availability*; a malicious coordinator can only force a protocol abort and cannot compromise key secrecy or forge signatures. In our (standard) model, the coordinator enforces that parties sign the same message, use the correct key shares, and never reuse a presignature. We model this by treating the coordinator as honest for protocol execution, but assume that the adversary may observe all messages the coordinator receives.
- **Communication model.** The parties and coordinator communicate via a synchronous network of pairwise secure channels. Crucially, we do not require a broadcast channel. Instead, messages can simply be relayed by the coordinator without requiring direct, encrypted point-to-point routing between every pair of signers. This approach minimizes pairwise networking overhead and naturally justifies our coordinator-centric architecture.

2.2. Protocol Approach

Modular Construction.

Our protocol utilizes a highly modular structure where each underlying functionality can be analyzed, implemented, and tested independently. We rely on two main ideal functionalities:

1. $\mathcal{F}_{\text{rss}}$ **(random secret sharing).** Provides $(t+1)$-out-of-n Shamir shares of random values and $(2t+1)$-out-of-n shares of zero.
2. $\mathcal{F}_{\text{triple}}$ **(batch multiplication triples).** Produces m triples $\{a_i, k_i, w_i = a_i \cdot k_i)\}_{i=1}^{m}$ in a single batch. This is what enables the amortized performance gains described above.

The main protocol Π_{ECDSA} is then proved secure in the $\{\mathcal{F}_{\text{rss}}, \mathcal{F}_{\text{triple}}\}$-hybrid model.

Our realization. In [KU25], $\mathcal{F}_{\text{rss}}$ is realized via pseudorandom secret sharing (PRSS) and pseudo-random secret sharing (PRZS) [CDI05], without a trusted dealer or broadcast channel. $\mathcal{F}_{\text{triple}}$ is realized from $\mathcal{F}_{\text{rss}}$ via an intermediate weak-multiplication functionality $\mathcal{F}_{\text{wmult}}$ that produces shares of products up to adversarial additive shifts, adapted from Chida et al. [CHIKGLN23]. A polynomial batch-verification check later detects any shift.

Offline Presigning phase. To generate m presignatures, the parties:

1. **Initialize a random-secret-sharing functionality** $\mathcal{F}_{\text{rss}}$, which provides $(t+1)$-out-of-n shares of random values and $(2t+1)$-out-of-n shares of zero.
2. **Generate multiplication triples.** Parties execute the batch triple-generation protocol, so each party P_j obtains secret shares of m triples $\{a_{i,j}, k_{i,j}, w_{i,j}\}_{i=1}^{m}$, where the underlying values satisfy $w_i = a_i \cdot k_i$. A batch verification check is performed; if it fails, the parties abort.
3. **Derive presignatures.** Each party P_j publishes $w_{i,j}$ and the group element $R_{i,j} = g^{k_{i,j}}$. From the reconstructed values w_i and R_i, each party computes $r_i = F(R_i)$ and its presignature share $k'_{i,j} = w_i^{-1} \cdot a_{i,j}$.

Since none of the above steps involve any specific private key, the resulting presignatures are key-independent and can later be used with any key shared by the parties. The entire batch presigning phase requires only *4 offline rounds*, making it highly efficient over standard networks.

Online signing phase. Given a hashed message h, a designated presignature index i, and a key share x_j, each party P_j computes a single signature share $s_j = k'_{i,j} \cdot (h + r_i \cdot x_j) + o_{i,j}$ (where $o_{i,j}$ is a zero-share that masks the individual contributions) and sends (r_i, s_i) to the coordinator. The coordinator interpolates the shares and outputs a standard ECDSA signature (r_i, s). This reduces the online signing phase to exactly **1 round** of communication; no party-to-party interaction is required during the actual signing process.

2.3. Security Property

KU has been proven to securely realize an ideal ECDSA signing functionality $\mathcal{F}_{\text{ECDSA}}$ in the $\{\mathcal{F}_{\text{rss}}, \mathcal{F}_{\text{triple}}\}$-hybrid model, against a static, malicious adversary corrupting up to $t < n/2$ parties (together with the semi-honest coordinator). The protocol guarantees the output of standard, unbiasable signatures; while an adversary can force an abort, they cannot inject bias into successfully generated signatures. Additionally, the threshold parameters t and n used for

the underlying Shamir secret-sharing of the signing key are identical to those used during the threshold signing execution. The proof does not use rewinding and carries over to the UC setting.

3. Reference Implementation

The reference implementation is fully functional for all core phases and is implemented entirely in Rust. The architecture is built upon the LFDT-Lockness/round-based[2] framework, providing a highly modular structure that mirrors the specification's separation of functionalities. The implementation covers the batch generation of key-independent presignatures, the non-interactive online signing phase, and the coordinator's orchestration logic.

3.1. Implementation of the Networking Model

For both LAN and WAN experiments, parties communicate through a central Coordinator using direct TCP streams. The Coordinator relays messages and routes presignature indices. In the WAN setting, parties are deployed as independent peers on geographically distributed hosts.

3.2. Testing

A primary challenge in testing threshold protocols is ensuring deterministic execution for reproducibility. We address this by allowing the underlying functionalities (such as $\mathcal{F}_{rss}$) and all operations to be initialized with fixed random seeds during local unit tests.

Because our protocol requires the active cooperation of all parties, we will evaluate the implementation's fault handling by simulating up to t corrupted parties. These tests are designed to verify that both malformed inputs and network-level disruptions, such as dropped TCP streams, reliably trigger a protocol abort.

4. Experimental Evaluation

Our benchmarks evaluate the KU protocol against the CGGMP [CGGMP20] protocol (implemented in the same language and environment) to contextualize the honest-majority advantage. Tests were conducted on common AWS EC2 c5.large instances.

The Report on Experimental Evaluation will cover performance measurements across configurations indexed by $(n, t, m,$ network delay$)$, reporting:

- *Perf1 (Memory):* Bytes stored per party during presigning and signing.

[2] https://github.com/LFDT-Lockness/round-based

- *Perf2 (Processing):* End-to-end wall-clock time for presigning and signing, broken down by phase.
- *Perf3 (Communication):* Bytes communicated per presignature and per signature.

4.1. Preliminary Results

We provide a direct performance evaluation that compares our Rust implementation of KU with [CGGMP20]. To clarify our performance metrics, we divide the execution into an *offline phase* (presignature generation) and an *online phase* (non-interactive signing).

Table 1 reports the local computation time for the complete end-to-end process (offline + online) without network round-trips. Tables 2 and 3 include coordinator communication latency, focusing on the amortized offline costs and the maximum online throughput, respectively. For offline batch presignature generation, KU yields massive amortized speedups, dropping to 1.3 ms per presignature for a batch of $10,000$. By combining this high-speed batch generation with our global pool of key-independent presignatures, KU sustains a high-volume throughput of up to ≈ 340 signatures per second.

1. *Local Execution (Offline + Online):* KU demonstrates extremely fast raw MPC execution without deployment or communication overhead. Table 1 details the end-to-end local computation time required to generate a single signature from scratch.

[CGGMP20] **(ms)**	**KU (ms)**
1067	15

Table 1: Local end-to-end signing latency ($n = 5, t = 2$, batch size $m = 1$) on a Mac M1.

2. *Offline Presignature Generation:* For batch generation, KU yields massive amortized speedups, dropping to 1.3 ms per presignature for a batch of 10,000.

Batch Size	[CGGMP20]	**KU**
1	2990ms	30ms
100	-	1.0ms/presig
10000	-	1.3ms/presig

Table 2: Amortized presignature latency ($n = 5, t = 2$) on EC2 c5.large instances (2 vCPU of Intel(R) Xeon(R) Platinum 8124M CPU @ 3.00GHz, 1 core, 4 GiB of RAM) deployed in a single Region.

3. *Sustained Throughput:* Table 3 demonstrates the practical impact of our protocol in a high-volume setting. When executing the entire protocol from scratch (Offline + Online phase back-to-back), KU heavily outpaces [CGGMP20]. More importantly, during the online phase, [CGGMP20] is bottlenecked by the rapid exhaustion of key-dependent presignatures. Because KU leverages a global, shared pool of key-independent presignatures that are

virtually always available, it sustains a massive online throughput of up to 333 signatures per second.

	[CGGMP20]	**KU**
End-to-end (w/o presigs)	0.42sig/sec	37 sig/sec
Online phase only (w. presigs)	100 sig/sec	333sig/sec

Table 3: Max throughput ($n = 5, t = 2$) on EC2 c5.large instances (2 vCPU of Intel(R) Xeon(R) Platinum 8124M CPU @ 3.00GHz, 1 core, 4 GiB of RAM) deployed in a single Region.

Platform. The baseline platform is expected to provide sufficient resources for realistic benchmarking and we do not anticipate particular challenges.

5. Licensing and Patent Claims

We intend to release the core implementation under a permissive open-source license.

We disclose a patent application that covers the building blocks underlying KU. DFNS owns one pending patent that protects methods underlying KU:

- SYSTEMS AND METHODS FOR PRESIGNATURE AND SIGNATURE GENERATION – PCT/IB2025/058940 https://patents.google.com/patent/US20260074916A1/en

References

[CCLST20] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. "Bandwidth-efficient threshold EC-DSA". In: *IACR International Conference on Public-Key Cryptography*. Springer. 2020, pp. 266–296. DOI: 10.1007/978-3-030-45388-6_10. Also at ia.cr/2020/084.

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. "Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation". In: *Theory of Cryptography Conference (TCC) 2005*. Vol. 3378. LNCS. Springer, 2005, pp. 342–362. DOI: 10.1007/978-3-540-30576-7_19.

[CGGMP20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. "UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts". In: *ACM Conf. on Computer and Communications Security (CCS)*. ACM, 2020, pp. 1769–1787. DOI: 10.1145/3372297.3423367. Also at ia.cr/2021/060.

[CHIKGLN23] Koji Chida, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Daniel Genkin, Yehuda Lindell, and Ariel Nof. "Fast Large-Scale Honest-Majority MPC for Malicious Adversaries". In: *J. Cryptology* 36.3 (2023). Preliminary version in Crypto 2018. DOI: 10.1007/S00145-023-09453-7. Also at ia.cr/2018/570.

[DKLS19] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. "Threshold ECDSA from ECDSA assumptions: The multiparty case". In: *2019 IEEE Symposium on Security and Privacy*. IEEE. 2019, pp. 1051–1066. DOI: 10.1109/SP.2019.00024. Also at ia.cr/2019/523.

[DKLS23] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. "Threshold ECDSA in Three Rounds". In: *IEEE Symp. on Security and Privacy 2023*. IEEE, 2023. DOI: 10.1109/SP54263.2024.00178. Also at ia.cr/2023/765.

[KU25] Jonathan Katz and Antoine Urban. "Honest-Majority Threshold ECDSA with Batch Generation of Key-Independent Presignatures". In: *IACR Communications in Cryptology* 2 (April 2025). DOI: 10.62056/a0l5wol7. Also at ia.cr/2024/2011.

[LN17] Yehuda Lindell and Ariel Nof. "A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest majority". In: *Proc. ACM Conference on Computer and Communications Security 2017*. ACM, 2017, pp. 259–276. DOI: 10.1145/3133956.3133999. Also at ia.cr/2017/816.

[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.