

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

Title: Quorus

Subtitle: Scalable Threshold ML-DSA from MPC

Version: 0.1 (2026-01-21)¹

Team name: Quorus: Quorus Team

Team members: Alexander Bienstock, Leo de Castro, Daniel Escudero, Antigoni Polychroniadou, Akira Takahashi

Abstract: In this document, we present an overview of our multi-party key and signature generation protocol for the module-lattice digital signature algorithm (ML-DSA). Our proposal includes an MPC-friendly variant of ML-DSA retaining compatibility with the FIPS 204 compliant verification algorithm, a distributed key generation (DKG) protocol, and an efficient threshold signing protocol with offline preprocessing. Our protocols are designed to provide strong security guarantees, such as post-quantum security and UC security, scalability to medium-sized groups of signers (e.g., up to 64) assuming an honest majority, and low signing latency in the online phase.

Proposed crypto-systems: ML-DSA (N1.4) and ML KeyGen (N4.3)

Keywords: Threshold Cryptography; NIST Threshold Call; ML-DSA

¹Preliminary 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: Alexander Bienstock^{i1,a1}, Leo de Castro^{i2,a1}, Daniel Escudero^{i3,a2}, Antigoni Polychroniadou^{i4,a1}, Akira Takahashi^{i5,a1}

Open Researcher and Contributor Identifiers (ORCID):

i1 (0000-0001-7640-4974); i2 (0009-0005-3190-6416); i3 (0000-0003-2375-0034); i4 (0009-0003-0125-2971); i5 (0000-0001-8556-3053)

Affiliations:

^{a1} J.P. Morgan AI Research & AlgoCRYPT CoE

^{a2} TACEO

Main contacts:

- **Team mailing list:** threshold-mlds-a-quorus@googlegroups.com
- **Primary technical contact person:** Leo de Castro, leo.decastro@jpmchase.com
- **Secondary contact person 1:** Alexander Bienstock, alex.bienstock@jpmchase.com
- **Secondary contact person 2:** Akira Takahashi, akira.takahashi@jpmorgan.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

In the upcoming call, we plan to propose multi-party threshold signature schemes compatible with the standardized ML-DSA verification process [LDKLS20; FIPS204]. Our submission will build upon the recent work of [BCEPT26], highlighting the following main features:

- **MPC-friendly variant of ML-DSA:** We specify an alternative version of the ML-DSA key generation and signing algorithms amenable to efficient secure multi-party computation (MPC) protocols. Importantly, this MPC-friendly variant outputs signatures that remain verifiable under the *unmodified* ML-DSA verification algorithm. Compared to the original ML-DSA as specified in [FIPS204], our version differs in that (1) it removes deterministic nonce generation and replaces it with randomized sampling, and (2) it includes an additional noise term in the vector w input to the Decompose function to enable proof of unforgeability under the leakage of rejected partial signatures. In particular, the second modification allows threshold signing protocols to compute Fiat-Shamir hashes in the clear, significantly reducing the complexity of the MPC protocol.
- **Distributed key generation:** We design a distributed key generation (DKG) protocol that allows a group of N parties to jointly generate an ML-DSA public key along with secret shares of the corresponding secret key, without relying on any trusted party. Along the way, we define an efficient MPC subprotocol for generating sharings of the small-norm secret vectors used in ML-DSA. As our DKG outputs public keys compatible with the original ML-DSA verification algorithm and standard Shamir sharings of the secret key over the ML-DSA modulus q , this building block is of independent interest.
- **Scalable threshold signing protocol with preprocessing:** We present an efficient threshold signing protocol that allows any subset of n out of N parties to jointly compute an ML-DSA signature, given secret shares of the secret key generated via our DKG protocol. Our protocol is designed to leverage offline preprocessing to significantly reduce the online signing latency, and to efficiently scale to a medium number of signers (e.g., $n \leq 64$), while tolerating up to $t < n/2$ malicious corruptions.
- **Stronger security of MPC-friendly ML-DSA and Universal Composability:** To retain post-quantum security, our MPC-friendly ML-DSA is designed to satisfy a stronger form of existential unforgeability under adaptive chosen-message attacks in the quantum random oracle model, even when the adversary can obtain incomplete signatures resulting from the rejection sampling step. We also prove that our distributed key generation and threshold signing protocols securely realize ideal functionalities in the Universal Composability (UC) framework, assuming an honest majority among the parties. Together, this implies unforgeability of the resulting threshold signature scheme, and as a consequence of the UC theorem, we furthermore obtain the guarantee that our protocols can be safely deployed as a subcomponent of larger, complex systems. This fits well with applications such as blockchain and cryptocurrencies, where threshold signatures are often used as a building block within a larger protocol.

2. Specification

2.1. Organization

In the final submission package, we plan to include a detailed specification of our proposed threshold signature schemes, organized as follows.

2.1.1. MPC-friendly ML-DSA

Following [BCEPT26], we first specify an MPC-friendly variant of the ML-DSA scheme summarized in Algorithm 1, which forms the basis of our threshold signature scheme. Notably, this variant produces signatures that remain verifiable under the original ML-DSA verification algorithm. The main differences from ML-DSA as specified in [FIPS204] are:

- In the key generation algorithm, the random seed ρ used to generate the matrix A is now sampled uniformly at random, instead of being derived from another random seed via a hash function.
- In the key generation algorithm, the secret vectors s and e are now directly sampled from S_η uniformly, instead of being generated by applying `ExpandS` to random seeds.
- In the signing algorithm, the nonce y is now sampled uniformly at random from $\tilde{S}_{\gamma_1}^\ell$, instead of being derived from a random seed via `ExpandMask`.
- In the signing algorithm, an additional noise term e_w uniformly sampled from S_η^k is added to the computation of w .
- The signing algorithm now omits the while-loop for rejection sampling and always outputs w_1 even if (z, h) gets rejected, instead of outputting \perp .

In particular, the last two changes are crucial for enabling efficient MPC protocols in a secure manner, as they allow the Fiat-Shamir hash c to be computed in the clear without leaking information about the secret key when a signature gets rejected.

2.1.2. MPC Protocols for ML-DSA Key Generation and Signing

With the MPC-friendly ML-DSA scheme in hand, the specification will proceed to describe the actual MPC protocol. This MPC protocol will have the following main building blocks:

- *Shamir Secret Sharing*. Standard Shamir secret sharings $[[x]]$ over prime field \mathbb{F}_p and $[[b]]_2$ over \mathbb{F}_2 . We call the former an “arithmetic sharing” and the latter a “binary sharing”. If at most t parties are corrupted, then the secrets are hidden from the adversary. These have the usual properties of Shamir Secret Sharing such as linearity.
- *Standard MPC Building Blocks*:

Algorithm 1: MPC-friendly ML-DSA (changes from [FIPS204] are highlighted)

KeyGen() 1: $\rho \leftarrow \{0, 1\}^{256}$ 2: $A \leftarrow \text{ExpandA}(\rho)$ 3: $(s, e) \leftarrow S_\eta^\ell \times S_\eta^k$ 4: $t \leftarrow As + e \bmod q$ 5: $(t_1, t_0) \leftarrow \text{Power2Round}_q(t, d)$ 6: $tr \leftarrow H(\rho, t_1)$ 7: $pk \leftarrow (\rho, t_1)$ 8: $sk \leftarrow (\rho, tr, s, e, t_1, t_0)$ 9: return (pk, sk)	$P_1(sk = (\rho, tr, s, e, t_1, t_0))$ 1: $A \leftarrow \text{ExpandA}(\rho)$ 2: $y \leftarrow \tilde{S}_{\gamma_1}^\ell; e_w \leftarrow S_\eta^k$ 3: $w \leftarrow Ay + e_w \bmod q$ 4: $(w_1, w_0) \leftarrow \text{Decompose}_q(w, 2\gamma_2)$ 5: $(w_1, \text{state} = (w_0, sk, y, e_w))$
$\text{Sign}(sk = (\rho, tr, s, e, t_1, t_0), \text{msg})$ 1: $(w_1, \text{state}) \leftarrow P_1(sk)$ 2: $\mu \leftarrow H(tr, \text{msg}); c \leftarrow H(\mu, w_1)$ 3: $(z, h) \leftarrow P_2(\text{state}, c)$ 4: return $(\sigma = (c, z, h), w_1)$	$P_2(\text{state}, c)$ 1: $z \leftarrow cs + y$ 2: $\bar{w}_0 \leftarrow w_0 - ce$ 3: if $\ z\ _\infty \geq \gamma_1 - \beta$ or $\ \bar{w}_0\ _\infty \geq \gamma_2 - \beta$ then return (\perp, \perp) 4: else 5: $h \leftarrow \text{MakeHint}_q(-ct_0, Az - ct + ct_0, 2\gamma_2)$ 6: if $\ ct_0\ _\infty \geq \gamma_2$ or $\text{HW}(h) > \omega$ then $h \leftarrow \perp$ 7: return (z, h)

- Open: opens a shared secret $\llbracket x \rrbracket$ to all parties.
- Mult: takes in two sharings $\llbracket x \rrbracket, \llbracket y \rrbracket_2$ and outputs sharings $\llbracket x \cdot y \rrbracket$ of the product of the secrets.
- Coin[D]: Samples a random value $x \in D$ and sends x to all parties.
- Rand[D]: Samples a random value $x \in D$ and outputs shares $\llbracket x \rrbracket$ of it to all parties.
- *Dabits [RW19] Generation*: Dabits samples random $b \in \{0, 1\}$ and outputs shares $(\llbracket b \rrbracket; \llbracket b \rrbracket_2)$ over both \mathbb{F}_p and \mathbb{F}_2 .
- *Edabits [EGKRS20] Generation*: Edabits samples random $r \in \mathbb{F}_p$, bit-decomposes it as $r = r_1 + r_2 \cdot 2 + \dots + r_m \cdot 2^{m-1}$ and distributes shares $(\llbracket r \rrbracket; \llbracket r_1 \rrbracket_2, \dots, \llbracket r_m \rrbracket_2)$.
- *Bitwise Less Than Constant*: BitLTC takes in binary sharings $\llbracket x_1 \rrbracket_2, \dots, \llbracket x_m \rrbracket_2$ and public bound B , and outputs sharing $\llbracket b \rrbracket_2$ where $x = \sum_{i=1}^m x_i 2^{i-1}$ and $b = (x < B)$. Utilizes Mult.
- *Less Than Constant*: LTC takes in arithmetic sharing $\llbracket x \rrbracket$, public bound B , and outputs $\llbracket b \rrbracket_2$ where $b = (x < B)$. Uses Edabits, Open, and BitLTC.
- *Bit Add*: BitAdd takes in binary sharings $(\llbracket x_1 \rrbracket_2, \dots, \llbracket x_m \rrbracket_2)$ and $(\llbracket y_1 \rrbracket_2, \dots, \llbracket y_m \rrbracket_2)$, outputs $(\llbracket z_1 \rrbracket_2, \dots, \llbracket z_{m+1} \rrbracket_2)$, where $\sum_{j=1}^{m+1} 2^{j-1} z_j = \sum_{i=1}^m 2^{i-1} x_i + \sum_{i=1}^m 2^{i-1} y_i$.
- *Bit Decomposition*: BitDec takes in arithmetic sharing $\llbracket s \rrbracket$, where $s = s_1 + s_2 \cdot 2 + \dots + s_m \cdot 2^{m-1}$ and outputs $\llbracket s_1 \rrbracket_2, \dots, \llbracket s_m \rrbracket_2$. Utilizes Open, Edabits, LTC, and BitAdd.
- *Distributed Key Generation*: DKG outputs public key of ML-DSA signature system and secret shares of the secret key. Utilizes Coin, Rand, and Open.

- *Preprocessing*: For each signature, Prep computes the message-independent part of the threshold signing protocol. Utilizes Rand, BitDec, Dabits, and Open.
- *Rejection Sampling*: RejSamp Performs the rejection sampling part of the ML-DSA signing protocol via MPC. Utilizes LTC, Coin, Dabits, Open, Rand, and Mult.
- *Threshold Signing Protocol*: Computes a signature. Utilizes Prep, Open, RejSamp.

2.2. System Model

Let N be the total number of parties participating in DKG and $n \leq N$ be the number of signers participating in the threshold signing protocol. Following [BCEPT26], we plan to support medium-sized groups of signers, e.g., $n \leq 64$. Unlike several other existing works on lattice-based threshold signatures, our protocol maintains a constant signature size independent of the number of signers n . We assume synchronous communication with pairwise encrypted and authenticated channels between all parties. For each signing group of n parties, we will assume that there is an honest majority among them: i.e., $t < n/2$, where t is the number of corrupted parties.

2.3. Security

Security of MPC-friendly ML-DSA: We first analyze security of our MPC-friendly variant of ML-DSA presented in Algorithm 1. In more detail, we prove that ML-DSA remains existentially unforgeable under adaptive chosen-message attacks (UF-CMA) in the quantum random oracle model (QROM) even when the signing algorithm is replaced with our MPC-friendly variant. We also define a stronger variant of the UF-CMA notion tailored to signature schemes that may output incomplete signatures, and prove that our MPC-friendly ML-DSA satisfies the UF-CMA_\perp notion, which essentially guarantees that even if an adversary can obtain an incomplete signature (w_\perp, c, \perp) on m , it still cannot produce a valid signature on m . Following [BCEPT26], we guarantee UF-CMA_\perp security of our MPC-friendly ML-DSA scheme in the QROM, assuming *unforgeability under no message attacks* (UF-NMA) security of the original ML-DSA scheme and under the standard module learning with errors (MLWE) assumption, which already underlies the security of ML-DSA.

Security of the threshold scheme: We provide the security analysis of our protocol in the Universal Composability (UC) framework [Can01]. In more detail, we prove that our protocol securely realizes the ideal functionality $\mathcal{F}_{\text{Sign}}$ computing the MPC-friendly ML-DSA signing algorithm, in the presence of a malicious adversary corrupting at most t of the n signers. Our protocol achieves *security-with-abort*, meaning that the adversary learns only the output signature and nothing else about the ML-DSA secret key, and all honest parties either receive the correct signature or abort.

Taken together, we conclude that our threshold signature scheme is unforgeable even against adversaries corrupting up to t signers and interacting with honest signers to obtain signatures on adaptively chosen messages. We will also discuss identifiable aborts and security under adaptive corruption in the full version of the specification.

3. Open-Source Implementation, Performance Evaluation, and License

We have implemented a preliminary version of an MPC protocol that evaluates our MPC-friendly signing algorithm. This C++ implementation is intended for research & benchmarking purposes; the implementation should *NOT* be considered safe for production uses.

Code structure: Our implementation is based on the Secure Computation Library², which is an open-source C++ library for implementing MPC protocols. This library also includes basic networked functionalities, although we have not yet benchmarked our protocol in a concrete network setting. Instead, we intend to update the code as new versions of the MPC protocol are developed; we are actively researching better methods to evaluate our MPC-friendly variant of ML-DSA signing.

Code progress and availability: Our implementation is available on Zenodo: <https://doi.org/10.5281/zenodo.17888654>, although access is restricted for now until we receive clearance for a full public release. We intend to host the code on GitHub within the coming weeks. As mentioned above, we expect this protocol to change substantially in the coming months as we work to develop better methods to evaluate our new signing algorithm in MPC.

Implementation of the networking model: We are considering only peer-to-peer communication in our analysis, and we are not assuming any additional network structure other than a basic PKI to establish pairwise encrypted and authenticated channels.

Testing: We expect our computation benchmarks to be immediately reproducible by anyone who downloads & compiles the source code. For a fully networked setting, our protocol should be evaluated as any other MPC protocol, where messages between parties may be lost or delayed arbitrarily. Our protocol is agnostic to the underlying methods used to implement reliable peer-to-peer networking, although any final implementation would need to undergo rigorous testing in adversarial network environments.

Performance evaluation: The performance evaluation of the current protocol has two main components: the communication sizes for each iteration of the rejection sampling protocol and the compute times for the rejection sampling iteration. As with most MPC protocols, we expect the overall performance to be heavily bottlenecked by the communication. The primary goal of the compute benchmarks are to show that the compute will not easily become the bottleneck.

License: We plan to release the code of our submission under the Apache License, Version 2.0.

Disclaimer

This paper was prepared in part for information purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates (“JP Morgan”), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information

²<https://github.com/anderspkd/secure-computation-library>

contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

References

- [BCEPT26] Alexander Bienstock, Leo de Castro, Daniel Escudero, Antigoni Polychroniadou, and Akira Takahashi. “Quorus: Efficient, Scalable Threshold ML-DSA Signatures from MPC”. In: *USENIX Security 2026*. 2026. Also at ia.cr/2025/904.
- [Can01] Ran Canetti. “Universally Composable Security: A New Paradigm for Cryptographic Protocols”. In: *42nd FOCS*. IEEE Computer Society Press, October 2001, pp. 136–145. DOI: [10.1109/SFCS.2001.959888](https://doi.org/10.1109/SFCS.2001.959888). Also at ia.cr/2000/067.
- [EGKRS20] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. “Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits”. In: *CRYPTO 2020, Part II*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12171. LNCS. Springer, Cham, August 2020, pp. 823–852. DOI: [10.1007/978-3-030-56880-1_29](https://doi.org/10.1007/978-3-030-56880-1_29). Also at ia.cr/2020/338.
- [FIPS204] Federal Information Processing Standards Publication. *FIPS 204: Module-Lattice-Based Digital Signature Standard*. Federal Information Processing Standards Publication <https://doi.org/10.6028/NIST.FIPS.204>. 2024. URL: <https://doi.org/10.6028/NIST.FIPS.204>.
- [LDKLSSSB20] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. *CRYSTALS-DILITHIUM*. Tech. rep. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions>. National Institute of Standards and Technology, 2020.
- [RW19] Dragos Rotaru and Tim Wood. “MABled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security”. In: *INDOCRYPT 2019*. Ed. by Feng Hao, Sushmita Ruj, and Sourav Sen Gupta. Vol. 11898. LNCS. Springer, Cham, December 2019, pp. 227–249. DOI: [10.1007/978-3-030-35423-7_12](https://doi.org/10.1007/978-3-030-35423-7_12). Also at ia.cr/2019/207.
- [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. 2025. DOI: [10.6028/NIST.IR.8214C](https://doi.org/10.6028/NIST.IR.8214C).