Dealing Haystack: Towards Trustless Haystack in the Optimistic Setting

Isabel Muñoz[1], Isaac Agudo[1,2], Marco López[1,2], Daniel Morales[2]

[1]University of Málaga [2]Decentralized Security

{isabel.munozruiz19, isaac, lopez.marco}@uma.es

damesca.crypto@protonmail.com

August 26, 2026

Abstract

Hash-based constructions occupy a distinctive position among post-quantum signatures: their security reduces to well-tested properties of hash functions rather than to newer assumptions such as lattices or isogenies. This work focuses on stateful schemes instead of stateless, because the former are considerably more efficient. However, they have the problem of state handling, since reusing a one-time key twice enables signature forgeries. Despite threshold signatures mitigate this problem by spreading trust among a set of disjoint parties, building them from hash-based schemes is difficult, since these lack the homomorphic structure needed to recombine partial signatures, and generic multiparty computation can be expensive for hash-based constructions. Kelsey, Lang and Lucks recently proposed Haystack, the first threshold scheme for hash-based signatures producing standard LMS or XMSS signatures, at the cost of a fully trusted setup and a large common reference value. We analyze Haystack along two dimensions: performance and security. First, as Haystack lacks an implementation and realistic benchmarking, we implement the protocol in Java and produce a network-aware evaluation of its viability in real deployments, concluding that it performs comparably to other post-quantum threshold schemes. Second, we relax the trust placed in the dealer. For that, we introduce a variant of the setup built on an optimistic, lightweight MPC-based partial-DKG. It does not remove the dealer's ability to forge, but it prevents it from impersonating trustees within the signing protocol, while preserving the standard signature format. Also, an optional succinct-argument layer provides public auditability. We further consider a full-DKG setting with no dealer and where the trustees run the entire setup under MPC. Both variants are implemented in MP-SPDZ and their costs have been analyzed.

1 Introduction

1.1 Context and motivation

1.1.1 Post-Quantum Cryptography

A cryptographically relevant quantum computer would break the discrete-logarithm and factoring assumptions on which essentially all deployed digital signatures rest [1]. Because of this, research in new post-quantum algorithms has received major traction, mainly pushed by the standardization process led by NIST. Unlike confidentiality, where the harvest-now-decrypt-later threat makes the deadline immediate, signatures are broken only prospectively. However, the exposure is governed by the lifetime of the verifying party rather than by that of the signer. Long-lived roots of trust are therefore the first casualties: code- and firmware-signing keys embedded in devices deployed for one or two decades must be quantum-resistant today, since the verifier cannot be updated once it has shipped. This is precisely the setting in which stateful hash-based signatures have been standardized and, in some jurisdictions, mandated.

Among post-quantum signature families, hash-based constructions occupy a distinctive position. Their security reduces to properties of the underlying hash function — in modern designs, essentially second-preimage resistance — rather than to the hardness of lattice or isogeny problems, which are comparatively recent and less well understood. The design principle dates to Lamport [2] and Merkle [3], and the intervening decades of cryptanalysis of standard hash functions transfer directly. For a root of trust expected to remain valid for twenty years, this conservatism is the point.

The family divides into stateless schemes, standardized as SLH-DSA, and stateful ones, of which LMS [4] and XMSS [5] are the standardized representatives, with usage requirements given in NIST SP 800-208 [6]. Stateful schemes buy substantially smaller signatures and faster verification at the price of a bounded number of signatures D and, more consequentially, of a signing state that must be maintained correctly. A clear example of stateful HBS adoption is the CNSA 2.0 suite[1], which incorporates PQ algorithms approved for National Security Systems (NSS) use. Among them, both LMS and XMSS have been included as algorithms for the specific application of signing firmware and software.

1.1.2 The state problem of HBS

A stateful scheme is built from D one-time key pairs whose public keys form the leaves of a Merkle tree. A signature reveals selected values from the hash chains of one such one-time key, together with the authentication path to the root. Revealing a second signature under the *same* one-time key is catastrophic: an observer of both can walk forward in the chains and forge signatures on a large

[1] https://media.defense.gov/2022/Sep/07/2003071836/-1/-1/0/CSI_CNSA_2.0_FAQ_.PDF

class of messages, without breaking any cryptographic assumption. Security therefore rests not only on the hash function but on an operational invariant — that no index is ever consumed twice — which the scheme itself cannot enforce.

This invariant is fragile in ways that have little to do with cryptography. Restoring a signing host from a backup, cloning or rolling back a virtual machine, a crash between consuming an index and persisting that fact, replicating a signer for availability, or plain operator error all reintroduce a state that has already been used. SP 800-208 responds with stringent handling requirements, but these constrain how the key is stored rather than eliminating the failure mode. The practical objection to stateful hash-based signatures is not their security argument, which is unusually strong, but the difficulty of guaranteeing this invariant across a device's operational lifetime.

1.1.3 Threshold constructions for HBS

Splitting the signing capability across several trustees addresses the problem at its source. If producing a signature requires the concurrent participation of k independent devices, and each refuses to sign twice under the same index, then key reuse demands that all k fail simultaneously and identically — a conjunction of independent faults rather than a single one. A restored backup on one trustee no longer suffices. Even a 2-of-2 configuration converts a single point of failure into a coincidence, which is a disproportionate gain relative to its cost.

The usual motivations for threshold signing apply in addition. Theft or compromise of one device no longer yields the signing key, a firmware-release key can be placed under the joint control of several individuals, and a trustee that becomes untrusted can be excluded without reissuing the public key. For high-value, long-lived signing roots — the same deployments that drive the adoption of stateful schemes in the first place — these properties are not incidental.

However, constructing threshold signatures from hash-based schemes is particularly difficult. First, threshold constructions for RSA or Schnorr signatures exploit homomorphic structure that hash functions deliberately lack. Second, generic multiparty computation over a hash-based signing algorithm is prohibitively expensive: an early attempt reported on the order of an hour to produce a single signature [7].

The scheme of Kelsey, Lang and Lucks [8] (namely *Haystack*[2]) circumvents this by exploiting a key property of Winternitz signing. Producing a signature amounts to selecting secret-key values at indices determined by the public message hash, and this selection commutes with additive sharing. Each trustee can therefore select locally from its own share, and combining the resulting shares yields the corresponding selection from the combined key. Haystack pays for this with a large public Common Reference Value (CRV) precomputed

[2] https://csrc.nist.gov/csrc/media/Projects/threshold-cryptography/documents/TCall-1/Haystack-PW01.pdf

at setup and with sharding of the index space, but produces signatures that are fully compatible with ordinary LMS or XMSS: verification is unchanged, and a verifier cannot distinguish a distributed signature from a single-signer one. We remark that sharding leaves us with cohorts of different cardinalities $\{|C_1|, |C_2|, ..., |C_m|\}$ — all $|C_i| \leq n$, given n the total number of signers —, where each cohort actually carries on a $|C_i|$-of-$|C_i|$ full threshold signature. It is worth to note that there exist a concurrent work named PRAWNS [9] (yet to be fully disclosed at the time of writing) that targets this exact problem: to achieve an standard t-of-n threshold signature for HBS. They do so by relying on a lattice-based threshold PRF built upon a key-homomorphic PRF [10]. However, despite noticed as a target problem, they do not address the trusted DKG yet. In this paper, we have the opposite direction: while we leave the threshold by sharding solution untouched, we do address the problem of the trusted DKG in the current version of Haystack.

1.1.4 Haystack implementation and validation

Haystack provides the construction and its security analysis, but, at the time of writing, no implementation of the scheme is publicly available. A first contribution of this paper is therefore to provide an independent implementation of the sharded construction over LMS and evaluate it experimentally. In particular, we validate that the distributed protocol can produce standard LMS signatures accepted by unmodified implementations, quantify the concrete costs of setup and local signing, and characterize distributed signing latency under controlled Trustee RTTs and coalition sizes. This implementation provides the practical baseline on which the extensions studied later in this paper are built.

1.1.5 The remaining assumption

Haystack requires a trusted dealer. During setup the dealer generates the complete key pair, derives every trustee's share, and is then instructed to delete all of it. Its correctness is thereby reduced to a single unverifiable act, performed by a party that, up to that moment, holds exactly the power the scheme is intended to deny to everyone else. This is one of the main concerns of this paper: how to minimize — or even remove — the power of the trusted dealer.

1.2 Contributions

This paper makes two complementary contributions around two dimensions of Haystack: performance and security. The specific contributions are:

- We provide an interoperable implementation of the sharded Haystack construction over LMS, turning the original construction — which only has a Python PoC and no deployment analysis — into an executable system that covers setup, trustee state management, and the two-round signing protocol. We validate interoperability against unmodified LMS

implementations and experimentally characterize local implementation costs and distributed signing latency under controlled Trustee RTTs for coalitions of $k = 3$, 5, and 10.

- We investigate how the trusted-dealer assumption in Haystack can be progressively relaxed through distributed key generation. We first consider a partial-DKG variant in which trustees generate their own secret material and lightweight MPC is used during CRV construction, weakening the dealer's ability to impersonate trustees while preserving the standard signature format; we also consider an optional succinct-argument layer for public auditability. We then investigate a full-DKG variant in which the dealer is removed and the CRV is generated jointly by the trustees through MPC, and implement a proof of concept to quantify the cost of generating the required hash-chain material.

1.3 Document structure

Section 2 recalls the stateful HBS setting and the Haystack construction, including its trust model. Section 3 presents our implementation of the original construction and its experimental validation, including interoperability, local performance, and controlled distributed network measurements. Section 4 then isolates the trusted-dealer assumption and presents the partial- and full-DKG variants, which are experimentally evaluated in Section 5. Finally, Section 6 concludes the paper and outlines future work.

2 Background and Haystack Baseline

2.1 Stateful Hash-Based Signatures

Hash-based signatures (HBS) derive their security solely from the properties of the underlying cryptographic hash function, rather than from number-theoretic assumptions such as factoring or discrete logarithms. This makes them attractive in a post-quantum setting. In this work we focus on *stateful* HBS, the family standardized as LMS [4] and XMSS [5], which our implementation builds upon.

The basic building block is a one-time signature (OTS) scheme, in our case the Winternitz construction. As the name suggests, a single OTS key pair may be used to sign exactly one message. Signing two distinct messages with the same key leaks enough of the secret key to allow forgeries. Internally, a Winternitz signature is built from hash chains over the private key and a checksum, which prevents an adversary from advancing the chains to forge the signature of a different message. For the purposes of this document the essential property is simple: *one key, one signature.*

A single OTS key is rarely useful on its own. To obtain a practical number of signatures, D one-time key pairs are generated and their public keys are placed as the leaves of a Merkle tree. The root of this tree is published as a single

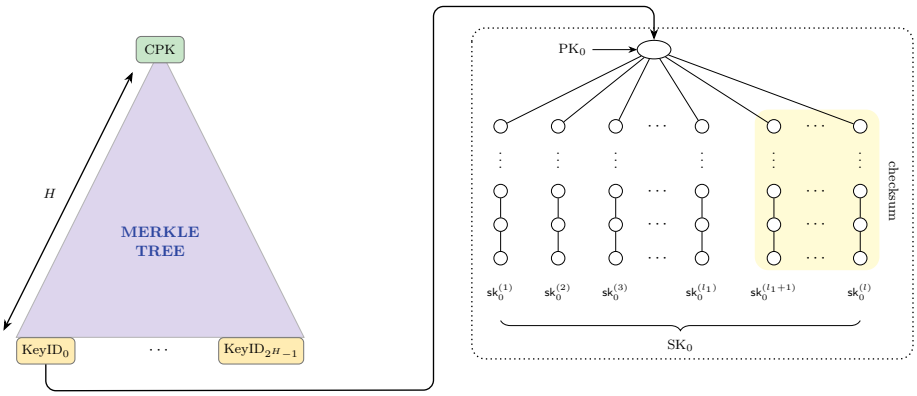


Figure 1: Structure of a stateful hash-based signature. The $D = 2^H$ one-time public keys form the leaves of a Merkle tree of height H, whose root is the composite public key (CPK). The leaf KeyID_0 is expanded to show its underlying Winternitz structure, with the message and checksum hash chains, the secret key seed SK_0 and the public key PK_0.

composite public key (CPK), committing to all D one-time keys at once (see Figure 1). Then, a signature carries a *Merkle path* — herein denoted PATH — which proves that the one-time key used is the leaf at a given position under the CPK. The parameter D is fixed at key-generation time and bounds the total number of signatures the key can ever produce.

Each one-time key is identified by a unique index, $\text{KeyID} \in \{0, \ldots, D-1\}$, which is visible to the verifier. The *state* of a stateful HBS signer is precisely the record of which KeyIDs have already been used. Maintaining this state correctly is the central practical challenge of these schemes and the property our entire contribution follows.

A signature in these schemes is a triple (R, PATH, Z), where R is a randomizer, PATH is the Merkle path described above and Z is the Winternitz one-time signature. Modern HBS use *randomized hashing*: the signature is computed over $h = \mathsf{Hash}(R, M)$ rather than over the message M directly. They also incorporate scheme-specific *tweaks* into hash calls so that distinct uses of the hash function are domain separated, avoiding multi-target preimage attacks. For example, abstracting away the scheme-specific encoding of these tweaks, Haystack uses $(1, \text{KeyID})$ as the tweak for the randomized message digest, which they write as $h = \mathsf{Hash}_{(1,\text{KeyID})}(R, M)$.

The defining constraint of stateful HBS is therefore the following: a one-time key, and hence a KeyID, must **never** be used to sign more than once. Any party observing two signatures produced under the same KeyID can recover enough of the one-time key to forge signatures on arbitrary messages. Every security and robustness consideration in the remainder of this document ultimately stems

from this single requirement.

2.2 The Haystack scheme

2.2.1 General vision

Haystack [8] turns any stateful HBS of the form described above into a threshold or distributed signature scheme, without altering the verification procedure. We first recall the distinction between the two notions. In a *threshold* k-of-ℓ scheme there are ℓ trustees and any subset of k of them can jointly produce a signature [11, 12]. A *distributed* signature scheme generalizes this to arbitrary authorized subsets [13]. For instance, one may require a designated trustee (say, a company's CEO) to take part in every signature, while any single additional trustee suffices to complete it.

Haystack involves four roles:

Dealer. It is a trusted, one-time, setup-only party: it runs the key generation of the underlying HBS, uses the trustee PRF keys to define shares of the signing material for each KeyID, constructs and publishes the complementary CRV data, distributes the PRF keys and shard information to the trustees, and then deletes the private setup material.

Trustees. They are the ℓ parties that jointly hold the delegated signing capability. Each trustee stores only its PRF key, K_t, and the record of which one-time keys it has already used to sign with.

Aggregator. It drives the signing protocol, communicating with each trustee and reading the CRV, but it holds no secrets and has no power beyond requesting signatures.

Verifier. It checks signatures exactly as in the standard HBS.

Notably, the aggregator does not need to be a separate entity: its role may be played by one of the trustees with no loss of security or performance.

For each CPK, the dealer produces a CRV: public data that the trustees cannot derive on their own and which the aggregator combines with the trustees' responses to assemble a signature. Only the aggregator ever accesses the CRV, neither the trustees nor the verifier need to read it. The CRV is indexed by each single one-time key and its associated content — per key — consists of four fields: CRV.R (randomizer), CRV.CHK (check value for the randomizer), CRV.PATH (Merkle tree PATH) and CRV.SK (Winternitz hash chain). The dealer generates them during setup as follows. For each secret value to be shared, it derives one share per trustee from that trustee's PRF key, K_t, and the KeyID. Then, it stores in the CRV the XOR of the true value (TV) with all k trustee shares (S_t):

$$\mathrm{CRV.field} \;=\; \mathrm{TV} \;\oplus\; \bigoplus_{t=1}^{k} \mathrm{S}_t.$$

This way, the aggregator can reconstruct the value only by combining the CRV field with the trustees' shares. Neither the CRV alone nor the trustees alone can reveal any secret.

The CRV has three properties that are essential to the entire scheme:

1. It is *public*, as it contains no secret information.
2. It is *untrusted*, as tampering with it can only prevent a valid signature from being produced (a denial of service) but never enables a forgery.
3. It is *large*. The size of an entry grows with the number of trustees k, the hash size parameter n, the Winternitz parameter w and the tree height H. For the full CRV, it grows linearly with the number of one-time keys $D = 2^H$. Table 1 breaks this down and Table 2 contains a real CRV size depending on chosen parameters.

Table 1: Size of the CRV. Here n is the hash output size, w the Winternitz parameter, $l = a + c$ the number of chains, where $a = n/w$ and $c = \lceil \frac{\log_2(a)}{w} \rceil + 1$ are the number of message and checksum chains, $H = \log_2 D$ the Merkle tree height, k the number of trustees in the coalition and $D = 2^H$ the total number of one-time keys.

Field	Size	Depends on
CRV.R	n	hash size
CRV.CHK	$k \cdot n$	hash size, number of trustees
CRV.PATH	$\log_2 D \cdot n$	hash size, number of keys
CRV.SK	$l \cdot 2^w \cdot n$	hash size, Winternitz params
Per one-time key	$\left(1 + k + \log_2 D + l \cdot 2^w\right) \cdot n$	individual fields
Full CRV	$D \cdot \left(1 + k + \log_2 D + l \cdot 2^w\right) \cdot n$	individual fields, number of keys

Table 2: Example CRV size for an LMS instance with $n = 256$ bits, $w = 4$ ($a = 64$, $c = 3$, then $l = 67$), $2^w = 16$, tree height $H = 15$ ($D = 2^{15}$ one-time keys) and a coalition of $k = 3$ trustees.

Field	Formula	Size
CRV.R	n	32 B
CRV.CHK	$k \cdot n$	96 B
CRV.PATH	$\log_2 D \cdot n$	480 B
CRV.SK	$l \cdot 2^w \cdot n$	34 304 B
Per one-time key		$\approx$ 34.1 KiB
Full CRV		$\approx$ 1.07 GiB

Signing proceeds in two sequential interaction rounds between the aggregator and the participating trustees, exchanged over point-to-point channels with no broadcast required. In the first round, the randomizer R is established, together

with a check value CHK that lets each trustee confirm that R is correct. The ordering of this round is security-relevant: all participating trustees must receive the message M before R is reconstructed and returned to them, so that no subset of $k-1$ trustees learns the randomizer before the full coalition has seen the message to be signed. In the second round, the trustees produce their shares of the Merkle path and of the one-time signature. The aggregator then combines these shares with the CRV to obtain the final signature (R, PATH, Z). Figure 2 summarizes the signing protocol.

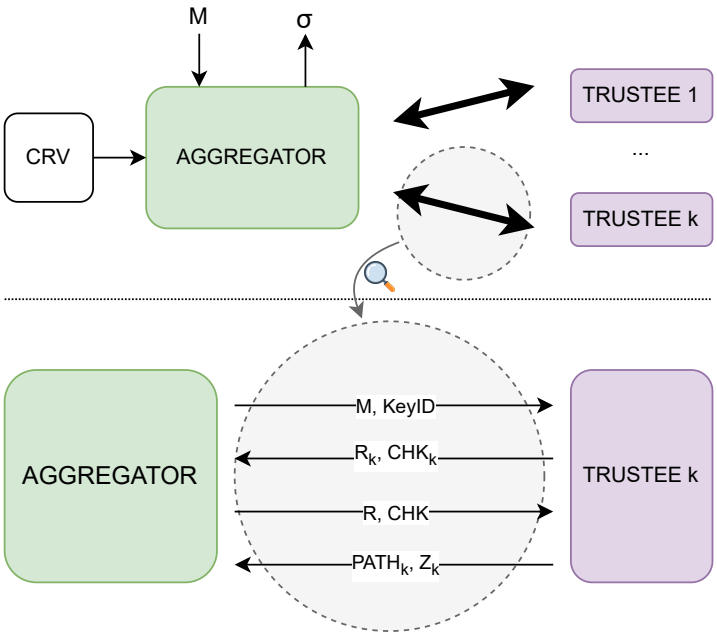


Figure 2: Overview of the signing protocol (top) and detailed two-round message exchange between the aggregator and a trustee k (down).

Requiring every member of a fixed coalition to participate gives a k-of-k construction, but generalizing it naively to k-of-ℓ creates a state-management problem. Two authorized coalitions may contain disjoint sets of honest trustees, so one coalition may be unaware that another has already consumed a particular KeyID. Haystack refers to this as the *Disjoint Honest Trustees Problem*. Rather than requiring a broadcast channel to synchronize state globally, Haystack uses *sharding*: the D one-time keys belonging to a CPK are partitioned among authorized coalitions, and each KeyID is assigned to exactly one coalition. The assignment is encoded in a coalition list CL indexed by KeyID. Consequently, two distinct coalitions are never given signing shares for the same one-time key, and state only needs

to remain consistent within the coalition that owns that shard. The cost of sharding is that the fixed signature budget D must also be partitioned among the authorized coalitions. If every coalition must support at least m signatures, the number of coalitions is bounded by $\lfloor D/m \rfloor$, which makes authorization structures with combinatorially many coalitions impractical.

A key property of the construction is that the resulting signature is *identical* to one produced by the underlying single-signer HBS. Verification is unchanged and the verifier cannot tell a distributed signature apart from an ordinary one. As a consequence, no custom verification code is needed and the verifier does not need to know anything about which coalitions are authorized to sign.

2.2.2 Trust depending on the role

Having described the four roles, we now make explicit the amount of trust each of them needs. This is the central question when reasoning about the security of the scheme and it will motivate the new directions explored in Section 4.

Dealer. It is the only fully trusted party. During setup it has access to everything: the complete keypair, all one-time keys, all trustee shares and PRF keys. The security of the scheme therefore rests on the dealer deleting every secret value once the CRV has been distributed. If it fails to do so, it retains the entire signing key and can forge signatures on its own. This is a strong and, crucially, unverifiable assumption: no external party can confirm that the dealer actually erased the keys.

Aggregator. It holds no secret and has no signing power. It only orchestrates the protocol and reads the CRV. A malicious aggregator can refuse to sign, send malformed data or abort the protocol, but it can never produce a forgery. In fact, the security of the scheme holds even when the aggregator is assumed to be entirely under the adversary's control. Its worst-case impact is therefore a denial of service, never a loss of integrity. This is why the aggregator may run on an untrusted machine and why its role can be played by one of the trustees at no security cost.

Trustees. The signing capability is split among the trustees so that no proper subset of a coalition can sign or recover the one-time key. All k trustees of the coalition (together with the CRV) are required to sign a message. An adversary that compromises up to $k-1$ trustees of a coalition learns nothing useful about that one-time key and must corrupt the entire coalition to forge a signature.

CRV. Tampering with it causes the resulting signature to fail verification, which is again a denial of service rather than a forgery. Such tampering can be detected by hashing the CRV and the denial of service can be mitigated by replicating it across several locations.

In summary, we can see that the only strong trust assumption left in the scheme concerns the dealer. We return to this assumption in Section 4 after

experimentally validating the original construction.

3 Haystack Implementation and Experimental Validation

3.1 Implementation

Haystack specifies the construction and its security analysis, but no implementation of the scheme is publicly available at the time of writing.[3] We therefore provide an independent implementation of the sharded construction over LMS and use it to evaluate interoperability and concrete performance.

Our implementation[4] is written in Java 17 and builds on BouncyCastle's LMS implementation (`bcprov-jdk18on` / `bcpkix-jdk18on`, v1.84). We extend it in two places: `LM_OTS_WITH_CHAIN` exposes the intermediate Winternitz hash-chain values required to construct threshold shares, and `LMSSerializer` encodes the reconstructed result as a standard RFC 8554 LMS signature. The resulting signatures can therefore be consumed by an unmodified LMS verifier.

The implementation covers the complete sharded workflow described in Section 2.2. The `Dealer` is an ephemeral setup process that generates the LMS key material and trustee PRF keys, constructs the CRVs and coalition list CL, publishes the public artifacts, distributes each K_t to its trustee, and terminates. Trustees run as independent gRPC services and persist both their PRF key and KeyID state in SQLite. The `Aggregator` exposes an HTTP signing interface, retrieves the coalition and public material associated with a KeyID, coordinates the participating trustees over gRPC, and reconstructs the final (R, PATH, Z) signature. Public CRVs and CL are held in a content-addressable HTTP store (CAS), with a public bulletin board containing the LMS public key and the identifiers needed to retrieve them.

Two implementation details are particularly relevant to the experimental evaluation. First, trustee shares of R, CHK, PATH, and the Winternitz-chain material are derived with a domain-separated KMAC-256 PRF [14]. A trustee consumes a KeyID upon entering Round 1 and never makes it available again, including after an aborted signing attempt. Second, trustee RPCs are parallelized within each protocol round. The Aggregator launches all Round 1 requests before waiting for their completion and reconstructing R and CHK; Round 2 follows the same pattern before reconstruction of PATH and Z. The protocol-mandated barrier between the two rounds is preserved.

[3] The Haystack authors have stated that they are developing an open-source Python implementation, currently covering LMS.

[4] Code, tests and benchmark results are available at https://anonymous.4open.science/r/threshold-hbs/

3.2 Validation and Local Performance

Before benchmarking the implementation, we verified that reconstructed signatures preserve the standard LMS interface. Signatures produced by `AggregatorSign` and serialized by `LMSSerializer` verify successfully with both BouncyCastle's unmodified LMS verifier and OpenSSL's LMS implementation, without any Haystack-specific verification logic. Unit and integration tests additionally cover dealer setup, trustee state transitions, coalition assignment, invalid authentication values, KeyID reuse, and complete distributed signing.

We first characterize the cryptographic cost independently of network communication using the in-process implementation. Measurements were performed on an Intel Core i7-13620H CPU @ 2.40 GHz with 16 GB of RAM (Windows 11, JDK 21.0.11), for $w \in \{1, 2, 4, 8\}$ and $k \in \{3, 5, 7, 10\}$. We report the mean of 15 repetitions after 8 warm-up iterations. Table 3 reports the $k = 3$ case together with the equivalent plain LMS baseline; coalition-size effects are examined explicitly in the distributed evaluation below.

Table 3: Cost of a complete threshold signature for a coalition of $k = 3$ trustees, $D = 32$, and $h = 5$, compared with the equivalent plain LMS signature. Mean $\pm$ standard deviation over 15 repetitions after 8 warm-up iterations.

w	threshold ($k = 3$) sign (ms)	plain LMS sign (ms)
1	3.5625 ± 0.0510	0.179
2	3.6136 ± 0.0315	0.142
4	8.0752 ± 0.1891	0.200
8	71.0534 ± 1.5174	1.277

Setup is performed only once, but its cost scales with the number $D = 2^h$ of one-time keys. For $w = 4$ and $k = 3$, we measure ≈ 8 ms of `ShardSetup` computation per KeyID. Table 4 extrapolates this cost linearly to larger LMS trees and also shows the speed ups achieved when parallelizing. The distributed experiments below therefore focus on steady-state signing rather than setup.

Table 4: Estimated `ShardSetup` cost for increasing LMS tree heights ($w = 4$, $k = 3$) in both sequential and parallel settings.

	Runtime			
h	sequential (extrapolated)	2 thr	4 thr	8 thr
5	256 ms	115.97 ms	71.24 ms	59.55 ms
10	8.192 s	3.378 s	1.940 s	1.389 s
15	4.37 min	18.086 s	12.027 s	9.829 s
20	2.33 h	21.078 s	13.673 s	10.564 s

3.3 Distributed Network Evaluation

We next study the interactive signing path under controlled network delay. The evaluation addresses three questions: how signing latency scales with Aggregator–Trustee RTT, whether increasing the coalition size changes that scaling, and how heterogeneous Trustee RTTs affect round and end-to-end signing latency. Traffic shaping is therefore restricted to Aggregator–Trustee communication; the client–Aggregator and Aggregator–CAS paths remain unmodified.

3.3.1 Experimental setup

The Aggregator, trustees, CAS, and setup components run as independent Docker containers and communicate through real TCP sockets. Aggregator–Trustee communication uses gRPC and public objects are retrieved from the CAS over HTTP. Controlled delay is introduced with Linux `tc/netem` on the host-side Docker veth interfaces: for a target RTT R, approximately $R/2$ is imposed in each direction. We introduce no artificial packet loss, jitter, or bandwidth limitation.

The campaigns were executed on a single Linux host equipped with an Intel Core Ultra 9 185H CPU (16 cores, 22 hardware threads) and 32 GB of RAM, running Arch Linux with kernel 7.1.4, OpenJDK 17.0.19, and Docker 29.6.2.

We evaluate fixed k-of-k coalitions with $k \in \{3, 5, 10\}$ using LMS `lms_sha256_n32_h10` and LMOTS `sha256_n32_w4`, giving $D = 1024$ one-time KeyIDs. Within each campaign, every KeyID is assigned to the same full coalition of active trustees, keeping the authorization structure fixed while varying only coalition size. All signatures use the same 1024-byte message.

The primary experiment considers an unshaped baseline and target RTTs of 20, 80, and 200 ms. The corresponding mean RTTs measured from the Aggregator namespace were 0.054, 20.107, 80.116, and 200.112 ms. Each coalition size is evaluated in a separate campaign beginning with 10 warm-up signatures. Measurements are then organized into five randomized blocks, each containing the four network profiles once in a reproducible pseudo-random order. After each profile transition, one conditioning signature is discarded before collecting 10 measured signatures. This gives 50 samples per $(k, \text{profile})$ condition, or 600 measured signatures across the primary experiment.

A second campaign isolates path heterogeneity for $k = 10$. It compares three conditions: all trustees at 20 ms RTT, nine trustees at 20 ms with one required trustee at 200 ms, and all trustees at 200 ms. The same warm-up, randomized-block, and conditioning procedure yields 50 measurements per condition and 150 measured signatures in total.

The experiment harness reserves a fresh KeyID persistently before every request and never returns it to the pool after an attempt, complementing the one-time state enforced independently by each trustee. All measured signatures were

successfully serialized as 2508-byte LMS signatures and verified with OpenSSL 4.0.1. Our primary metric is the Aggregator's instrumented end-to-end signing time, `aggregator_total_ns`; client-observed HTTP latency is used only as a consistency check.

3.3.2 RTT and coalition size

Figure 3 shows a nearly linear increase in signing latency with RTT, with little separation among coalition sizes. Baseline median latency ranges from 29.79 ms for $k = 3$ to 32.72 ms for $k = 10$. At approximately 200 ms RTT, all three topologies converge near 441 ms. Variability is low throughout: IQRs range from 2.47 to 6.03 ms across all conditions and are therefore barely visible at the scale of the figure.

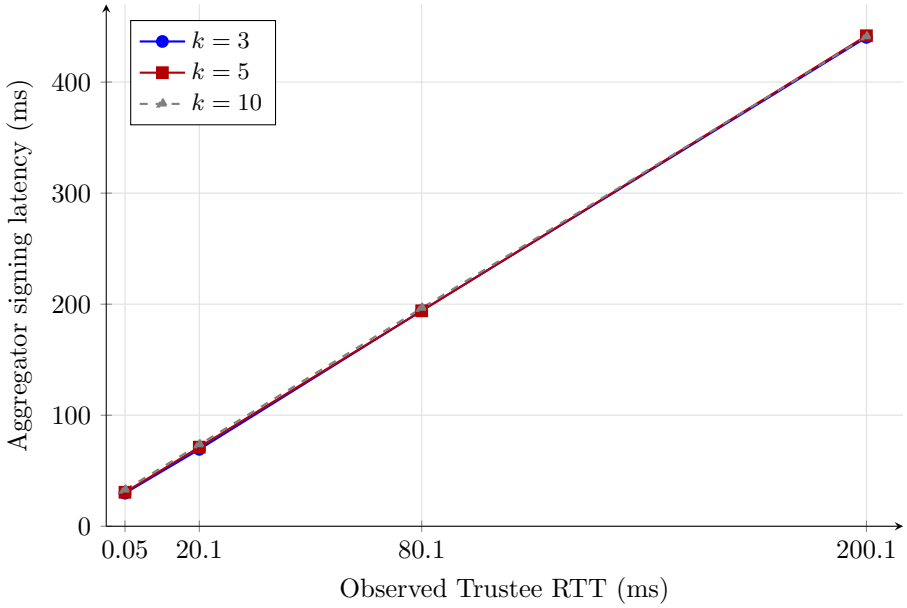


Figure 3: Aggregator signing latency as a function of measured Aggregator–Trustee RTT. Points show the median over 50 signatures per condition; error bars span the first and third quartiles and are largely obscured by the markers at this scale.

To quantify this relationship while accounting for temporal drift, we fit signing latency against measured RTT independently within each randomized block, using the four profile medians of that block. The mean slopes across the five blocks are 2.059 ± 0.011, 2.056 ± 0.011, and 2.042 ± 0.007 ms/ms for $k = 3$, 5, and 10, respectively, where the uncertainty is the standard deviation across blocks. Thus, throughout the evaluated range,

$$\Delta T_{\text{sign}} \approx 2\,\Delta\text{RTT}.$$

Coalition size primarily affects the local component of signing rather than its sensitivity to network delay. Increasing k from 3 to 10 raises the unshaped median by only 2.93 ms, whereas the fitted RTT slopes remain effectively unchanged. Hence, under the evaluated conditions, larger coalitions incur additional fan-out/fan-in and processing cost but do not materially amplify RTT-dependent latency.

3.3.3 Why signing follows two RTTs

The protocol-level instrumentation explains the factor of two. Across coalition sizes, the fitted RTT slopes are 1.021–1.031 for Round 1 and 1.021–1.027 for Round 2. Together, the two rounds account for approximately 88% of Aggregator latency in the baseline and 99.1–99.2% at 200 ms RTT. For $k = 3$, for example, their median durations increase from 14.40 and 11.78 ms in the baseline to 220.40 and 216.85 ms at 200 ms RTT, while the remaining Aggregator work remains approximately 3.4 ms.

Per-RPC timing confirms that this behavior results from parallel fan-out within each round rather than sequential contact with the trustees. Median launch skew is 0.040 ms in Round 1 and 0.032 ms in Round 2, and a complete round exceeds its slowest RPC by a median of only 0.049 ms. Since Round 2 cannot begin until Round 1 completes, a homogeneous network contributes approximately one RTT to each of two sequential barriers, explaining the observed 2 RTT dependence.

3.3.4 Heterogeneous paths

The heterogeneous campaign tests whether the same model holds when trustee RTTs differ. Starting with all ten trustees at 20 ms, increasing the RTT of only one required trustee to 200 ms raises median signing latency by 352.66 ms. Increasing the remaining nine trustees to 200 ms adds only a further 13.84 ms. The single slow path alone increases Round 1 and Round 2 medians by 176.34 and 176.76 ms, respectively.

In all 50 signatures measured with one slow trustee, that trustee was both the longest-running and the last-finishing RPC in each round. The median gap between complete round duration and the wall-clock span of its RPCs was only 0.20 ms for Round 1 and 0.25 ms for Round 2. Thus,

$$T_{\text{round}} \approx \max_i T_{\text{RPC},i}.$$

When path heterogeneity is dominated by propagation delay, the corresponding first-order network model is therefore

$$T_{\text{network}} \approx 2 \max_i \text{RTT}_i.$$

Client-observed latency provides an independent consistency check: across the primary experiment, its median difference from the instrumented Aggregator latency is only 1.81 ms.

3.3.5 Scope of the result

The experiment isolates RTT rather than total communication cost. Although coalition size has little effect on the RTT-dependent component of latency, the serialized Protocol Buffer payload increases from approximately 11.10 KB per signature at $k = 3$ to 39.26 KB at $k = 10$, excluding gRPC, HTTP/2, TCP, IP, and link-layer overhead. The results therefore do not imply that communication cost is independent of coalition size or that the same behavior would hold on bandwidth-constrained links.

Likewise, all containers execute on a single Linux host and network distance is emulated with `netem`. The measurements characterize signing as a function of controlled Aggregator–Trustee RTT; they do not attempt to reproduce packet loss, jitter, congestion, bandwidth constraints, or other effects of a geographically distributed Internet deployment.

Overall, the implementation demonstrates that sharded Haystack can preserve both one-time state discipline and the standard LMS verification interface in a distributed deployment. Its network-sensitive signing path is governed by the two sequential interaction rounds: under homogeneous paths each contributes approximately one RTT, while under heterogeneous paths each is bounded by the slowest required trustee. This implementation provides the concrete Haystack baseline for the trust-reduction mechanisms introduced next.

4 Minimizing trust assumptions in Haystack

4.1 Limitations in Haystack: the dealer assumption

As established in Section 2.2.2, Haystack removes trust from the aggregator and the CRV, and requires an entire coalition to be compromised through the trustees. However, it retains a fully trusted dealer to runt he setup. We now isolate the capabilities that follow from this assumption before considering how each can be relaxed.

4.1.1 Dealer's power

The dealer runs $\mathsf{StatefulGen}(D)$ in the clear and therefore obtains the composite secret key CSK in full, together with every trustee key $K_1, \ldots, K_k$ and every per-leaf randomizer R. In a realistic implementation, this material may be

represented compactly: a single n-bit seed determines every chain start $\mathrm{SK}_{\mathrm{KeyID}}[i][0]$, and hence every hash chain, one-time public key, the Merkle tree and CPK.ROOT.

Security therefore depends on the dealer deleting all secret material after distributing the CRV, an act that no external party can verify without additional assumptions such as generation inside a Hardware Security Module. This setup authority gives the dealer three distinct capabilities, which we separate because they are addressed by different mechanisms.

1. **Universal forgery.** Holding CSK, the dealer can produce a valid signature under any KeyID, and thus impersonate any coalition.
2. **Perfect deniability.** By [8, Footnote-5], the triple (R, PATH, Z) produced by the dealer alone presents the same distribution to one produced by a genuine coalition. No verifier, and no trustee inspecting the signature, can attribute it.
3. **Silent violation of state safety.** The dealer can silently sign under a KeyID that no trustee ever consumed. The scheme's principal practical benefit (that a one-time key reuse requires all members of a coalition to fail simultaneously) therefore does not hold against it. This would easily lead to multiple signatures under the same key.

The severity of the above depends on the deployment. Where a single organization owns the dealer, the aggregator and all trustees, trusting the dealer is unproblematic: that organization controls the signing key by construction, and the threshold structure serves mainly to guard against accidental one-time key reuse. The assumption becomes untenable precisely in the setting that motivates threshold signatures in the first place, namely when the roles are split across mutually distrusting entities, and when the trustees are expected to derive assurance from the fact that no single party can sign on their behalf. This setting motivates the trust-reduction mechanisms considered below.

4.1.2 Our contributions

We seek to weaken the dealer assumption along two axes, corresponding to two distinct capabilities identified above. First, to remove the dealer's ability to impersonate a coalition undetectably, so that a signature which did not involve the genuine trustees can be recognized as such; this is addressed by the partial-DKG of Section 4.2, and does not require the dealer to forget anything. Second, to remove the existence of any single party holding a compressing key for the secret material, so that there is nothing to retain and nothing to delete; this is addressed by the full-DKG of Section 4.5. Throughout, we impose two constraints inherited from Haystack. First, the verification algorithm and the signature format must remain unchanged, so that signatures stay standard LMS objects and no verifier needs to be aware of the distributed setting. Second, the construction must continue to rely only on secure hash and pseudorandom functions, since assurance from well-studied symmetric primitives is the principal

advantage of hash-based schemes over others such as lattice- and isogeny-based threshold constructions.

4.2 Partial DKG: MPC assisted CRV construction

The partial-DKG does not prevent the dealer from forging valid LMS signatures, because the dealer still learns the complete CSK. Instead, it provides what we call *in-protocol impersonation resistance*: the dealer cannot impersonate the trustees within the distributed signing protocol because it never learns their K_t or SK_t. Dealer- and coalition-generated signatures remain identical once aggregated, but the dealer can no longer produce a valid protocol transcript on behalf of the genuine trustees. In this sense, its forgery capability moves from an in-band, indistinguishable action to an out-of-band one.

To achieve these properties we rely on MPC techniques when running the KeyGen algorithm. More specifically, to build a CRV field, the dealer and the trustees engage in a very lightweight MPC where they input their partial shares and collaboratively compute the XOR, i.e.

$$\mathrm{CRV.field} = [\mathrm{TV}] \oplus \bigoplus_{t=1}^{k} [\mathrm{S}_t]$$

where [a] represents a as a private input (we abstract from the underlying MPC protocol used), TV stands for the true value held by the dealer and S_t stands for the share of trustee t. Note that [a] $\oplus$ [b] means collaboratively computing the XOR of two secrets. We present the algorithm for the partial-DKG in Algorithm 1.

Notice that MPC is used to compute the values inside the CRV, but also the seed. The reason for doing it this way is twofold: (1) it prevents the dealer from choosing the seed unilaterally, so the resulting LMS master seed depends on fresh randomness contributed by the participating trustees, and (2) the seed is only seen in plaintext by the dealer, but the MPC protects the trustees' partial seeds.

Another aspect to notice is that CHK cannot be computed anymore as defined in Haystack without using some kind of OPRF where the dealer provides the input $(\mathrm{KeyID}, \mathrm{R}, \mathrm{n})$ and the trustee t provides the evaluation key K_t. This would be needed because the dealer should not learn K_t and the trustees should not learn R before round 1 of the signing protocol has finished. Instead, we have the dealer just adding a new entry

$$\mathrm{CRV.CHK} = \mathsf{Hash}(\mathrm{KeyID}, R).$$

After round 1, once R is provided by the aggregator, each trustee can verify this binding directly. This also reduces the CHK contribution to the CRV from $k \cdot n$ bits to n bits. A key remark here is that the check can only be applied if the trustees have access to the CRV. Otherwise, they could use a Merkle Tree to

Algorithm 1 Haystack setup with Partial-DKG

Trustee key generation:
Each trustee samples K_t locally.

Seed agreement (coin-flip):
Each trustee samples seed_t and together with the dealer they compute $[\mathrm{seed}] = \bigoplus_t [\mathrm{seed}_t]$. Only the dealer opens seed.

Dealer runs standard LMS StatefulGen**:**
$\mathrm{CPK}, \mathrm{CSK} \leftarrow \mathsf{StatefulGen}(D, \mathrm{seed})$.

Trustee shares are WOTS chains:
Generate $\mathrm{SK}_t[i][0] = \mathsf{PRF}_{K_t}(\mathrm{KeyID}, i)$ and then hash-chain.

Joint CRV construction (per leaf):
$\mathrm{CRV.R} = [\mathrm{R}] \oplus \bigoplus_{t=1}^{k} [\mathrm{R}_t]$
$\mathrm{CRV.PATH} = [\mathrm{PATH}] \oplus \bigoplus_{t=1}^{k} [\mathrm{PATH}_t]$
$\mathrm{CRV.SK} = [\mathrm{SK}] \oplus \bigoplus_{t=1}^{k} [\mathrm{SK}_t]$ ▷ This is XOR applied independently to each cell of the SK matrix.
$\mathrm{CRV.CHK} = \mathrm{Hash}(\mathrm{KeyID}, \mathrm{R})$.
$\mathrm{CRV.TPK} = H(PK_1 || ... || PK_t)$.

verify, by storing the root computed over CHK[KeyID] and have the aggregator supply CHK[KeyID] with a *log D*-node opening in round 2.

A second change is that trustee shares of SK are no longer arbitrary pseudo-random values: each trustee builds an actual Winternitz hash chain. This structure makes every SK_t a valid WOTS key and therefore gives trustee t a corresponding public key,

$$\mathrm{PK}_t = \mathsf{Hash}(\mathrm{SK}_t[0][2^w - 1] \parallel \ldots \parallel \mathrm{SK}_t[l-1][2^w - 1]).$$

Its partial signature Z_t can consequently be verified against PK_t, unlike the pseudo-random shares of Haystack. Since structured keys alone could still be fabricated by the dealer, we additionally include CRV.TPK as a commitment to the trustees' partial public keys. This binds the CRV to the genuine trustee keys and prevents the dealer from substituting its own shares within the protocol.

4.2.1 Using lightweight MPC

Even when the partial-DKG needs MPC, it can be done in a very lightweight manner in the semi-honest model because it only involves XOR operations, i.e., $\bigoplus_t [S_t]$, which is fully linear. For that, pairwise-masked XOR aggregation suffices. Each pair of trustees (t, t') derives a shared mask stream $m_{t,t'}$ from a pairwise seed and then trustee t sends $y_t = SK_t \oplus \bigoplus_{t' \neq t} m_{t,t'}$. Notice that masks are canceled in the dealer's global XOR $\bigoplus_t y_t$ because each mask appears

exactly twice. This method allows us to obtain a 1 round protocol with no heavy preprocessing rather than deriving the pairwise masks.

4.2.2 Trust model

The partial DKG changes the Haystack trust model without eliminating the dealer's residual signing power. Because the dealer still learns CSK, it can produce valid LMS signatures independently; what it can no longer do is impersonate the genuine trustees within the distributed protocol. The distinction therefore relies on dealer-generated signatures occurring out of band.

The MPC setup additionally requires private and authenticated channels, and long-term trustee identities. These identities bind the contributions used to construct the CRV to the genuine trustees and prevent the dealer from replacing them with Sybil participants.

Verification of trustee participation can then follow two models. In the optimistic model[5], the aggregated signature is accepted by default but may be challenged by requesting a *proof of distribution*: the partial signatures Z_t are checked against their corresponding PK_t, together with CRV.TPK. The aggregator may perform this verification and is assumed semi-honest for this challenge-verification step, although it holds no secret material. We remark that trustees can identify dealer forgeries (and therefore submit a challenge) if they see a valid signature under a KeyID they haven't removed from their local state tracker $\mathrm{keylist}_t$ (see Algorithm 11 in [8]).

Alternatively, the signature may always carry $\{\mathrm{Z}_t\}_{t\in[k]}$, with $\{\mathrm{PK}_t\}$ and the CRV publicly available to the verifier. This removes the optimistic assumption but increases the additional signature material linearly with k. The ordinary LMS component remains unchanged, while the verification material is carried as an optional extension:

$$\sigma = ((R, \mathrm{PATH}, Z), \{\mathrm{Z}_t\}_{t\in[k]}).$$

Finally, the protocol remains secure against trustee reconstruction as long as the dealer does not collude with $k-1$ trustees; otherwise, it can unblind the remaining trustee's SK_t and produce a partial signature on its behalf. This is the corresponding k-of-k threshold assumption.

4.3 Audit layer with succinct arguments

The partial-DKG of Section 4.2 makes trustee participation verifiable, but the partial signatures are verified by the aggregator. Public verification could be obtained by attaching $\{\mathrm{Z}_t\}_{t\in[k]}$ to every signature and making $\{\mathrm{PK}_t\}_{t\in[k]}$ and the CRV publicly available. This preserves the standard LMS signature as the base object, but the additional verification material grows linearly with k.

[5] Optimistic models are common in the blockchain setting, e.g., optimistic roll-ups, because they trade unconditional verification for efficiency.

An alternative to this, in order to achieve a signature with sublinear length, would be to rely on succinct arguments. By using, e.g., a SNARK [15], the aggregator would be able to provide a single proof ρ for the relation that captures the verification of each partial signature Z_t, linked to the trustees known identities, i.e.,

$$\begin{aligned}&\mathsf{R}_{\mathsf{aud-sig}}(\text{params}, \theta, \omega)\\ &\quad \theta = (\mathrm{CPK}, M, \sigma, \mathrm{CRV.TPK[KeyID]}, \{\mathrm{PK}_t\})\\ &\quad \omega = (\{\mathrm{Z}_t\}))\end{aligned}$$

and the verifications inside the argument are

$$\begin{aligned}&\forall t \in [k] : \mathsf{WOTS.Verify}(\mathrm{Z}_t; \mathrm{PK}_t) \wedge\\ &\mathrm{CRV.TPK[KeyID]} = \mathsf{Hash}(\mathrm{PK}_1 \| \ \ldots \| \ \mathrm{PK}_k) \wedge\\ &\mathrm{Z} = \mathrm{Z}_{\mathrm{CRV}} \oplus \bigoplus_{t \in [k]} \mathrm{Z}_t.\end{aligned}$$

This way, public verifiers can check (efficiently) with only access to $\{\mathrm{PK}_t\}_{t\in[k]}$, but without the need of receiving all $\{\mathrm{Z}_t\}_{t\in[k]}$, i.e., the signature would be $\sigma = ((R, \mathrm{PATH}, \mathrm{Z}), \rho)$. The circuit for the proof ρ would need k WOTS verifications, which is $\approx k \cdot p \cdot (2^w - 1)$ hash calls (plus the TPK hash and signature linkage, which is negligible). Notice that this solution introduces a similar problem than in the MPC case, i.e., computing SHA-256 hashes (needed for standard-compatibility) inside the SNARK can become very expensive.

To address this setting more generally, we recall to the work by Khaburzaniya et al. [16], which studies approaches to produce hash-based signatures using STARKs [17], for both the aggregated and threshold settings. One key result is that they provide a lower bound of 50 for the number of signatures to be aggregated. The reason is that the proof is large (because STARKs are transparent, without setup), ranging from 100 to 200 KB.

4.4 Signature anonymity and accountability

Attributing a signature to its coalition and hiding which coalition produced it are complementary goals: a scheme is *coalition-accountable* with respect to a class of observers exactly when it fails to be *coalition-anonymous* against that class. Therefore, the interesting question is determining which observers sit on each side of the boundary, and whether that placement is a design parameter or a consequence of the construction. We distinguish *external verifiers*, holding only $(\mathrm{CPK}, M, \sigma)$, from *infrastructure observers*, who additionally hold the public data needed to produce signatures — for Haystack, the coalition list CL and the CRV.

In Haystack, the placement is structural. Every signature carries a public KeyID, extractable from PATH and indispensable for verification. Sharding

makes CL a total map from KeyID to coalitions, so attribution is a table lookup rather than a cryptographic operation, and it is available to every infrastructure observer. Against an external verifier the guarantee weakens only to pseudonymity, not anonymity: the KeyID is a persistent label, so signatures by one coalition remain linkable.

In succinct-argument approaches such as [16], disclosure is decided by the relation. Exposing the signing subset as a public input gives accountability and no anonymity, which is why distinct authorized subsets yield distinguishable proofs. On the other hand, proving subset membership in-circuit against a commitment to the trustee set, with only the threshold public, instead gives (weak) anonymity and unlinkability against every observer. Attribution is then lost entirely unless a tracing trapdoor is added, which reintroduces a trusted opener. The freedom is paid for in the verification interface: signatures are no longer standard LMS objects and a proof-friendly hash is required.

That the sharded setting forecloses anonymity is a property of sharding, not of hash-based signatures. The group signature of [18], a 1-of-ℓ construction over a stateful HBS, achieves signer anonymity from hash assumptions alone, at the cost of a designated opener holding a tracing capability (the classical accountable-ring compromise, transplanted to the post-quantum setting). Regarding PRAWNS [9], authors state to achieve a classic $t\text{-}of\text{-}n$ HBS where neither the public key nor the signature reveal t or n. However, they rely on lattice assumptions for their core threshold PRF construction.

Haystack thus occupies the accountability end of this axis by construction, and receives verifier compatibility in exchange. The audit layer of this section does not move it: since KeyID $\mapsto C$ is already total, a succinct argument attesting participation reveals nothing new to any holder of CL, and its only privacy cost (disclosing C and $|C|$ to external verifiers) is incurred solely when the optional tier is consulted.

4.5 Full DKG: eliminating the dealer entirely

The partial DKG introduced in Section 4.2 still requires a dealer that learns the true values before they are masked. Full DKG removes the dealer, since the k trustees generate the CRV jointly through MPC, without any participant learning the corresponding unmasked value.

We focus specifically on the way to obtain the SK hash-chain matrix, since it is by far the heaviest component of the CRV (see Table 1). One important aspect to notice is that each column of the SK matrix is generated independently from the others, which let us room for MPC speed up by parallelization through SIMD techniques.

First, for notation simplicity, let's assume we have just selected one column (chain) of the SK matrix. Then, let $\mathsf{CRV.SK}_i$ be the value on the i-th position of the chain in the $\mathsf{CRV.SK}$ field and let SK_t^i be the value from the t-th trustee

on the i-th position of the chain share. Let S_i be the real value on the i-th position of the chain that is never revealed, but computed via MPC in order to get the values from the CRV. Then, to compute each $\mathsf{CRV.SK}_i$, we need to distinguish two cases: the base one ($i = 0$) and the rest of the chain.

The base case is handled slightly different because $\mathsf{CRV.SK}_0$ is not derived from anything. Instead it is sampled independently at random and published, and the (never stored) virtual-dealer value at position $i = 0$ is defined from it

$$[S_0] = \mathsf{CRV.SK}_0 \oplus \bigoplus_{j=1}^{k} [SK_j^0].$$

The equation above is the opposite order to every subsequent position, where $\mathsf{CRV.SK}_i$ is computed from S_i rather than the other way around. Picking $\mathsf{CRV.SK}_0$ independently, rather than letting it coincide with $\bigoplus_j \mathsf{SK}_j^0$, avoids a degenerate case in which S_0 would always take the same value for a given set of trustee inputs.

For $i \geq 1$, $\mathsf{CRV.SK}_i$ just follows the recursive definition

$$[\mathsf{CRV.SK}_i] \;=\; H\left([\mathsf{CRV.SK}_{i-1}] \oplus \bigoplus_{j=1}^{k} [\mathsf{SK}_j^{i-1}]\right) \oplus \bigoplus_{j=1}^{k} [\mathsf{SK}_j^{i}].$$

Figure 4 depicts this and works through the first three positions explicitly for $k = 3$ trustees. The leftmost column shows the virtual dealer's value at each position purely for exposition as it is never written to storage, transmitted or held by any single party at any point in time. It exists only fleetingly, inside the MPC computation, immediately before being re-masked and revealed as the corresponding $\mathsf{CRV.SK}_i$.

We evaluate this construction experimentally in Section 5.2.

5 Trustless setup evaluation

Unlike the complete validation for Haystack in Section 3, this section focuses on the evaluation of the additional MPC setup cost introduced by both the partial- and full-DKG constructions. Both variants have been implemented using MP-SPDZ [19], a general purpose MPC framework that incorporates many different MPC protocols as backend. For our particular use case, we relied on the `semi-bin` protocol — its foundations come from [20,21] — with semi-honest security. The code for both variants is available online [6] [7]. We measured all the tests on an Intel Core i7-13620H CPU @ 2.40 GHz with 16 GB of RAM, Windows 11.

[6] https://anonymous.4open.science/r/threshold-hbs-partial-dkg/
[7] https://anonymous.4open.science/r/threshold-hbs-full-dkg/

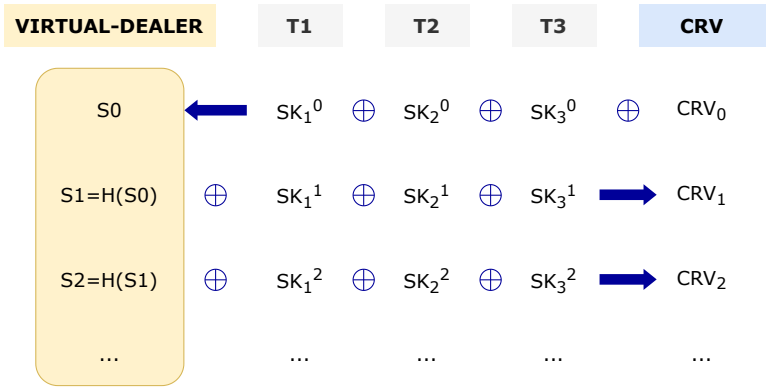


Figure 4: First three positions of a CRV chain for $k = 3$ trustees. The virtual dealer's column (shaded) is never materialized outside the MPC computation. Only the trustees' shares and the resulting CRV_i values are ever stored or revealed.

5.1 Partial-DKG proof-of-concept evaluation

For the sake of simplicity, we have run the partial-DKG setup with $k = 3$ trustees and with the results averaged over 5 runs. The parameters match the real size used in the paper's cost estimates: $w = 4$, and therefore $p = 67$ chains of $2^w = 16$ steps each, with $n = 32$ bytes per node, leading to a matrix of $67 \times 16 \times 32 = 34,304$ bytes. The results from running time and communication costs are shared in Table 5. It can be seen that this procedure is actually very lightweight and fast (compared to the ≈ 8 ms in the baseline algorithm) because it only requires XORing the inputs. However, the bottleneck will be determined by the network profile (RTT between the dealer and the trustees).

Table 5: Cost of the partial DKG, in terms of running time and communication cost.

Metric	Mean	Std. dev.
Time, dealer (ms)	31.44	0.93
Data sent, dealer (MB)	0.1157	—
Global data (MB)	0.2571	—

5.2 Full-DKG proof-of-concept evaluation

The full-DKG has also been tested with $k = 3$ trustees and averaged over 5 runs. In this case, the full hash-chain is computed by MPC, and therefore it requires SHA-256 evaluation on private inputs. For that, we rely on the MP-SPDZ built-in of the SHA-256 circuit — `Compiler.circuit.sha256` —. In order to

provide the results, we first focus on the cost of a single hash chain — with different lengths, i.e., $\mathsf{n_steps} \in \{4, 16, 256\}$ — and then we also provide some insights on the differences between computing the whole matrix sequentially or using parallelization speed ups, i.e., SIMD techniques.

First, we clarify the properties that are measured. Global communication captures the total of the 3 parties, and while the amount of data sent per party is symmetric, the number of communication rounds (round-trips) is asymmetric: P_0 performs twice the communication rounds of P_1 or P_2, because it is the MPC coordinator. The VM rounds is the number of sequential synchronization steps the circuit's Beaver-triple evaluation requires (essentially its multiplicative depth). Then, the number of bit triples determines the number of AND-gate triples consumed — 1 fixed-size Bristol-Fashion SHA-256 circuit has 22,573 AND gates —. Finally, it is worth to notice that the results obtained include preprocessing — offline phase —.

Table 6: Cost of generating a full CRV column with $k = 3$ trustees.

n_steps	Running time (s)	Global data (MB)	VM rounds	Bit triples
4	1.60	6.41	4,826	67,719
16	11.99	19.35	24,122	338,595
256	107.15	290.70	410,042	5,756,115

Table 7: Cost of generating a full CRV with different number of columns with $k = 3$ trustees, MP-SPDZ `semi-bin` protocol, averaged over 5 runs per configuration. (This one is for $n_{steps} = 16$)

n_{cols}	Running time (s)	Bit triples (AND gates)	VM rounds	Global data sent (MB)	Comm rounds (P_0)	Comm rounds (P_i)
1	1.785	338,595	24,122	19.351	96,602	48,349
2	3.924	677,190	48,242	35.522	193,172	96,669
4	7.978	1,354,380	96,482	71.018	386,330	193,325
8	16.794	2,708,760	192,962	138.856	772,628	386,621
16	31.705	5,417,520	385,922	274.532	1,545,224	773,213
32	75.978	10,835,040	771,842	549.038	3,090,434	1,546,413

Table 6 shows the cost to compute a single CRV column with different lengths parametrized by n_steps, while Table 7 shows the cost to sequentially compute the full matrix — with a fixed $\mathsf{n_steps} = 16$ — for different number of columns. Unlike clock time, which is subject to measurement noise, the protocol level metrics scale exactly linearly with n_steps:

$$\text{bit triples} = 22{,}573 \times (\mathsf{n_steps} - 1),$$
$$\text{VM rounds} \approx 1{,}608 \times \mathsf{n_steps} - 1{,}606.$$

Each additional SHA-256 evaluation in the chain therefore costs a constant 22,573 non-linear operations regardless of its position. Because the per-step cost is constant, the cost of a complete SK matrix ($p = 67$ chains) or of the full CRV

(D leaves) can be estimated directly from these figures using the scaling factors given in the paragraph above, without having to run either at full size. However, since hash chains are independently generated, we noticed that the MPC run can be sped up by parallelization techniques, i.e., using SIMD, which are already provided by the MP-SPDZ backend. The comparison between the sequential and parallelized cases is shown in Figure 5, which confirms a significant speed up, on the order of one magnitude.

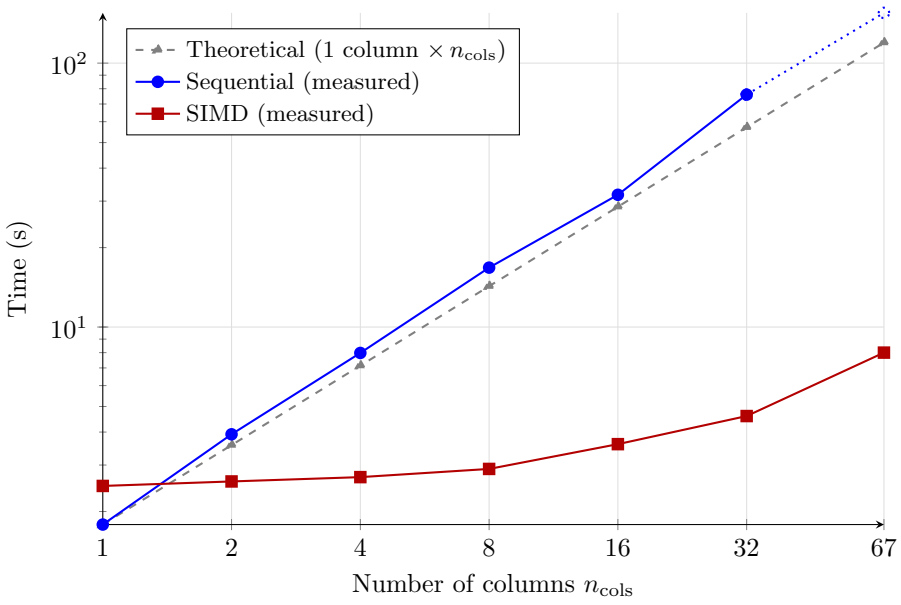


Figure 5: Running time (s) to complete the full DKG between 3 trustees for different number of columns. Two settings are compared: *sequential* and *parallel (SIMD)* computation of the columns.

The costs shown in Table 7 and Figure 5 reflect the impact of the MPC. Although SIMD introduces a significant improvement, it still takes ≈ 8 seconds (without considering network delays) to produce a full matrix of 67 columns, compared to the ≈ 8 ms and 31.44 ms it takes in the baseline and Partial-DKG protocols, respectively. Since this is for a single one-time key, building a full tree of $D = 2^{15}$ leaves can go easily to more than one day of computation. As it happens with other cryptographic applications built on arithmetic circuits, it is worth analyzing the potential benefit of using arithmetization-oriented hash functions such as Poseidon2 as an alternative. Concurrent work by Adomnicăi [22] evaluates Poseidon2 hash chains directly within MP-SPDZ and reports that, for their most MPC-friendly instantiation (the 31-bit KoalaBear prime field, state width $t = 16$, S-box exponent $d = 3$), a single 2-to-1 compression call requires a multiplicative depth of only 28 and consumes 296 preprocessed field elements — compared to the 22,573 non-linear operations we measured per SHA-256

evaluation in the binary-circuit setting —. While the two figures are not directly comparable, the gap is large enough to suggest that switching the hash function inside the same full-DKG construction described above could plausibly bring the full-tree cost down by orders of magnitude, turning a costly one-time setup into a efficient one. Their work further reports that, for an LMS-style scheme using the same Winternitz parameter $w = 4$ as we have been using and combined with batched precomputation across four parallel signatures, signing completes in under one second on a 3-party, 20 ms round-trip-time network. While this reinforces Poseidon2 as a promising direction for a full-DKG implementation of our own scheme, there are two main details to be kept in account. First, that a Poseidon2-based instantiation would no longer produce standard LMS signatures — forcing updates on the verfier code —, and second, that it would replace a well-studied hash function with another one which is under active cryptanalysis[8] [19, 23, 24]. In fact, it is worth noting that the Ethereum Foundation has recently — at the time of writing — decided to drop Poseidon for SHA and BLAKE[9] in their long-term roadmap. Therefore, whether the trade introduced by Poseidon is worthwhile is a question we leave open.

6 Conclusions and future work

This work addresses the stateful HBS construction of Haystack and two of its current limitations. The first limitation is the lack of network-aware benchmarking, which we provide through a Java implementation of Haystack together with experiments run under different network profiles. These allow us to conclude that signing is efficient and comparable to other post-quantum threshold schemes [25, 26] (in both communication and runtime), with $\approx$ 8 ms per signature in the local setting and $\approx 2\,\Delta\text{RTT}$ additional cost under real network conditions (cost dominated by slowest trustee). Haystack also requires a trusted setup which is large, but its generation can be highly parallelized even for large trees, lowering theoretical hours to seconds. The second limitation is the amount of trust placed in the dealer, which can trivially produce forgeries indistinguishable from signatures generated by the threshold protocol. We analyze two directions that mitigate this issue while keeping the protocol as close as possible to the original and leaving the signatures produced in their standard form. The Partial-DKG shows that, at minimal additional cost through a lightweight MPC, one obtains an optimistic setting in which forgeries by the dealer become detectable. The Full-DKG then addresses the costly case in which the setup is run entirely under MPC and the dealer is removed altogether. While these costs are likely prohibitive for most use cases (from a few milliseconds in the baseline to the order of 10-100 seconds for a single leaf), we show that SIMD techniques substantially improve setup runtime by parallelizing the construction of the independent hash chains of a single one-time key.

[8] https://www.poseidon-initiative.info/

[9] https://www.cryptotimes.io/2026/08/13/ethereum-foundation-drops-poseidon-hash-for-sha-blake-in-l1-pivot/

As future work, we aim to extend the benchmarking of the baseline Haystack scheme to geographically distributed cloud instances, yielding more realistic measurements of the network's impact on the protocol. The same applies to the Partial- and Full-DKGs, which will replace the baseline trusted setup in our deployment so that the effect of these distributed setups can be assessed under real network conditions. Finally, both the trust assumptions and the performance of the two DKGs leave considerable room for improvement (principally in the Full-DKG) through the exploration of alternative MPC protocols and concrete techniques for running the setup more efficiently.

Acknowledgment

This work has been partially supported by...

References

[1] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134.

[2] L. Lamport, "Constructing Digital Signatures from a One Way Function," Tech. Rep. CSL-98, Oct. 1979, edition: SRI International. [Online]. Available: https://www.microsoft.com/en-us/research/publication/constructing-digital-signatures-one-way-function/

[3] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," in *Advances in Cryptology — CRYPTO '87*, C. Pomerance, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 369–378.

[4] D. McGrew, M. Curcio, and S. Fluhrer, "Leighton-Micali Hash-Based Signatures," RFC 8554, Apr. 2019. [Online]. Available: https://www.rfc-editor.org/info/rfc8554

[5] A. Huelsing, D. Butin, S.-L. Gazdag, J. Rijneveld, and A. Mohaisen, "XMSS: eXtended Merkle Signature Scheme," RFC 8391, May 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8391

[6] D. Cooper, D. Apon, Q. Dang, M. Davidson, M. Dworkin, and C. Miller, "Recommendation for stateful hash-based signature schemes," National Institute of Standards and Technology, Gaithersburg, MD, United States, NIST Special Publication NIST Special Publication 800-208, 2020.

[7] D. Cozzo and N. P. Smart, "Sharing the LUOV: Threshold post-quantum signatures," in *Cryptography and Coding – 17th IMA International Conference, IMACC 2019*, ser. Lecture Notes in Computer Science, M. Albrecht, Ed., vol. 11929. Springer, 2019, pp. 128–153.

[8] J. Kelsey, N. Lang, and S. Lucks, "Turning hash-based signatures into distributed signatures and threshold signatures: Delegate your signing capability, and distribute it among trustees," *IACR Communications in Cryptology*, vol. 2, no. 2, 2025.

[9] D. Boneh, Benedikt, Justin, Y. Litchev, and K. Nazirkhanova, "PRAWNS," 2026. [Online]. Available: https://lclhost.org/blog/prawns/

[10] D. Boneh, K. Lewi, H. Montgomery, and A. Raghunathan, "Key Homomorphic PRFs and Their Applications," 2015. [Online]. Available: https://eprint.iacr.org/2015/220

[11] Y. Desmedt, "Society and group oriented cryptography: a new concept," in *Advances in Cryptology — CRYPTO '87*, C. Pomerance, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 120–127.

[12] V. Shoup, "Practical threshold signatures," in *Advances in Cryptology — EUROCRYPT 2000*, B. Preneel, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 207–220.

[13] J. Herranz, C. Padró, and G. Sáez, "Distributed rsa signature schemes for general access structures," in *Information Security*, C. Boyd and W. Mao, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 122–136.

[14] J. Kelsey, S.-j. Change, and R. Perlner, "SHA-3 derived functions: cSHAKE, KMAC, TupleHash and ParallelHash," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST SP 800-185, Dec. 2016. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf

[15] A. Nitulescu, "zk-SNARKs: A Gentle Introduction," p. 50.

[16] I. Khaburzaniya, K. Chalkias, K. Lewi, and H. Malvai, "Aggregating and thresholdizing hash-based signatures using starks," in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 393–407. [Online]. Available: https://doi.org/10.1145/3488932.3524128

[17] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," 2018.

[18] R. E. Bansarkhani and R. Misoczki, "G-Merkle: A Hash-Based Group Signature Scheme From Standard Assumptions," 2018. [Online]. Available: https://eprint.iacr.org/2018/274

[19] N. Keller and A. Rosemarin, "Mind the Middle Layer: The HADES Design Strategy Revisited," 2020, publication info: Published by the IACR in EUROCRYPT 2021. [Online]. Available: https://eprint.iacr.org/2020/179

[20] M. Keller, E. Orsini, and P. Scholl, "MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer," 2016, publication info:

Published elsewhere. Major revision. ACM CCS 2016. [Online]. Available: https://eprint.iacr.org/2016/505

[21] M. Keller, "MP-SPDZ: A Versatile Framework for Multi-Party Computation," 2020, publication info: Published elsewhere. Minor revision. ACM CCS 2020. [Online]. Available: https://eprint.iacr.org/2020/521

[22] A. Adomnicăi, "Towards practical multi-party hash chains using arithmetization-oriented primitives: With applications to threshold hash-based signatures," *IACR Communications in Cryptology*, vol. 2, no. 4, 2026.

[23] T. Ashur, T. Buschman, and M. Mahzoun, "Algebraic Cryptanalysis of HADES Design Strategy: Application to POSEIDON and Poseidon2," 2023, publication info: Preprint. [Online]. Available: https://eprint.iacr.org/2023/537

[24] Z. Zhao, A. Sanso, G. Vitto, and J. Ding, "Graeffe-Based Attacks on Poseidon and NTT Lower Bounds," 2025, publication info: Preprint. [Online]. Available: https://eprint.iacr.org/2025/1916

[25] S. Celi, R. d. Pino, T. Espitau, G. Niot, and T. Prest, "Efficient Threshold ML-DSA," 2026, publication info: Published elsewhere. Major revision. USENIX Security '26. [Online]. Available: https://eprint.iacr.org/2026/013

[26] A. Bienstock, L. d. Castro, D. Escudero, A. Polychroniadou, and A. Takahashi, "Quorus: Efficient, Scalable Threshold ML-DSA Signatures from MPC," 2025. [Online]. Available: https://eprint.iacr.org/2025/1163