

Nonstop Multihop? Constructions and Lower Bounds for Re-Aggregatable Multisignatures

Lucjan Hanzlik¹ Julian Loss² Omar Renawi^{1,3}

¹CISPA Helmholtz Center for Information Security
{hanzlik, omar.renawi}@cispa.de

²Ruhr University Bochum
lossjulian@gmail.com

³Saarland University

Abstract. Multisignatures compress many signatures on a common message into a single aggregate. In large distributed systems, aggregation occurs over multiple hops, where partial aggregates may overlap. The recent hint-free multisignatures of Hofheinz, Reichle, and Wagner (TCC 2026) eliminate aggregation-topology hints, but verification still requires the complete set of participating public keys. This leaves open whether the signer-set information itself can be eliminated. We answer this question negatively. We introduce *idempotent multisignatures* to capture unrestricted aggregation of overlapping aggregates and prove that any such scheme has a signature size linear in the number of signers.

To circumvent this lower bound, we introduce *unique multisignatures*, a sequential alternative that eliminates this metadata and retains constant-size signatures. We construct unique multisignatures using incrementally verifiable computations. Finally, replacing signature chains in authenticated Dolev–Strong Byzantine Broadcast with our primitive enables two-round termination when the sender is honest and reduces communication from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ against $t < n$ adaptive corruptions.

1 Introduction

In (non-interactive) multisignatures (MS) [22], a group of N independent signers jointly endorse a message $\mathbf{m}$ by producing N signature shares. Once these shares are collected, they can be combined into a single constant-size aggregate signature, ideally no larger than an individual share. This aggregate can then be publicly verified to demonstrate that all N signers endorsed $\mathbf{m}$. Multisignatures are a fundamental cryptographic primitive with numerous applications, including Bitcoin [31,28,32] and Proof-of-Stake systems [17,6,7].

When the number of signers is small, all signature shares can be collected and aggregated directly in a single stage. This approach, however, does not scale to large networks such as Proof-of-Stake blockchains [17], where requiring every signer to broadcast a signature share to the entire network would generate

prohibitive communication overhead. In Ethereum, for example, tens of thousands of validators¹ sign attestations using the non-interactive BLS multisignature scheme [8,6,7]. Rather than broadcasting every signature share globally, validators first send their shares, via a gossip protocol, to a subset of *designated aggregators* within their committees. Each designated aggregator combines the shares it receives into a partial aggregate and broadcasts it to the global network. A *block proposer* then combines several such partial aggregates into the final aggregate signature. This multi-hop aggregation process substantially reduces global network traffic.

Fault tolerance, however, creates a complication. Each committee has, on average, 16 aggregators [16], and different aggregators receive shares from overlapping sets of validators. Consequently, the same validator’s signature share may appear in multiple partial aggregates. The block proposer, therefore, cannot blindly combine partial aggregates from the same committee, since doing so may include the same share multiple times in the final signature. At the same time, selecting a single partial aggregate is undesirable because that aggregate may represent only a small fraction of the committee. Thus, the simple aggregation procedure used in a single-hop setting, where no auxiliary information accompanies a share or aggregate, is insufficient in the multi-hop setting.

Ethereum addresses this problem by attaching a bitmask to every partial aggregate that identifies exactly which validators contributed signature shares [17]. After receiving the partial aggregates and their corresponding bitmasks, the block proposer selects a collection whose bitmasks are pairwise disjoint, so that no validator is counted more than once. Among such collections, the proposer aims to maximize the number of participating validators, corresponding to the number of set bits in the bitwise OR of the selected masks. This solution has two significant drawbacks. First, the bitmask introduces an overhead that grows linearly with the number of validators, eliminating the constant-size advantage of multisignatures. Second, selecting the optimal collection of partial aggregates yields the *Attestation Aggregation and Packing Problem* (AAPP), an NP-hard optimization problem [19,37].

One can avoid the AAPP by replacing the bitmask with a vector of counters that records the multiplicity of each validator’s signature share. Partial aggregates could then be combined without requiring the signer sets to be disjoint, while the counters would track duplicate contributions. This approach, however, does not solve the underlying communication problem: if ψ upper-bounds the multiplicity of any validator’s share, the counter vector incurs an overhead of $N \cdot \log_2(\psi)$ bits. Hence, both approaches require auxiliary information whose size grows linearly with the number of signers.

Constant-size Signatures. Ideally, we would like to aggregate signatures arbitrarily while keeping the sizes of both the partial and final aggregate signatures constant with respect to the number of validators. In particular, aggregation

¹ Approximately a 1/32 fraction of all active validators; see <https://beaconcha.in/charts/validators>.

should not require broadcasting a bitmask or maintaining per-signer multiplicity information. Achieving this goal is currently listed as an open problem in Ethereum research [18]. We formalize this ideal functionality through the notion of *non-interactive idempotent multisignatures*. This leads to the first question addressed in this work:

Can we construct a non-interactive idempotent multisignature with constant-size signatures that supports repeated aggregation over overlapping signer sets, without tracking the multiplicity of individual signers?

We answer this question *negatively*. We prove that such a primitive cannot exist. Otherwise, it would imply an unconditional lossless compression algorithm capable of compressing random data beyond information-theoretic limits [11,35,34]. In fact, our impossibility result is stronger: it continues to hold even when the signature size is $N - 1$ bits.

A Practical Alternative. The impossibility result suggests that tracking the exact set of participating signers is the only approach if one simultaneously requires unrestricted aggregation and constant-size signatures. Fortunately, many consensus applications do not need to know *which* participants signed. It is often sufficient to verify only *how many* distinct participants contributed to an aggregate signature.

Motivated by this observation, we move away from arbitrary multi-parent aggregation and instead adopt a sequential aggregation paradigm. This gives rise to our notion of *non-interactive unique multisignatures*. In a unique multisignature scheme, signatures are aggregated sequentially, and each aggregation step increases the signature’s *depth*. An aggregate signature of depth $k \in [N]$ convinces a verifier that k distinct signers participated in signing the message, without revealing their identities.

The depth need not be encoded explicitly in the signature. To preserve constant-size signatures, the verifier may instead test validity for each possible value $k \in [N]$. Thus, the signature itself remains a constant size regardless of the number of signers. This leads to the second question of this work:

Can we construct unique multisignatures with constant-size signatures?

We answer this question *positively*, but with a caveat. Relying on the depth to verify how many signers have signed a message has an obvious flaw: a single malicious signer can sign the message and increase the counter k times to convince the verifier that k different signers have signed the message. Preventing such a malicious behavior under standard assumption ² requires encoding the signers identities in the aggregate signatures, which brings us back to the same impossibility result introduced above. To break this cycle, we show that such a primitive with constant-size signatures can be constructed assuming the signers can be restricted to contribute at most once to an aggregate signature. This can be done using trusted hardware environments or one-shot signatures [2,36]. Both

² The meaning of “standard assumptions” will be clarified in the following.

approaches can be used to limit the number of signing operations each signer can perform in the protocol run, ensuring they can sign at most once per epoch. We provide such a construction and demonstrate its practicality by providing a byzantine broadcast protocol on top of it.

Byzantine Broadcast with Early Termination. The Dolev-Strong protocol [14] is an authenticated Byzantine Broadcast protocol enabling a group of n parties $P_1, \dots, P_n$ to reach agreement over a synchronous network. Assume without loss of generality that P_1 is the sender with an input value v , and that $1 \leq t \leq n - 1$ is the number of corrupted parties. The protocol ensures with overwhelming probability that after $t + 1$ rounds: (i) all honest parties terminate and decide some value v' (*Termination*), (ii) all honest parties decide on the same value v' (*Agreement*), and (iii) if the sender is honest, then all honest parties decide on v (*Validity*).

A drawback of this protocol prevents honest parties from terminating before completing $t + 1$ rounds. A malicious sender P_1 colluding with corrupted parties can exploit early termination to equivocate and break Agreement. The main issue allowing this attack is the attacker's ability to sign two distinct values. Restricting the signing ability of each signer to one signature per protocol run can solve this issue. Since our construction of unique multisignatures provides exactly this restriction, it is a perfect fit to substitute the typical signature chains in standard Dolev-Strong byzantine broadcast implementations.

Because the signing ability is restricted to one signature per epoch, if the sender is honest, every honest party terminates safely in round 2. Otherwise, a malicious sender can force the protocol to run for $t + 1$ rounds. Due to the constant size of our unique multisignature, the protocol messages sent by the parties are of size $1 + 2 \cdot k$, where k is the constant signature size. Therefore, the communication complexity of our protocol is reduced from the original $\mathcal{O}(n^3)$ down to $\mathcal{O}(n^2 \cdot (l + k))$, where $l := 1$ and $2 \cdot k$ is the signature size.

1.1 Related Work

Authenticated Byzantine Broadcast. Dolev and Strong [14] gave the classical deterministic authenticated Byzantine Broadcast protocol, tolerating up to $t = n - 1$ corruptions in $t + 1$ rounds. Dolev and Reischuk [15] established fundamental lower bounds on the communication required for authenticated Byzantine agreement. More recently, Momose and Ren [30] obtained quadratic communication under weaker resilience assumptions, including $t < n/2$ with threshold signatures and trusted setup.

Early Termination. A related line of work studies protocols whose termination time depends on the actual number f of corruptions. Loss and Nielsen [25] give a deterministic, adaptively secure protocol tolerating up to $t = n - 1$ corruptions and terminating in $\mathcal{O}(\min f^2, t + 1)$ rounds. Deligios, Klasovita, and Liu-Zhang [13] subsequently obtain asymptotically optimal early termination for

$t < (1 - \varepsilon)n$. In contrast, our approach enables early termination by strengthening the authentication primitive: unique multisignatures prevent a signer from authenticating two conflicting values within the same protocol epoch.

Randomized Protocols. Randomization provides another approach to improving upon the deterministic $t + 1$ -round bound. Garay, Katz, Koo, and Ostrovsky [20] study authenticated broadcast in the presence of a dishonest majority. Subsequent works obtain improved round complexity in different dishonest-majority regimes [9,41,40]. These results are largely orthogonal to ours: we retain a deterministic protocol and instead use unique multisignatures to prevent equivocation and reduce authentication overhead.

Hint-Free Multisignatures. Most closely related to our cryptographic motivation, Hofheinz, Reichle, and Wagner [21] recently introduced *hint-free multisignatures* for multi-hop aggregation. Their notion removes auxiliary information describing how an aggregate was formed, while verification still depends on the set of participating public keys. Our setting asks whether one can go further and avoid tracking signer identities or multiplicities altogether. Our impossibility result rules out this stronger form of arbitrary idempotent aggregation with constant-size signatures, thereby motivating unique multisignatures that certify only the number of participating signers.

2 Technical Overview

BLS Signatures and Aggregation. A standard BLS signature operates over a bilinear group, where a signer with secret key $\mathbf{sk}$ hashes a message $\mathbf{m}$ and computes the signature as $\sigma := \mathbf{H}(\mathbf{m})^{\mathbf{sk}}$. To verify the signature, the verifier uses a pairing to check whether $e(\sigma, g) = e(\mathbf{H}(\mathbf{m}), \mathbf{pk})$, where $\mathbf{pk} := g^{\mathbf{sk}}$. A powerful feature of BLS is non-interactive multisignature aggregation, which allows multiple signatures $\sigma_1, \dots, \sigma_N$ on the same message $\mathbf{m}$ to be combined simply by multiplying them into $\sigma = \prod_i \sigma_i$. This results in a compact signature regardless of the number of signers. Furthermore, this aggregate signature can be verified under the aggregate public key, computed as a simple product $\mathbf{apk} = \prod_i \mathbf{pk}_i$. Similar to standard BLS, verification of the aggregate signature is performed by checking $e(\sigma, g) = e(\mathbf{H}(\mathbf{m}), \mathbf{apk})$, which holds since

$$e(\sigma, g) = e\left(\prod_i \sigma_i, g\right) = e(\mathbf{H}(\mathbf{m})^{\mathbf{sk}}, g) = e(\mathbf{H}(\mathbf{m}), g^{\mathbf{sk}}) = e(\mathbf{H}(\mathbf{m}), \mathbf{apk}),$$

where $\mathbf{sk} := \sum_i \mathbf{sk}_i$.

Note that we are not operating in the plain public key model: We assume that every signer provides a Proof of Possession (PoP) [6,26,33] of the secret key $\mathbf{sk}_i$ corresponding to its public key $\mathbf{pk}_i$, which is how Ethereum’s Proof of Stake works. Without PoP, special mitigations must be implemented against rogue-key attacks [7].

Multi-Hop Aggregation. In highly distributed systems with many signers, collecting all signature shares to aggregate them at once does not scale due to the communication overhead that floods the network. To reduce this overhead, one can use multi-hop aggregation by dividing the signer set PK into smaller subsets PK_i such that $\text{PK} = \cup_i \text{PK}_i$. After aggregating the shares of each subset PK_i inside smaller subnetworks, only the partial aggregations of PK_i are sent to the global network, which are then aggregated together to generate the final signature. However, this introduces the following issue: Let $\text{PK} := \{\text{pk}_1, \text{pk}_2, \text{pk}_3\}$, $\text{PK}_1 := \{\text{pk}_1, \text{pk}_2\}$, and $\text{PK}_2 := \{\text{pk}_2, \text{pk}_3\}$. Assume all sign the same message, producing $\sigma_i := H(m)^{\text{sk}_i}$, and the signature shares inside the corresponding subnetwork are aggregated as $\Sigma_1 := \sigma_1 \cdot \sigma_2$ and $\Sigma_2 := \sigma_2 \cdot \sigma_3$. Aggregating both partial aggregations yields $\Sigma := \Sigma_1 \cdot \Sigma_2 = \sigma_1 \cdot \sigma_2^2 \cdot \sigma_3$. Consequently, verifying this signature under the aggregate public key $\text{apk} := \text{pk}_1 \cdot \text{pk}_2 \cdot \text{pk}_3$ fails if we use the standard verification equation, since

$$\begin{aligned} e(\Sigma, g) &= e(H(m)^{(\text{sk}_1 + 2 \cdot \text{sk}_2 + \text{sk}_3)}, g) = e(H(m), g^{(\text{sk}_1 + 2 \cdot \text{sk}_2 + \text{sk}_3)}) \\ &\neq e(H(m), g^{(\text{sk}_1 + \text{sk}_2 + \text{sk}_3)}) = e(H(m), \text{apk}). \end{aligned}$$

To verify correctly, the verifier requires knowledge of the multiplicity of each signature share in the final aggregate signature. This issue can be resolved using one of the following approaches:

- **Bitvector-Guided Aggregation:** To avoid the multiplicity issue, a system can use a bitvector to track participants and only aggregate disjoint sets. However, because aggregators often collect overlapping sets of signatures, these overlapping aggregates cannot be merged, as the bitvector cannot represent multiplicities.
- **Encoding the Multiplicities:** If a protocol wants to force the multi-hop aggregation of overlapping sets, one solution is to encode a vector of counters that tells the verifier the exact multiplicity of each public key. However, this means replacing the simple 1-bit-per-signer bitvector with multiple bits per signer, which adds significant data overhead.
- **Tracking the Aggregation Topology:** Another solution is to let the verifier use the exact tree of how the aggregates were merged to compute the verification equation. However, encoding and sending this random topology as a hint creates a large communication overhead.
- **The Normalization Approach:** A recent solution by Hofheinz et al. [21] introduces hint-free multi-signatures using Indistinguishability Obfuscation [3,23]. Their approach uses a public normalization procedure that automatically removes the redundant signature shares during aggregation, thus reducing the multiplicity of every key to exactly 1. While this completely eliminates the need to track multiplicities or aggregation topologies, the verifier still requires the exact set of participating public keys (e.g., an N -bit bitvector) as an explicit input.

Unfortunately, all these approaches incur an overhead that is at least linear in the number of signers.

Impossibility of $(N-1)$ -Bit Aggregate Signatures. We show that multisignature schemes supporting multi-hop aggregation for N signers while preserving a signature size of $N-1$ bits do not exist. We formalize this type of signature as a new primitive called *Idempotent Multisignature* and demonstrate that the existence of such a primitive implies a compression algorithm capable of losslessly compressing random data.

Intuitively, if an idempotent multisignature scheme can encode exactly who signed and who did not within a signature of size $N-1$, it is essentially compressing a linear-length bitvector (where 1 denotes participation and 0 denotes absence). We can frame this signature aggregation as an information-theoretic transmission game. First, for a fixed sequence length N (e.g., $N=4$), we generate N key pairs (pk_i, sk_i) and produce individual signature shares σ_i for a fixed, arbitrary message $m := 0$. This setup acts as a public dictionary that is generated once and is entirely independent of the data to be transmitted.

Now, suppose a sender wishes to transmit an arbitrary N -bit sequence, for example, $B := 1001$. To encode B , the sender aggregates the pre-computed signature shares corresponding to the set bits (positions 1 and 4), producing the aggregate $y := \text{SigAgg}(\{\sigma_1, \sigma_4\})$.

The receiver, given y (and having access to the public dictionary of individual shares σ_i), iteratively checks positions $i = 1, \dots, N$ to deduce whether the i -th signer participated. For the first bit ($i = 1$), the receiver computes a test aggregate $y_1 := \text{SigAgg}(\{\sigma_2, \sigma_3, \sigma_4\})$. They then aggregate y and y_1 to form $Y_1 := \text{SigAgg}(\{y, y_1\})$. Because y contains σ_1 , the combined aggregate Y_1 contains all four shares and will successfully verify against the full aggregate public key. Since the receiver knows σ_1 was not in y_1 , its presence in Y_1 implies it must have come from y , allowing the receiver to deduce $b_1 = 1$. Similarly, for the second bit, the receiver computes $y_2 := \text{SigAgg}(\{\sigma_1, \sigma_3, \sigma_4\})$ and $Y_2 := \text{SigAgg}(\{y, y_2\})$. In this case, Y_2 will fail to verify because neither y nor y_2 contains σ_2 , leading the receiver to correctly deduce $b_2 = 0$. By repeating this process, a sender can transmit any arbitrary bit string of length N compressed within a signature of size $N-1$, which directly violates established information-theoretic compression bounds.

Sequential Aggregation. As an alternative to idempotent multisignatures, we study a variant of sequential aggregate signatures [27,26]. A directed-path-style aggregation eliminates the situation resulting in one of the shares being aggregated multiple times, assuming the signers behave honestly. While falling back to sequential aggregation comes with its efficiency limitations, it reopens the question of whether we can achieve multi-hop aggregation while keeping the signature size constant.

Unique Multisignatures. Unique Multisignatures are a variant of sequential aggregate signatures. In contrast to typical aggregate signatures that allow the signers to sign arbitrary distinct messages, unique multisignatures impose strict pattern on the messages being signed. Concretely, the signers sign messages of the form (m, e, τ) , where m is a message signed by all signers, e is an epoch,

and τ is the current depth of the signature. Crucially, message-signature pairs can only be aggregated for the same $(\mathbf{m}, e)$. In the e -th protocol run, N sign the same message $\mathbf{m}$. The i -th signer in the signing chain for $i \in [N]$ signs $(\mathbf{m}, e, i)$, consequently, the signing sequence is of the form:

$$(\perp, (\mathbf{m}, e, 0)) \rightarrow (\sigma_1, (\mathbf{m}, e, 1)) \rightarrow (\sigma_2, (\mathbf{m}, e, 2)) \rightarrow \dots \rightarrow (\sigma_\tau, (\mathbf{m}, e, N)),$$

where $(\sigma_N, (\mathbf{m}, e, N))$ is the final aggregate signature in epoch e of depth N . Unfortunately, as highlighted earlier, unique multisignatures exist only if the signers behave honestly. For example, an adversarial signer in the sequence can receive a signature of depth j , signs the message $k > 1$ times, and forwards a signature of depth $j + k$ instead of $j + 1$ to the next signer. To encode $B := 1001$, the encoder sequentially signs and aggregates a message $\mathbf{m} := 0$ under the keys $\mathbf{sk}_1$ and $\mathbf{sk}_4$ (the positions corresponding to 1) sequentially in epoch $e := 1$: $\sigma'_1 \leftarrow^{\$} \text{Sign}(\text{PK}, \mathbf{sk}_1, \mathbf{m}, e, 1)$; $\sigma_1 \leftarrow \text{SigAgg}(\text{PK}, \mathbf{m}, e, (\perp, 0), (\sigma'_1, 1))$; $\sigma'_4 \leftarrow \text{Sign}(\text{PK}, \mathbf{sk}_4, \mathbf{m}, e, 2)$; $\sigma_4 \leftarrow \text{SigAgg}(\text{PK}, \mathbf{m}, e, (\sigma_1, 1), (\sigma'_4, 2))$. Given the compression σ_4 , the decoder iterates over all bits b_i for $i := 1, \dots, 4$. For b_1 , it computes $\Sigma'_1 \leftarrow^{\$} \text{Sign}(\text{PK}, \mathbf{sk}_1, \mathbf{m}, e, 3)$ and $\Sigma_1 \leftarrow \text{SigAgg}(\text{PK}, \mathbf{m}, e, (\sigma_4, 2), (\Sigma'_1, 3))$. It then tries to verify $\text{VerifyAgg}(\text{PK}, \mathbf{m}, e, 3, \Sigma_1)$, which should output 0 because the signature is valid for depth 2 since only two unique signers participated in its signing. Consequently, since adding a share corresponding to $\mathbf{sk}_1$ did not increase the depth of the signature, the decoder learns that a share from $\mathbf{sk}_1$ has been aggregated in σ_4 ; therefore, $b_1 = 1$.

For the second bit b_2 , the decoder analogously computes Σ'_2 and Σ_2 . However, since $b_2 = 0$, no share corresponding to $\mathbf{sk}_2$ is aggregated in σ_4 , hence aggregating Σ'_2 increases the depth of σ_4 , hence $\text{VerifyAgg}(\text{PK}, \mathbf{m}, e, 3, \Sigma_2) = 1$, and the decoder infers that no share from $\mathbf{sk}_2$ was included in σ_4 and therefore $b_2 = 0$. The rest of the bits can be decoded analogously.

Restrictive Signing Oracle A Restrictive Signing Oracle (RSO) provides access to the signing functionality of an underlying digital signature scheme DS, while restricting the caller to obtaining at most one signature per epoch. In particular, the caller never obtains the signing key $\mathbf{sk}$. During initialization, upon invocation of **KeyGen**, the oracle generates a key pair $(\mathbf{pk}, \mathbf{sk})$, stores $\mathbf{sk}$ internally, and returns $\mathbf{pk}$. Signatures are requested through the interface **Sign**, which takes as input a message $\mathbf{m}$ and an epoch e . If no signature has previously been issued for epoch e , the oracle computes and returns a valid signature σ on $\mathbf{m}$ and marks e as used. Any subsequent signing request for the same epoch is rejected with $\perp$. There are two natural approaches to realizing an RSO. The first relies on trusted hardware that protects the signing key and maintains state across signing operations. The second relies on one-shot signatures, a cryptographic primitive in which a signing key can be used to produce at most one signature [2,36]. The former can be instantiated using commodity hardware, such as trusted execution environments or hardware authentication tokens, whereas the latter fundamentally exploits quantum computing.

- **Hardware Assumptions.** A generic way to implement an RSO is to use a trusted execution environment (TEE) [10]. A TEE isolates security-sensitive

code and data from the rest of the system, including privileged software such as the operating system or hypervisor. The RSO can therefore generate and store sk within the protected environment and maintain a trusted state recording of which epochs have already been used. Provided that this state is protected against rollback, the TEE can enforce the one-signature-per-epoch restriction even when the surrounding software is malicious. Moreover, remote attestation can bind the public key pk to a particular attested program, allowing a third party to verify that pk was generated by an implementation that enforces the required signing policy. A more lightweight realization may be possible using hardware FIDO authenticators, such as YubiKey, Google Titan, or Nitrokey devices. FIDO2 and WebAuthn implement public-key-based challenge-response authentication in which an authenticator generates a credential key pair and uses the corresponding private key to sign authentication challenges [5,38]. Hardware authenticators can additionally provide attestation evidence about the authenticator and the provenance of the generated credential.

Of particular interest for our setting is the optional signature-counter mechanism supported by WebAuthn authenticators. When implemented, the authenticator maintains a counter that increases after successful authentication operations, and the resulting counter value is included in the signed authenticator data. Consequently, for an authenticator that provides a monotonically increasing, rollback-resistant counter, the counter value can be interpreted as a device-generated epoch identifier. Under this interpretation, each counter value is associated with at most one signature, giving an RSO-like interface for sequential epochs.³

- **One-Shot Signatures.** One-shot signatures (OSS) provide a particularly natural cryptographic realization of the restriction underlying an RSO. Informally, an OSS signing key permits the generation of a signature on a single message, after which it cannot be used to produce a valid signature on a second message. The notion was introduced by Amos, Georgiou, Kiayias, and Zhandry [2]. More recently, Shmueli and Zhandry gave a standard-model construction under cryptographic assumptions [36]. Such signatures fundamentally rely on quantum signing keys and exploit the inability to copy quantum information freely.

This property is stronger than what double-authentication-preventing signatures (DAPS) provide. In a DAPS scheme, multiple signatures can, in principle, be generated, but signing two conflicting messages associated with the same address reveals the signer’s secret key [4]. Thus, DAPS discourages equivocation by making it punishable, whereas OSS prevents the production of two signatures in the first place.

A single OSS key pair naturally realizes only a single epoch. To obtain an RSO supporting multiple epochs, one can assume an upper bound E on the

³ This requires an authenticator that implements a suitable signature counter and protects it against rollback. Moreover, if epochs are chosen independently by the application rather than identified with the authenticator counter, additional trusted state is required to enforce the one-signature-per-epoch restriction.

number of epochs and generate an independent OSS key pair (pk_e, sk_e) for every $e \in [E]$. The RSO uses sk_e to answer the signing request associated with epoch e . To avoid a public key whose size grows linearly with E , the public keys $\{pk_e\}_{e \in [E]}$ can be committed to using a Merkle tree. The RSO public key then consists only of the Merkle root, while a signature for epoch e is accompanied by pk_e and the corresponding Merkle authentication path.

We provide a construction of a unique multisignature with constant signature size using Incrementally Verifiable Computations. This approach can be combined with a hardware signing oracle or one-shot signatures.

Byzantine Broadcast with Early Termination. Early termination is generally unsafe in the standard Dolev–Strong protocol. Suppose an honest party P_i terminates in round t . A malicious sender P_1 colluding with $t - 1$ corrupted parties can violate the Agreement as follows. During the first $t - 1$ rounds, the corrupted parties exchange two conflicting values, $v := 1$ and $v' := 0$, only among themselves, thereby constructing valid signature chains for both values. In round t , they send the chain for v to an honest party P_i and the chain for v' to another honest party P_j . If P_i terminates immediately and outputs v , then P_j may output v' , since it has not yet learned about the conflicting value v . Agreement is therefore violated. Waiting one additional round prevents this attack. In round $t + 1$, the honest parties forward the values they received, causing P_i and P_j to learn both v and v' . They can then detect the equivocation and output the prescribed default value, thereby preserving Agreement. This illustrates why the standard Dolev–Strong protocol requires parties to execute all $t + 1$ rounds: an honest party cannot safely terminate upon receiving its first valid signature chain, because a conflicting chain may still exist elsewhere in the network. The attack fundamentally relies on the sender’s ability to authenticate two conflicting values within the same protocol execution. If each participant is restricted to signing at most one value per execution, such equivocation becomes impossible. Our notion of unique multisignatures provides precisely this guarantee by allowing each signer to contribute at most one signature per epoch. This makes unique multisignatures a natural replacement for the signature chains used in standard implementations of Dolev–Strong Byzantine Broadcast.

With this restriction, once an honest party receives a valid value carrying the sender’s signature, it can safely sign the value, broadcast the resulting aggregate signature, and terminate. No conflicting value can later be accepted, because producing such a value would require a second valid signature from the sender in the same epoch, which is prohibited by the unique-signing property. Thus, sender equivocation is ruled out cryptographically rather than detected through an additional round of communication.

Moreover, unique multisignatures keep the authentication information constant size. Instead of carrying a signature chain whose length grows with the number of rounds, a message contains only a constant-size aggregate authentication object. Consequently, replacing Dolev–Strong signature chains with unique multisignatures reduces the protocol’s overall communication complexity from

$\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ while simultaneously enabling safe early termination for honest parties.

3 Preliminaries

Let $\lambda \in \mathbb{N}$ denote the security parameter. Unless stated otherwise, all algorithms and adversaries are probabilistic polynomial-time in λ . For $n \in \mathbb{N}$, we write $[n] := \{1, \dots, n\}$. For a binary string $x \in \{0, 1\}^*$, $|x|$ denotes its bit length; for a finite set S , $|S|$ denotes its cardinality. We use $x \leftarrow y$ for deterministic assignment and $x \leftarrow^{\$} \mathcal{D}$ to denote sampling x according to a distribution or randomized experiment $\mathcal{D}$; if $\mathcal{D}$ is a finite set, the sampling is uniform. Unless explicitly indicated, probabilities are taken over all random coins of the setup, the algorithms, and the adversary.

3.1 Digital Signatures

We recall the definition of standard digital signatures.

Definition 1 (Digital Signatures). A digital signature scheme $\text{DS} := (\text{KeyGen}, \text{Sign}, \text{Verify})$ is defined as follows:

- $\text{KeyGen}(1^\lambda)$ The randomized key generation algorithm takes as input a security parameter 1^λ , and it outputs a key pair (pk, sk) .
- $\text{Sign}(\text{sk}, \text{m})$ The randomized signing algorithm takes as input a secret key sk and a message m , and it outputs a signature σ .
- $\text{Verify}(\text{pk}, \text{m}, \sigma)$ The deterministic verification algorithm takes as input a public key pk , a message m , and a signature σ . It outputs a bit $b \in \{0, 1\}$.

A digital signature scheme DS must satisfy correctness and existential unforgeability. For completeness, we provide the full definitions in the supplementary material.

3.2 Broadcast

Network and Parties. We consider a set of parties $\mathcal{P} := \{P_1, \dots, P_n\}$, where P_1 is the designated sender and holds an input $v \in \{0, 1\}$. Before the protocol begins, a trusted registration functionality binds each party P_i to a public key pk_i and gives all parties authenticated, read-only access to $\text{PK} := \text{pk}_{i \in [n]}$. The adversary cannot modify these bindings. The parties communicate over authenticated point-to-point channels in a synchronous network. An execution proceeds in numbered, lock-step rounds. Every message sent by an honest party to an honest party in a given round is delivered by the end of that round.

Adaptive Corruptions. The adversary $\mathcal{A}$ may adaptively corrupt up to $t < n$ parties during the execution, based on its current view. Corruptions are permanent. Upon corrupting a party, $\mathcal{A}$ obtains its local state and controls all of its subsequent actions and outgoing messages. Note that corrupting an RSO-equipped party does not reveal the RSO's, as they are inaccessible even to the party itself.

Definition 2 (Byzantine Broadcast). Let Π be a protocol that is executed by N parties $P_1, \dots, P_N$, where P_1 , the designated sender, holds an input v and all parties terminate upon output. We say that Π is adaptively secure if, for every adaptive adversary corrupting at most t parties, the following properties hold with overwhelming probability:

- **Termination:** All honest parties eventually terminate and decide on some value v' .
- **Agreement:** All honest parties decide on the same value v' .
- **Validity:** If the sender P_1 is honest, then every honest party decides on v .

3.3 Compression

Definition 3 (Compression in the Random Oracle Model). For a security parameter λ , a hash function H treated as a random oracle, we define compression schemes $\text{CMP}^H := (\text{Setup}^H, \text{Encode}^H, \text{Decode}^H)$ as follows:

- $\text{Setup}^H(1^\lambda, N)$ The setup algorithm takes as input a security parameter 1^λ and a maximum bit length N , and outputs a public parameter pp . We assume pp specifies the input domains $\mathcal{X}_{\text{pp}} := \{0, 1\}^N$ and $\mathcal{Y}_{\text{pp}}$ of Encode^H and Decode^H , respectively.
- $\text{Encode}^H(\text{pp}, x)$ The compression algorithm takes as input a string $x \in \mathcal{X}_{\text{pp}}$ and outputs a compressed string $y \in \mathcal{Y}_{\text{pp}}$.
- $\text{Decode}^H(\text{pp}, y)$ The decompression algorithm takes as input a compressed string $y \in \mathcal{Y}_{\text{pp}}$, and it outputs a decompressed string $x \in \mathcal{X}_{\text{pp}}$.

Theorem 1 (Compression Limit in the Random Oracle Model). Let $\text{CMP}^H := (\text{Setup}^H, \text{Encode}^H, \text{Decode}^H)$ be a δ -lossless compression scheme in the random oracle model for a constant $\delta \in (0, 1)$. That is, for all security parameters λ , all maximum bit lengths N , all random oracles H_{pp} , all public parameters $\text{pp} \leftarrow^{\$} \text{Setup}^H(1^\lambda, N)$, and all $x \in \mathcal{X}_{\text{pp}}$, it holds

$$\Pr_H \left[\text{Decode}^H(\text{pp}, \text{Encode}^H(\text{pp}, x)) = x \mid H \text{ extends } H_{\text{pp}} \right] \geq \delta.$$

Then it holds $|\mathcal{Y}_{\text{pp}}| \geq \delta \cdot |\mathcal{X}_{\text{pp}}|$.

Proof. The proof relies on a simple averaging argument similar to [12, Fact 8.1.]. As $\Pr[\text{Decode}^H(\text{pp}, \text{Encode}^H(\text{pp}, x)) = x] \geq \delta$, there exists a realization of a random oracle $H \in \mathcal{H}$ extending H_{pp} such that there is a fraction of $\mathcal{X}_{\text{pp}}$, say $\mathcal{X}'_{\text{pp}} \subseteq \mathcal{X}_{\text{pp}}$, with $|\mathcal{X}'_{\text{pp}}| \geq \delta \cdot |\mathcal{X}_{\text{pp}}|$, and it holds for all $x' \in \mathcal{X}'_{\text{pp}}$ that $\text{Decode}^H(\text{pp}, \text{Encode}^H(\text{pp}, x')) = x'$. Thus, the cardinality of the image of Encode^H is at least of size $|\mathcal{X}'_{\text{pp}}|$, i.e. $|\mathcal{Y}_{\text{pp}}| \geq |\mathcal{X}'_{\text{pp}}| \geq \delta \cdot |\mathcal{X}_{\text{pp}}|$. $\square$

3.4 Incrementally Verifiable Computation (IVC)

Incrementally Verifiable Computation (IVC) [39] allows a prover to prove that an initial state x_0 reaches another state x_j by repeatedly applying a transition function $\mathcal{F}$ for j times, i.e.,

$$x_j := \underbrace{\mathcal{F}(\mathcal{F}(\dots \mathcal{F}(x_0)))}_{j \text{ times}}.$$

Crucially, the proof size and the verifier's time are sublinear in the depth j . In this work, we follow the definitions of [24], which assume a constant-size proof.

Definition 4 (Incrementally Verifiable Computation). *An Incrementally Verifiable Computation (IVC) scheme is a tuple of algorithms $\text{IVC} = (\text{Setup}, \text{KeyGen}, \text{Prove}, \text{Verify})$ where:*

- $\text{Setup}(1^\lambda)$: The randomized setup algorithm takes as input a security parameter 1^λ and outputs a common reference string crs .
- $\text{KeyGen}(\text{crs}, \mathcal{F})$: The deterministic key generation algorithm takes as input a common reference string crs and a transition function $\mathcal{F}$. It outputs a prover key prk and a verifier key vrk .
- $\text{Prove}(\text{prk}, x_0, x_{j-1}, x_j, j, \pi_{j-1}, \sigma_{j-1})$: The randomized prover algorithm takes as input a prover key prk , an initial state x_0 , two states x_{j-1} and x_j , a step index j , a proof π_{j-1} , and a witness σ_{j-1} . It outputs an updated proof π_j .
- $\text{Verify}(\text{vrk}, x_0, x_j, j, \pi_j)$: The deterministic verifier algorithm takes as input a verifier key vrk , an initial state x_0 , a state x_j , a proof depth j , and a proof π_j . It outputs a bit $b \in \{0, 1\}$.

An IVC scheme $\text{IVC} = (\text{Setup}, \text{KeyGen}, \text{Prove}, \text{Verify})$ must satisfy the following properties:

Definition 5 (Perfect Completeness). *An IVC scheme $\text{IVC} := (\text{Setup}, \text{KeyGen}, \text{Prove}, \text{Verify})$ satisfies perfect completeness if for any adversary $\mathcal{A}$:*

$$\Pr \left[\begin{array}{l} \text{Verify}(\text{vrk}, x_0, x_0, 0, \perp) = 1 \wedge \\ \text{Verify}(\text{vrk}, x_0, x_j, j, \pi_j) = 1 \end{array} \middle| \begin{array}{l} \text{crs} \leftarrow^{\$} \text{Setup}(1^\lambda), \\ (\mathcal{F}, j, x_0, x_{j-1}, \sigma_{j-1}, \pi_{j-1}) \leftarrow^{\$} \mathcal{A}(\text{crs}), \\ (\text{prk}, \text{vrk}) \leftarrow \text{KeyGen}(\text{crs}, \mathcal{F}), \\ x_j \leftarrow \mathcal{F}(x_{j-1}, \sigma_{j-1}), \\ \text{Verify}(\text{vrk}, x_0, x_{j-1}, j-1, \pi_{j-1}) = 1, \\ \pi_j \leftarrow^{\$} \text{Prove}(\text{prk}, x_0, x_{j-1}, x_j, j, \pi_{j-1}, \sigma_{j-1}) \end{array} \right] = 1.$$

Definition 6 ($(t_{ks}, t'_{ks}, \varepsilon_{ks})$ -Knowledge Soundness). *An IVC scheme $\text{IVC} := (\text{Setup}, \text{KeyGen}, \text{Prove}, \text{Verify})$ satisfies $(t_{ks}, t'_{ks}, \varepsilon_{ks})$ -knowledge soundness if for all $j \in \mathbb{N}$, and for any adversary $\mathcal{A}$ running in time at most t_{ks} , there exists an extractor $\mathcal{E}$ running in time at most t'_{ks} such that:*

$$\left| \begin{array}{l} \Pr_{r \leftarrow^{\$} \{0,1\}^\lambda} \left[\begin{array}{l} \text{Verify}(\text{vrk}, x_0, x_j, j, \pi_j) = 1 \\ x_j = x_j^* \text{ where} \\ x_{i+1} \leftarrow \mathcal{F}(x_i, w_i) \\ \forall i \in \{0, \dots, j-1\} \end{array} \middle| \begin{array}{l} \text{crs} \leftarrow^{\$} \text{Setup}(1^\lambda), \\ (\mathcal{F}, x_0, x_j, \pi_j) \leftarrow \mathcal{A}(\text{crs}, r), \\ (\text{prk}, \text{vrk}) \leftarrow \text{KeyGen}(\text{crs}, \mathcal{F}) \end{array} \right] \\ - \Pr_{r \leftarrow^{\$} \{0,1\}^\lambda} \left[\begin{array}{l} \text{crs} \leftarrow^{\$} \text{Setup}(1^\lambda), \\ (\mathcal{F}, x_0, x_j^*, \pi_j) \leftarrow \mathcal{A}(\text{crs}, r), \\ (w_0, \dots, w_{j-1}) \leftarrow \mathcal{E}(\text{crs}, r) \end{array} \right] \end{array} \right| \leq \varepsilon_{ks}.$$

Definition 7 (Succinctness). *There exists a polynomial p such that for all security parameters $\lambda \in \mathbb{N}$, all step indices $j \in \mathbb{N}$, and all transition functions $\mathcal{F}$, the proof size and the running time of `Verify` are strictly bounded by $p(\lambda, |\text{vrk}|)$.*

4 Idempotent Multisignatures

In this section, we introduce the notion of non-interactive idempotent multisignatures, a variant of standard multisignatures [22,29,7] allowing for repeated aggregation of signature shares while preserving a fixed total aggregate-signature size at most $N - 1$ bits, where N is the number of signers. The key requirement for such a primitive is a property we call *idempotent correctness*. This property ensures that the multiplicity of a signature share in an aggregate signature does not affect the aggregate key required to verify the signature correctly.

To formalize this, we distinguish between a multiset (which allows repeated elements) and its *underlying set* (which contains only the unique elements, ignoring repetitions). Consider a set of three signers $\{S_1, S_2, S_3\}$, each using their secret key sk_i to generate a signature share σ_i on the same message m . Let $\mathcal{S} := \{\sigma_1, \sigma_2, \sigma_3\}$ be the standard set of these shares. The standard aggregate signature $\sigma \leftarrow \text{SigAgg}(\mathcal{S})$ is valid under the aggregate key $\text{apk} \leftarrow \text{KeyAgg}(\{\text{pk}_1, \text{pk}_2, \text{pk}_3\})$. The idempotence property ensures that for any finite multiset Σ whose underlying set is exactly $\mathcal{S}$, the multi-aggregated signature $\sigma' \leftarrow \text{SigAgg}(\Sigma)$ remains valid under the exact same aggregate key apk for the message m .

Definition 8 (Non-interactive Idempotent Multisignatures). *A non-interactive idempotent multisignature scheme $\text{IMS} := (\text{Setup}, \text{KeyGen}, \text{KeyAgg}, \text{Sign}, \text{SigAgg}, \text{Verify})$ is defined as follows:*

- $\text{Setup}(1^\lambda)$: *The randomized setup algorithm takes as input a security parameter 1^λ and outputs a public parameter pp . We assume that pp implicitly encodes λ .*
- $\text{KeyGen}(\text{pp}; \text{rnd})$: *The deterministic⁴ key generation algorithm takes as input a public parameter pp , and randomness rnd , and it outputs a pair (sk, pk) .*
- $\text{KeyAgg}(\text{PK})$: *The deterministic key aggregation algorithm takes as input a set of public keys $\text{PK} := \{\text{pk}_1, \dots, \text{pk}_t\}$ for any integer $t \geq 1$, and it outputs an aggregate key apk .*
- $\text{Sign}(\text{sk}, \text{m})$: *The randomized signing algorithm takes as input a secret key sk and a message m and outputs a signature share $\bar{\sigma}$.*
- $\text{SigAgg}(\Sigma)$: *The deterministic signature aggregation algorithm takes as input a finite set of signatures Σ (which can be basic signature shares or partial aggregations). It outputs an aggregate signature σ .*
- $\text{Verify}(\text{apk}, \text{m}, \sigma)$: *The deterministic verification algorithm takes an aggregate key apk , a message m , and a signature σ as input. It outputs a bit $b \in \{0, 1\}$.*

⁴ The random coin rnd is given explicitly as input.

Note that we define **KeyGen** this way to enable ad-hoc key generation by instantiating **KeyGen** as a pseudorandom function PRF with **rnd** being the PRF key.

An IMS must satisfy idempotent correctness (definition 9), existential unforgeability (definition 10), and efficiency (definition 11).

Definition 9 (Idempotent Correctness). We say that an Idempotent Multisignatures scheme $\text{IMS} := (\text{Setup}, \text{KeyGen}, \text{KeyAgg}, \text{Sign}, \text{SigAgg}, \text{Verify})$ satisfies idempotent correctness if for any security parameter $\lambda \in \mathbb{N}$, any integer $N \geq 1$, and any message $m \in \{0, 1\}^*$, the following holds:

- Generate public parameters $\text{pp} \leftarrow^{\$} \text{Setup}(1^\lambda)$.
- Sample uniformly random coins $\text{rnd}_i \leftarrow^{\$} \{0, 1\}^\lambda$ and compute key pairs $(\text{sk}_i, \text{pk}_i) \leftarrow \text{KeyGen}(\text{pp}; \text{rnd}_i)$ for all $i \in [N]$.
- Compute the aggregate public key $\text{apk} \leftarrow \text{KeyAgg}(\{\text{pk}_1, \dots, \text{pk}_N\})$.
- Let $\sigma_i \leftarrow^{\$} \text{Sign}(\text{sk}_i, m)$ for $i \in [N]$. Define the complete set of basic signature shares as $\Sigma := \{\sigma_1, \dots, \sigma_N\}$.
- For any integer $k \geq 1$, let $V_1, \dots, V_k$ be a collection of arbitrary, **not necessarily disjoint**, non-empty subsets of Σ such that $\bigcup_{j=1}^k V_j = \Sigma$.
- For all $j \in [k]$, let $\sigma'_j \leftarrow \text{SigAgg}(V_j)$ be the partial aggregation of subset V_j .
- Let $\sigma \leftarrow \text{SigAgg}(\{\sigma'_1, \dots, \sigma'_k\})$ be the final aggregate signature.

It holds that $\Pr[\text{Verify}(\text{apk}, m, \sigma) = 1] = 1$.

Definition 10 (Existential Unforgeability (EUF-CMA)). For an idempotent multisignature scheme $\text{IMS} := (\text{Setup}, \text{KeyGen}, \text{KeyAgg}, \text{Sign}, \text{SigAgg}, \text{Verify})$, an adversary $\mathcal{A}$, and a security parameter $\lambda \in \mathbb{N}$, we define the unforgeability game $\text{EUF-CMA}_{\text{IMS}}^{\mathcal{A}}$ as follows:

- **Setup:** Sample random coins $\text{rnd} \leftarrow^{\$} \{0, 1\}^\lambda$. Run $\text{pp} \leftarrow^{\$} \text{Setup}(1^\lambda)$ and $(\text{sk}^*, \text{pk}^*) \leftarrow \text{KeyGen}(\text{pp}; \text{rnd})$. Provide pp and the target public key pk^* to the adversary $\mathcal{A}$.
- **Online Phase:** $\mathcal{A}$ may adaptively request signatures on arbitrarily chosen messages m . Upon receiving a query for m , return $\bar{\sigma} \leftarrow^{\$} \text{Sign}(\text{sk}^*, m)$. Let $\mathcal{Q}$ be the set of all messages queried by $\mathcal{A}$ to this oracle.
- **Output Determination:** When $\mathcal{A}$ terminates, it outputs a tuple $(\text{PK}, m^*, \sigma^*)$. Compute $\text{apk} \leftarrow \text{KeyAgg}(\text{PK})$. Output 1 if all of the following conditions hold: 1) $\text{Verify}(\text{apk}, m^*, \sigma^*) = 1$, 2) $\text{pk}^* \in \text{PK}$, 3) $m^* \notin \mathcal{Q}$. Otherwise, output 0.

The advantage of $\mathcal{A}$ is defined as:

$$\text{Adv}_{\text{IMS}}^{\text{EUF-CMA}}(\mathcal{A}) := \Pr \left[\text{EUF-CMA}_{\text{IMS}}^{\mathcal{A}} = 1 \right].$$

We say that IMS is $(t_{\text{euf}}, \varepsilon_{\text{euf}})$ -EUF-CMA-secure if for all adversaries $\mathcal{A}$ running in time at most t_{euf} , it holds that

$$\text{Adv}_{\text{IMS}}^{\text{EUF-CMA}}(\mathcal{A}) \leq \varepsilon_{\text{euf}}.$$

Definition 11 (Efficiency). We say that an Idempotent Multisignatures scheme $\text{IMS} := (\text{Setup}, \text{KeyGen}, \text{KeyAgg}, \text{Sign}, \text{SigAgg}, \text{Verify})$ is efficient if for any security parameter $\lambda \in \mathbb{N}$ and any number of signers $N \geq 1$, the following holds: The size of any basic signature share output by **Sign** is bounded by $\text{poly}(\lambda)$, and the size of any valid aggregate signature σ output by **SigAgg** is at most $N - 1$ bits.

Theorem 2. In the random oracle model, if there exists an efficient idempotent multisignature scheme satisfying idempotent correctness and (t, ε) -EUF-CMA security for some $N \cdot \varepsilon < 1/2$, then there exists a δ' -lossless compression scheme $\text{CMP}^H := (\text{Setup}^H, \text{Encode}^H : \mathcal{X} \rightarrow \{0, 1\}^M, \text{Decode}^H : \{0, 1\}^M \rightarrow \mathcal{X})$ for $M \leq N - 1$, $\delta' := 1 - N \cdot \varepsilon$, a realization of the random oracle H , and an input domain $\mathcal{X}$, such that $M < \log(|\mathcal{X}| \cdot \delta')$, which violates the bound in **theorem 1**.

We provide the compression algorithm in figure 1 and defer the full proof with the formal losslessness analysis to section C

5 Unique Multisignatures

Non-interactive unique multisignatures are a variation of sequential aggregate signatures [27,26] with an aggregate signature size of $\text{poly}(\lambda) + \log(N)$, where N is the number of signers. One of the key features of unique multisignatures is that they restrict the signing and aggregation functionalities. In unique multisignatures, signing is divided into epochs e such that every signer signs a single message m per epoch. An epoch can be seen as an identifier of a protocol execution; hence, each signer is allowed to issue a single signature per protocol execution. It must be infeasible for any signer to sign two messages m and m' with $m \neq m'$ in the same epoch e . We call this property *Non-Equivocation*.

Additionally, no coalition of N dishonest signers can generate a final aggregate signature attesting that $N' > N$ signers have signed a message m in epoch e . Intuitively, this property ensures that each signer can sign a message m in some epoch e at most once. We call this property *Uniqueness*. We provide formal definitions in the following.

Definition 12 (Unique Multisignatures). A Unique Multisignature scheme $\text{UMS} := (\text{Setup}, \text{KeyGen}, \text{Sign}, \text{Verify}, \text{SigAgg}, \text{VerifyAgg})$ is defined by the following algorithms:

- **Setup**($1^\lambda, N$): The randomized setup algorithm takes as input a security parameter 1^λ and a number of signers N , and outputs public parameters pp .
- **KeyGen**(pp): The randomized key generation algorithm takes as input public parameters pp and outputs a secret key sk and a public key pk .
- **Sign**($\text{PK}, \text{sk}, m, e, \tau$): The randomized signing algorithm takes as input a public key set PK , a secret key sk , a message m , an epoch counter e , and a depth parameter τ , and outputs a signature share σ .
- **Verify**($\text{PK}, \text{pk}, m, e, \tau, \sigma$): The deterministic verification algorithm takes as input a public key set PK , a public key pk , a message m , an epoch counter e , a depth parameter τ , and a signature σ , and outputs a bit $b \in \{0, 1\}$.

$\text{Setup}^H(1^\lambda, N)$	$\text{Decode}^H(\text{pp}, y)$
<pre> 1: $S := []$ 2: $\text{pp}_{ims} \leftarrow^{\\$} \text{IMS.Setup}(1^\lambda)$ 3: for $i = 1 \dots N$ do 4: $k_i \leftarrow^{\\$} H(i)$ 5: $(\text{pk}_i, \text{sk}_i) \leftarrow \text{IMS.KeyGen}(\text{pp}_{ims}; k_i)$ 6: $S[i] \leftarrow^{\\$} \text{IMS.Sign}(\text{sk}_i, 0)$ 7: done 8: $\text{pp} := (N, \text{pp}_{ims}, \mathcal{X} := \{0, 1\}^N, S)$ 9: return pp </pre>	<pre> 1: $(N, \text{pp}_{ims}, \mathcal{X}, S) \leftarrow \text{pp}$ 2: for $i = 1 \dots N$ do 3: $k_i \leftarrow^{\\$} H(i)$ 4: $(\text{pk}_i, \text{sk}_i) \leftarrow \text{IMS.KeyGen}(\text{pp}_{ims}; k_i)$ 5: done 6: $\text{apk} \leftarrow \text{IMS.KeyAgg}(\{\text{pk}_1, \dots, \text{pk}_N\})$ 7: for $i = 1 \dots N$ do 8: $S_i := \emptyset$ 9: for $j \in [N] \setminus \{i\}$ do 10: $S_i := S_i \cup \{S[j]\}$ 11: done 12: $\sigma_{S_i} := \text{IMS.SigAgg}(S_i)$ 13: $\sigma_i := \text{IMS.SigAgg}(\{\sigma_{S_i}, y\})$ 14: $x_i := \text{IMS.Verify}(\text{apk}, 0, \sigma_i)$ 15: done 16: $x \leftarrow (x_1, \dots, x_N)$ 17: return x </pre>
$\text{Encode}^H(\text{pp}, x)$	
<pre> 1: $(N, \text{pp}_{ims}, \mathcal{X}, S) \leftarrow \text{pp}$ 2: $(x_1, \dots, x_N) \leftarrow x$ 3: $S_x := \emptyset$ 4: for $i = 1 \dots N$ do 5: if $x_i = 1$ then 6: $S_x := S_x \cup \{S[i]\}$ 7: fi 8: done 9: $y := \text{IMS.SigAgg}(S_x)$ 10: return y </pre>	

Fig. 1. A construction of δ' -lossless compression scheme $\text{CMP}^H := (\text{Setup}^H, \text{Encode}^H, \text{Decode}^H)$ using an idempotent multisignature scheme IMS, where H is a hash function treated as a random oracle.

- $\text{SigAgg}(\text{PK}, \mathbf{m}, e, (\sigma_1, \tau_1), (\sigma_2, \tau_2))$: The signature aggregation algorithm takes as input a public key set PK , a message $\mathbf{m}$, an epoch e , two signatures σ_1 and σ_2 of depths τ_1 and τ_2 , respectively. It outputs an aggregate signature σ and a number of shares τ .
- $\text{VerifyAgg}(\text{PK}, \mathbf{m}, e, \tau, \sigma)$: The deterministic aggregate verification algorithm takes as input a public key set PK , a message $\mathbf{m}$, an epoch e , a number of shares τ , and a signature σ . It outputs a bit $b \in \{0, 1\}$.

Definition 13 (Correctness). Let $\text{UMS} := (\text{Setup}, \text{KeyGen}, \text{Sign}, \text{Verify}, \text{SigAgg}, \text{VerifyAgg})$ be a unique multisignature scheme. We say that UMS satisfies correctness if for any security parameter 1^λ , any number of signers N , any message $\mathbf{m}$, any epoch $e \in \mathbb{N}$, any depth $\tau \in [N]$, and any signer $i \in [N]$, the following holds:

- Run $\text{pp} \leftarrow^{\$} \text{Setup}(1^\lambda, N)$.

- Generate N key pairs $(pk_j, sk_j) \leftarrow^{\$} \text{KeyGen}(\text{pp})$ for $j \in [N]$ and set $\text{PK} := \{pk_j\}_{j \in [N]}$.
- Generate $\sigma \leftarrow^{\$} \text{Sign}(\text{PK}, sk_i, m, e, \tau)$. Then,

$$\Pr[\text{Verify}(\text{PK}, pk_i, m, e, \tau, \sigma) = 1] = 1.$$

Definition 14 (Sequential Correctness). Let $\text{UMS} := (\text{Setup}, \text{KeyGen}, \text{Sign}, \text{Verify}, \text{SigAgg}, \text{VerifyAgg})$ be a unique multisignature scheme. We say that UMS is sequentially correct if for any security parameter 1^λ , any number of signers N , any message m , any epoch $e \in \mathbb{N}$, and any ordered sequence of $k \in [N]$ distinct signers $(i_1, \dots, i_k)$ where $i_j \in [N]$, the following holds:

- Run $\text{pp} \leftarrow^{\$} \text{Setup}(1^\lambda, N)$ and initialize an empty public key set $\text{PK} \leftarrow \emptyset$.
- Generate N key pairs $(pk_i, sk_i) \leftarrow^{\$} \text{KeyGen}(\text{pp})$ for $i \in [N]$ and set $\text{PK} \leftarrow \text{PK} \cup \{pk_i\}_{i \in [N]}$.
- Set the initial aggregation state to empty: $\sigma'_0 \leftarrow \perp$, $\tau'_0 \leftarrow 0$.
- For $j = 1, \dots, k$, the signers iteratively sign and aggregate their shares with the previous partial aggregation:

$$\begin{aligned} \sigma_j &\leftarrow^{\$} \text{Sign}(\text{PK}, sk_{i_j}, m, e, j) \\ (\sigma'_j, j) &\leftarrow^{\$} \text{SigAgg}(\text{PK}, m, e, (\sigma'_{j-1}, j-1), (\sigma_j, j)) \end{aligned}$$

- Then, for the final aggregate signature σ'_k and the final number of shares k , it holds that:

$$\Pr[\text{VerifyAgg}(\text{PK}, m, e, k, \sigma'_k) = 1] = 1.$$

Definition 15 (Uniqueness). For a unique multisignature scheme $\text{UMS} := (\text{Setup}, \text{KeyGen}, \text{Sign}, \text{Verify}, \text{SigAgg}, \text{VerifyAgg})$, a security parameter λ , a total number of signers $N \in \mathbb{N}$, a corruption bound $t \in [N]$, and an adversary $\mathcal{A}$, we define the game $\text{UNQ}_{\text{UMS}}^{\mathcal{A}}(t, N)$ as follows:

- **Setup Phase.** Generate $\text{pp} \leftarrow^{\$} \text{Setup}(1^\lambda, N)$, initialize an empty public key set $\text{PK} \leftarrow \emptyset$, an empty set of corrupted parties $\mathcal{C} \leftarrow \emptyset$, a table $\mathcal{K}[\cdot] \leftarrow \emptyset$, and a table $\mathcal{H}[\cdot] \leftarrow \emptyset$. For every $i \in [N]$, generate $(pk_i, sk_i) \leftarrow^{\$} \text{KeyGen}(\text{pp})$, set $\mathcal{K}[i] := (pk_i, sk_i)$, and update $\text{PK} \leftarrow \text{PK} \cup \{pk_i\}$.
- **Signing Phase:** Invoke $\mathcal{A}(\text{pp}, \text{PK})$. $\mathcal{A}$ gets access to the following two oracles, which it may query adaptively and in an interleaved fashion.
 - **Signing Oracle.** On input an index i , a message m , an epoch counter e , and a depth parameter τ , return $\perp$ if $i \notin [N]$ or $i \in \mathcal{C}$. Otherwise, parse $(pk_i, sk_i) \leftarrow \mathcal{K}[i]$, generate $\sigma \leftarrow^{\$} \text{Sign}(\text{PK}, sk_i, m, e, \tau)$. If $\sigma \neq \perp$, update $\mathcal{H}[(m, e)] \leftarrow \mathcal{H}[(m, e)] \cup \{i\}$. Return σ to $\mathcal{A}$.
 - **Corruption Oracle.** On input an index i , return $\perp$ if $i \notin [N]$, $i \in \mathcal{C}$, or $|\mathcal{C}| \geq t$. Otherwise, update $\mathcal{C} \leftarrow \mathcal{C} \cup \{i\}$, parse $(pk_i, sk_i) \leftarrow \mathcal{K}[i]$, and return sk_i to $\mathcal{A}$.
- **Output Determination:** $\mathcal{A}$ outputs a tuple $(m^*, e^*, \tau^*, \sigma^*)$. Output 1 iff:

$$\text{VerifyAgg}(\text{PK}, m^*, e^*, \tau^*, \sigma^*) = 1 \quad \text{and} \quad \tau^* > |\mathcal{H}[(m^*, e^*)] \cup \mathcal{C}|.$$

Otherwise, output 0.

We say UMS is $(t_{\text{unq}}, \varepsilon_{\text{unq}})$ -Unique if for all adversaries $\mathcal{A}$ running in time t_{unq} , it holds that

$$\Pr[\text{UNQ}_{\text{UMS}}^{\mathcal{A}}(t, N) = 1] \leq \varepsilon_{\text{unq}}.$$

Intuitively, the adversary wins if it can generate a valid aggregate signature with depth τ^* larger than the number of parties that either signed (m^*, e^*) while honest or were corrupted. This can happen by (i) repeated aggregation of one of the signature shares, (ii) forging a signature share corresponding to an honest party that neither signed (m^*, e^*) nor was corrupted, or (iii) generating more than one signature share for one of the corrupted parties. Therefore, a scheme allowing a signer to generate an arbitrary number of signatures per epoch cannot satisfy this notion.

Definition 16 (Non-Equivocation). For a unique multisignature scheme $\text{UMS} := (\text{Setup}, \text{KeyGen}, \text{Sign}, \text{Verify}, \text{SigAgg}, \text{VerifyAgg})$, an adversary $\mathcal{A}$, a security parameter λ , and a number of signers N , we define the game $\text{Eq}_{\text{UMS}}^{\mathcal{A}}$ as follows:

- **Setup:** Generate public parameters $\text{pp} \leftarrow^{\$} \text{Setup}(1^\lambda, N)$, initialize an empty public key set $\text{PK} \leftarrow \emptyset$ and a secret key set $\text{SK} \leftarrow \emptyset$, and invoke $\mathcal{A}(\text{pp})$.
- **Online Phase:** $\mathcal{A}$ may adaptively send up to N distinct KeyGen requests. For each request, generate $(\text{pk}, \text{sk}) \leftarrow^{\$} \text{KeyGen}(\text{pp})$, update $\text{PK} \leftarrow \text{PK} \cup \{\text{pk}\}$ and $\text{SK} \leftarrow \text{SK} \cup \{\text{sk}\}$, and return (pk, sk) to $\mathcal{A}$. Moreover, $\mathcal{A}$ is given access to a signing oracle: On input (sk, m, e, τ) , return $\perp$ if $\text{sk} \notin \text{SK}$ and otherwise return $\text{Sign}(\text{PK}, \text{sk}, m, e, \tau)$.
- **Output Determination:** $\mathcal{A}$ outputs a tuple $(\text{pk}, m_1, \tau_1, \sigma_1, m_2, \tau_2, \sigma_2, e)$. Output 1 iff $m_1 \neq m_2$, $\text{pk} \in \text{PK}$, and $\text{Verify}(\text{PK}, \text{pk}, m_1, e, \tau_1, \sigma_1) = \text{Verify}(\text{PK}, \text{pk}, m_2, e, \tau_2, \sigma_2) = 1$. Otherwise, output 0.

We say UMS is $(t_{\text{eq}}, \varepsilon_{\text{eq}})$ -non-equivocation secure if for all adversaries $\mathcal{A}$ running in time t_{eq} , it holds that

$$\Pr[\text{Eq}_{\text{UMS}}^{\mathcal{A}} = 1] \leq \varepsilon_{\text{eq}}.$$

Definition 17 (Efficiency). A unique multisignature scheme UMS is efficient if there exists a polynomial p such that for any security parameter $\lambda \in \mathbb{N}$, any number of signers N , any public key set PK , any message m , any epoch $e \in \mathbb{N}$, any depth $\tau \in \mathbb{N}$, and any valid aggregate signature σ (i.e., such that $\text{VerifyAgg}(\text{PK}, m, e, \tau, \sigma) = 1$), it holds that $|\sigma| \leq p(\lambda) + \log(N)$.

Constant-Size UMS. Note that efficiency (definition 17) tolerates an aggregate signature size logarithmic in the number of signers N . However, it is easy to see that any UMS satisfying this efficiency definition can be transformed into another unique multisignature scheme UMS' with a constant aggregate signature size by modifying the algorithm VerifyAgg to drop the signature depth τ from its input. In this case, UMS' . VerifyAgg has to search for the largest depth $\tau \in [N]$ for which $\text{UMS}.$ $\text{VerifyAgg}(\dots, \tau, \dots) = 1$.

5.1 On the Impossibility of Unique Multisignatures under Standard Assumptions

In this section, we show that Unique Multisignatures are impossible to construct under standard assumptions.

This impossibility stems from the combination of the Uniqueness and Efficiency properties due to their contradictory nature. Uniqueness requires the verification algorithm to reject if the aggregate key contains a duplicate constituent (one of the public keys was aggregated more than once). To this end, the verification algorithm needs the aggregate key to encode and reveal its constituents; therefore, it must encode a per-key counter specifying how many times each public key occurs. Constructing such an encoding while preserving Efficiency violates the compression bound (**theorem 1**) because it is equivalent to encoding an arbitrary amount (linear in the number of signers) of data in a constant-size codeword (independent of the number of signers).

Theorem 3. *In the plain random oracle model, if a (t, ε) -unique multisignature scheme UMS satisfying efficiency (definition 17) exists, then there exists a δ -lossless compression scheme $\text{CMP}^H := (\text{Setup}^H, \text{Encode}^H : \mathcal{X} \rightarrow \{0, 1\}^M, \text{Decode}^H : \{0, 1\}^M \rightarrow \mathcal{X})$ for an input domain $\mathcal{X}$ and $\delta \geq 1 - N \cdot \varepsilon$, such that $M < \log(|\mathcal{X}| \cdot \delta)$.*

We provide the compression algorithm in figure 2 and defer the full proof with the formal losslessness analysis to section C.

5.2 An IVC Transition Function ($\mathcal{F}_{\text{agg}}^{\text{DS}}$)

In this section, we describe a transition function with respect to a digital signature scheme $\text{DS} := (\text{KeyGen}, \text{Sign}, \text{Verify})$.

- **State.** The state x_j consists of a tuple $(\text{PK}, \text{m}, e, j)$, where PK is a public key set, m is a message, $e \in \mathbb{N}$ is an epoch, and $j \in \mathbb{N}$ is the state index. The initial state is $x_0 := (\text{PK}, \text{m}, e, 0)$.
- **Transition Function** $\mathcal{F}_{\text{agg}}^{\text{DS}}(x_{j-1}, w_j)$. On input the previous state $x_{j-1} = (\text{PK}, \text{m}, e, j-1)$ and a witness $w_j = (\text{pk}_j, \sigma_j)$, return x_{j-1} if $\text{pk}_j \notin \text{PK}$ or $\text{DS.Verify}(\text{pk}_j, (\text{m}, e, (\text{PK}, j)), \sigma_j) = 0$. Otherwise, output an updated state $x_j := (\text{PK}, \text{m}, e, j)$.

5.3 Unique Multi Signatures under Stronger Assumptions

In this section, we propose a construction of a unique multisignature scheme UMS. The construction relies on a restrictive signing oracle $\text{RSO}_{[\text{DS}]}$ for an EUF-CMA-secure digital signature scheme, ensuring that no signer can generate more than one signature per epoch. We formally define the oracle below. This combination directly ensures that no malicious signer can sign two distinct values in the same epoch; hence, the non-equivocation property holds. However, it is insufficient for uniqueness. In particular, ensuring uniqueness requires preventing the aggregation algorithm from repeatedly aggregating the same share.

Setup($1^\lambda, N$)	Decode(pp, y)
<pre> 1: PK $\leftarrow \emptyset$ 2: pp_{ums} $\leftarrow^{\\$}$ UMS.Setup($1^\lambda, N$) 3: for $i = 1 \dots N$ do 4: (pk_i, sk_i) $\leftarrow^{\\$}$ UMS.KeyGen(pp_{ums}) 5: PK $\leftarrow$ PK $\cup$ {pk_i} 6: done 7: pp := (N, pp_{ums}, $\mathcal{X} := \{0, 1\}^N$, {sk_i}_{i$\in$[N]}, PK) 8: return pp </pre>	<pre> 1: (N, pp_{ums}, $\mathcal{X}$, {sk_i}_{i$\in$[N]}, PK) $\leftarrow$ pp 2: m := 0, e := 1, $\sigma'_0 := \perp$; $\tau := 0$ 3: if $y = \perp$ then // $x = 0 \dots 0$ 4: $\tau = 1$ 5: else 6: for $j = N \dots 1$ do 7: if UMS.VerifyAgg(PK, m, e, j, y) = 1 then 8: $\tau := j + 1$ 9: break // signature depth found, exit the loop 10: fi 11: done 12: for $i = 1 \dots N$ do 13: $\sigma_i \leftarrow^{\\$}$ (UMS.Sign(PK, sk_i, m, e, τ), τ) 14: (σ'_i, τ'_i) $\leftarrow^{\\$}$ UMS.SigAgg(PK, m, e, (y, $\tau - 1$), σ_i) 15: if UMS.VerifyAgg(PK, m, e, τ, σ'_i) = 0 then 16: $x_i := 1$ 17: else 18: $x_i := 0$ 19: fi 20: done $x := (x_1, \dots, x_N)$ 21: return x </pre>
Encode(pp, x)	
<pre> 1: (N, pp_{ums}, $\mathcal{X}$, {sk_i}_{i$\in$[N]}, PK) $\leftarrow$ pp 2: m := 0; e := 1, $\sigma'_0 := \perp$, $\tau := 0$ 3: ($x_1, \dots, x_N$) $\leftarrow$ x 4: for $i = 1 \dots N$ do 5: if $x_i = 1$ then 6: $\tau := \tau + 1$ 7: $\sigma_{\tau-1} := (\sigma'_{\tau-1}, \tau - 1)$ 8: $\sigma_\tau \leftarrow^{\\$}$ (UMS.Sign(PK, sk_i, m, e, τ), τ) 9: (σ'_τ, τ) $\leftarrow^{\\$}$ UMS.SigAgg(PK, m, e, $\sigma_{\tau-1}, \sigma_\tau$) 10: fi 11: done 12: $y := \sigma'_\tau$ 13: return y </pre>	

Fig. 2. A construction of an unconditional compression algorithm $\text{CMP} := (\text{Setup}, \text{Encode}, \text{Decode})$ from a given unique multisignature scheme UMS.

We achieve this via sequentiality: A signature σ_2 of depth τ_2 on a message $\mathbf{m}$ can be aggregated with another signature σ_1 of depth τ_1 on the same message $\mathbf{m}$ iff $\tau_2 = \tau_1 + 1$. Note that this condition is checked implicitly when evaluating IVC.Prove . It remains to ensure efficiency. To this end, we treat signature aggregation as a transition function updating a state $x_i := (\text{PK}, \mathbf{m}, e, \tau_i)$ sequentially:

$$(\text{PK}, \mathbf{m}, e, 0) \rightarrow (\text{PK}, \mathbf{m}, e, 1) \rightarrow \dots \rightarrow (\text{PK}, \mathbf{m}, e, N).$$

Then, we rely on incrementally verifiable computations to generate proofs for the current state attesting that τ_i distinct signature shares have been aggregated.

Restrictive Signing Oracle. We formalize the notion of restrictive signing oracles (RSO), an abstraction modeling any technique restricting the signing ability of parties in a distributed system. Such an oracle is constructed with respect to a digital signature scheme to fully control the parties' access to their secret keys. Concretely, it allows each party to sign a single message per epoch. To this end, signers must not be able to extract their secret keys from the oracle; otherwise, they could sign arbitrarily. Consequently, any event where a party successfully signs two messages in the same epoch must result from breaking the EUF-CMA security of the underlying signature scheme.

Definition 18 (Restrictive Signing Oracle). A restrictive signing oracle $\text{RSO}_{[\text{DS}]}$ for a digital signature scheme DS is defined by the following interfaces:

- **KeyGen**(i): The randomized key generation algorithm takes as input an identifier i , and outputs a public key pk_i .
- **Sign**(i, m, e, aux): On input an identifier i , a message m , an epoch counter e , and auxiliary data aux , it outputs a signature σ .

Definition 19 (Correctness). A restrictive signing oracle $\text{RSO}_{[\text{DS}]}$ for a digital signature scheme $\text{DS} := (\text{KeyGen}, \text{Sign}, \text{Verify})$ satisfies correctness if for all $\lambda \in \mathbb{N}$, all identifiers i , all messages m , all epoch counters $e \in \mathbb{N}$, all auxiliary data aux ,

$$\Pr \left[\text{DS}.\text{Verify}(\text{pk}_i, (m, e, \text{aux}), \sigma) = 1 \mid \begin{array}{l} \text{pk}_i \leftarrow^{\$} \text{RSO}_{[\text{DS}]}.\text{KeyGen}(i) \\ \sigma \leftarrow^{\$} \text{RSO}_{[\text{DS}]}.\text{Sign}(i, m, e, \text{aux}) \end{array} \right] = 1.$$

Definition 20 (One-Shot Security). For a number of signers $N \in \mathbb{N}$, a digital signature scheme $\text{DS} := (\text{KeyGen}, \text{Sign}, \text{Verify})$, a restrictive signing oracle $\text{RSO}_{[\text{DS}]} := (\text{KeyGen}, \text{Sign})$, and an adversary $\mathcal{A}$, we define the game $\text{OS}_{\text{RSO}_{[\text{DS}]}}^{\mathcal{A}}$ as follows:

- **Setup:** Initialize $\text{RSO}_{[\text{DS}]}$ with a public key set $\text{PK} := \emptyset$, and invoke $\mathcal{A}(1^\lambda, N)$.
- **Online Phase:** $\mathcal{A}$ is given oracle access to $\text{RSO}_{[\text{DS}]}.\text{KeyGen}(\cdot)$ which it may query on at most N distinct identifiers. For each query on an identifier i , generate a public key $\text{pk}_i \leftarrow^{\$} \text{RSO}_{[\text{DS}]}.\text{KeyGen}(i)$, add it to PK , and return it to $\mathcal{A}$. Similarly, $\mathcal{A}$ is given access to $\text{RSO}_{[\text{DS}]}.\text{Sign}(\cdot)$ to acquire signatures on tuples of the form (i, m, e, aux) .
- **Output Determination:** $\mathcal{A}$ eventually outputs a tuple $(\text{pk}^*, e^*, m_1, \text{aux}_1, \sigma_1, m_2, \text{aux}_2, \sigma_2)$. Output 1 iff all of the following hold: 1) $\text{DS}.\text{Verify}(\text{pk}^*, (m_1, e^*, \text{aux}_1), \sigma_1) = \text{DS}.\text{Verify}(\text{pk}^*, (m_2, e^*, \text{aux}_2), \sigma_2) = 1$, 2) $(m_1, \text{aux}_1) \neq (m_2, \text{aux}_2)$, 3) $\text{pk}^* \in \text{PK}$. Otherwise, output 0.

The advantage of $\mathcal{A}$ is defined as: $\text{Adv}_{\text{RSO}_{[\text{DS}]}}^{\text{OS}}(\mathcal{A}) := \Pr \left[\text{OS}_{\text{RSO}_{[\text{DS}]}}^{\mathcal{A}} = 1 \right]$. For a $(t_{\text{euf}}, \varepsilon_{\text{euf}})$ -EUF-CMA-secure digital signature scheme DS, we say that the oracle $\text{RSO}_{[\text{DS}]}$ is one-shot secure if for all adversaries $\mathcal{A}$ running in time t_{euf} , it holds that: $\text{Adv}_{\text{RSO}_{[\text{DS}]}}^{\text{OS}}(\mathcal{A}) \leq N \cdot \varepsilon_{\text{euf}}$. Intuitively, we assume that the restrictive signing oracle perfectly stores each secret key and enforces at most one signing query per

key and epoch. Hence, a winning output contains a valid signature on a message that the oracle did not sign under one of at most N public keys. Guessing this public key incurs a factor N , yielding $\text{Adv}_{\text{RSO}_{[\text{DS}]}}^{\text{OS}}(\mathcal{A}) \leq N \cdot \varepsilon_{\text{euf}}$.

Construction of Unique Multisignatures. We assume there exists a restrictive signing oracle $\text{RSO}_{[\text{DS}]} := (\text{KeyGen}, \text{Sign})$ for a digital signature scheme DS . Furthermore, we assume the existence of an incrementally verifiable computation scheme IVC .

Construction 1 (Unique Multisignature). Let $\text{IVC} := (\text{Setup}, \text{KeyGen}, \text{Prove}, \text{Verify})$ be a succinct IVC scheme satisfying completeness and knowledge soundness. Let $\mathcal{F}_{\text{agg}}^{\text{DS}}$ be the transition function defined in section 5.2. Let PK be a public key set and initialize a counter $i := 0$.

- $\text{Setup}(1^\lambda, N)$. Generate $\text{crs} \leftarrow^{\$} \text{IVC.Setup}(1^\lambda)$ and $(\text{prk}, \text{vrk}) \leftarrow \text{IVC.KeyGen}(\text{crs}, \mathcal{F}_{\text{agg}}^{\text{DS}})$, set $\text{pp} := (\text{prk}, \text{vrk})$, and return pp .
- $\text{KeyGen}(\text{pp})$. Increment $i := i + 1$, invoke $\text{pk} \leftarrow^{\$} \text{RSO}_{[\text{DS}]}.\text{KeyGen}(i)$, set $\text{sk} := i$, and return (pk, sk) .
- $\text{Sign}(\text{PK}, \text{sk}, \text{m}, e, \tau)$. Return $\sigma \leftarrow^{\$} \text{RSO}_{[\text{DS}]}.\text{Sign}(\text{sk}, \text{m}, e, (\text{PK}, \tau))$.
- $\text{Verify}(\text{PK}, \text{pk}, \text{m}, e, \tau, \sigma)$. Return $\text{DS.Verify}(\text{pk}, (\text{m}, e, (\text{PK}, \tau)), \sigma)$.
- $\text{SigAgg}(\text{PK}, \text{m}, e, (\sigma_1, \tau_1), (\sigma_2, \tau_2))$. Parse $(\text{prk}, \text{vrk}) \leftarrow \text{pp}$, and let $x_0 := (\text{PK}, \text{m}, e, 0)$, $x_{\tau_1} := (\text{PK}, \text{m}, e, \tau_1)$, and $x_{\tau_2} := (\text{PK}, \text{m}, e, \tau_2)$. Abort and output $\perp$ if

$$(\sigma_1 \neq \perp \vee \tau_1 \neq 0) \wedge \text{IVC.Verify}(\text{vrk}, x_0, x_{\tau_1}, \tau_1, \sigma_1) = 0 \\ \vee \forall \text{pk} \in \text{PK}, \text{Verify}(\text{PK}, \text{pk}, \text{m}, e, \tau_2, \sigma_2) = 0.$$

Otherwise, let pk be the key under which the signature σ_2 verifies and generate $\sigma \leftarrow^{\$} \text{IVC.Prove}(\text{prk}, x_0, x_{\tau_1}, x_{\tau_2}, \tau_2, \sigma_1, (\text{pk}, \sigma_2))$, and output (σ, τ_2) .

- $\text{VerifyAgg}(\text{PK}, \text{m}, e, \tau, \sigma)$. Parse $(\text{prk}, \text{vrk}) \leftarrow \text{pp}$, set $x_0 := (\text{PK}, \text{m}, e, 0)$ and $x_\tau := (\text{PK}, \text{m}, e, \tau)$. Return $\text{IVC.Verify}(\text{vrk}, x_0, x_\tau, \tau, \sigma)$.

Theorem 4 (Correctness of UMS). Let UMS be the unique multisignature scheme defined in construction 1. If the restrictive signing oracle $\text{RSO}_{[\text{DS}]}$ satisfies correctness, then UMS satisfies correctness.

Theorem 5 (Sequential Correctness of UMS). Let UMS be the unique multisignature scheme defined in construction 1. If the restrictive signing oracle $\text{RSO}_{[\text{DS}]}$ satisfies correctness and the incrementally verifiable computation scheme IVC satisfies perfect completeness, then UMS satisfies sequential correctness.

Theorem 6 (Uniqueness of UMS). Let UMS be the unique multisignature scheme defined in construction 1 for a number of signers N and a corruption

Table 1. Comparison of deterministic synchronous Byzantine broadcast protocols. Here, t is the corruption bound, f is the actual number of corruptions, $c = n - f$, and κ is the signature size. We assume constant-size broadcast values, i.e., $\ell = O(1)$.

Protocol	General rounds	Improved rounds	Worst-case rounds	Communication
Albouy et al. [1]	$t + 1$	$\max\{2, t + 3 - c\}$	$t + 1$	$\mathcal{O}(n!(1 + t\kappa)/(n - t - 1)!)^*$
Loss-Nielsen [25]	$\mathcal{O}(\min\{f^2, t\})$	$\mathcal{O}(f)$	$\mathcal{O}(t)$	$\mathcal{O}(n^4(\min\{f^2, t\} + \kappa))$
Deligios et al. [13]	$\mathcal{O}(\min\{f^2, t\})$	$\mathcal{O}(f)$	$\mathcal{O}(t)$	$\mathcal{O}(n^4(\min\{f^2, t\} + \kappa))$
Our work	$t + 1$	2	$t + 1$	$\mathcal{O}(n^2)$

*The communication complexity of Albouy et al. is not stated in the paper; we derive this bound from the explicit forwarding of signature chains in Algorithm 2.

bound $t \in [N]$, and let IVC , DS , and $\text{RSO}_{[\text{DS}]}$ be the primitives used in the construction. If IVC is $(t_{ks}, t'_{ks}, \varepsilon_{ks})$ -knowledge sound, DS is $(t_{euf}, \varepsilon_{euf})$ -EUF-CMA secure, and $\text{RSO}_{[\text{DS}]}$ is OS secure, then UMS is $(t_{unq}, \varepsilon_{unq})$ -Unique, where:

$$\varepsilon_{unq} \leq \varepsilon_{ks} + 2 \cdot N \cdot \varepsilon_{euf} \quad \text{and} \quad t_{unq} \approx \min(t_{ks}, t_{euf} - t'_{ks})$$

Theorem 7 (Non-Equivocation of UMS). *Let UMS be the unique multisignature scheme defined in construction 1 for a number of signers N , and let $\text{RSO}_{[\text{DS}]}$ be the restrictive signing oracle used in the construction with respect to a digital signature scheme DS . If $\text{RSO}_{[\text{DS}]}$ is OS secure, then UMS is $(t_{eq}, \varepsilon_{eq})$ -non-equivocation secure, where $\varepsilon_{eq} \leq N \cdot \varepsilon_{euf}$ and $t_{eq} \approx t_{euf}$.*

For completeness, we provide the full proofs in section C.

6 Efficient Byzantine Broadcast with Early Termination

In this section, we propose an efficient variant of the Dolev-Strong protocol with *good-case* early termination. In particular, our protocol allows honest parties to terminate in round 2 when the sender is honest. Our protocol also improves communication efficiency by replacing the linearly growing signature chains of the original Dolev-Strong protocol with a single constant-size unique multisignature section 5. This reduces the overall communication complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$, outperforming existing approaches as shown in table 1.

Early Termination. In the standard Dolev-Strong protocol, honest parties cannot safely terminate immediately after accepting a valid value. A malicious sender, together with corrupted parties, may maintain two conflicting signature chains and reveal them to different honest parties in different rounds. Therefore, an honest party that terminates as soon as it sees the first valid chain may decide on a value before learning that another conflicting chain exists. Waiting until round $t + 1$ prevents this attack: any value that can still be extended to a valid chain must eventually be forwarded by an honest party, allowing all honest parties to observe the relevant values before deciding.

Dolev-Strong Protocol with Early Termination

The protocol is executed by a set of parties $\mathcal{P} := \{P_1, \dots, P_N\}$ for a unique epoch e (increased for each execution). Assume w.l.o.g. P_1 is the sender with a public key pk_1 and input value $v \in \{0, 1\}$.

Each of the parties $P_2, \dots, P_N$ maintain three (initially empty) sets $\mathcal{V}^i, \mathcal{S}_0^i, \mathcal{S}_1^i$.

- **The Sender P_1 .** Generate $\sigma'_1 \leftarrow^{\$} \text{Sign}(\text{PK}, \text{sk}, v, e, 1)$ and $(\sigma_1, 1) \leftarrow^{\$} \text{SigAgg}(\text{PK}, v, e, (\perp, 0), (\sigma'_1, 1))$, broadcast (v, σ'_1, σ_1) , and terminate.
- **Party P_i for $i = 2 \dots N$.** For all rounds $r = 1, \dots, t + 1$, do the following:
 1. If a message $(v, \sigma'_1, \sigma_{r-1})$ was accepted in round $r - 1$, generate $\sigma'_r \leftarrow^{\$} \text{Sign}(\text{PK}, \text{sk}, v, e, r)$ and $(\sigma_r, r) \leftarrow^{\$} \text{SigAgg}(\text{PK}, v, e, (\sigma_{r-1}, r - 1), (\sigma'_r, r))$, broadcast (v, σ'_1, σ_r) , and proceed to the decision phase.
 2. When receiving a message (v, σ'_1, σ_r) in round r , if $v \notin \mathcal{V}^i$, $\text{Verify}(pk_1, m, e, 1, \sigma'_1) = 1$, and $\text{VerifyAgg}(\text{PK}, v, e, r, \sigma_r) = 1$, accept the message v by updating $\mathcal{V}^i \leftarrow \mathcal{V}^i \cup \{v\}$ and $\mathcal{S}_v^i \leftarrow \mathcal{S}_v^i \cup \{(\sigma'_1, \sigma_r)\}$.
- **Decision Phase.** Output 1 if $\mathcal{V}^i = \{1\}$ and 0 otherwise.

Fig. 3. A variant of the Dolev-Strong broadcast protocol that relies on a Unique Multisignature scheme $\text{UMS} := (\text{Setup}, \text{KeyGen}, \text{Sign}, \text{Verify}, \text{SigAgg}, \text{VerifyAgg})$. We assume all parties generated their keys and populated PK with their public keys.

Our protocol removes the source of this problem using the non-equivocation property of unique multisignatures. Every accepted value contains a valid signature share under the sender's public key for the same epoch. Hence, except with negligible probability, two conflicting values cannot both carry valid sender authentication in the same execution. Consequently, once an honest party accepts a valid value, no conflicting value can later be accepted by another honest party. The party can therefore broadcast the accepted value and terminate immediately, rather than waiting until round $t + 1$ as in the standard Dolev-Strong protocol. In particular, when the sender is honest, all honest parties accept the sender's value in round 1 and can terminate in round 2.

Communication Complexity. The communication complexity of the protocol is $\mathcal{O}(n^2)$. The sender broadcasts once in round 1, and every other honest party broadcasts at most once: after accepting a value, it updates the aggregate signature, broadcasts it to all parties, and terminates. Hence, the protocol contains at most n broadcasts by honest parties. Since each broadcast sends the same message to $\mathcal{O}(n)$ parties, this results in $\mathcal{O}(n^2)$ point-to-point messages. Moreover, each message contains only the value v , the sender's individual signature share, and a constant-size unique multisignature aggregate. Thus, the authentication information does not introduce an additional linear factor in the number of parties, reducing the communication complexity from $\mathcal{O}(n^3)$ for the standard Dolev-Strong protocol to $\mathcal{O}(n^2)$.

Theorem 8 (Adaptive Security). *Let Π be the Dolev-Strong-based broadcast protocol with early termination defined in figure 3, executed by n parties while tolerating at most $t < n$ adaptive corruptions. Suppose that the underlying*

ing unique multisignature scheme UMS satisfies correctness and sequential correctness, is $(t_{unq}, \varepsilon_{unq})$ -Unique for corruption bound t , and is $(t_{eq}, \varepsilon_{eq})$ -non-equivocation secure. Then Π satisfies Termination with probability 1, Validity with probability at least $1 - \varepsilon_{eq}$, and Agreement with probability at least $1 - \varepsilon_{eq} - \varepsilon_{unq}$. In particular, if ε_{unq} and ε_{eq} are negligible, then Π is adaptively t -secure.

Proof. Let $\mathcal{A}$ be an adaptive adversary corrupting at most t parties throughout the protocol execution.

Termination. Every honest party either accepts a value in some round $r \leq t$ and terminates after broadcasting it in round $r + 1$, or reaches the end of round $t + 1$ and executes the decision phase. Therefore, every honest party eventually terminates and outputs a value. To show validity and agreement, we define and rule out the following bad events.

- Let E_1 be the event that there exist two distinct values $v \neq v'$ and two signatures σ_1, σ'_1 such that

$$\text{UMS.Verify}(\text{PK}, \text{pk}_1, v, e, 1, \sigma_1) = \text{UMS.Verify}(\text{PK}, \text{pk}_1, v', e, 1, \sigma'_1) = 1.$$

- Let E_2 be the event that an honest party accepts a value v in round $t + 1$ with a valid aggregate signature σ_{t+1} of depth $t + 1$, but $\mathcal{H}[(v, e)] \setminus \mathcal{C} = \emptyset$, where $\mathcal{H}[(v, e)]$ is the set of parties that signed (v, e) while being honest and $\mathcal{C}$ denotes the final set of corrupted parties.

We bound the probabilities of these events through the following two claims. For completeness, we provide proofs for both claim in the supplemental material.

Claim 1. *There exists a reduction $\mathcal{R}$ such that $\Pr[E_1] \leq \Pr[\text{Eq}_{\text{UMS}}^{\mathcal{R}} = 1] \leq \varepsilon_{eq}$.*

Claim 2. *There exists a reduction $\mathcal{R}$ such that $\Pr[E_2] \leq \Pr[\text{UNQ}_{\text{UMS}}^{\mathcal{R}}(t, n) = 1] \leq \varepsilon_{unq}$.*

Validity. Suppose the sender P_1 is honest and holds input v . In round 1, P_1 generates a valid signature share σ'_1 on $(v, e, 1)$ and a valid aggregate signature σ_1 of depth 1, and broadcasts (v, σ'_1, σ_1) to all parties. By correctness and sequential correctness of UMS, every honest party accepts v in round 1.

Consider any value $v' \neq v$. For an honest party to accept v' , per figure 3, there must exist a signature σ''_1 such that

$$\text{UMS.Verify}(\text{PK}, \text{pk}_1, v', e, 1, \sigma''_1) = 1.$$

At the same time, the signature generated by the honest sender satisfies

$$\text{UMS.Verify}(\text{PK}, \text{pk}_1, v, e, 1, \sigma'_1) = 1.$$

Hence, the acceptance of $v' \neq v$ implies E_1 . By claim 1, this event occurs with probability at most ε_{eq} . Therefore, except with probability at most ε_{eq} , every

honest party accepts only v and outputs v .

Agreement. Assume an execution in which neither E_1 nor E_2 occurs.

We first observe that at most one value can be accepted by honest parties during the execution. Assume for the sake of contradiction that two distinct values v and v' are accepted, possibly by different honest parties and in different rounds. By the acceptance rule in figure 3, there exist signatures σ_1 and σ'_1 such that $\text{UMS.Verify}(\text{PK}, \text{pk}_1, v, e, 1, \sigma_1) = \text{UMS.Verify}(\text{PK}, \text{pk}_1, v', e, 1, \sigma'_1) = 1$.

Since $v \neq v'$, this implies E_1 , contradicting our assumption. Therefore, throughout the execution, all honest parties that accept a value accept the same value.

Furthermore, we notice that if the sender is honest, then the validity argument above already implies agreement. Therefore, we assume in the following that the sender is corrupted.

Next, consider the case in which no honest party accepts a value. In this case, every honest party outputs 0, and agreement holds. Therefore, it remains to consider the case in which at least one honest party accepts a value and show that this implies that every honest party accepts this value.

Let r be the first round in which an honest party accepts a value, and fix an honest party P_i that accepts a value v in round r . If $v = 0$, then no honest party can accept 1, since all values accepted by honest parties are identical. Consequently, every honest party outputs 0.

It remains to consider the case $v = 1$. We observe that the first honest acceptance of 1 must occur by round t , i.e., $r \leq t$. Indeed, an aggregate signature accepted in round $t + 1$ has depth $t + 1$. Since at most t parties are corrupted, such an aggregate cannot be generated solely from corrupted parties without violating the uniqueness of UMS.

Formally, this is captured by E_2 . Since E_2 does not occur, whenever an honest party accepts 1 in round $t + 1$, it holds that $\mathcal{H}[(1, e)] \setminus \mathcal{C} \neq \emptyset$. Fix $j \in \mathcal{H}[(1, e)] \setminus \mathcal{C}$. Since the sender is corrupted, $j \neq 1$. Thus, P_j is an honest non-sender. By the protocol, P_j can generate its signature share in some round $s \leq t + 1$ only after accepting 1 in round $s - 1$. Consequently, P_j must have accepted 1 by round t . Hence, whenever an honest party accepts 1 in round $t + 1$, some honest party has already accepted 1 by round t . It follows that the first honest acceptance of 1 occurs in some round $r \leq t$.

Since P_i remains honest throughout the execution, in round $r + 1$ it generates its signature share, extends the aggregate signature accepted in round r , and broadcasts the resulting message. By correctness and sequential correctness of UMS, the resulting aggregate signature is valid.

By the choice of r , no honest party accepted a value before round r . Every honest party that accepts a value in round r accepts 1 and therefore already holds the same value as P_i . Every other honest party remains active in round $r + 1$ and, by synchrony, receives the message broadcast by P_i and accepts 1 by the end of that round. Thus, every honest party outputs 1.

We conclude that Agreement holds whenever neither E_1 nor E_2 occurs. Therefore, Agreement can fail only if at least one of these events occurs. By claim 1

and claim 2, we have

$$\Pr[\text{Agreement fails}] \leq \Pr[E_1 \cup E_2] \leq \Pr[E_1] + \Pr[E_2] \leq \varepsilon_{eq} + \varepsilon_{unq}.$$

Hence, Π satisfies Agreement with probability at least $1 - \varepsilon_{eq} - \varepsilon_{unq}$. $\square$

References

1. Timoth   Albouy, Davide Frey, Michel Raynal, and Fran  ois Ta  ani. Good-Case Early-Stopping Latency of Synchronous Byzantine Reliable Broadcast: The Deterministic Case. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing (DISC 2022)*, volume 246 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:22, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum f  r Informatik.
2. Ryan Amos, Marios Georgiou, Aggelos Kiayias, and Mark Zhandry. One-shot signatures and applications to hybrid quantum/classical authentication. pages 255–268, 2020.
3. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. pages 1–18, 2001.
4. Mihir Bellare, Bertram Poettering, and Douglas Stebila. Deterring certificate subversion: Efficient double-authentication-preventing signatures. pages 121–151, 2017.
5. Nina Bindel, Cas Cremers, and Mang Zhao. FIDO2, CTAP 2.1, and WebAuthn 2: Provable security and post-quantum instantiation. pages 1471–1490, 2023.
6. Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. pages 31–46, 2003.
7. Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. pages 435–464, 2018.
8. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. pages 514–532, 2001.
9. T.-H. Hubert Chan, Rafael Pass, and Elaine Shi. Sublinear-round byzantine agreement under corrupt majority. pages 246–265, 2020.
10. Victor Costan, Ilia A. Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. pages 857–874, 2016.
11. Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2nd edition, 2006.
12. Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and PRGs. pages 649–665, 2010.
13. Giovanni Deligios, Ivana Klasovita, and Chen-Da Liu-Zhang. Asymptotically optimal early termination for dishonest majority broadcast. pages 116–144, 2025.
14. D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.
15. Danny Dolev and R  diger Reischuk. Bounds on information exchange for byzantine agreement. pages 132–140, 1982.
16. Ben Edgington. Upgrading ethereum: Aggregator selection. https://eth2book.info/latest/part2/building_blocks/aggregator/. Accessed: 2026-05-14.
17. Ben Edgington. Upgrading ethereum: Signatures. https://eth2book.info/capella/part2/building_blocks/signatures/. Accessed: 2026-05-14.

18. Ethereum Foundation Research. Open problems in ethereum research. <https://efdn.notion.site/open-problems-in-ethereum-research>, 2026. Accessed: 2026-05-12.
19. Victor Farazdaghi. Attestation aggregation heuristics. Ethereum Research (ethresear.ch), April 2020. Accessed: 2026-05-15.
20. Juan A. Garay, Jonathan Katz, Chiu-Yuen Koo, and Rafail Ostrovsky. Round complexity of authenticated broadcast with a dishonest majority. pages 658–668, 2007.
21. Dennis Hofheinz, Michael Reichle, and Benedikt Wagner. Hint-free multi-signatures. Cryptology ePrint Archive, Paper 2026/723, 2026.
22. K. ITAKURA. A public-key cryptosystem suitable for digital multisignature. *NEC Research and Development*, 71:1–8, 1983.
23. Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. pages 60–73, 2021.
24. Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. pages 359–388, 2022.
25. Julian Loss and Jesper Buus Nielsen. Early stopping for any number of corruptions. pages 457–488, 2024.
26. Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. pages 465–485, 2006.
27. Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. pages 74–90, 2004.
28. Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr multi-signatures with applications to Bitcoin. Cryptology ePrint Archive, Report 2018/068, 2018.
29. Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: Extended abstract. pages 245–254, 2001.
30. Atsuki Momose and Ling Ren. Optimal communication complexity of authenticated byzantine agreement. Cryptology ePrint Archive, Report 2020/1569, 2020.
31. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
32. Jonas Nick, Tim Ruffing, and Yannick Seurin. MuSig2: Simple two-round Schnorr multi-signatures. pages 189–221, 2021.
33. Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. pages 228–245, 2007.
34. Khalid Sayood. *Introduction to Data Compression*. Morgan Kaufmann, 5th edition, 2017.
35. Claude E Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.
36. Omri Shmueli and Mark Zhandry. On one-shot signatures, quantum vs. classical binding, and obfuscating permutations. pages 350–383, 2025.
37. Michael Sproul and Sigma Prime. Optimising attestation packing. Sigma Prime Blog, November 2022. Accessed: 2026-05-15.
38. Enis Ulqinaku, Hala Assal, AbdelRahman Abdou, Sonia Chiasson, and Srdjan Capkun. Is real-time phishing eliminated with FIDO? Social engineering downgrade attacks against FIDO protocols. pages 3811–3828, 2021.
39. Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. pages 1–18, 2008.
40. Jun Wan, Hanshen Xiao, Srinivas Devadas, and Elaine Shi. Round-efficient byzantine broadcast under strongly adaptive and majority corruptions. pages 412–456, 2020.

41. Jun Wan, Hanshen Xiao, Elaine Shi, and Srinivas Devadas. Expected constant round byzantine broadcast under dishonest majority. pages 381–411, 2020.

A RSO from One-Shot Signatures

We now show how to instantiate a restrictive signing oracle using quantum information, relying on the notion of one-shot signatures. To do so, we first recall the definition of one-shot signatures introduced by Amos et al. [2].

Definition 21 (One-shot Signatures).

A one-shot signature scheme is a tuple of algorithms $(\text{KeyGen}, \text{Sign}, \text{Verify})$ with the following syntax:

- $\text{KeyGen}(\text{crs})$: On input a common reference string crs , outputs a classical public key pk and a quantum secret key sk .
- $\text{Sign}(\text{sk}, \text{m})$: On input a secret key sk and a message m , outputs a signature σ .
- $\text{Verify}(\text{crs}, \text{pk}, \text{m}, \sigma)$: On input a common reference string crs , a public key pk , a message m , and a signature σ , outputs a bit b .

Definition 22 (Correctness).

If $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{crs})$, then $\text{Verify}(\text{crs}, \text{pk}, \text{m}, \text{Sign}(\text{sk}, \text{m})) = 1$ for any message m with overwhelming probability.

Definition 23 (One-Shot Security). For every quantum polynomial-time algorithm $\mathcal{A}$, we have:

$$\Pr \left[\begin{array}{l} \text{Verify}(\text{crs}, \text{pk}, \text{m}_0, \sigma_0) = 1 \\ \text{Verify}(\text{crs}, \text{pk}, \text{m}_1, \sigma_1) = 1 \end{array} \middle| \begin{array}{l} \text{crs} \leftarrow \{0, 1\}^n \\ (\text{pk}, \{(\text{m}_0, \sigma_0), (\text{m}_1, \sigma_1)\}) \leftarrow \mathcal{A}(\text{crs}) \end{array} \right] \leq \varepsilon(\lambda)$$

We now show that one-shot signatures naturally realize the notion of a restrictive signing oracle. We assume that the number of epochs is upper bounded by $e_u(\lambda)$, which is polynomial in the security parameter.

We define $\text{RSO.KeyGen}(i)$ and $\text{RSOSign}(i, \text{m}, e, \text{aux})$ as follows. Throughout, we assume that a common reference string crs is given:

- $\text{RSO.KeyGen}(i)$: Create e_u key pairs for the one-shot signature scheme by running $(\text{pk}_j, \text{sk}_j) \leftarrow \text{KeyGen}(\text{crs})$ for each $j \in [e_u]$. Output the public key $\text{PK}_i = (\text{pk}_1, \dots, \text{pk}_{e_u})$ and store $\text{SK}_i = (\text{sk}_1, \dots, \text{sk}_{e_u})$.
- $\text{RSO.Sign}(i, \text{m}, e, \text{aux})$: Recover $\text{SK}_i = (\text{sk}_1, \dots, \text{sk}_{e_u})$, set $M = (\text{m} || \text{aux})$, and output $\text{Sign}(\text{sk}_e, M)$.

Lemma 1. *The above construction is a correct and one-shot secure restrictive signing oracle instantiation.*

Proof. Correctness of the restrictive signing oracle follows directly from the correctness of the underlying one-shot signature scheme.

For one-shot security, suppose that there exists an adversary $\mathcal{A}$ that breaks the one-shot security of the RSO scheme. That is, the adversary outputs a valid

tuple $(\text{PK}^*, e^*, \mathbf{m}_1, \text{aux}_1, \sigma_1, \mathbf{m}_2, \text{aux}_2, \sigma_2)$. This means that there exists a public key $\text{pk}_e^* \in \text{PK}^*$ and messages $M_1 = (\mathbf{m}_1 || \text{aux}_1)$ and $M_2 = (\mathbf{m}_2 || \text{aux}_2)$ such that

$$\text{Verify}(\text{pk}_e^*, M_1, \sigma_1) = \text{Verify}(\text{pk}_e^*, M_2, \sigma_2) = 1.$$

Therefore, $(\text{pk}_e^*, (M_1, \sigma_1), (M_2, \sigma_2))$ is a valid forgery against the one-shot security of the underlying signature scheme for the public key pk_e^* . $\square$

B Extended Preliminaries

B.1 Digital Signatures

Definition 24 (Correctness). A digital signature scheme DS satisfies correctness if for all $\lambda \in \mathbb{N}$ and all messages $\mathbf{m}$,

$$\Pr \left[\text{Verify}(\text{pk}, \mathbf{m}, \sigma) = 1 \mid (\text{pk}, \text{sk}) \xleftarrow{\$} \text{KeyGen}(1^\lambda), \sigma \xleftarrow{\$} \text{Sign}(\text{sk}, \mathbf{m}) \right] = 1.$$

Definition 25 (Existential Unforgeability (EUF-CMA)). Let $\text{DS} := (\text{KeyGen}, \text{Sign}, \text{Verify})$ be a digital signature scheme and $\mathcal{A}$ be an adversary. For any security parameter $\lambda \in \mathbb{N}$, we define the game $\text{EUF-CMA}_{\text{DS}}^{\mathcal{A}}$ as follows:

- **Setup:** Compute $(\text{pk}, \text{sk}) \xleftarrow{\$} \text{KeyGen}(1^\lambda)$ and invoke $\mathcal{A}(\text{pk})$.
- **Online Phase:** $\mathcal{A}$ is given oracle access to $\text{Sign}(\text{sk}, \cdot)$. Let $\mathcal{Q}$ be the set of all messages queried by $\mathcal{A}$ to this oracle.
- **Output Determination:** $\mathcal{A}$ eventually outputs a forgery pair $(\mathbf{m}^*, \sigma^*)$. The game outputs 1 if $\text{Verify}(\text{pk}, \mathbf{m}^*, \sigma^*) = 1$ and $\mathbf{m}^* \notin \mathcal{Q}$. Otherwise, it outputs 0.

The advantage of $\mathcal{A}$ is defined as:

$$\text{Adv}_{\text{DS}}^{\text{EUF-CMA}}(\mathcal{A}) := \Pr \left[\text{EUF-CMA}_{\text{DS}}^{\mathcal{A}} = 1 \right].$$

We say that the scheme DS is $(t_{\text{euf}}, \varepsilon_{\text{euf}})$ -EUF-CMA secure if for all adversaries $\mathcal{A}$ running in time at most t_{euf} , it holds that

$$\text{Adv}_{\text{DS}}^{\text{EUF-CMA}}(\mathcal{A}) \leq \varepsilon_{\text{euf}}.$$

C Deferred Proofs

Theorem 2. In the random oracle model, if there exists an efficient idempotent multisignature scheme satisfying idempotent correctness and (t, ε) -EUF-CMA security for some $N \cdot \varepsilon < 1/2$, then there exists a δ' -lossless compression scheme $\text{CMP}^{\text{H}} := (\text{Setup}^{\text{H}}, \text{Encode}^{\text{H}} : \mathcal{X} \rightarrow \{0, 1\}^M, \text{Decode}^{\text{H}} : \{0, 1\}^M \rightarrow \mathcal{X})$ for $M \leq N - 1$, $\delta' := 1 - N \cdot \varepsilon$, a realization of the random oracle H , and an input domain $\mathcal{X}$, such that $M < \log(|\mathcal{X}| \cdot \delta')$, which violates the bound in **theorem 1**.

Proof. Given an efficient Idempotent Multisignatures scheme $\text{IMS} := (\text{Setup}, \text{KeyGen}, \text{KeyAgg}, \text{Sign}, \text{SigAgg}, \text{Verify})$, we construct a compression scheme $\text{CMP}^{\text{H}} := (\text{Setup}^{\text{H}}, \text{Encode}^{\text{H}}, \text{Decode}^{\text{H}})$ as depicted in figure 1.

Now, we analyze the δ -losslessness of our construction. For any $x \in \mathcal{X}$, let $y := \text{Encode}^{\text{H}}(x)$ and $x' := \text{Decode}^{\text{H}}(y)$. For all $\iota \in [N]$, we distinguish two cases:

- $x_\iota = 1$: By construction, since $x_\iota = 1$, Encode^{H} ensures that $S[\iota] = \text{IMS.Sign}(\text{sk}_\iota, 0)$ is an aggregant of y . Additionally, Decode^{H} ensures that σ_ι contains all $S[j] = \text{IMS.Sign}(\text{sk}_j, 0)$ for $\iota \neq j \in [N]$ (from σ_{S_ι}) as well as $\text{IMS.Sign}(\text{sk}_\iota, 0)$ (from y). Consequently, each signature share in S appears at least once as an aggregant in σ_ι . Per the idempotent correctness property of IMS, $\text{IMS.Verify}(\text{apk}, 0, \sigma_\iota)$ evaluates to 1 with probability 1. As Decode^{H} computes $x'_\iota := \text{IMS.Verify}(\text{apk}, 0, \sigma_\iota)$, it holds that $\Pr[x'_\iota = 1] = \Pr[x'_\iota = x_\iota] = 1$.
- $x_\iota = 0$: In this case, Encode^{H} does not include $S[\iota]$ as an aggregant in y , and neither Decode^{H} includes it as an aggregant in σ_{S_ι} , hence $S[\iota] = \text{IMS.Sign}(\text{sk}_\iota, 0)$ is not an aggregant of σ_ι . Consequently, it must hold that $\text{IMS.Verify}(\text{apk}, 0, \sigma_\iota) = 0$ per the EUF-CMA security of IMS with probability at least $1 - \varepsilon$ since IMS is (t, ε) -EUF-CMA secure. Consequently, since Decode^{H} outputs $x'_\iota := \text{IMS.Verify}(\text{apk}, 0, \sigma_\iota)$, it holds that $\Pr[x'_\iota = 0] = \Pr[x'_\iota = x_\iota] \geq 1 - \varepsilon$. We show this formally in claim 3.

Claim 3. *For any $x \in \mathcal{X}$ with $y := \text{Encode}^{\text{H}}(x)$ and $x' := \text{Decode}^{\text{H}}(y)$, and any $\iota \in [N]$ with $x_\iota = 0$, there exists a reduction $\mathcal{R}$ with*

$$\Pr[x'_\iota = 1] \leq \text{Adv}_{\text{IMS}}^{\text{EUF-CMA}}(\mathcal{R}) \leq \varepsilon.$$

Proof. For a fixed $x \in \mathcal{X}$ and $\iota \in [N]$ such that $x_\iota = 0$, we construct a reduction $\mathcal{R}$ that plays the EUF-CMA game against the IMS challenger. $\mathcal{R}$ receives the public parameters pp_{ims} and the challenger's public key pk^* , and proceeds as follows:

1. Set $\text{pk}_\iota := \text{pk}^*$.
2. For all $j \in [N] \setminus \{\iota\}$, generate key pairs $(\text{pk}_j, \text{sk}_j) \leftarrow \text{IMS.KeyGen}(\text{pp}_{\text{ims}}; \text{rnd}_j)$ using randomly sampled coins rnd_j , and compute the basic signature shares $S[j] \leftarrow \text{IMS.Sign}(\text{sk}_j, 0)$.
3. Compute the partial aggregate signature $y := \text{IMS.SigAgg}(\{S[j] \mid x_j = 1\})$. Note that because $x_\iota = 0$, this step is computable without knowing sk_ι or $S[\iota]$.
4. Compute $\sigma_{S_\iota} := \text{IMS.SigAgg}(\{S[j] \mid j \in [N] \setminus \{\iota\}\})$ and $\sigma_\iota := \text{IMS.SigAgg}(\{\sigma_{S_\iota}, y\})$ as done in Decode^{H} .
5. Output the forgery $(\text{PK}, \text{m}^*, \sigma^*) := (\{\text{pk}_1, \dots, \text{pk}_N\}, 0, \sigma_\iota)$ to the challenger.

Notice that $\mathcal{R}$ perfectly simulates the execution of Encode^{H} and Decode^{H} without querying challenger's signing oracle for the target message 0. Furthermore, the key pk^* is included in the keyset PK. By the definition of Decode^{H} , the bit x'_ι evaluates to 1 iff $\text{IMS.Verify}(\text{apk}, 0, \sigma_\iota) = 1$. In this case, σ_ι is a valid forgery under the aggregate key $\text{apk} = \text{IMS.KeyAgg}(\text{PK})$, hence

$$\Pr[x'_\iota = 1] \leq \text{Adv}_{\text{IMS}}^{\text{EUF-CMA}}(\mathcal{R}) \leq \varepsilon.$$

□

Overall, we have

$$\begin{aligned}
& \Pr \left[\text{Decode}^H(\text{Encode}^H(x)) = x \right] = \Pr \left[\text{Decode}^H(y) = x \mid y = \text{Encode}^H(x) \right] \\
&= \Pr \left[x' = x \mid y = \text{Encode}^H(x), x' = \text{Decode}^H(y) \right] \\
&= \Pr \left[\forall i \in [N], x'_i = x_i \mid y = \text{Encode}^H(x), x' = \text{Decode}^H(y) \right] \\
&= 1 - \Pr \left[\exists \iota \in [N], x'_\iota \neq x_\iota \mid y = \text{Encode}^H(x), x' = \text{Decode}^H(y) \right] \\
&= 1 - \Pr \left[\exists \iota \in [N], \bigvee_{x'_\iota = 0 \neq x_\iota} \bigvee_{x'_\iota = 1 \neq x_\iota} \mid y = \text{Encode}^H(x), x' = \text{Decode}^H(y) \right] \\
&\geq 1 - \Pr \left[\exists \iota \in [N], x'_\iota = 0 \neq x_\iota \mid y = \text{Encode}^H(x), x' = \text{Decode}^H(y) \right] \\
&\quad - \Pr \left[\exists \iota \in [N], x'_\iota = 1 \neq x_\iota \mid y = \text{Encode}^H(x), x' = \text{Decode}^H(y) \right] \\
&\geq 1 - 0 - N \cdot \varepsilon = 1 - N \cdot \varepsilon.
\end{aligned}$$

Our construction is δ' -lossless for $\delta' := 1 - N \cdot \varepsilon$, and its codeword length M is identical to the signature length output by `IMS.SigAgg`. Since `IMS` must be efficient (in the sense of definition 11), it holds that $M \leq N - 1$. However, by **theorem 1**, it must hold that $M \geq \log(|\mathcal{X}| \cdot \delta') = \log(2^N \cdot \delta') = N + \log(\delta')$. Since $N \cdot \varepsilon < 1/2$, we have $\delta' > 1/2$ and therefore $\log(\delta') > -1$. Hence, $N - 1 < N + \log(\delta')$, which is a contradiction. Therefore, `IMS` cannot exist. □

Theorem 3. *In the plain random oracle model, if a (t, ε) -unique multisignature scheme `UMS` satisfying efficiency (definition 17) exists, then there exists a δ -lossless compression scheme $\text{CMP}^H := (\text{Setup}^H, \text{Encode}^H : \mathcal{X} \rightarrow \{0, 1\}^M, \text{Decode}^H : \{0, 1\}^M \rightarrow \mathcal{X})$ for an input domain $\mathcal{X}$ and $\delta \geq 1 - N \cdot \varepsilon$, such that $M < \log(|\mathcal{X}| \cdot \delta)$.*

Proof. Let $\text{UMS} := (\text{Setup}, \text{KeyGen}, \text{Sign}, \text{Verify}, \text{SigAgg}, \text{VerifyAgg})$ be an arbitrary unique multisignature scheme. We construct a compression scheme $\text{CMP}^H := (\text{Setup}^H, \text{Encode}^H, \text{Decode}^H)$ as depicted in figure 2.

It remains to show that our scheme is δ -lossless. For any $x \in \mathcal{X}$, let $y := \text{Encode}^H(x)$ and $x' := \text{Decode}^H(y)$. For all $\iota \in [N]$, we distinguish two cases:

- $x_\iota = 1$: By construction, since $x_\iota = 1$, Encode^H ensures the generation of a signature share σ_ι on $(\mathbf{m}, e)$ under the secret key sk_ι and its aggregation in the aggregate signature y . Upon decoding, Decode^H generates a fresh signature share on $(\mathbf{m}, e)$ using the secret key sk_ι , aggregates it with y to generate a fresh aggregate signature σ'_ι of depth $\tau'_\iota := \tau$, where $\tau - 1$ is the depth of y . Finally, Decode^H computes $x'_\iota := 1 - \text{UMS.VerifyAgg}(\text{PK}, \mathbf{m}, e, \tau'_\iota, \sigma'_\iota)$. Per the uniqueness property, $\text{UMS.VerifyAgg}(\text{PK}, \mathbf{m}, e, \tau'_\iota, \sigma'_\iota) = 0$, except with probability at most ε . Thus, except with probability ε , it must hold that $x_\iota = x'_\iota$. We state this formally in the following claim.

Claim 4. For any $x \in \mathcal{X}$ with $y := \text{Encode}^H(x)$ and $x' := \text{Decode}^H(y)$, and any $\iota \in [N]$ with $x_\iota = 1$, there exists a reduction $\mathcal{R}$ such that

$$\Pr[x'_\iota = 0] \leq \text{Adv}_{\text{UMS}}^{\text{UNQ}}(\mathcal{R}) \leq \varepsilon.$$

Proof. For a fixed arbitrary string $x \in \mathcal{X}$ and an index $\iota \in [N]$ where $x_\iota = 1$, assume Decode^H outputs $x'_\iota = 0$. We construct a reduction $\mathcal{R}$ that plays the game $\text{UNQ}_{\text{UMS}}^{\mathcal{R}}$.

During the setup phase, $\mathcal{R}$ receives the public parameters pp and the public key set PK .

$\mathcal{R}$ emulates $\text{Encode}^H(x)$ to compute the aggregate signature y of depth $\tau := H$, where H is the Hamming weight of x . For every index j where $x_j = 1$, $\mathcal{R}$ queries the signing oracle on $(j, \mathbf{m}, e, \tau)$ at the corresponding depth. Therefore, after emulating $\text{Encode}^H(x)$, it holds that

$$\mathcal{H}[(\mathbf{m}, e)] = \{j \in [N] \mid x_j = 1\}.$$

Because $x_\iota = 1$ and $x'_\iota = 0$, emulating the decoder at index ι yields an aggregate signature σ' of depth $\tau := H+1$ such that $\text{UMS.VerifyAgg}(\text{PK}, \mathbf{m}, e, \tau, \sigma') = 1$. In particular, $\mathcal{R}$ queries the signing oracle on $(\iota, \mathbf{m}, e, H+1)$ and aggregates the resulting signature share with y as done in Decode^H .

Since $x_\iota = 1$, the index ι already belongs to $\mathcal{H}[(\mathbf{m}, e)]$, and therefore this additional signing query does not increase the size of $\mathcal{H}[(\mathbf{m}, e)]$. Furthermore, $\mathcal{R}$ makes no corruption queries. Hence,

$$|\mathcal{H}[(\mathbf{m}, e)] \cup \mathcal{C}| = H.$$

$\mathcal{R}$ outputs the tuple $(\mathbf{m}, e, H+1, \sigma')$. Because

$$\text{UMS.VerifyAgg}(\text{PK}, \mathbf{m}, e, H+1, \sigma') = 1$$

and

$$H+1 > |\mathcal{H}[(\mathbf{m}, e)] \cup \mathcal{C}| = H,$$

$\mathcal{R}$ satisfies all winning conditions. Therefore, we have:

$$\Pr[x'_\iota = 0] \leq \text{Adv}_{\text{UMS}}^{\text{UNQ}}(\mathcal{R}) \leq \varepsilon.$$

□

Thus, we have

$$\Pr[x'_\iota = 1 \mid x_\iota = 1] \geq 1 - \varepsilon.$$

- $x_\iota = 0$: In this case, Encode^H generates the aggregate signature y without including a signature share from sk_ι in it. Then Decode^H generates a signature share for sk_ι as $\sigma_\iota := (\text{UMS.Sign}(\text{sk}_\iota, \mathbf{m}, e, \tau+1), \tau+1)$, where τ is the depth of y . Then, Decode^H aggregates it with y as $(\sigma'_\iota, \tau'_\iota) \leftarrow^{\$} \text{UMS.SigAgg}(\text{PK}, \mathbf{m}, e, (y, \tau), \sigma_\iota)$. As σ'_ι was generated by aggregating $\tau+1$ signature shares from $\tau+1$ distinct signers, it follows by definition 14 that

$\text{UMS.VerifyAgg}(\text{PK}, \mathbf{m}, e, \tau + 1, \sigma'_\ell) = 1$ with probability 1, hence Decode^H computes $x'_\ell := 0$, hence

$$\Pr[x'_\ell = 0 \mid x_\ell = 0] = 1.$$

Overall, we have

$$\begin{aligned} & \Pr[\text{Decode}^H(\text{Encode}^H(x)) = x] = \Pr[\text{Decode}^H(y) = x \mid y = \text{Encode}^H(x)] \\ &= \Pr[x' = x \mid y = \text{Encode}^H(x), x' = \text{Decode}^H(y)] \\ &= \Pr[\forall i \in [N], x'_i = x_i \mid y = \text{Encode}^H(x), x' = \text{Decode}^H(y)] \\ &= 1 - \Pr[\exists \ell \in [N], x'_\ell \neq x_\ell \mid y = \text{Encode}^H(x), x' = \text{Decode}^H(y)] \\ &= 1 - \Pr[\exists \ell \in [N], \bigvee_{x'_\ell = 0} x'_\ell = 0 \neq x_\ell \mid y = \text{Encode}^H(x), x' = \text{Decode}^H(y)] \\ &\geq 1 - \Pr[\exists \ell \in [N], x'_\ell = 0 \neq x_\ell \mid y = \text{Encode}^H(x), x' = \text{Decode}^H(y)] \\ &\quad - \Pr[\exists \ell \in [N], x'_\ell = 1 \neq x_\ell \mid y = \text{Encode}^H(x), x' = \text{Decode}^H(y)] \\ &\geq 1 - N \cdot \varepsilon - 0 = 1 - N \cdot \varepsilon. \end{aligned}$$

Our construction is δ -lossless for $\delta := 1 - N \cdot \varepsilon$, and it has a codeword size identical to the signature size $|y|$ output by UMS.SigAgg . Since UMS must be efficient (in the sense of definition 17), it holds that $|y| \leq \text{poly}(\lambda) + \log(N) \equiv \mathcal{O}(1) + \log(N)$. However, per **theorem 1**, it must hold that $|y| \geq \log(|\mathcal{X}| \cdot \delta) = \log(2^N \cdot \delta) = N + \log(\delta)$. For a sufficiently large N such that $\mathcal{O}(1) + \log(N) < N + \log(\delta)$, we have a contradiction. Therefore, UMS cannot exist in the plain random oracle model. $\square$

Theorem 4 (Correctness of UMS). *Let UMS be the unique multisignature scheme defined in construction 1. If the restrictive signing oracle $\text{RSO}_{[\text{DS}]}$ satisfies correctness, then UMS satisfies correctness.*

Proof. Let $\mathbf{m}$ be a message, e an epoch, $\tau \in [N]$ a depth, and $i \in [N]$ a signer. By construction,

$$\sigma \stackrel{\S}{\leftarrow} \text{UMS.Sign}(\text{PK}, \text{sk}_i, \mathbf{m}, e, \tau) = \text{RSO}_{[\text{DS}]}.\text{Sign}(\text{sk}_i, \mathbf{m}, e, (\text{PK}, \tau)).$$

By the correctness of $\text{RSO}_{[\text{DS}]}$, it holds that

$$\text{DS.Verify}(\text{pk}_i, (\mathbf{m}, e, (\text{PK}, \tau)), \sigma) = 1$$

with probability 1. By definition of UMS.Verify , this is equivalent to

$$\text{UMS.Verify}(\text{PK}, \text{pk}_i, \mathbf{m}, e, \tau, \sigma) = 1.$$

Therefore, UMS satisfies correctness. $\square$

Theorem 5 (Sequential Correctness of UMS). *Let UMS be the unique multisignature scheme defined in construction 1. If the restrictive signing oracle $\text{RSO}_{[\text{DS}]}$ satisfies correctness and the incrementally verifiable computation scheme IVC satisfies perfect completeness, then UMS satisfies sequential correctness.*

Proof. Let $\mathbf{m}$ be a message, e an epoch, and suppose k honest signers $(i_1, \dots, i_k)$ iteratively compute an aggregate signature σ'_k . By definition, verifying this aggregate signature is equivalent to verifying the underlying IVC proof:

$$\text{UMS.VerifyAgg}(\text{PK}, \mathbf{m}, e, k, \sigma'_k) = \text{IVC.Verify}(\text{vrk}, x_0, x_k, k, \sigma'_k).$$

To show this evaluates to 1, we simply need to show that every state transition in the sequence is valid. At any step $j \in [k]$, the honest signer i_j computes

$$\sigma_j \leftarrow^{\$} \text{UMS.Sign}(\text{PK}, \text{sk}_{i_j}, \mathbf{m}, e, j) = \text{RSO}_{[\text{DS}]}.\text{Sign}(\text{sk}_{i_j}, \mathbf{m}, e, (\text{PK}, j)).$$

By the correctness of $\text{RSO}_{[\text{DS}]}$, this signature is valid, i.e.,

$$\text{DS.Verify}(\text{pk}_{i_j}, (\mathbf{m}, e, (\text{PK}, j)), \sigma_j) = 1.$$

When UMS.SigAgg folds this step into the chain, it provides the witness $w_j := (\text{pk}_{i_j}, \sigma_j)$. Because pk_{i_j} is registered in PK and σ_j is valid, the transition function evaluates successfully, mapping the previous state x_{j-1} to the new state x_j :

$$\mathcal{F}_{\text{agg}}^{\text{DS}}((\text{PK}, \mathbf{m}, e, j-1), (\text{pk}_{i_j}, \sigma_j)) = (\text{PK}, \mathbf{m}, e, j).$$

Since $\perp$ is a valid initial proof for the initial state $x_0 := (\text{PK}, \mathbf{m}, e, 0)$, and every subsequent step perfectly satisfies the transition function $\mathcal{F}_{\text{agg}}^{\text{DS}}$, the perfect completeness of IVC guarantees that every updated proof σ'_j is valid. In particular, the final updated proof σ'_k is valid. Therefore, the verification algorithm outputs 1. $\square$

Theorem 6 (Uniqueness of UMS). *Let UMS be the unique multisignature scheme defined in construction 1 for a number of signers N and a corruption bound $\mathfrak{t} \in [N]$, and let IVC, DS, and $\text{RSO}_{[\text{DS}]}$ be the primitives used in the construction. If IVC is $(t_{ks}, t'_{ks}, \varepsilon_{ks})$ -knowledge sound, DS is $(t_{euf}, \varepsilon_{euf})$ -EUF-CMA secure, and $\text{RSO}_{[\text{DS}]}$ is OS secure, then UMS is $(t_{unq}, \varepsilon_{unq})$ -Unique, where:*

$$\varepsilon_{unq} \leq \varepsilon_{ks} + 2 \cdot N \cdot \varepsilon_{euf} \quad \text{and} \quad t_{unq} \approx \min(t_{ks}, t_{euf} - t'_{ks})$$

Proof. Let $\mathcal{A}$ be an adversary playing the uniqueness game against UMS. We bound the advantage $\text{Adv}_{\text{UMS}}^{\text{UNQ}}(\mathcal{A})$ through a sequence of games.

Game $\mathbf{G}_0$. This is the original $\text{UNQ}_{\text{UMS}}^{\mathcal{A}}$ game.

Game $\mathbf{G}_1$. This game is identical to $\mathbf{G}_0$, except that we define a bad event $\mathbf{E}_3$ and the game aborts and outputs 0 if this event occurs. We fix the random coins r of the adversary $\mathcal{A}$. When $\mathcal{A}$ eventually terminates and outputs a valid forgery tuple $(\mathbf{m}^*, e^*, \tau^*, \sigma^*)$, it must hold that $\text{UMS.VerifyAgg}(\text{PK}, \mathbf{m}^*, e^*, \tau^*, \sigma^*) = 1$. Per construction, this implies that σ^* is a valid IVC proof. That is, $\text{IVC.Verify}(\text{vrk}, x_0, x_{\tau^*}, \tau^*, \sigma^*) = 1$, where $\text{pp} = (\text{prk}, \text{vrk})$, $x_0 := (\text{PK}, \mathbf{m}^*, e^*, 0)$, and $x_{\tau^*} := (\text{PK}, \mathbf{m}^*, e^*, \tau^*)$.

Upon successful verification, the game invokes the knowledge extractor $(w_1, \dots, w_{\tau^*}) \leftarrow \mathcal{E}(\text{crs}, r)$ to extract the witness sequence. We define the bad event $\mathbf{E}_3$ to occur if the extracted witness chain contains an invalid transition. Formally, $\mathbf{E}_3$ triggers if there exists an index $j \in [\tau^*]$ such that for the extracted witness $w_j = (\text{pk}_j, \sigma_j)$, either $\text{pk}_j \notin \text{PK}$ or the signature is invalid for the state transition, i.e. $\text{DS.Verify}(\text{pk}_j, (\mathbf{m}^*, e^*, (\text{PK}, j)), \sigma_j) \neq 1$.

Claim 5. *If IVC is $(t_{ks}, t'_{ks}, \varepsilon_{ks})$ -knowledge sound, then*

$$\Pr[\mathbf{E}_3] \leq \varepsilon_{ks}.$$

Proof. We construct a reduction $\mathcal{R}_1$ that acts as a malicious prover against the knowledge soundness of IVC. $\mathcal{R}_1$ receives the common reference string crs and random coins r from the challenger. It computes $(\text{prk}, \text{vrk}) \leftarrow \text{IVC.KeyGen}(\text{crs}, \mathcal{F}_{\text{agg}}^{\text{DS}})$, constructs the UMS public parameters $\text{pp} := (\text{prk}, \text{vrk})$, initializes $\text{PK} \leftarrow \emptyset$, $\mathcal{C} \leftarrow \emptyset$, $\mathcal{K}[\cdot] \leftarrow \emptyset$, and $\mathcal{H}[\cdot] \leftarrow \emptyset$. For every $i \in [N]$, it generates $(\text{pk}_i, \text{sk}_i) \xleftarrow{\$} \text{UMS.KeyGen}(\text{pp})$, sets $\mathcal{K}[i] := (\text{pk}_i, \text{sk}_i)$, and updates $\text{PK} \leftarrow \text{PK} \cup \{\text{pk}_i\}$. It then invokes $\mathcal{A}(\text{pp}, \text{PK})$ using the coins r and answers its signing and corruption oracle queries exactly as in $\mathbf{G}_0$, perfectly simulating $\mathbf{G}_0$.

When $\mathcal{A}$ outputs its valid forgery $(\mathbf{m}^*, e^*, \tau^*, \sigma^*)$, $\mathcal{R}_1$ computes the states $x_0 := (\text{PK}, \mathbf{m}^*, e^*, 0)$ and $x_{\tau^*} := (\text{PK}, \mathbf{m}^*, e^*, \tau^*)$, and outputs the tuple $(\mathcal{F}_{\text{agg}}^{\text{DS}}, x_0, x_{\tau^*}, \sigma^*)$ to its challenger.

By knowledge soundness, except with probability at most ε_{ks} , whenever $\mathcal{R}_1$ outputs a verifying proof, $\mathcal{E}$ outputs a valid sequence of witnesses. By definition, $\mathbf{E}_3$ occurs exactly when $\mathcal{R}_1$ generates a proof that satisfies the verification relation, but the corresponding extractor $\mathcal{E}(\text{crs}, r)$ fails to output a sequence of witnesses satisfying the transition function $\mathcal{F}_{\text{agg}}^{\text{DS}}$ (i.e., invalid witnesses sequence). Consequently, we have that $\Pr[\mathbf{E}_3] \leq \varepsilon_{ks}$ and hence the claim. $\square$

Per the claim, we have

$$|\Pr[\mathbf{G}_1 = 1] - \Pr[\mathbf{G}_0 = 1]| \leq \Pr[\mathbf{E}_3] \leq \varepsilon_{ks}.$$

Game $\mathbf{G}_2$. This game is identical to $\mathbf{G}_1$, except that we define a bad event $\mathbf{E}_4$ and the game aborts and outputs 0 if this event occurs. Upon extracting the witness sequence $(w_1, \dots, w_{\tau^*}) \leftarrow \mathcal{E}(\text{crs}, r)$ where each witness w_j is a signature-key pair $(\text{pk}_j^*, \sigma_j^*)$, we define $\mathbf{E}_4$ to occur if the chain contains a signature generated under a public key pk_i such that $i \notin \mathcal{H}[(\mathbf{m}^*, e^*)] \cup \mathcal{C}$. Formally, $\mathbf{E}_4$ occurs if there exist indices $j \in [\tau^*]$ and $i \in [N] \setminus (\mathcal{H}[(\mathbf{m}^*, e^*)] \cup \mathcal{C})$ such that for the extracted witness $w_j = (\text{pk}_j^*, \sigma_j^*)$, it holds that $\text{pk}_j^* = \text{pk}_i$.

Claim 6. *If DS is $(t_{\text{euf}}, \varepsilon_{\text{euf}})$ -EUF-CMA secure, then*

$$\Pr[\mathbf{E}_4] \leq N \cdot \varepsilon_{\text{euf}}.$$

Proof. We construct a reduction $\mathcal{R}_2$ that runs the EUF-CMA game against DS. $\mathcal{R}_2$ samples $i \xleftarrow{\$} [N]$ uniformly at random and receives a challenger public key pk^* from the EUF-CMA challenger. It simulates $\mathbf{G}_2$ for $\mathcal{A}$, except that it embeds $\text{pk}_i := \text{pk}^*$ and sets $\text{sk}_i := i$. For every $\ell \in [N] \setminus \{i\}$, it generates $\text{pk}_\ell \xleftarrow{\$} \text{RSO}_{[\text{DS}]}.\text{KeyGen}(\ell)$ and sets $\text{sk}_\ell := \ell$. It initializes $\mathcal{K}, \mathcal{C}$, and $\mathcal{H}$, constructs $\text{PK} := \{\text{pk}_\ell\}_{\ell \in [N]}$, initializes an empty set $\mathcal{Q}_i \leftarrow \emptyset$, and invokes $\mathcal{A}(\text{pp}, \text{PK}; r)$.

Whenever $\mathcal{A}$ queries the signing oracle for signer i on (m, e, τ) , if $e \in \mathcal{Q}_i$, $\mathcal{R}_2$ returns $\perp$. Otherwise, it updates $\mathcal{Q}_i \leftarrow \mathcal{Q}_i \cup \{e\}$, forwards the message $(\text{m}, e, (\text{PK}, \tau))$ to the EUF-CMA challenger's signing oracle, returns the resulting signature to $\mathcal{A}$, and updates $\mathcal{H}[(\text{m}, e)]$. All signing queries for the remaining signers and all corruption queries are answered as in $\mathbf{G}_2$. If $\mathcal{A}$ corrupts signer i , $\mathcal{R}_2$ aborts.

When $\mathcal{A}$ outputs its valid forgery $(\text{m}^*, e^*, \tau^*, \sigma^*)$, $\mathcal{R}_2$ uses the fixed random coins r to run the extractor $(w_1, \dots, w_{\tau^*}) \leftarrow \mathcal{E}(\text{crs}, r)$. If $\mathbf{E}_4$ occurs and i is one of the indices such that $i \notin \mathcal{H}[(\text{m}^*, e^*)] \cup \mathcal{C}$ and there exists $j \in [\tau^*]$ with $w_j = (\text{pk}_i, \sigma_j^*)$, then

$$\text{DS.Verify}(\text{pk}_i, (\text{m}^*, e^*, (\text{PK}, j)), \sigma_j^*) = 1.$$

Since $i \notin \mathcal{H}[(\text{m}^*, e^*)]$, $\mathcal{R}_2$ never queried its EUF-CMA oracle for the message $(\text{m}^*, e^*, (\text{PK}, j))$. Therefore, $\mathcal{R}_2$ outputs $((\text{m}^*, e^*, (\text{PK}, j)), \sigma_j^*)$ as a valid forgery.

$\mathcal{R}_2$ guesses such an index i with probability at least $1/N$. Hence,

$$\text{Adv}_{\text{DS}}^{\text{EUF-CMA}}(\mathcal{R}_2) \geq \frac{\Pr[\mathbf{E}_4]}{N},$$

hence

$$\Pr[\mathbf{E}_4] \leq N \cdot \varepsilon_{\text{euf}}.$$

□

Per this claim, we have

$$|\Pr[\mathbf{G}_2 = 1] - \Pr[\mathbf{G}_1 = 1]| \leq \Pr[\mathbf{E}_4] \leq N \cdot \varepsilon_{\text{euf}}.$$

Finally, we bound the probability that the adversary wins $\mathbf{G}_2$. By the winning conditions of the game, if $\mathcal{A}$ wins, it outputs a valid forgery with depth

$$\tau^* > |\mathcal{H}[(\text{m}^*, e^*)] \cup \mathcal{C}|.$$

Because $\mathbf{G}_2$ did not abort, the extracted witness sequence $(w_1, \dots, w_{\tau^*}) \leftarrow \mathcal{E}(\text{crs}, r)$ is valid and contains no signatures generated under a public key pk_i such that $i \notin \mathcal{H}[(\text{m}^*, e^*)] \cup \mathcal{C}$. Therefore, for all $j \in [\tau^*]$, the extracted key pk_j^* must belong to the set

$$\{\text{pk}_i \mid i \in \mathcal{H}[(\text{m}^*, e^*)] \cup \mathcal{C}\}.$$

Since the sequence contains

$$\tau^* > |\mathcal{H}[(m^*, e^*)] \cup \mathcal{C}|$$

valid signatures, there must exist two distinct indices $i, j \in [\tau^*]$ ($i \neq j$) such that $\text{pk}_i^* = \text{pk}_j^*$ by the pigeonhole principle. This, however, violates the one-shot security of the underlying $\text{RSO}_{[\text{DS}]}$. We show this formally in the following claim.

Claim 7. *If $\text{RSO}_{[\text{DS}]}$ is OS secure, then*

$$\Pr[\mathbf{G}_2 = 1] \leq N \cdot \varepsilon_{\text{euf}}.$$

Proof. We construct a reduction $\mathcal{R}_3$ that plays the game $\text{OS}_{\text{RSO}_{[\text{DS}]}}^{\mathcal{R}_3}$. $\mathcal{R}_3$ gets oracle access to $\text{RSO}_{[\text{DS}]}.\text{KeyGen}(\cdot)$ and $\text{RSO}_{[\text{DS}]}.\text{Sign}(\cdot)$.

$\mathcal{R}_3$ sets up the UMS public parameters $\text{pp} \leftarrow^{\$} \text{UMS.Setup}(1^\lambda, N)$, initializes $\text{PK} \leftarrow \emptyset$, $\mathcal{C} \leftarrow \emptyset$, $\mathcal{K}[\cdot] \leftarrow \emptyset$, and $\mathcal{H}[\cdot] \leftarrow \emptyset$. For every $i \in [N]$, it generates $\text{pk}_i \leftarrow^{\$} \text{RSO}_{[\text{DS}]}.\text{KeyGen}(i)$, sets $\text{sk}_i := i$ and $\mathcal{K}[i] := (\text{pk}_i, \text{sk}_i)$, and updates $\text{PK} \leftarrow \text{PK} \cup \{\text{pk}_i\}$. It then invokes $\mathcal{A}(\text{pp}, \text{PK})$.

Whenever $\mathcal{A}$ queries the signing oracle on (i, m, e, τ) , $\mathcal{R}_3$ forwards the underlying signing request $(i, m, e, (\text{PK}, \tau))$ to its challenger's Sign oracle, updates $\mathcal{H}[(m, e)] \leftarrow \mathcal{H}[(m, e)] \cup \{i\}$ if the resulting signature is not $\perp$, and returns the response to $\mathcal{A}$. Whenever $\mathcal{A}$ corrupts a signer i , $\mathcal{R}_3$ updates $\mathcal{C} \leftarrow \mathcal{C} \cup \{i\}$ and returns $\text{sk}_i = i$. All subsequent invocations of the underlying $\text{RSO}_{[\text{DS}]}.\text{Sign}$ oracle are forwarded to the challenger.

Eventually, $\mathcal{A}$ outputs its valid forgery $(m^*, e^*, \tau^*, \sigma^*)$. $\mathcal{R}_3$ uses the fixed random coins r to run the extractor $(w_1, \dots, w_{\tau^*}) \leftarrow \mathcal{E}(\text{crs}, r)$. As established above, if $\mathcal{A}$ wins $\mathbf{G}_2$, the extracted chain must contain two witnesses $w_i = (\text{pk}^*, \sigma_i^*)$ and $w_j = (\text{pk}^*, \sigma_j^*)$ at distinct indices $i \neq j$ under the same public key pk^* . By the verification conditions of the chain, it must hold that

$$\text{DS.Verify}(\text{pk}^*, (m^*, e^*, (\text{PK}, i)), \sigma_i^*) = \text{DS.Verify}(\text{pk}^*, (m^*, e^*, (\text{PK}, j)), \sigma_j^*) = 1.$$

$\mathcal{R}_3$ sets $\text{aux}_1 := (\text{PK}, i)$ and $\text{aux}_2 := (\text{PK}, j)$, and outputs the tuple $(\text{pk}^*, e^*, m^*, \text{aux}_1, \sigma_i^*, m^*, \text{aux}_2, \sigma_j^*)$ to its challenger. Because $i \neq j$, it follows that $\text{aux}_1 \neq \text{aux}_2$, hence $(m^*, \text{aux}_1) \neq (m^*, \text{aux}_2)$. Additionally, it holds that

$$\text{DS.Verify}(\text{pk}^*, (m^*, e^*, \text{aux}_1), \sigma_i^*) = \text{DS.Verify}(\text{pk}^*, (m^*, e^*, \text{aux}_2), \sigma_j^*) = 1.$$

Furthermore, pk^* was generated through the challenger's $\text{RSO}_{[\text{DS}]}.\text{KeyGen}$ oracle and hence belongs to PK . This satisfies all winning conditions of the OS game. Since $\mathcal{R}_3$ perfectly simulates the environment for $\mathcal{A}$ and constructs a valid winning tuple whenever $\mathcal{A}$ wins $\mathbf{G}_2$, we have:

$$\Pr[\mathbf{G}_2 = 1] \leq \text{Adv}_{\text{RSO}_{[\text{DS}]}}^{\text{OS}}(\mathcal{R}_3) \leq N \cdot \varepsilon_{\text{euf}}.$$

□

Overall, we have

$$\begin{aligned}
\mathbf{Adv}_{\text{UMS}}^{\text{UNQ}}(\mathcal{A}) &= \mathbf{Adv}^{\mathbf{G}_0}(\mathcal{A}) \\
&\leq \mathbf{Adv}^{\mathbf{G}_1}(\mathcal{A}) + \varepsilon_{ks} \\
&\leq \mathbf{Adv}^{\mathbf{G}_2}(\mathcal{A}) + N \cdot \varepsilon_{euf} + \varepsilon_{ks} \\
&\leq 2 \cdot N \cdot \varepsilon_{euf} + \varepsilon_{ks}.
\end{aligned}$$

□

Theorem 7 (Non-Equivocation of UMS). *Let UMS be the unique multisignature scheme defined in construction 1 for a number of signers N , and let $\text{RSO}_{[\text{DS}]}$ be the restrictive signing oracle used in the construction with respect to a digital signature scheme DS. If $\text{RSO}_{[\text{DS}]}$ is OS secure, then UMS is $(t_{eq}, \varepsilon_{eq})$ -non-equivocation secure, where $\varepsilon_{eq} \leq N \cdot \varepsilon_{euf}$ and $t_{eq} \approx t_{euf}$.*

Proof. Let $\mathcal{A}$ be an adversary playing the game $\text{Eq}_{\text{UMS}}^{\mathcal{A}}$. We construct a reduction $\mathcal{R}_4$ that plays the game $\text{OS}_{\text{RSO}_{[\text{DS}]}}^{\mathcal{R}_4}$. $\mathcal{R}_4$ receives oracle access to $\text{RSO}_{[\text{DS}]}.\text{KeyGen}(\cdot)$ and $\text{RSO}_{[\text{DS}]}.\text{Sign}(\cdot)$.

$\mathcal{R}_4$ sets up the UMS public parameters $\text{pp} \leftarrow^{\$} \text{UMS.Setup}(1^\lambda, N)$, initializes an empty public key set $\text{PK} \leftarrow \emptyset$ and a counter $i := 0$, and invokes $\mathcal{A}(\text{pp})$. Whenever $\mathcal{A}$ makes a UMS.KeyGen request, $\mathcal{R}_4$ increments $i := i + 1$, forwards i to its challenger's $\text{RSO}_{[\text{DS}]}.\text{KeyGen}$ oracle to obtain pk_i , updates $\text{PK} \leftarrow \text{PK} \cup \{\text{pk}_i\}$, and returns (pk_i, i) to $\mathcal{A}$. Whenever $\mathcal{A}$ makes a signing-oracle query (i, m, e, τ) , $\mathcal{R}_4$ forwards $(i, \text{m}, e, (\text{PK}, \tau))$ to its challenger's $\text{RSO}_{[\text{DS}]}.\text{Sign}$ oracle and returns its response to $\mathcal{A}$.

Eventually, $\mathcal{A}$ outputs its winning tuple $(\text{pk}, \text{m}_1, \tau_1, \sigma_1, \text{m}_2, \tau_2, \sigma_2, e)$. By the winning condition of the game Eq, it must hold that $\text{m}_1 \neq \text{m}_2$, $\text{UMS.Verify}(\text{PK}, \text{pk}, \text{m}_1, e, \tau_1, \sigma_1) = \text{UMS.Verify}(\text{PK}, \text{pk}, \text{m}_2, e, \tau_2, \sigma_2) = 1$, and $\text{pk} \in \text{PK}$.

$\mathcal{R}_4$ sets $\text{aux}_i := (\text{PK}, \tau_i)$ for $i \in \{1, 2\}$ and outputs the tuple $(\text{pk}, e, \text{m}_1, \text{aux}_1, \sigma_1, \text{m}_2, \text{aux}_2, \sigma_2)$ to its challenger. Because $\text{m}_1 \neq \text{m}_2$, it follows that $(\text{m}_1, \text{aux}_1) \neq (\text{m}_2, \text{aux}_2)$ and it holds that $\text{DS.Verify}(\text{pk}, (\text{m}_1, e, \text{aux}_1), \sigma_1) = \text{DS.Verify}(\text{pk}, (\text{m}_2, e, \text{aux}_2), \sigma_2) = 1$. This satisfies the winning conditions of the OS game. Since $\mathcal{R}_4$ perfectly simulates the game for $\mathcal{A}$ and constructs a valid winning tuple whenever $\mathcal{A}$ wins $\text{Eq}_{\text{UMS}}^{\mathcal{A}}$, we have:

$$\mathbf{Adv}_{\text{UMS}}^{\text{Eq}}(\mathcal{A}) \leq \mathbf{Adv}_{\text{RSO}_{[\text{DS}]}}^{\text{OS}}(\mathcal{R}_4) \leq N \cdot \varepsilon_{euf}.$$

□

Claim 1. *There exists a reduction $\mathcal{R}$ such that $\Pr[\text{E}_1] \leq \Pr[\text{Eq}_{\text{UMS}}^{\mathcal{R}} = 1] \leq \varepsilon_{eq}$.*

Proof. Let $\mathcal{A}$ be an adaptive adversary against the protocol in figure 3. We construct a reduction $\mathcal{R}$ that plays the game $\text{Eq}_{\text{UMS}}^{\mathcal{R}}$.

$\mathcal{R}$ receives the public parameters pp and makes n UMS.KeyGen requests to obtain key pairs $(\text{pk}_i, \text{sk}_i)$ for all $i \in [n]$. It sets

$$\text{PK} := \{\text{pk}_i\}_{i \in [n]}$$

and assigns $\mathbf{pk}_1$ to the sender P_1 .

$\mathcal{R}$ simulates the protocol execution for $\mathcal{A}$. Whenever a simulated party invokes $\text{UMS.Sign}(\text{PK}, \mathbf{sk}_i, \mathbf{m}, e, \tau)$, $\mathcal{R}$ queries its signing oracle on $(\mathbf{sk}_i, \mathbf{m}, e, \tau)$ and uses the returned signature. Signing requests made after a party is corrupted are forwarded analogously.

All honest-party computations are performed according to figure 3. Whenever $\mathcal{A}$ adaptively corrupts a party, $\mathcal{R}$ provides its local state and lets $\mathcal{A}$ control the party for the remainder of the execution.

If E_1 occurs, then there exist two distinct values $v \neq v'$ and signatures σ_1, σ'_1 such that

$$\text{UMS.Verify}(\text{PK}, \mathbf{pk}_1, v, e, 1, \sigma_1) = \text{UMS.Verify}(\text{PK}, \mathbf{pk}_1, v', e, 1, \sigma'_1) = 1.$$

$\mathcal{R}$ outputs

$$(\mathbf{pk}_1, v, 1, \sigma_1, v', 1, \sigma'_1, e)$$

to its challenger.

Since $v \neq v'$, $\mathbf{pk}_1 \in \text{PK}$, and both signatures verify under $\mathbf{pk}_1$ in the same epoch, this tuple satisfies all winning conditions of the non-equivocation game. Therefore,

$$\Pr[E_1] \leq \Pr[\text{Eq}_{\text{UMS}}^{\mathcal{R}} = 1] \leq \varepsilon_{eq}.$$

□

Claim 2. *There exists a reduction $\mathcal{R}$ such that $\Pr[E_2] \leq \Pr[\text{UNQ}_{\text{UMS}}^{\mathcal{R}}(\mathbf{t}, n) = 1] \leq \varepsilon_{unq}$.*

Proof. Let $\mathcal{A}$ be an adaptive adversary against the protocol in figure 3. We construct a reduction $\mathcal{R}$ that plays the game $\text{UNQ}_{\text{UMS}}^{\mathcal{R}}(\mathbf{t}, n)$.

During the setup phase, $\mathcal{R}$ receives the public parameters $\mathbf{pp}$ and the public key set PK . It invokes $\mathcal{A}$ and simulates the protocol execution according to figure 3.

Whenever an honest party P_i generates a signature share on a value v in epoch e at depth r , $\mathcal{R}$ queries its signing oracle on (i, v, e, r) and returns the resulting signature to the simulated party. Whenever $\mathcal{A}$ corrupts P_i , $\mathcal{R}$ queries its corruption oracle on i and thereafter lets $\mathcal{A}$ control that party.

Suppose E_2 occurs. Then some honest party accepts a value v with an aggregate signature $\sigma_{\mathbf{t}+1}$ satisfying

$$\text{UMS.VerifyAgg}(\text{PK}, v, e, \mathbf{t} + 1, \sigma_{\mathbf{t}+1}) = 1,$$

while

$$\mathcal{H}[(v, e)] \setminus \mathcal{C} = \emptyset.$$

Hence,

$$\mathcal{H}[(v, e)] \subseteq \mathcal{C},$$

and therefore

$$|\mathcal{H}[(v, e)] \cup \mathcal{C}| = |\mathcal{C}| \leq \mathbf{t}.$$

$\mathcal{R}$ outputs

$$(v, e, \mathbf{t} + 1, \sigma_{\mathbf{t}+1})$$

to its challenger. Since

$$\mathbf{t} + 1 > |\mathcal{H}[(v, e)] \cup \mathcal{C}|,$$

and the aggregate signature verifies at depth $\mathbf{t} + 1$, $\mathcal{R}$ satisfies both winning conditions of the uniqueness game. Thus,

$$\Pr[\mathbf{E}_2] \leq \Pr[\text{UNQ}_{\text{UMS}}^{\mathcal{R}}(\mathbf{t}, n) = 1] \leq \varepsilon_{\text{unq}}.$$

□