Enabling Threshold Custody for the Lightning Network with Nested Threshold Multi-Signatures

Paul Gerhart[*1], Nadav Kohen[*2], Jesse Posner[3], and Matias Furszyfer[2]

[1] *TU Wien, Vienna, Austria*
[2] *Chaincode Labs, New York, United States*
[3] *Vora, San Francisco, United States*

Abstract

The Bitcoin Lightning Network secures hundreds of millions of dollars, yet channel endpoints rely on vulnerable single online keys. Although threshold signatures are routinely used to protect on-chain Bitcoin, no practical deployment has been possible for Lightning channels. This is because thresholdizing a Lightning party requires nesting a threshold signature scheme inside of an established two-party MuSig2 protocol without altering its nonce exchange or message flow.

In this work, we resolve this limitation by formalizing nested threshold multi-signatures, a new cryptographic primitive for thresholdizing one participant inside a multi-signature protocol. As an instance of this primitive, we present Iceberg, the first construction for nested threshold MuSig2 signatures. Iceberg enables one side of a Lightning channel to operate as a t-of-n threshold group while appearing to the counterparty as a standard MuSig2 participant. As a result, threshold custody can be deployed unilaterally on today's Lightning Network without requiring any modifications to Bitcoin, the Lightning protocol, or channel counterparties.

We prove the security of Iceberg, integrate a prototype into a production Lightning node, and benchmark its performance. Our measurements show that thresholdizing a Lightning channel incurs only modest overhead, since a threshold group tolerating one corrupted member sustains over 93% of the payment throughput of an unmodified endpoint.

Keywords. Lightning Network, Threshold Custody, Nested Threshold Multi-Signatures, MuSig2, Schnorr Signatures

1 Introduction

The Bitcoin Lightning Network [PD16], a payment-channel layer that settles Bitcoin off-chain, now carries *real money* at *institutional scale*. The value locked in its public channels reached a record of roughly 5,600 BTC, about $490 million, in December 2025 [Bit25]. This value is held across on the order of 15,000 public nodes and tens of thousands of channels [1ML25], and because most channels are private these figures are *lower bounds*. Major exchanges including Bitfinex, OKX, Kraken, Binance, and Coinbase now settle customer deposits and withdrawals over Lightning [lne25], which is cheaper and faster than settling on-chain and increasingly what customers expect.

[*] Authors contributed equally to this work.

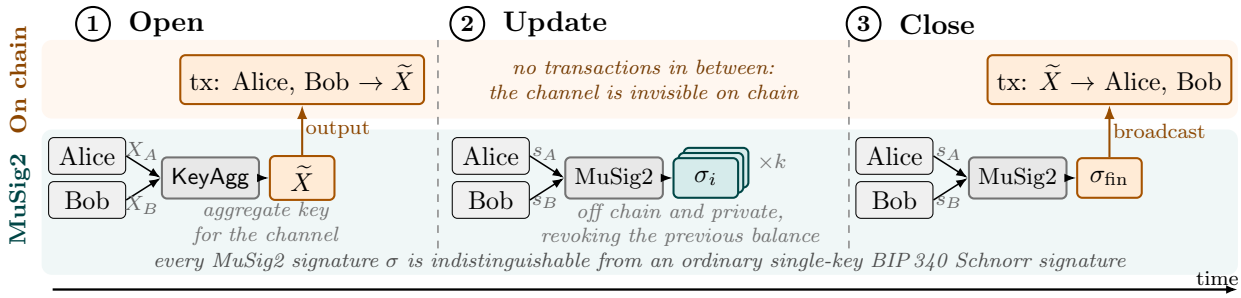

Figure 1: **A Lightning channel has one aggregate key and a sequence of MuSig2 signatures.** Alice and Bob lock funds under one aggregate MuSig2 key $\widetilde{X} = \mathsf{KeyAgg}(\{X_A, X_B\})$, which the funding transaction Alice, Bob $\to \widetilde{X}$ carries in its output. Every balance update and the close is a single MuSig2 2-of-2 signature formed from the partial signatures s_A and s_B. Only the funding and the closing transaction $\widetilde{X} \to$ Alice, Bob reach the chain; everything in between stays private and off-chain.

However, every payment-channel in the Lightning Network rests on a fragile foundation. On each side of the channel, there is a single online key holding the channel endpoint's funds, and a single key is a single point of failure. Relying on a single key can fail in two ways. The key can be *compromised*: in 2019, attackers drained 7,000 BTC from a single exchange hot wallet in one transaction that passed every security check [Bin19]. Or the key can be *lost*: hundreds of millions of dollars sit permanently beyond reach on one encrypted drive whose owner forgot the password [Pop21]. Neither is recoverable, and both kinds of failure recur throughout Bitcoin's history. This is already a serious problem for Bitcoin custody. In Lightning, it becomes even more acute, since keeping a channel live and routing payments requires the key to stay online, ruling out the protection of cold storage.

On-chain, a single signing key being a single point of failure is largely a solved problem. Threshold signing [Des94] (t-of-n) distributes signing authority across n parties so that any t of them can sign while no single key corruption is fatal, because the compromise of up to $t-1$ shares does not allow forging signatures, and similarly the loss of up to $n-t$ shares is survivable, as long as t shares remain. Threshold custody is standard practice for institutions and serious self-custodians [LN18, SP21]. Bitcoin supports it natively in Script by listing several keys and requiring a quorum of signatures [And11]. Modern Schnorr-based schemes such as MuSig2 [NRS21] (specified in BIP 327 [NRJ22]) and FROST [KG20a] achieve the same function more compactly and privately, collapsing the whole group into one aggregate key and signature. However, the protection that on-chain custody takes *for granted is missing* for Lightning channels.

To see why, consider how a Lightning channel works (Figure 1). Two parties, say Alice and Bob, open a channel by locking funds into a single output they jointly control, namely a 2-of-2 that, on modern taproot channels [Osu22], takes the form of one aggregate MuSig2 key [NRS21]. Spending it requires both parties to sign, so neither can move the funds alone. Because it is an aggregate key, the output is indistinguishable on-chain from an ordinary single-key address. For this reason, the rest of the world cannot even tell a channel is open. From then on, Alice and Bob update their shared balance privately and off-chain, each update re-signing a transaction spending that funding output using MuSig2. When they decide to close the channel, they broadcast the latest agreed state, spending the funding output and distributing the funds according to the final channel balance, either cooperatively or, if necessary, unilaterally.

Alice's funding key is one half of that 2-of-2 signing key, and on her side, *an online single point of*

failure she would like to distribute. As of today, Alice *cannot do so*, as the channel's structure stands in the way at multiple stages. On the legacy channels Bitcoin first supported [Lig25], the 2-of-2 funding output is a Bitcoin Script, and Alice could in principle widen her half into a threshold script, but not alone, since it would require changing the Lightning protocol both peers run. Furthermore, it would force a non-standard script to be negotiated at every channel open, enlarge every transaction and its fee, and leave an on-chain footprint that advertises her key-management setup, an undesirable privacy leak. Taproot channels [WNT20, Osu22], which the ecosystem is now adopting for their privacy and smaller transactions, further complicate matters for would-be non-standard script users, since the two sides aggregate into a single MuSig2 key spent by one ordinary signature, and MuSig2 gives each party just one key to contribute, with nothing visible to widen. Either way, with Lightning moving to taproot channels, a threshold Alice can actually use must live inside one ordinary-looking MuSig2 key. In addition, as of today, Alice can only use a threshold if the changes she makes require no changes from Bob, the protocol, or the chain. Therefore, thresholdizing Alice's side of the channel calls for a scheme that creates threshold partial signatures within an outer MuSig2 session, which is the new primitive we introduce in this work formalized as a *nested threshold multi-signature*.

Finding constructions for a nested threshold multi-signature has been a long-standing open problem for the Bitcoin community [Ste19b, Ste19a, KC23, Sed19, Shl20, Zmn25]. The main reason why finding a nested threshold multi-signature for MuSig2 is hard is due to the nonce R of a Schnorr signature (R, s) that MuSig2 computes. Lightning fixes the session nonce R a full round-trip before the transaction it will sign exists [Osu22], and once Alice has sent her part of it to Bob she cannot revise it. This is fine if Alice is a single signer, but if she is thresholdized, her signer set may change in between rounds. Whichever signing set of Alice's devices is online when the transaction arrives has to complete the signature under a nonce that was fixed before the transaction existed, and possibly chosen by a different signing set. In the literature, this issue is solved using stateless threshold signatures such as Arctic [KG25] using deterministic nonces and replicated secret sharing [ISN87, CDI05]. Unfortunately, using stateless signatures does not work in the Lightning setting for at least two reasons. First, stateless Schnorr signatures (including Arctic) typically derive the nonce from the message to be signed and the current signing session to avoid key leakage that would arise if the two different messages are signed with the same randomness. Yet, in the Lightning setting Alice knows neither the message to be signed nor the final signing session when she needs to issue her part of the nonce R. Second, if the signers in Lightning would be stateless, they could be convinced to sign an outdated channel state which would make them subject to Lightning's punishment mechanisms. Hence, any solution for thresholdized Lightning must at least safely carry the latest channel state to be meaningful. We discuss the restrictions on nested threshold multi-signatures imposed by the structure of Lightning channels alongside our mitigations of these in Section 2.

Given that threshold custody for Lightning is highly desirable for the Bitcoin community and yet not solved as of today we ask the question:

Can threshold custody be deployed in today's Lightning Network?

1.1 Our Contribution

In this paper, we answer this question in the affirmative.

- We introduce the notion of *nested threshold multi-signatures*, a novel cryptographic primitive designed to t-of-n threshold-sign partial signatures of a given multi-signature scheme (Section 4).

- We present the first nested threshold multi-signature Iceberg, that allows t-of-n threshold sign partial MuSig2 signatures [NRS21, NRJ22] (Section 5).
- We prove the unforgeability of Iceberg (Section 6) with optimal signing thresholds dictated by the Lightning environment. In particular, unforgeability holds if an adversary compromises up to $t-1$ signing key shares, and any $2t-1$ signing shares are sufficient to produce a valid signature. (Cf. Section 2 for a discussion on the optimality of these parameters).
- We show that Iceberg is a drop-in replacement for MuSig2 at a Lightning endpoint. Its first-round output is exactly the message a single MuSig2 participant would send, and the channel it funds is an ordinary single-key Taproot output. In other words, an Iceberg endpoint is compatible with the Lightning protocol as deployed. As a result, threshold custody can be added unilaterally, with no change to Bitcoin, to the protocol, or to the counterparty.
- We integrate Iceberg into a production Lightning node and measure it over taproot channels (Section 7). Making one endpoint a 2-of-4 group adds 3.8 ms per payment, which is 6.7% of the work a payment already costs. That group only saturates an additional core at 260 payments per second.

1.2 Related Work

Threshold signatures [Des94, DDFY94, GRJK00, Sho00, GJKR96, GJKR07, GJKR03] have become a practical solution for distributed custody of digital assets, with recent constructions providing efficient signing protocols for Schnorr [Sch90] signatures [KG20a, BCK+22, CGRS23, CKM23, CKK+25, Che25, BDLR25b, BDLR25a, GLRS26, KRT24, NRT26, BCL+25] and ECDSA [GGN16, GG18, LN18, DKLs19, DOK+20, GKSŚ20, CGG+20, CCL+20, DJN+20, YCX21, ANO+22]. However, existing threshold schemes assume that the threshold group constitutes the complete signer. Lightning creates a fundamentally different setting, in which only one participant of an existing MuSig2 execution is thresholdized, while the remaining participant and the overall signing protocol remain unchanged. This requires threshold signing to operate inside an existing multi-signature protocol, rather than replacing it.

None of the existing threshold constructions support this setting, since they either bind their nonce generation to the signing set, incur high round complexity, or require the message to be known at nonce creation time (cf. Table 1). We restrict this section to the schemes most relevant for deploying threshold custody in Lightning and refer to [CKK+25] for a broader survey of Schnorr threshold and multi-signature schemes.

The FROST family [KG20a, BCK+22, CGRS23, CKK+25] builds its aggregate nonce from the contributions of the quorum that signs, which binds the nonce to that quorum and rules it out for a nested endpoint. Stateless schemes [GKMN21, KOR23, FYZ+25] remove the binding by deriving the nonce deterministically, and pay for it by taking the message as an input, and Arctic [KG25] is the most recent of them, and [NRSW20a, GLRS26] meet the same obstacle. As discussed in the introduction, fully stateless signers cannot be used for thresholding Lightning, as signing an old channel state would lead to punishment of the signer set. Yet, Iceberg follows Arctic's pattern of deterministic nonces over replicated publicly-verifiable secret sharing, but seeds the derivation with the state of the channel rather than with the message and the signer set resolving this issue (Section 2).

Iceberg operates inside a multi-signature protocol, making the multi-signature literature equally relevant to our setting. Bellare and Neven [BN06] introduced the first Schnorr multi-signature

	FROST [KG20a]	Arctic [KG25]	MuSig-DN [NRSW20a]	Seurin [Seu24]	**Iceberg**
Nestable in MuSig2	✗	✗	✗	✓	✓
Dynamic online signers	✗	✓	✗	✓	✓
State-aware	✗	✗	✗	✓	✓
Deterministic nonces	✗	✓	✓	✓	✓
Honest majority†	✗	✓	–	✓	✓
Complete, proven construction	✓	✓	✓	✗	✓
Deployable in today's Lightning.	✗	✗	✗	✗	✓

Table 1: Feature comparison of Iceberg with most relevant threshold-Schnorr and nesting schemes (✓ yes, ✗ no). † a ✓ denotes an assumed honest majority, and MuSig-DN is an n-of-n multi-signature, so the axis does not apply (–).

in the plain public-key model, and MuSig [MPSW19] added key aggregation, allowing a group of signers to appear as a single signer on chain. Achieving two-round Schnorr multi-signatures proved challenging, and Drijvers et al. [DEF+19] showed that natural constructions are vulnerable, while MuSig2 [NRS21], standardized as BIP 327 [NRJ22], and DWMS [AB21] achieve two rounds by providing each signer with $\nu = 2$ nonce commitments. MuSig-DN [NRSW20a] instead derives nonces deterministically with public verifiability, at the cost of a zero-knowledge proof per signature. Subsequent works further improve the analysis or avoid rewinding-based proofs [DOTT21, BD21, TZ23, PW23, PW24, BW25]. Lightning uses MuSig2 as its signing protocol and does not permit replacing it with an alternative multi-signature scheme. Therefore, Iceberg must operate within the MuSig2 execution itself rather than a different underlying construction.

Placing a threshold on one side of a MuSig2 channel was sketched informally by Seurin [Seu24] in an unpublished note. The first formal treatment of nesting a multi-signature scheme into MuSig2 was done by Kohen [Koh26]. However Kohen nests a multi-signature (n-out-of-n) rather than a threshold signature (t-out-of-n) into MuSig2. Since in this setting all signers are required to participate in every signing session, the signer set remains fixed and the availability and compromise challenges that motivate Lightning custody are not addressed.

2 Technical Overview

A Lightning channel between two parties locks their coins in a single on-chain output, a *funding output* guarded by a 2-of-2 multisignature. The two parties then transact without touching that output again, co-signing a sequence of *commitment transactions* that each encode the current balance, where each new one revokes its predecessor. Should a party try to settle the channel on an old, revoked state, Lightning's *penalty mechanism* lets the counterparty claim all of the channel's funds, and that threat keeps both sides committed to the latest balance. Bitcoin's Taproot upgrade lets this 2-of-2 be realised with MuSig2 [NRS21], a two-round multi-signature scheme that aggregates several public keys into a single one. The two funding keys aggregate to one key $\widetilde{X}$, and any spend both parties agree to, in particular a cooperative channel close, is a single Schnorr signature under

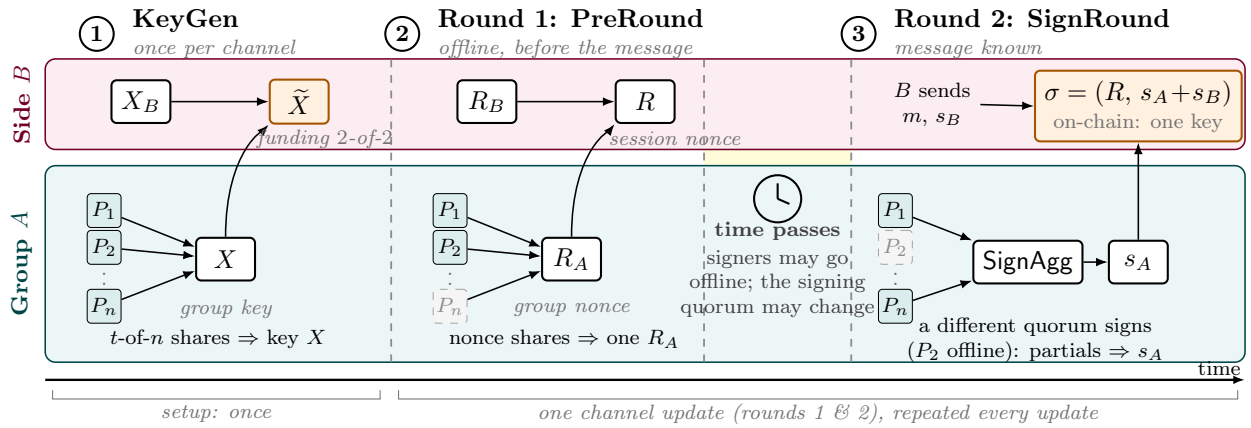

Figure 2: A thresholdized Lightning channel. **(1) KeyGen** (once per channel) fixes the group key X from replicated VPSS shares of $P_1, \ldots, P_n$; X fills one slot of the funding 2-of-2 $\widetilde{X} = \mathsf{KeyAgg}(\{X, X_B\})$, on chain a single key-spend. Each update then runs two rounds: **(2) PreRound**, offline and message-free, turns the session identifier *sid* into one group nonce R_A (the same for every quorum) that joins B's R_B into R; **(3) SignRound**, once m is known, has the online quorum output partials that $\mathsf{SignAgg}$ combines into s_A, yielding the MuSig2 signature $\sigma = (R, s_A + s_B)$ under $\widetilde{X}$. Between the rounds signers may go offline, so the quorum that precommits the nonce need not be the one that signs.

$\widetilde{X}$. On chain this is a key-path spend indistinguishable from an ordinary single-key payment, since the verifier sees one key and one signature, with no trace of the two parties behind $\widetilde{X}$.

Our goal is to turn one side of the channel from a single signer into a group of signers. Instead of a single funding key, we want n parties, a user's devices or an exchange's servers, of which any $2t-1$ suffice to sign and which stays secure while fewer than t are compromised. Throughout this overview, side A is this group and side B is the channel counterparty.

Achieving this would be straightforward if we could redesign how Lightning channels work. Our constraint is that we retrofit it into today's channels, so the scheme can be adopted with no change on the counterparty's side. This means B keeps running plain two-party MuSig2 and the chain keeps verifying one Schnorr signature under $\widetilde{X}$. The group must therefore occupy the key slot A holds in the funding 2-of-2 and, at each round, present exactly what a single MuSig2 signer would.

Figure 2 shows the channel this produces, with side B unchanged and side A a group, and tracks what B and the chain see at each stage. At key generation the group's n parties present one group key X, which fills side A's slot and aggregates with X_B into the funding key $\widetilde{X}$ just as two ordinary keys would. In the first round the group presents one nonce R_A, fixed before the message exists, which aggregates with R_B into the session nonce R. In the second round the group presents one partial signature s_A, which adds to s_B to give the final signature $\sigma = (R, s_A + s_B)$, on chain a single-key spend. The group presents one key, one nonce and one partial signature, so at no point can B or the chain tell side A from a lone MuSig2 signer.

Our setting. The Lightning protocol and the practical deployment context together shape what such a group can look like. Some of the following constraints may seem cherry-picked, but the rest of this section shows that they are already required of a threshold lightning construction.

(C1) Signers go offline. Keys may sit in cold storage, so an honest member is routinely absent

Threshold t	Corruptions	Signing quorum	Minimal group
2	1	3	4
3	2	5	7
4	3	7	10
5	4	9	13

Table 2: Possible Iceberg thresholds. A threshold t tolerates up to $t-1$ corruptions, needs a signing quorum of $2t-1$ online members, and requires a group of at least $n \geq 3t-2$.

rather than malicious. The quorum that begins a signing session need not be the one that finishes it, and absent members cannot simply be counted as corruptions.

(C2) Nonces precede the message. In the simple taproot channels protocol [Osu22], each side exchanges its MuSig2 nonce a full round-trip before the commitment transaction it will sign even exists. The group must therefore commit to a single nonce R_A before knowing the message m, and since the signing quorum may not yet be fixed at that point, the nonce cannot depend on who will eventually sign.

(C3) Nonces cannot change. Once R_A is sent to the counterparty it is fixed for this session. The outer MuSig2 session is unmodified BIP 327, so the group cannot supply a different nonce after the fact. Should the quorum that finishes the second round differ from the one that started the first, the nonce already committed to B must stay the same.

(C4) The group must agree on the live state. Were a corrupt minority of $t-1$ members to convince an honest one to sign a superseded commitment transaction, the counterparty would invoke the penalty mechanism and claim all channel funds. The group must therefore reach consensus on the current commitment number before signing.

(C5) Signer sets are small. A typical group is a user's few devices or the handful of keys an exchange distributes across staff and servers, and many more would be operationally unwieldy. We therefore expect n to be at most a dozen.

Offline signers and small signer sets are practical assumptions rather than protocol constraints, but they are realistic for the deployments we envision. Consensus on the live state carries an immediate consequence for the group's parameters. Agreeing with up to $t-1$ faulty members is Byzantine agreement, which requires a strict honest supermajority and thus $n \geq 3t-2$. Table 2 lists, for each Iceberg threshold up to a group of twelve, the corruptions it tolerates, the online quorum it needs, and the smallest group that supports it.

Strawman approaches. Now that we have established the setting, let us explore why existing threshold and multi-signature schemes do not present solutions to the task of constructing a thresholdized Lightning signer. We consider two strawman approaches, MuSig2 [NRS21] and FROST [KG20a].

MuSig2 seems like the natural starting point, since the channel already uses it. And indeed one MuSig2 instance can be nested inside another, since the inner group's aggregate key is itself an ordinary public key and can occupy side A's slot while the group runs MuSig2 internally [Koh26]. This does not solve our problem, though, because nesting only splits side A into an n-of-n group,

where every one of the n parties must be online for each signing. Tolerating offline signers is one of our goals as is improving key management, so a multi-signature is not enough, and we need a genuine t-of-n threshold.

This calls for a threshold signature, which produces a signature from any t of the n parties and lets the absent ones stay offline. We try FROST [KG20a], the standard such scheme, to see where threshold Schnorr signatures break when nested into Lightning. To sign in FROST, each signer commits two nonces (D_i, E_i). A signing quorum C forms the aggregate nonce $R = D\,E^b$ from the products D, E of these commitments and a binding factor b that hashes the quorum C and the message m, and the partial signatures are combined by Lagrange interpolation over C. The nonce R therefore depends on which members form the quorum C, and this is not specific to FROST. Most threshold Schnorr schemes relying on Shamir secret sharing [Sha79] build the aggregate nonce from fresh per-signer contributions R_i whose discrete logarithm only the contributing signer knows, so the nonce is tied to the particular set of signers that produced it. A different quorum brings different contributions and reconstructs a different R [CKM23, KG20b]. The group commits the single nonce R_A to the counterparty in the first round, but in Lightning (and more generally in most threshold-nested signing settings) a different quorum may finish the second round, and it would reconstruct a different R_A than the one already promised to B.

Design decision 1, replicated secret sharing. FROST and its relatives [KG20b] violate two requirements that nesting places on the group's nonce. First, the nonce must be the same whichever quorum produces it, or the second round would contradict the R_A already committed in the first (constraint C3). Second, it must remain producible by a member that was offline in the first round, since the finishing quorum need not be the starting one (constraint C1). With nonces committed or shared in the first round, an absent member holds nothing and cannot sign in the second. We meet both by deriving the nonce *deterministically* from *replicated* shares.

Replicated secret sharing [ISN87] splits the secret into one share $\phi_{\mathbf{a}}$ for each subset $\mathbf{a}$ of $t-1$ members, distributed to every member *outside* $\mathbf{a}$, so that the secret is their sum $\mathsf{sk} = \sum_{\mathbf{a}} \phi_{\mathbf{a}}$. Any subset of $t-1$ members, $\mathbf{a}$, miss precisely the share indexed by $\mathbf{a}$ and learn nothing about the aggregate secret, while any larger set holds every summand $\phi_{\mathbf{a}}$. Because the secret is this one fixed sum and every qualifying quorum holds all of its terms, reconstruction does not depend on who is present, unlike the per-signer aggregates of FROST or Sparkle that are assembled from whoever happens to sign.

Instead of sampling a fresh nonce and sharing it in the first round, each member derives its contribution from its long-lived shares through a fixed pseudorandom function, evaluated on a per-session input, in the manner of pseudorandom secret sharing [CDI05]. Nothing is exchanged in the first round, so a member absent then can still recompute its contribution in the second from shares it has held since key generation [CDI05].

Put together, every quorum feeds the same shares into the same function and (given the same seed) obtains the same nonce, available to any qualifying set at any time.

The primary trade-off of this approach is storage complexity. Each member holds one share per $(t-1)$-subset it lies outside of, resulting in $\binom{n-1}{t-1}$ shares. As the number of shares grows quickly with n, replicated sharing is dismissed as impractical for large deployments. In our setting it is not, because we expect signer sets to be small (constraint C5), so the share count stays modest.

Design decision 2, unique pre-message nonces. A deterministic nonce is a function of its seed, so every nonce requirement is a seed requirement. A deterministic nonce is safe only if it is never reused across two different messages. If a group were to sign two messages under one nonce R, the two partial signatures $s = r + c\,x$ and $s' = r + c'\,x$ on challenges $c \neq c'$ would, by the two-special-soundness of Schnorr, expose the signing key as $x = (s - s')/(c - c')$. Therefore, the seed must vary whenever the challenge does.

The literature derives the nonce from the message itself [KG25], so an identical message gives an identical nonce and signing the same message twice merely repeats one signature rather than exposing a key. We cannot follow this, because the nonce is fixed in the first round, before the message, the commitment transaction to be signed, exists (constraint C2). Instead we use our Lightning protocol setting's state as part of our seed. Every commitment transaction carries a commitment number that is unique and strictly increasing over the channel's life, and it is known before the transaction is assembled. We seed the nonce with this number as a part of the session identifier *sid*, giving a nonce that is fixed before the message yet still varies from one channel state to the next, at no added cost.

Since the derivation depends on *sid* alone, two commitment transactions that carry the same commitment number share a nonce, and the key leakage above returns.

Design decision 3, consensus on the live state. The gap left by deterministic nonces is that one *sid* may be signed under two messages, as nothing in the derivation prevents the group from being driven to sign two different commitment transactions under the same commitment number. If that happens, the two partial signatures share the nonce R but carry distinct challenges, and that leaks the signing key.

Nesting makes this worse, because the outer session runs MuSig2 with $\nu = 2$ nonces per signer. Two signatures on one *sid* do not merely repeat but place the attacker in the concurrent ROS setting [BLL+21] that $\nu = 2$ exists to resist, with the group as the target. The defence against that setting is more nonces, and we cannot add them, since the outer session is unmodified BIP 327. We therefore cannot prevent the reuse within our construction.

Instead, we use a mechanism any thresholdized channel already relies on. The group must agree on the current commitment number regardless of our scheme (constraint C4), since signing a superseded state lets the counterparty invoke the penalty mechanism and seize the channel. An honest member therefore signs at most one message under any one *sid*. The bound $n \geq 3t-2$ secures an honest supermajority, which keeps a corrupt minority from forcing a second one, preventing nonce reuse. If an implementation does want to retry a failed signing attempt at the top-level MuSig2, it has to agree on the protocol messages that change the *sid*.

The channel already requires this agreement, so this method of keeping our deterministic nonces safe comes with no additional cost for our construction.

Iceberg. Putting these three design choices together yields a t-of-n signing protocol that interacts with the outer session like a single MuSig2 participant. At key generation, the group's secret key sk is shared among the members with replicated secret sharing [ISN87]. The key is split additively into $\binom{n}{t-1}$ summands $\mathsf{sk} = \sum_{\mathbf{a}} \phi_{\mathbf{a}}$, one field element $\phi_{\mathbf{a}} \in \mathbb{Z}_p$ per $(t-1)$-sized subset $\mathbf{a}$ of the members, and each $\phi_{\mathbf{a}}$ is handed to every member *outside* $\mathbf{a}$. A member thus holds the $\binom{n-1}{t-1}$ summands whose subset excludes it, and any t members jointly hold all summands and so could reconstruct sk, while any $t-1$ miss the summand for their own subset and learn nothing. The members' shares

determine a single group key $X = g^{\mathsf{sk}}$, which occupies side A's slot in the funding 2-of-2. The shares can be set up by a trusted dealer or a distributed key generation, and the same shares later serve both keys and nonces.

The first round runs offline for the current session identifier sid, before the message exists. For each of the two nonce slots $i \in \{1, 2\}$, every member evaluates a pseudorandom function on $i \| sid$ under each of its shares, $\mathsf{H}_{\mathrm{prf}}(\phi_{\mathbf{a}}, i \| sid)$, and combines the results into its two nonce shares $(R_{1,k}, R_{2,k})$, which it broadcasts. A present quorum first checks the shares for well-formedness, that the replicated copies of each $\phi_{\mathbf{a}}$ produced the same value, and then aggregates them into the group's two nonces, bound together into the nonce R_A handed to the counterparty. Since the nonce shares are a deterministic function of sid alone and every share $\phi_{\mathbf{a}}$ is replicated across all members outside $\mathbf{a}$, any qualifying quorum recomputes the same R_A, so a member absent in this round is not locked out of the next.

In the second round, once the message m and the counterparty's nonce arrive, each member forms the binding factor $\check{b}$ and the challenge c, both computable from the public transcript, and returns its partial signature

$$s_k = r_{1,k} + r_{2,k}\,\check{b} + c\,a\,x_k,$$

where $r_{i,k}$ are its nonce-share scalars from the first round and x_k its signing-key share. Lagrange interpolation over the quorum turns the $\{s_k\}$ into side A's single partial signature s_A, which the outer MuSig2 session adds to s_B to form the final signature. The protocol is two rounds, and the first is precomputable before the message exists. This is the same as the round structure of MuSig2 itself, so the group adds no rounds to the channel protocol.

2.1 Proving Security

Proving security for a nested threshold multi-signature cannot be done with existing security models, as nested threshold multi-signatures neither fit pure threshold nor standard multi-signature frameworks. Instead, they represent a threshold group occupying a single slot within an outer multi-signature scheme. Threshold unforgeability games verify a forgery under the group's own key, so the group is the entire signing side and no outer session exists. The recent work of Kohen nests a MuSig2 session inside of MuSig2 [Koh26], where the inner signing session verifies against an outer aggregate key. We need exactly that, but the nested signer there is itself a multi-signature over a fixed set of members, and every one of them signs every session. Nesting a threshold signature adds three things that [Koh26] does not model, namely a corruption threshold inside the nested signer, a quorum that may differ between the two rounds (constraint C1), and agreement on session state that the channel forces on the group (constraint C4). We define unforgeability for nested threshold multi-signatures by modifying the security model of [Koh26] accordingly, so that the adversary controls every outer co-signer, corrupts up to $t-1$ members of the group, and opens concurrent signing sessions on messages and outer contexts of its choosing. In addition, the adversary can decide to switch the signing quorum from one round to the next. The adversary breaks unforgeability of a nested threshold multi-signature if it can produce a signature that verifies under an outer aggregate key without having collected enough honest contributions from the remaining honest parties to have assembled one itself.

Our final goal is now to prove security of Iceberg with this new security model. First, we observe that by using replicated secret sharing to make each protocol round quorum-independent, the t-out-of-n shared messages exchanged during an Iceberg round can be mapped directly to an *additive*

combination of shares. Recall that under replicated secret sharing, a secret key summand is defined for each $(t-1)$-sized subset of signers, and each party receives precisely the shares corresponding to the subsets it does *not* belong to [ISN87]. This yields $\binom{n}{t-1}$ underlying key summands in total. As all protocol messages in Iceberg are deterministicly computed from the shared key, the messages exchanged among any valid quorum in Iceberg can be deterministically expanded into $\binom{n}{t-1}$ individual protocol messages. This mapping allows us to convert the t-out-of-n threshold signature Iceberg into a virtual $\binom{n}{t-1}$-out-of-$\binom{n}{t-1}$ multi-signature scheme, which we denote as $\mathsf{Iceberg}_{\text{base}}$. In Theorem 1, we formalize this share conversion and prove that any adversary breaking the unforgeability of Iceberg can be transformed into an efficient adversary breaking the unforgeability of $\mathsf{Iceberg}_{\text{base}}$.

It is thus sufficient to show that $\mathsf{Iceberg}_{\text{base}}$ is unforgeable. We achieve this via a reduction from $\mathsf{Iceberg}_{\text{base}}$ to the nested multi-signature scheme NestedMuSig2 [Koh26]. Specifically, we embed the EUF-CMA challenge key of the honest signer in the security experiment of NestedMuSig2 into a single secret summand of $\mathsf{Iceberg}_{\text{base}}$ unknown to the threshold adversary. We then simulate all remaining computations by combining honest evaluations over known summands with queries to the EUF-CMA signing oracle for the challenge summand. Careful programming of the aggregation hash function H_{agg} ensures that the partial signatures returned by the oracle match the expected distributions in $\mathsf{Iceberg}_{\text{base}}$. Finally, because NestedMuSig2 comes with security proofs in both the Random Oracle Model (ROM, using $\nu = 8$ at depth 2) and the ROM + Algebraic Group Model (AGM, using $\nu = 2$ as in BIP 327), $\mathsf{Iceberg}_{\text{base}}$ (and by extension Iceberg) inherits both security bounds.

In all, we show that Iceberg fits threshold Lightning's operational constraints demonstrating that provably secure threshold custody for Lightning channels is achievable.

3 Preliminaries

Notation. We write $x \leftarrow y$ for assignment and $x \leftarrow\!\$\ X$ for uniform sampling. A group description $(\mathbb{G}, p, g)$ denotes a cyclic group $\mathbb{G}$ of prime order p with generator g. For $n \in \mathbb{N}$ we write $[n] = \{1, \ldots, n\}$, $\|$ for the concatenation of bitstrings, and $|S|$ for the cardinality of a set S. For a group of n members with threshold t, subsets of size $t-1$ are set in bold, so $\mathbf{A} = \binom{[n]}{t-1}$ is the family of all $(t-1)$-subsets of $[n]$, and $\mathbf{A}_k = \{\mathbf{a} \in \mathbf{A} : k \notin \mathbf{a}\}$ are the subsets whose shares member k holds.

Communication model. We work in the synchronous communication model. The endpoints of the higher-level multi-signature are connected by pairwise authenticated channels. The parties of the thresholdized endpoint are likewise connected by pairwise authenticated channels. They further use an untrusted aggregator node, which may be any of the participants or another party that facilitates communication (though this leaks privacy if another party is used).

Multi-signatures. A multi-signature scheme lets n signers, each holding an ordinary key pair, jointly produce a single signature verifiable under one aggregate key. It provides a key generation KeyGen that outputs a signer's key pair, a key aggregation KeyAgg that maps a list of public keys to one aggregate key $\widetilde{X}$, an interactive signing protocol, and a verification Verify. The signature is an ordinary one under $\widetilde{X}$, so on chain the group is indistinguishable from a single key.

MuSig2. MuSig2 [NRS21] instantiates this over $\mathbb{G}$ and is the scheme Lightning uses. Signer i holds $X_i = g^{x_i}$, and KeyAgg maps a key list L to $\widetilde{X} = \prod_i X_i^{a_i}$ with coefficients $a_i = \mathsf{H}_{\text{agg}}(L, X_i)$ that bind

each key to the whole list. Signing runs in two rounds. The first round is message-independent, and each signer samples ν nonces and sends $R_{i,j} = g^{r_{i,j}}$ for $j \in [\nu]$ (BIP 327 uses $\nu = 2$), and the nonces are aggregated slot-wise to $R_j = \prod_i R_{i,j}$. In the second round, once the message m is known, a binding factor $b = \mathsf{H}_{\mathrm{non}}(\widetilde{X}, (R_1, \ldots, R_\nu), m)$ combines the aggregates into the session nonce $R = \prod_j R_j^{b^{j-1}}$, and each signer returns the partial signature $s_i = c\, a_i\, x_i + \sum_j r_{i,j}\, b^{j-1}$ for the challenge $c = \mathsf{H}_{\mathrm{sig}}(\widetilde{X}, R, m)$. The partials sum to s, and $\sigma = (R, s)$ verifies as a Schnorr signature under $\widetilde{X}$. Since $\widetilde{X}$ is itself an ordinary public key, one MuSig2 aggregate key can occupy a signer slot of another, a *nested* MuSig2 [Koh26], which is the structure Iceberg uses to hide a group inside one side of the channel. We recall the pseudocode of the nested scheme NestedMuSig2 in Figure 10 in Section A.

Unforgeability. A (nested) multi-signature is EUF-CMA secure if no efficient adversary that controls every signer but one honest signer, and that opens concurrent signing sessions of its choice, can forge a signature under an aggregate key involving the honest key on a message it never asked the honest signer to sign. We recall the formal game in Figure 6 in Section A. Our security reduces to the EUF-CMA security of NestedMuSig2, which holds under standard assumptions [Koh26], and Section 6 makes the reduction and its instantiation precise.

Verifiable pseudorandom secret sharing. A verifiable pseudorandom secret sharing (VPSS) [KG25] is a t-of-n secret sharing whose long-lived shares let the members repeatedly derive sharings of pseudorandom values, one for each public tag w, without interaction. It provides a key generation $\mathsf{KeyGen}(n, t, \mu)$ that distributes long-lived shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ of a secret sk, a share derivation $\mathsf{Gen}(k, \mathsf{sk}_k, w)$ that outputs member k's share for tag w, and algorithms $\mathsf{VPSS}_1.\mathsf{Verify}$, Agg, and Recover that check the shares of a quorum and interpolate the derived value in the exponent or in the scalar, where the quorum parameter μ fixes how many shares verification needs. We require three properties. Under *verifiability* a quorum can check that the shares it received are consistent, under *uniqueness* every qualifying quorum obtains one and the same value for each tag, and under *pseudorandomness* the derived values are indistinguishable from random to any coalition of at most $t - 1$ members.

A VPSS from replicated secret sharing. We use the construction of [KG25], in which every share doubles as a key of a pseudorandom function. Key generation splits the secret $\mathsf{sk} = \sum_{\mathbf{a} \in \mathbf{A}} \phi_{\mathbf{a}}$ into one summand $\phi_{\mathbf{a}} \leftarrow\!\$\ \mathbb{Z}_p$ per subset $\mathbf{a} \in \mathbf{A}$ and hands $(\mathbf{a}, \phi_{\mathbf{a}})$ to every member outside $\mathbf{a}$, so member k holds $\mathsf{sk}_k = \{(\mathbf{a}, \phi_{\mathbf{a}})\}_{\mathbf{a} \in \mathbf{A}_k}$. Given a tag w, Gen evaluates $\mathsf{H}_{\mathrm{prf}}$ under each held summand and combines the results into member k's share. Across members these are Shamir shares of degree $t - 1$ of the value $\sum_{\mathbf{a}} \mathsf{H}_{\mathrm{prf}}(\phi_{\mathbf{a}}, w)$. Two features of this construction drive our design. First, Gen is deterministic, so a member can derive its share for any tag at any time from sk_k alone. Second, for a quorum parameter $\mu \geq 2t - 1$, verifiability and uniqueness hold information-theoretically, since a quorum of μ members contains at least t honest shares, which determine the degree-$(t-1)$ polynomial and expose every inconsistent share. Whenever $\mathsf{VPSS}_1.\mathsf{Verify}$ accepts, Agg therefore returns $g^{\sum_{\mathbf{a}} \mathsf{H}_{\mathrm{prf}}(\phi_{\mathbf{a}}, w)}$, the same value for every qualifying quorum. We recall the scheme, denoted VPSS_1, in Figure 8 in Section A, and a variant VPSS_0 without share conversion appears in Section 6.

4 Nested Threshold Multi-Signatures

We propose a new cryptographic primitive, namely *nested threshold multi-signatures*. Unlike conventional threshold signatures, which replace an entire signer with a threshold group, a nested threshold multi-signature allows a t-of-n group of members to occupy a single signer slot of an outer multi-signature protocol. This enables threshold signing within an existing multi-signature execution while leaving the outer protocol unchanged.

We define the primitive with respect to a two-round multi-signature scheme (Section 3) with key aggregation KeyAgg and verification Verify, whose signing session we call the *outer session*. We state syntax and security for one level of nesting, the setting of Section 2, and deeper nestings generalize as in [Koh26].

Syntax. A nested threshold multi-signature scheme extends a two-round threshold signature scheme with interfaces that connect the inner threshold execution to the outer multi-signature session. We follow the two-round threshold-signature formalization of [CKK+25] and the multi-signature formalization of [NRSW20b]. Nesting changes the interfaces in three places. First, each round ends in an aggregation algorithm that condenses the members' shares into the single message the outer session expects from the group's slot. Second, the signing round receives the entire outer context in which the group signs. Third and most importantly, the two rounds do not share per-member state, and they are linked only through a session identifier sid, so a member absent in the first round can still serve in the second (constraint C1). A nested threshold multi-signature scheme thus consists of the six algorithms ($\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{PreRound}, \mathsf{PreAgg}, \mathsf{SignRound}, \mathsf{SignAgg}$) with the following interfaces.

- $\mathsf{Setup}(1^\lambda) \to par$: reads the security parameter and outputs public parameters par, which fix the space $\mathcal{SID}$ of admissible session identifiers and are an implicit input to all other algorithms.
- $\mathsf{KeyGen}(k, n, t, A) \to (\mathsf{pk}, \mathsf{sk}_k)$: run by member k on the group size n and the threshold t, interacting with the other members through the interface A. It outputs the group's public key pk, which fills the group's slot of the outer key list, and k's secret share sk_k.
- $\mathsf{PreRound}(k, \mathsf{sk}_k, sid) \to out_k$: the message-independent first round, which reads a session identifier $sid \in \mathcal{SID}$ and outputs member k's share out_k of the group's first-round message.
- $\mathsf{PreAgg}(\mathsf{pk}, C, \{out_j\}_{j \in C}) \to out$: verifies and aggregates the first-round shares of a quorum C into the group's first-round message out in the outer session.
- $\mathsf{SignRound}(k, \mathsf{sk}_k, \mathsf{pk}, sid, m, out, ctx) \to s_k$: the second round, which reads the message m, the group's aggregate first-round message out, and the outer context ctx, consisting of the group's position in the outer key list, the co-signers' public keys, and the outer session's aggregate first-round message. It outputs member k's partial signature s_k.
- $\mathsf{SignAgg}(C, \{s_k\}_{k \in C}) \to s$: combines the partial signatures of a quorum C into the group's second-round message s.

A nested threshold multi-signature scheme has no verification algorithm of its own. Once the outer session combines the group's second-round message with those of the co-signers, the result is an ordinary signature of the outer scheme under the aggregate key $\widetilde{X}$ and is verified by Verify.

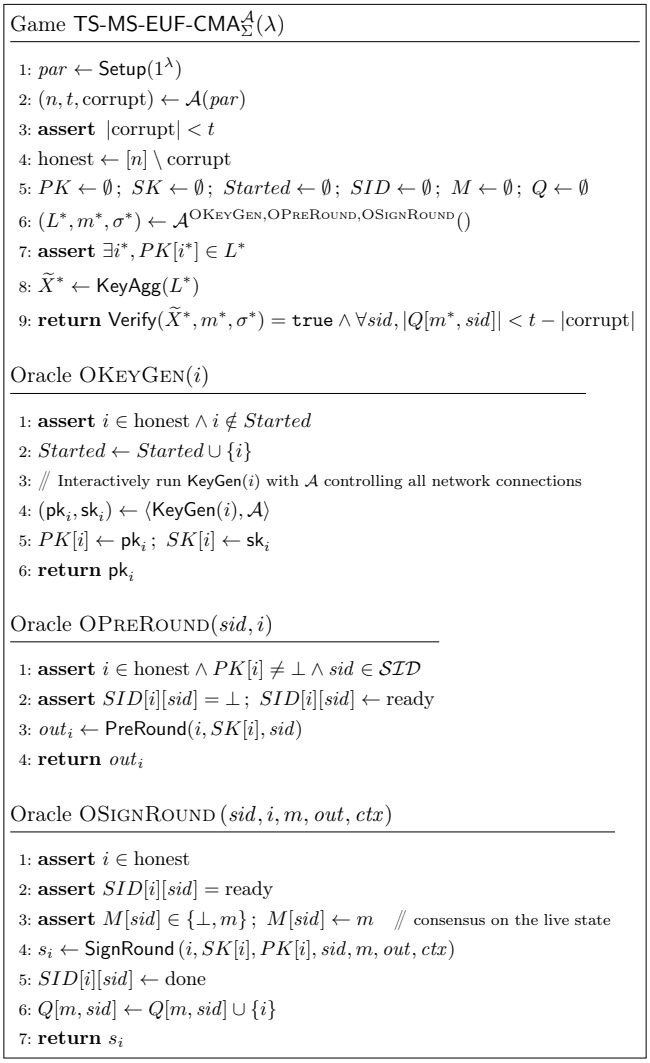

Figure 3: The TS-MS-EUF-CMA security game for a nested threshold multi-signature scheme Σ, where KeyAgg and Verify are the key aggregation and verification of the outer scheme. The assertion on $M[sid]$ models the consensus on the live state (constraint C4), so across all honest members, each session identifier is signed on at most one message.

Correctness. Fix a quorum size $\mu \geq t$, the number of shares its aggregation algorithms require. Correctness requires that an honest outer session completes to a valid signature under $\widetilde{X}$. For every quorum C of at least μ honest members, PreAgg and SignAgg produce a first-round and a second-round message that a single signer of the outer scheme holding the secret key of pk could have produced, and these messages are the same for every qualifying quorum, as constraint C3 demands.

Security. We define unforgeability of nested threshold multi-signature schemes through the experiment TS-MS-EUF-CMA, and our notation follows [Koh26]. The adversary statically corrupts fewer than t members, controls all signers of the outer session, and controls the network during the interactive key generation of the honest members. It interacts with the honest members through a

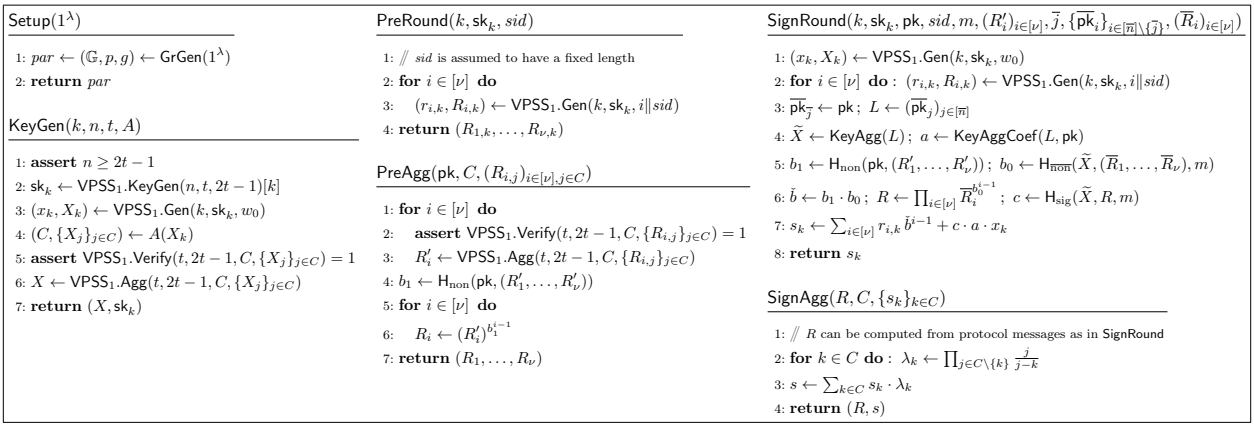

Figure 4: The protocol $\mathsf{Iceberg}[\mathsf{GrGen}, \nu, t]$. In SignRound, the values $(R'_i)_{i\in[\nu]}$ are the aggregate pre-nonces computed in PreAgg, and $(\overline{R}_i)_{i\in[\nu]}$ are the aggregate nonces of the outer session.

key generation oracle OKeyGen and two signing oracles OPreRound and OSignRound, which return a member's first-round share for a session identifier $sid \in \mathcal{SID}$ of the adversary's choice, and the member's partial signature for a message and an outer context of the adversary's choice. Across all honest members, each session identifier is signed on at most one message, modeling the consensus on the live state (constraint C4). The adversary wins if it outputs a fresh forgery, that is a signature that the outer scheme verifies under the aggregate key of a key list containing the group key, on a message m^* such that, for every sid, fewer than $t - |\mathrm{corrupt}|$ honest members completed the second round. We define $\mathsf{Adv}^{\mathsf{TS\text{-}MS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\Sigma}(\lambda) = \Pr\left[\mathsf{TS\text{-}MS\text{-}EUF\text{-}CMA}^{\mathcal{A}}_{\Sigma}(\lambda) = \mathtt{true}\right]$ and provide the formal experiment in Figure 3. Note that since we assume that $\binom{n}{t}$ is small by constraint C5, an adversary that fully adaptively corrupts signers can be reduced to a static security by a standard guessing argument.

5 Our Scheme

We describe our nested threshold multi-signature scheme Iceberg. For a high-level overview, we refer to Section 2 and to Figure 4 for pseudocode.

Setup and key generation. To set up Iceberg we select a group description $(\mathbb{G}, p, g)$, and the hash functions $\mathsf{H}_{\mathrm{prf}}, \mathsf{H}_{\mathrm{agg}}, \mathsf{H}_{\mathrm{non}}, \mathsf{H}_{\overline{\mathrm{non}}}, \mathsf{H}_{\mathrm{sig}} : \{0,1\}^* \to \mathbb{Z}_p$ are those of the outer session. Session identifiers are bitstrings of a fixed length, $\mathcal{SID} = \{0,1\}^\ell$, and the key tag w_0 is chosen outside $\mathcal{SID}$. Key generation wraps the key generation of VPSS_1, where the members obtain replicated shares sk_k of a secret $\mathsf{sk} = \sum_{\mathbf{a}\in\mathbf{A}} \phi_{\mathbf{a}}$ with quorum parameter $\mu = 2t-1$, from a trusted dealer or a distributed key generation. Each member derives its key share $(x_k, X_k) \leftarrow \mathsf{VPSS}_1.\mathsf{Gen}(k, \mathsf{sk}_k, w_0)$ at a fixed public tag w_0 reserved for the key, the share commitments of a quorum C are checked with $\mathsf{VPSS}_1.\mathsf{Verify}$, and Agg interpolates the group key $X = g^{\mathsf{sk}}$. This key occupies the group's slot of the outer key list, so to the outer session the group looks like one ordinary signer.

Nonce precomputation. The first round runs before the message exists and fixes the group's nonce for the session identifier sid. For each nonce slot $i \in [\nu]$, member k derives $(r_{i,k}, R_{i,k}) \leftarrow$

$\mathsf{VPSS}_1.\mathsf{Gen}(k, \mathsf{sk}_k, i\|sid)$ and broadcasts the group elements. PreAgg checks the shares of a quorum C with $\mathsf{VPSS}_1.\mathsf{Verify}$, interpolates each slot to the pre-nonce R'_i, derives the binding factor $b_1 \leftarrow \mathsf{H}_{\mathrm{non}}(\mathsf{pk}, (R'_1, \ldots, R'_\nu))$, and outputs the bound nonces $R_i \leftarrow (R'_i)^{b_1^{i-1}}$. This mirrors SignAgg followed by SignAggExt in NestedMuSig2, so the output is exactly the first-round message a single MuSig2 participant would send. Since Gen is deterministic and every summand is replicated, every qualifying quorum computes the same nonces, and a member absent in this round recomputes its shares later from sk_k alone.

Signing. The second round runs once the message m and the outer context ctx are known, that is, the group's position $\overline{j}$ in the outer key list, the co-signer keys $\{\overline{\mathsf{pk}}_i\}$, and the outer aggregate nonces $(\overline{R}_i)$. Member k recomputes its key share and nonce shares via Gen, forms the outer key list L with pk at position $\overline{j}$, and computes the aggregate key $\widetilde{X} \leftarrow \mathsf{KeyAgg}(L)$ together with the group's aggregation coefficient $a \leftarrow \mathsf{KeyAggCoef}(L, \mathsf{pk})$. Two binding factors enter the response, the inner $b_1 \leftarrow \mathsf{H}_{\mathrm{non}}(\mathsf{pk}, (R'_1, \ldots, R'_\nu))$ and the outer $b_0 \leftarrow \mathsf{H}_{\overline{\mathrm{non}}}(\widetilde{X}, (\overline{R}_1, \ldots, \overline{R}_\nu), m)$. With $\check{b} = b_1 \cdot b_0$, the session nonce $R = \prod_i \overline{R}_i^{b_0^{i-1}}$, and the challenge $c = \mathsf{H}_{\mathrm{sig}}(\widetilde{X}, R, m)$, member k outputs the partial signature

$$s_k \leftarrow \textstyle\sum_{i\in[\nu]} r_{i,k}\, \check{b}^{i-1} + c \cdot a \cdot x_k.$$

Aggregation and verification. SignAgg combines the partial signatures of a quorum C by Lagrange interpolation, $s \leftarrow \sum_{k\in C} s_k \cdot \lambda_k$, and outputs (R, s). The value s is the group's second-round message in the outer session. The outer session adds the partial signatures of the co-signers, and the result verifies as an ordinary Schnorr signature under $\widetilde{X}$. For Lightning we instantiate $\nu = 2$ and nesting depth $\Lambda = 2$, so the outer session is unmodified BIP 327 MuSig2 and the counterparty runs the protocol it already runs today.

6 Security

We prove the security of Iceberg by reducing it to the EUF-CMA security of NestedMuSig2, in two steps. First, in Theorem 1, we reduce Iceberg to the scheme $\mathsf{Iceberg}_{\mathrm{base}}$ (Figure 9), which replaces VPSS_1 by the variant VPSS_0 without share conversion (Figure 7), where every member outputs one contribution per summand, and aggregation deduplicates the replicated copies instead of interpolating. Both schemes compute the same keys, nonces, and signatures, so the reduction is a wrapper that converts shares back and forth. Then, in Theorem 2, we reduce $\mathsf{Iceberg}_{\mathrm{base}}$ to NestedMuSig2, and we describe this reduction at a high level before stating the theorem.

Lemma 1. *If* $\mathsf{Iceberg}_{\mathrm{base}}$ *is* TS-MS-EUF-CMA *secure then* Iceberg *is* TS-MS-EUF-CMA *secure. In particular, for every adversary* $\mathcal{A}$ *against* $\mathsf{TS\text{-}MS\text{-}EUF\text{-}CMA}^{\mathcal{A}}_{\mathsf{Iceberg}}(\lambda)$ *there exists an adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\mathsf{TS\text{-}MS\text{-}EUF\text{-}CMA}}_{\mathcal{B},\mathsf{Iceberg}_{\mathrm{base}}}(\lambda) \geq \mathsf{Adv}^{\mathsf{TS\text{-}MS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Iceberg}}(\lambda).$$

Proof. The algorithm $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$ that simulates the three TS-MS-EUF-CMA oracles by forwarding all oracle queries and performing share conversion, from replicated secret sharing to Shamir secret sharing, as follows.

Since VPSS_0 and VPSS_1 have identical key generation algorithms, $\mathcal{B}$ forwards all OKeyGen queries from $\mathcal{A}$ to the OKeyGen oracle provided by its environment until that phase of key generation is complete, making sure to store all extracted secret PRF keys. For each queried index

k, this results in the environment returning $(X_{k,\mathbf{a}})_{\mathbf{a}\in\mathbf{A}_k}$, from which $\mathcal{B}$ computes and returns to $\mathcal{A}$ the value $X_k \leftarrow \prod_{\mathbf{a}\in\mathbf{A}_k} X_{k,\mathbf{a}}^{L'_{\mathbf{a}}(k)}$, where $L'_{\mathbf{a}}(k) = \prod_{\ell\in\mathbf{a}} \frac{\ell-k}{\ell}$ as in $\mathsf{VPSS}_1.\mathsf{Gen}$. Upon subsequently receiving $(C, \{X_j\}_{j\in C})$ from $\mathcal{A}$, $\mathcal{B}$ asserts that $\mathsf{VPSS}_1.\mathsf{Verify}(t, 2t-1, C, \{X_j\}_{j\in C}) = 1$ and then obtains, for each $j \in C$, the share vector $(X_{j,\mathbf{a}})_{\mathbf{a}\in\mathbf{A}_j}$, either as observed in the interaction with the environment for honest j, or by computing $\mathsf{VPSS}_0.\mathsf{Gen}(j, \mathsf{sk}_j, w_0)$ from the shares sk_j extracted during key generation for corrupt j. Then $\mathcal{B}$ gives $\{(X_{j,\mathbf{a}})_{\mathbf{a}\in\mathbf{A}_j}\}_{j\in C}$ to the environment and receives X, which is returned to $\mathcal{A}$.

Upon receiving a OPREROUND(sid, k) query from $\mathcal{A}$, $\mathcal{B}$ forwards it to the $\mathsf{Iceberg}_{\text{base}}$ OPREROUND oracle, receiving $(R_{1,k}, \ldots, R_{\nu,k})$, where $R_{i,k} = (R_{i,k,\mathbf{a}})_{\mathbf{a}\in\mathbf{A}_k}$. Then, $\mathcal{B}$ computes

$$R'_{i,k} \leftarrow \prod_{\mathbf{a}\in\mathbf{A}_k} R_{i,k,\mathbf{a}}^{L'_{\mathbf{a}}(k)}$$

and returns $(R'_{1,k}, \ldots, R'_{\nu,k})$ to $\mathcal{A}$.

Upon receiving OSIGNROUND queries from $\mathcal{A}$, $\mathcal{B}$ forwards these queries to the $\mathsf{Iceberg}_{\text{base}}$ OSIGNROUND oracle, receiving $(s_{k,\mathbf{a}})_{\mathbf{a}\in\mathbf{A}_k}$, and then computes and returns to $\mathcal{A}$ the value

$$s_k \leftarrow \sum_{\mathbf{a}\in\mathbf{A}_k} s_{k,\mathbf{a}} \cdot L'_{\mathbf{a}}(k).$$

Finally, $\mathcal{B}$ outputs the forgery output by $\mathcal{A}$, which is valid in the $\mathsf{Iceberg}_{\text{base}}$ game since both schemes share the same key aggregation and signature verification. The above perfectly simulates the $\mathsf{Iceberg}$ environment to $\mathcal{A}$, noting that both VPSS_0 and VPSS_1 are information-theoretically verifiable and unique. □

Reducing to NestedMuSig2. The reduction behind Theorem 2 is a straight-line simulation. It never rewinds the adversary, and it introduces no assumption beyond the security of NestedMuSig2, which has multiple security proofs under different parameter constraints and cryptographic assumptions. Its starting point is the observation that a group running $\mathsf{Iceberg}_{\text{base}}$ already behaves like a depth-two NestedMuSig2 session. The group key is the product of the summand keys $g^{\mathsf{H}_{\text{prf}}(\phi_{\mathbf{a}}, w_0)}$, one per subset $\mathbf{a}$, so we may view the summands themselves as the innermost signers of a nesting, where the outer level is the MuSig2 session with the counterparty, and the inner level aggregates the summand keys into the group key. The reduction makes this view literal by programming the key aggregation oracle H_{agg} so that the aggregate of the summand keys equals the group key, and it hides the EUF-CMA challenge key in exactly one summand, one whose PRF key the corrupted members do not hold.

Every value the adversary sees is then computed honestly, with one exception. For all summands except the challenge one, the reduction knows the PRF keys and runs the real protocol. For the challenge summand it cannot evaluate H_{prf}, so it substitutes the honest signer of the EUF-CMA game. The nonce shares for a session identifier come from the OPREROUND oracle, and the corresponding component of a partial signature comes from the OSIGNROUND oracle. This substitution is consistent because, in $\mathsf{Iceberg}_{\text{base}}$, the component a member reports for a summand depends only on the summand and the session, not on the member, so one oracle response answers every honest member's query for the same *sid*, and caching it preserves the determinism the adversary expects. Furthermore, the consensus assertion in the game guarantees that each *sid* is signed on at most one message, so the single Sign' call that the EUF-CMA game permits per session is all the reduction ever needs.

A forgery translates because if for every *sid*, fewer than $t - |\text{corrupt}|$ honest members signed m^*, then, for a suitable choice of the challenge summand, no component of the challenge summand for m^* was ever released. The reduction then never queried OSignRound on m^*, and the forged signature is a valid NestedMuSig2 forgery on a fresh message, and Theorem 1 lifts the result from $\mathsf{Iceberg}_{\text{base}}$ to Iceberg.

Theorem 2. *If the nested multi-signature scheme* $\mathsf{NestedMuSig2}[\mathsf{GrGen}, \nu]$ *is EUF-CMA secure, then the nested threshold multi-signature scheme* $\mathsf{Iceberg}[\mathsf{GrGen}, \nu, t]$ *is TS-MS-EUF-CMA secure in the random oracle model for* H_{agg}, H_{prf}, H_{non}, $\mathsf{H}_{\overline{\text{non}}}$, $\mathsf{H}_{\text{sig}} : \{0,1\}^* \to \mathbb{Z}_p$.

Precisely, let $N = \binom{n}{t-1}$ *be the number of summands of the replicated sharing, which is also the size of the key list the reduction hands to the* NestedMuSig2 *session. For any adversary* $\mathcal{A}$ *against* Iceberg *running in time at most* τ *and making at most* q_{agg} *queries to* H_{agg}, *there exists an adversary* $\mathcal{C}$ *against* NestedMuSig2 *running in time at most*

$$\tau' = \tau + O(N) \cdot \tau_{\text{exp}},$$

where τ_{exp} *is the time of an exponentiation in* $\mathbb{G}$, *such that*

$$\mathsf{Adv}^{\text{EUF-CMA}}_{\mathcal{C},\mathsf{NestedMuSig2}}(\lambda) \geq \mathsf{Adv}^{\text{TS-MS-EUF-CMA}}_{\mathcal{A},\mathit{Iceberg}}(\lambda) - \frac{N \cdot q_{agg}}{2^\lambda}.$$

Proof. We proceed by constructing an elementary wrapper, $\mathcal{C}$, around $\mathcal{B}$ from Theorem 1 that simulates the TS-MS-EUF-CMA environment given access to the EUF-CMA environment for NestedMuSig2. The result then follows since, as we argue below, a forgery in the simulated game translates into an EUF-CMA forgery against NestedMuSig2.

First, $\mathcal{C}$ performs $\mathsf{VPSS}_0.\mathsf{KeyGen}$ simulating all honest parties, allowing it to extract all adversarial PRF secrets by the honest majority assumption, which assures that the honest parties make up a valid quorum. Index the subsets in $\mathbf{A}$ as $\mathbf{a}_1, \ldots, \mathbf{a}_\gamma$ with $\gamma = |\mathbf{A}|$ and, WLOG, let $\phi_{\mathbf{a}_1}$ be a PRF key not known to the adversary at the end of secret sharing.

Next, $\mathcal{C}$ embeds the EUF-CMA challenge key X^* by setting $X_1 \leftarrow X^*$, so that the corresponding summand key becomes $(X^*)^{a_1}$ for the programmed coefficient a_1 below. Specifically, for each $j > 1$, $\mathcal{C}$ computes $x'_j \leftarrow \mathsf{H}_{\text{prf}}(\phi_{\mathbf{a}_j}, w_0)$, picks a_j randomly, and then sets $x_j \leftarrow x'_j / a_j$ and $X_j \leftarrow g^{x_j}$. Finally, $\mathcal{C}$ programs $\mathsf{H}_{\text{agg}}(L, X_j)$ to be equal to a_j for all j, where $L \leftarrow \{X_1, \ldots, X_\gamma\}$. This ensures that $\mathsf{KeyAgg}(L) = \prod_j X_j^{a_j} = (X^*)^{a_1} \prod_{j>1} g^{x'_j}$. The programming fails with only negligible probability since the value X_1 was just drawn uniformly at random and is an element of L. Concretely, if q_{agg} is a bound on the number of queries the adversary makes to H_{agg} then the programming fails with probability at most $\frac{N \cdot q_{agg}}{2^\lambda}$. Lastly, $\mathcal{C}$ continues to execute key generation honestly, except that $(X^*)^{a_1}$ is used in place of $g^{\mathsf{H}_{\text{prf}}(\phi_{\mathbf{a}_1}, w_0)}$.

Upon receiving OPreRound queries, $\mathcal{C}$ honestly makes all expected *sid*-related assertions and computes PreRound for all but the unknown summand $\mathbf{a}_1$ in which the challenge key is embedded. For that summand, $\mathcal{C}$ calls the EUF-CMA OPreRound oracle. All future calls with the same session identifier use a cached response for the challenge summand to maintain consistency. Since we are assuming state-agreement is being used so that it is impossible to validly query the same sid in two different inputs to the second round, this simulation is faithful.

Lastly, upon receiving OSignRound queries, $\mathcal{C}$ honestly makes all verification assertions and then computes SignRound honestly for all but the unknown summand $\mathbf{a}_1$. For that summand, $\mathcal{C}$ invokes the EUF-CMA OSignRound oracle. Because of the programming of H_{agg} during key

generation, the response from the nested MuSig2 signing oracle yields a valid $\mathsf{Iceberg}_{\mathrm{base}}$ partial signature.

We conclude that $\mathcal{C}$ perfectly simulates the expected environment, so long as no bad event happens in $\mathsf{H}_{\mathrm{agg}}$, since everything is computed honestly except for the values relating to the embedded challenge, all of which are drawn from a distribution identical to an honest execution. □

Corollary 3. *Let* GrGen *be a group generation algorithm for which the algebraic one-more discrete logarithm problem is hard. By Theorem 2 and the security of* NestedMuSig2[GrGen, $\nu = 2$] *for sessions of depth two [Koh26], the scheme* Iceberg[GrGen, $\nu = 2, t$] *nested in an unmodified BIP 327 session is TS-MS-EUF-CMA secure in the algebraic group model for* GrGen *and the random oracle model for* $\mathsf{H}_{\mathrm{agg}}$, $\mathsf{H}_{\mathrm{prf}}$, $\mathsf{H}_{\mathrm{non}}$, $\mathsf{H}_{\overline{\mathrm{non}}}$, $\mathsf{H}_{\mathrm{sig}}$.

7 Prototype and Evaluation

We support our work with a prototype and a performance evaluation. Our prototype [Fur] implements Iceberg over `secp256k1` and ports it into `eclair`, a production Lightning implementation. Beyond demonstrating that Iceberg can thresholdize today's Lightning with only moderate overhead, we structure our evaluation around four questions.

1. *Micro benchmarks.* How expensive is Iceberg in computation and in wire bytes across (t, n)?
2. *Drop-in replacement.* Does Iceberg slot into a production Lightning node without changes to its channel logic?
3. *Thresholdized Lightning throughput.* How many payments per second can an Iceberg Lightning endpoint sustain on a single core?
4. *System bottleneck.* Does the threshold signature limit a thresholdized channel, or does the channel logic dominate it?

Experimental setup. We implement Iceberg over `secp256k1` with BIP 340 x-only keys [WNR20] and BIP 327 key aggregation [NRJ22]. The outer session is unmodified BIP 327 with $\nu = 2$, so the counterparty sees stock MuSig2 throughout. All measurements run on a dedicated machine with an AMD EPYC 7543 CPU, ten cores made available to the benchmark and a 4 GB measurement heap. We measure at two layers, namely the primitive in isolation and a whole payment driven through two `eclair` channel state machines. Both run over the parameter family $t \in \{2, 3, 4, 5\}$ with n up to ten, resulting in twenty configurations.[1]

Micro benchmarks. We first measure the local cost of Iceberg and compare it to single-key MuSig2. One ordinary MuSig2 participant spends 42.0 μs on its partial signature, with a standard deviation of 5.4 across all twenty configurations, measured over 200 iterations. A group instead spends 1 042 to 7 358 μs on the same partial signature (Table 3). The reported numbers are totals across the group, since every member runs in one process here. Spread over the members at $(3, 7)$, a member that signs in both rounds does 14.4× the work of one participant, and a member that only contributes to the first round 1.5×. Aggregation costs additional 392 μs per signature.

[1] For every t, the signing quorum has at least $2t - 1$ members.

	Group size n							
t	3	4	5	6	7	8	9	10
2	1 042	1 226	1 242	1 099	1 056	1 083	1 087	1 072
3			2 138	2 156	2 214	2 287	2 270	2 441
4					3 685	3 841	4 180	4 557
5							6 240	7 358

Table 3: Total processor time the group spends on one partial signature, in μs, measured in C over 200 iterations. Empty cells cannot exist, since the quorum of $2t - 1$ has to fit inside the group.

On the wire the group still acts as a single signer, sending the counterparty 66 bytes in the first round and 32 in the second, in accordance with BIP 327. The traffic inside the group costs 2 208 bytes per payment at $(3, 7)$, which is $2t - 1$ nonce shares in each of six first rounds and t signature shares in each of two second rounds. Each share carries a one-byte member index on top of the 66 or 32 bytes it transports. Beyond that traffic, a member stores 100 bytes of long-lived share material at $(2, 4)$, 484 at $(3, 7)$ and 2 692 at $(4, 10)$.

Drop-in replacement. Next, we demonstrate that Iceberg is a drop-in replacement for MuSig2. `eclair` reaches MuSig2 through the `secp256k1-kmp` bindings, and every call it makes for a taproot channel goes through that interface. We implement Iceberg behind that interface so `eclair` obtains a public key, a public nonce and a partial signature from the group wherever it previously obtained them from one signer. Therefore, we can use Iceberg in `eclair` without changing the channel logic or the message flow.

Next, we verified that our binding is valid with the checks included by `eclair`. A channel enters its operating state only after `eclair` rebuilds the commitment transaction and runs it through Bitcoin's script interpreter against the real funding output, on both sides. Every payment we time therefore settled against a genuine aggregate Schnorr signature. We also replay the whole path against complete signing sessions recorded from the reference implementation. Given the same seeds, session identifier and message, our prototype reproduces the reference byte for byte at each member's public share, at the group public key, at the aggregate group nonce and at the group signature. We run this replay at $(2, 3)$, $(3, 5)$, $(3, 7)$, $(4, 7)$ and $(5, 9)$. Beyond that we run ten sequential payments on one channel. We also pin both orderings of the two public keys, since key aggregation sorts them and the group lands first about half the time. Finally, we run the group as the party that opens the channel and as the party that accepts it. None of this changes `eclair`'s own behaviour, since 473 of its channel tests pass unchanged and the build that produced our timings passed all seventeen of its correctness checks.

We additionally verified that a Iceberg group never revises the nonce it sent even with a changing signing quorum (constraints C1 and C3). We verify that Iceberg is compliant with these constraints by completing a session in which the members that produce the second round did not contribute to the first. The resulting signature verifies under the aggregate key at every configuration we replay.

Thresholdized Lightning throughput. We then measure the sustained throughput an `eclair` channel achieves once Iceberg thresholdizes one of its two endpoints (Table 4). Making one endpoint a group costs 3 844 μs per payment at $(2, 4)$ and 8 246 μs at $(3, 7)$, which is 6.7% and 14.3% of a payment's CPU time. The largest configuration we deployed, $(4, 10)$, adds 16 786 μs, yet all config-

Tolerates	Configuration	Quorum	Added (μs)	Relative
1 faulty member	$(2, 4)$	3	$3\,844 \pm 157$	6.7%
2 faulty members	$(3, 7)$	5	$8\,246 \pm 202$	14.3%
3 faulty members	$(4, 10)$	7	$16\,786 \pm 247$	29.1%

Table 4: Added cost of one payment at the three deployable configurations, against a pooled control of $57\,599\,\mu$s.

Configuration	Payments per second	Against bare MuSig2	With member verification
bare MuSig2	17.4		
$(2, 4)$	16.3	−6.3%	15.8 (−9.0%)
$(3, 7)$	15.2	−12.5%	14.5 (−16.4%)
$(4, 10)$	13.4	−22.6%	12.6 (−27.5%)

Table 5: Payment throughput on one core, derived from the measured cost per payment. The last column adds the optional verification of a member's partial signature.

	Group size n							
t	3	4	5	6	7	8	9	10
2	3 884	3 844	3 618	3 960	3 954	n/r	3 910	4 096
3			7 482	7 954	8 246	8 271	8 746	8 836
4					12 528	13 535	15 185	16 786
5							22 573	26 124

Table 6: Added cost of one payment in μs, over 1 500 paired iterations per configuration.

urations retain substantial throughput headroom over current Lightning payment rates. Across the deployable family the growth is therefore quadratic.

The added work is fixed per payment, so it converts into throughput on one core (Table 5). A bare channel sustains 17.4 payments per second, a $(2, 4)$ group 16.3 and a $(3, 7)$ group 15.2. Matching the bare rate therefore takes 6.7 to 29.1 percent more machines, and 9.9 to 38.0 percent if the members verify each other's partial signatures.

Those three configurations pin n to t, since a deployable group has $3t - 2$ members, so they cannot show which of the two drives the cost. We therefore sweep the whole family and ask whether the cost follows the fault tolerance alone or also the number of key shares a member holds (Table 6). A member holds $\binom{n-1}{t-1}$ key shares, which is two to nine along the $t = 2$ row and twenty to eighty-four along the $t = 4$ row. The $t = 2$ row is flat, because its seven added shares move the cost by 211 μs against intervals of about ± 160. The $t = 4$ row rises by $4\,258\,\mu$s instead, because its sixty-four added shares cost roughly $67\,\mu$s each. The fault tolerance therefore sets the bulk of the cost, and the shares add to it only once the threshold makes them numerous. The entry marked n/r is not resolved, because its interval spans zero.

System bottleneck. Our micro benchmarks show that at $(3, 7)$ a group signs 53 times more slowly than one participant. At the same time, a payment's compute grows by only 14.3%. Therefore,

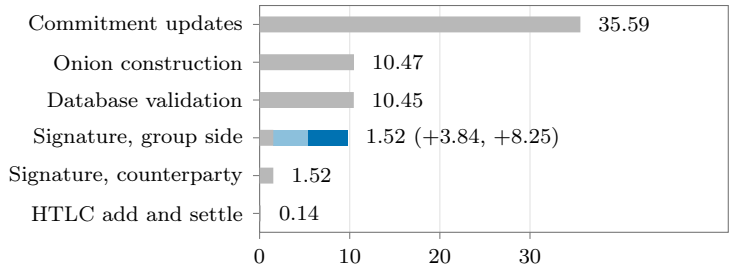

Figure 5: Computation splits of a single payment, over 500 iterations on a warm JVM. The two appended segments are what a $(2, 4)$ and a $(3, 7)$ group adds to the single signature it replaces.

Configuration	Group (μs)	Displaced signer (μs)	Measured added (μs)	Residual (μs)
$(2, 4)$	5 672	1 633	3 844	−195
$(3, 7)$	9 946	1 539	8 246	−161
$(4, 10)$	18 557	1 421	16 786	−350

Table 7: Cryptography timed inside a live payment against the cost measured through the channel.

we ask what the system bottleneck in a thresholdized channel is. To do this, we decompose a payment into the work `eclair` performs and compare it to the group's contribution (Figure 5). A payment costs 59.7 ms, of which the two commitment updates are 59.6%, the sender's onion construction is 17.5% and the test database validating its own writes is another 17.5%. The funding signatures of both parties are 5.1%, and only half of that, on average 1 517 μs, belongs to the party a group replaces. The substitution therefore starts from a fortieth of a payment. The commitment updates carry most of that, and inside them the MuSig2 signing takes microseconds. The rest is therefore transaction construction, one signature per HTLC, per-commitment key derivation, state persistence and message passing between the two channel state machines.

We also time the group's cryptography and the single-key signer it replaces inside the same payment, and hold their difference against the slowdown the channel shows (Table 7). The two agree to within 350 μs at every deployable configuration, so a group costs no more than the cryptography it adds. A naive combination of the micro benchmark and the payment decomposition would predict a much larger slowdown. Scaling a fortieth of a payment by the 53× micro-benchmark factor suggests a payment should more than double in time. However, the measured growth is only 14.3%. This is because, within the live payment flow, the group costs 6.5× the signer it replaces rather than 53×. We attribute this gap to the fixed overhead of the `secp256k1-kmp` bindings, which affects both signers.

Scope and limitations. We deliberately exclude the network between the members from our benchmarks. Since a group is the devices of one user or the machines of one company, we assume co-location on a low-latency local network. A payment still drives six first rounds and two second rounds inside the group, so a deployment adds one round trip for each of them. Furthermore, our prototype covers the active payment path alone, and mutual close, force close, splicing and channel announcements are not yet wired to a threshold signer. Channel announcements in particular need an ECDSA signature over the funding key, which Iceberg does not produce. Finally, our benchmarks

evaluate payments carrying a single HTLC. Because every key on the group's side is a group key and each HTLC needs its own signature, the cost grows with the number of HTLCs in flight.

8 Conclusion and Future Work

Iceberg allows a group of signers to jointly operate a single participant in a MuSig2 session such that the other participant sees the message flow of a single signer throughout. Either endpoint of a Lightning channel, or both independently, can use threshold custody with no change to Bitcoin, the Lightning protocol, or its counterparty. We prove Iceberg secure and demonstrate its practicality through an integration with `eclair`: a 2-of-4 endpoint sustains over 93% of the theoretical maximum payment throughput of an unmodified endpoint, leaving substantial headroom over current Lightning payment rates.

More broadly, Iceberg shows that thresholdization can be achieved by nesting a threshold group inside an existing multi-signature protocol rather than replacing it. An open question is which other multi-signature protocols admit such nesting and what properties are required to support it. For Lightning, a further question is how much a group can change about itself while the channel stays open: proactively refreshing its shares, repairing the share of a member that has lost it, or changing the membership or the threshold. Each of these has to preserve the aggregate key, since that key is the channel's funding output and changing it means closing and reopening the channel.

Acknowledgements

We would like to thank Adam Everspaugh, Leo Nash, and Jonas Nick for helpful discussions in the early stages of this project. Paul Gerhart's research has been supported by the Google PhD Fellowship in Privacy, Safety, and Security.

References

[1ML25] 1ML. Lightning network search and analysis engine: Statistics. https://1ml.com/statistics, 2025. Public node and channel counts, Bitcoin mainnet; 15,000 nodes, tens of thousands of channels, December 2025.

[AB21] Handan Kilinç Alper and Jeffrey Burdges. Two-round trip Schnorr multi-signatures via delinearized witnesses. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 157–188, Virtual Event, August 2021. Springer, Cham.

[And11] Gavin Andresen. BIP 11: M-of-N standard transactions. Bitcoin Improvement Proposal, https://github.com/bitcoin/bips/blob/master/bip-0011.mediawiki, 2011.

[ANO+22] Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. Low-bandwidth threshold ECDSA via pseudorandom correlation generators. In *2022 IEEE Symposium on Security and Privacy*, pages 2554–2572. IEEE Computer Society Press, May 2022.

[BCK+22] Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 517–550. Springer, Cham, August 2022.

[BCL+25] Renas Bacho, Yanbo Chen, Julian Loss, Stefano Tessaro, and Chenzhi Zhu. Adaptively secure partially non-interactive threshold schnorr signatures in the AGM. Cryptology ePrint Archive, Report 2025/1953, 2025.

[BD21] Mihir Bellare and Wei Dai. Chain reductions for multi-signatures and the HBMS scheme. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 650–678. Springer, Cham, December 2021.

[BDLR25a] Renas Bacho, Sourav Das, Julian Loss, and Ling Ren. Adaptively secure three-round threshold schnorr signatures from DDH. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 390–422. Springer, Cham, August 2025.

[BDLR25b] Renas Bacho, Sourav Das, Julian Loss, and Ling Ren. Glacius: Threshold schnorr signatures from DDH with full adaptive security. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part II*, volume 15602 of *LNCS*, pages 304–334. Springer, Cham, May 2025.

[Bin19] Binance. Security breach update. https://www.binance.com/en/support/articles/360028031711, 2019. 7,000 BTC stolen from a single hot wallet, May 2019.

[Bit25] Bitcoin Visuals. Lightning network statistics. https://bitcoinvisuals.com/lightning, 2025. Public capacity record 5,600 BTC across 15,000 nodes, December 2025.

[BLL+21] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 33–53. Springer, Cham, October 2021.

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 390–399. ACM Press, October / November 2006.

[BW25] Renas Bacho and Benedikt Wagner. T-spoon: Tightly secure two-round multi-signatures with key aggregation. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 256–290. Springer, Cham, August 2025.

[CCL+20] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 266–296. Springer, Cham, May 2020.

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 342–362. Springer, Berlin, Heidelberg, February 2005.

[CGG+20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1769–1787. ACM Press, November 2020.

[CGRS23] Hien Chu, Paul Gerhart, Tim Ruffing, and Dominique Schröder. Practical Schnorr threshold signatures without the algebraic group model. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 743–773. Springer, Cham, August 2023.

[Che25] Yanbo Chen. Round-efficient adaptively secure threshold signatures with rewinding. *IACR Communications in Cryptology*, 2(2), 2025.

[CKK+25] Elizabeth C. Crites, Jonathan Katz, Chelsea Komlo, Stefano Tessaro, and Chenzhi Zhu. On the adaptive security of FROST. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 480–511. Springer, Cham, August 2025.

[CKM23] Elizabeth C. Crites, Chelsea Komlo, and Mary Maller. Fully adaptive Schnorr threshold signatures. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 678–709. Springer, Cham, August 2023.

[DDFY94] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *26th ACM STOC*, pages 522–533. ACM Press, May 1994.

[DEF+19] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy*, pages 1084–1101. IEEE Computer Society Press, May 2019.

[Des94] Yvo Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, July/August 1994.

[DJN+20] Ivan Damgård, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Jakob Illeborg Pagter, and Michael Bæksvang Østergaard. Fast threshold ECDSA with honest majority. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 382–400. Springer, Cham, September 2020.

[DKLs19] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy*, pages 1051–1066. IEEE Computer Society Press, May 2019.

[DOK+20] Anders P. K. Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. Securing DNSSEC keys via threshold ECDSA from generic MPC. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, *ESORICS 2020, Part II*, volume 12309 of *LNCS*, pages 654–673. Springer, Cham, September 2020.

[DOTT21] Ivan Damgård, Claudio Orlandi, Akira Takahashi, and Mehdi Tibouchi. Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 99–130. Springer, Cham, May 2021.

[Fur] Matias Furszyfer. Iceberg benchmark. https://github.com/furszy/benchmark-iceberg.

[FYZ+25] Qi Feng, Kang Yang, Kaiyi Zhang, Xiao Wang, Yu Yu, and Xiang Xie. Stateless deterministic multi-party EdDSA signatures with low communication. In Tibor Jager and Jiaxin Pan, editors, *PKC 2025, Part V*, volume 15678 of *LNCS*, pages 268–297. Springer, Cham, May 2025.

[GG18] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1179–1194. ACM Press, October 2018.

[GGN16] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 2016*, volume 9696 of *LNCS*, pages 156–174. Springer, Cham, June 2016.

[GJKR96] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 354–371. Springer, Berlin, Heidelberg, May 1996.

[GJKR03] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure applications of Pedersen's distributed key generation protocol. In Marc Joye, editor, *CT-RSA 2003*, volume 2612 of *LNCS*, pages 373–390. Springer, Berlin, Heidelberg, April 2003.

[GJKR07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007.

[GKMN21] François Garillot, Yashvanth Kondi, Payman Mohassel, and Valeria Nikolaenko. Threshold Schnorr with stateless deterministic signing from standard assumptions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 127–156, Virtual Event, August 2021. Springer, Cham.

[GKSŚ20] Adam Gągol, Jędrzej Kula, Damian Straszak, and Michał Świętek. Threshold ECDSA for decentralized asset custody. Cryptology ePrint Archive, Report 2020/498, 2020.

[GLRS26] Paul Gerhart, Davide Li Calsi, Luigi Russo, and Dominique Schröder. Fully-adaptive two-round threshold Schnorr signatures from DDH. In *Advances in Cryptology – EUROCRYPT 2026*, pages 251–283. Springer Nature Switzerland, 2026. IACR Cryptology ePrint Archive, Paper 2025/1478.

[GRJK00] Rosario Gennaro, Tal Rabin, Stanislaw Jarecki, and Hugo Krawczyk. Robust and efficient sharing of RSA functions. *Journal of Cryptology*, 13(2):273–300, March 2000.

[ISN87] M. Ito, A. Saito, and Takao Nishizeki. Secret sharing schemes realizing general access structure. In *Proc. IEEE Global Telecommunication Conf. (Globecom'87)*, pages 99–102, 1987.

[KC23] Carla Kirk-Cohen. LN summit 2023 notes. lightning-dev mailing list, https://www.mail-archive.com/lightning-dev@lists.linuxfoundation.org/msg03262.html, 2023. Section "Multi-Sig Channel Parties".

[KG20a] Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In *Selected Areas in Cryptography (SAC) 2020*, 2020. IACR Cryptology ePrint Archive, Paper 2020/852.

[KG20b] Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 34–65. Springer, Cham, October 2020.

[KG25] Chelsea Komlo and Ian Goldberg. Arctic: Lightweight and stateless threshold Schnorr signatures. In *Public-Key Cryptography (PKC) 2025*, 2025. IACR Cryptology ePrint Archive, Paper 2024/466.

[Koh26] Nadav Kohen. Nested MuSig2. In *Advances in Cryptology – ASIACRYPT 2026*, 2026.

[KOR23] Yashvanth Kondi, Claudio Orlandi, and Lawrence Roy. Two-round stateless deterministic two-party Schnorr signatures from pseudorandom correlation functions. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 646–677. Springer, Cham, August 2023.

[KRT24] Shuichi Katsumata, Michael Reichle, and Kaoru Takemure. Adaptively secure 5 round threshold signatures from MLWE/MSIS and DL with rewinding. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 459–491. Springer, Cham, August 2024.

[Lig25] Lightning Network Specification. BOLT #3: Bitcoin transaction and script formats. https://github.com/lightning/bolts/blob/master/03-transactions.md, 2025.

[LN18] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1837–1854. ACM Press, October 2018.

[lne25] Major exchanges on the lightning network. https://bitcoinmagazine.com/markets/bitcoins-lightning-network-capacity-hits-new-all-time-high, 2025.

[MPSW19] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr multi-signatures with applications to Bitcoin. *DCC*, 87(9):2139–2164, 2019.

[NRJ22] Jonas Nick, Tim Ruffing, and Elliott Jin. BIP 327: MuSig2 for BIP340-compatible multi-signatures. Bitcoin Improvement Proposal, https://github.com/bitcoin/bips/blob/master/bip-0327.mediawiki, 2022.

[NRS21] Jonas Nick, Tim Ruffing, and Yannick Seurin. MuSig2: Simple two-round Schnorr multi-signatures. In *Advances in Cryptology – CRYPTO 2021*, 2021. IACR Cryptology ePrint Archive, Paper 2020/1261.

[NRSW20a] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. In *ACM SIGSAC Conference on Computer and Communications Security (CCS) 2020*, 2020. IACR Cryptology ePrint Archive, Paper 2020/1057.

[NRSW20b] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1717–1731. ACM Press, November 2020.

[NRT26] Guilhem Niot, Michael Reichle, and Kaoru Takemure. Adaptively-secure three-round threshold Schnorr from DL. In *Advances in Cryptology – EUROCRYPT 2026*, pages 3–32. Springer Nature Switzerland, 2026.

[Osu22] Olaoluwa Osuntokun. Simple taproot channels. https://github.com/lightning/bolts/pull/995, 2022. Lightning BOLTs Pull Request #995 (feature 80/81).

[PD16] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf, 2016. DRAFT Version 0.5.9.2, 14 January 2016.

[Pop21] Nathaniel Popper. Lost passwords lock millionaires out of their Bitcoin fortunes. The New York Times, https://www.nytimes.com/2021/01/12/technology/bitcoin-passwords-wallets-fortunes.html, 2021.

[PW23] Jiaxin Pan and Benedikt Wagner. Chopsticks: Fork-free two-round multi-signatures from non-interactive assumptions. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 597–627. Springer, Cham, April 2023.

[PW24] Jiaxin Pan and Benedikt Wagner. Toothpicks: More efficient fork-free two-round multi-signatures. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part I*, volume 14651 of *LNCS*, pages 460–489. Springer, Cham, May 2024.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, New York, August 1990.

[Sed19] Ken Sedgwick. hsmd: Added fields to hsm_sign_remote_commitment_tx to allow complete validation. ElementsProject/lightning Pull Request #3363, https://github.com/ElementsProject/lightning/pull/3363, 2019.

[Seu24] Yannick Seurin. Threshold MuSig2. https://gist.github.com/yannickseurin/c5a76b7180102219c77d4b63fe387445, 2024. Informal note.

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[Shl20] Omer Shlomovits. Battlement: A quorum based design for lightning network key management. https://zengo.com/wp-content/uploads/Battlement.pdf, 2020.

[Sho00] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 207–220. Springer, Berlin, Heidelberg, May 2000.

[SP21] Jacob Swambo and Antoine Poinsot. Risk framework for Bitcoin custody operation with the revault protocol. In Matthew Bernhard, Andrea Bracciali, Lewis Gudgeon, Thomas Haines, Ariah Klages-Mundt, Shin'ichiro Matsuo, Daniel Perez, Massimiliano Sala, and Sam Werner, editors, *FC 2021 Workshops*, volume 12676 of *LNCS*, pages 3–20. Springer, Berlin, Heidelberg, March 2021.

[Ste19a] Chris Stewart. Lightning 101 for exchanges: Private key management. Talk, The Lightning Conference, https://www.youtube.com/watch?v=14GKUfxfPMU&t=1201s, 2019. Transcript: https://btctranscripts.com/lightning-conference/2019/2019-10-19-chris-stewart-private-key-management.

[Ste19b] Chris Stewart. Lightning 101 for exchanges: Security part 3 – private key management. Suredbits blog, https://web.archive.org/web/20240424085231/https://suredbits.com/lightning-101-for-exchanges-security-part-3-private-key-management/, 2019.

[TZ23] Stefano Tessaro and Chenzhi Zhu. Threshold and multi-signature schemes from linear hash functions. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 628–658. Springer, Cham, April 2023.

[WNR20] Pieter Wuille, Jonas Nick, and Tim Ruffing. BIP 340: Schnorr signatures for secp256k1. Bitcoin Improvement Proposal, https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki, 2020.

[WNT20] Pieter Wuille, Jonas Nick, and Anthony Towns. BIP 341: Taproot: SegWit version 1 spending rules. Bitcoin Improvement Proposal, https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki, 2020.

[YCX21] Tsz Hon Yuen, Handong Cui, and Xiang Xie. Compact zero-knowledge proofs for threshold ECDSA with trustless setup. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 481–511. Springer, Cham, May 2021.

[Zmn25] ZmnSCPxj. Flattening nested 2-of-2 of a 1-of-1 and a k-of-n. Delving Bitcoin, https://delvingbitcoin.org/t/flattening-nested-2-of-2-of-a-1-of-1-and-a-k-of-n/2018, 2025.

Ethics Considerations

This paper proposes a nested threshold multi-signature scheme, proves it secure, and evaluates a prototype that integrates with an existing production Lightning implementation. We consider the ethical implications of conducting this research and of publishing its results, following a stakeholder-based analysis.

Stakeholders. The primary stakeholders are the operators of Lightning channel endpoints, from individuals who hold their own keys to exchanges that settle customer deposits over Lightning, and the users whose funds those endpoints hold. Further stakeholders are the maintainers of the Lightning implementations and of the Bitcoin libraries our prototype builds on, and the wider Bitcoin community that decides which channel types to adopt. Indirect stakeholders are adversaries who might study the work to understand the defences of a thresholdized endpoint.

Potential benefits. A Lightning channel holds real funds under a single online key, and both compromise and loss of that key are unrecoverable. Our work removes that single point of failure without asking the counterparty, the protocol or the chain to change, so the protection can be adopted by one side alone. The benefit falls to the party that adopts it and does not depend on anyone else acting.

Potential harms and dual use. Our construction is defensive. It introduces no attack techniques and it exploits no weaknesses of Bitcoin, MuSig2 or the Lightning protocol. The one failure mode we analyse, the leakage of a signing key when one nonce is used for two messages, is a textbook property of Schnorr signatures and is the reason our third design decision exists. We assess the risk that this work enables concrete harm as low, and as outweighed by the benefit of removing an online single point of failure from a system that holds hundreds of millions of dollars.

Research methodology and data handling. Our evaluation runs a prototype in a controlled environment. Both channel state machines run on our own machines against an in-memory database, and no channel we measure is opened on mainnet, carries real funds, or connects to a third party's node. No data from real users is collected, processed or analysed, so the research involves no human subjects, no personal data, and no experiment on a live system.

Decision to publish. We conclude that publication is justified. The work strengthens the custody of a system that is already carrying institutional volumes of value, and it introduces no new offensive capability. We have tried to state the limits of the prototype clearly enough that no reader mistakes it for a production system.

A Deferred Preliminaries

We recall the EUF-CMA security game and pseudocode of NestedMuSig2 [Koh26], its corresponding relevant security results, and the verifiable pseudorandom secret sharing VPSS_1 of Arctic [KG25].

All of the keys at the leaves of a cosigner tree must cooperate in generating partial signatures in order for the aggregate multi-signature to be computed, just as is the case for a non-nested multi-signature scheme. Thus, the unforgeability game for nested multi-signature schemes is derived

directly from the unforgeability game for security under concurrent signing sessions for two-round multi-signatures. Namely, the adversary is given a challenge pk^* and concurrent oracle access to PreRound and SignRound, where it may set the message to be signed, cosigner tree in which the honest signer is embedded, and all aggregate first round messages as the input to its SignRound oracle, and must produce a cosigner tree containing the challenge along with a forgery for the aggregate key on a message not queried to the SignRound oracle under that cosigner tree. This is defined rigorously in Figure 6.

The EUF-CMA security of NestedMuSig2, as defined in Figure 6, is exactly what we reduce the security of Iceberg to in this paper. Under this notion of security, NestedMuSig2 has two security results from [Koh26], one proven in the ROM and the other in the ROM+AGM.

Theorem 4. *Let* GrGen *be a group generation algorithm for which the AOMDL problem is hard. The nested multi-signature scheme* $\mathsf{NestedMuSig2}[\mathsf{GrGen}, \nu = 8, D = 2]$ *is EUF-CMA in the random oracle model for* $\mathsf{H}_{\mathrm{agg}}$, $\mathsf{H}_{\mathrm{non}}$, $\mathsf{H}_{\overline{\mathrm{non}}}$, $\mathsf{H}_{\mathrm{sig}} : \{0,1\}^* \to \mathbb{Z}_p$.

Precisely, for any adversary $\mathcal{A}$ *against* $\mathsf{NestedMuSig2}[\mathsf{GrGen}, \nu = 8]$ *running in time at most* τ, *making at most* q_s OSignRound *queries and at most* q_h *queries to each random oracle, and such that the size of* L_d *in any signing session and in the forgery is at most* N, *and the maximum depth of any signing session is at most* 2, *there exists an algorithm* $\mathcal{E}$ *taking as input group parameters* $(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$, *running in time at most*

$$\tau' = 8(\tau + q(3N + 3\nu - 3))\tau_{\exp} + O(qN)$$

where $q = (N+4)q_h + (2N+3)q_s + 2N + 1$ *and* $\tau_{\exp}$ *is the time of an exponentiation in* $\mathbb{G}$, *making at most* $8q_s$ DLog_g *queries, and solving the AOMDL problem with an advantage*

$$\mathsf{Adv}^{\mathsf{AOMDL}}_{\mathcal{E},\mathsf{GrGen}}(\lambda) \geq \frac{\epsilon^8}{16q^7} - \frac{16q + 2\binom{8q+2}{2}^2 + 12}{2^\lambda},$$

where $\epsilon = \mathsf{Adv}^{\text{EUF-CMA}}_{\mathcal{A},\mathsf{NestedMuSig2}[\mathsf{GrGen},\nu=8,D=2]}(\lambda)$.

Theorem 5. *Let* GrGen *be a group generation algorithm for which the AOMDL problem is hard. Then the multi-signature scheme* $\mathsf{NestedMuSig2}[\mathsf{GrGen}, \nu = 2, D = 2]$ *is EUF-CMA in the algebraic group model for* GrGen, *and the random oracle model for* $\mathsf{H}_{\mathrm{agg}}$, $\mathsf{H}_{\mathrm{non}}$, $\mathsf{H}_{\overline{\mathrm{non}}}$, $\mathsf{H}_{\mathrm{sig}} : \{0,1\}^* \to \mathbb{Z}_p$.

Precisely, for any algebraic adversary $\mathcal{A}$ *against* $\mathsf{NestedMuSig2}[\mathsf{GrGen}, \nu = 2]$ *running in time at most* τ, *making at most* q_s OSignRound *queries, at most* q_h *queries to each random oracle, such that the size of* L *in any signing session and in the forgery is at most* N, *and such that the depth,* Λ, *of any signing session or forgery is at most* 2, *there exists an algorithm* $\mathcal{B}$ *running in time at most*

$$\tau' = \tau + O(qN) \cdot \tau_{\exp} + O(q^2),$$

where $q = \binom{2q_h+2q_s+2}{2} + 2N(q_s+1) + Nq_h$ *and* $\tau_{\exp}$ *is the time of an exponentiation in* $\mathbb{G}$ *and making at most* $2q_s$ DLog_g *queries such that*

$$\mathsf{Adv}^{\mathsf{AOMDL}}_{\mathcal{B},\mathsf{GrGen}}(\lambda) \geq \mathsf{Adv}^{\text{EUF-CMA}}_{\mathcal{A},\mathsf{NestedMuSig2}[\mathsf{GrGen},\nu=2]}(\lambda) - \frac{30q^4}{2^\lambda}.$$

Lastly, in Figure 8 below, we recall the definition of the Verifiable Pseudorandom Secret Sharing function VPSS_1 as defined in [KG25]. This protocol allows a group of n parties with an honest majority to perform replicated secret sharing on pseudorandom function seeds to generate an arbitrary

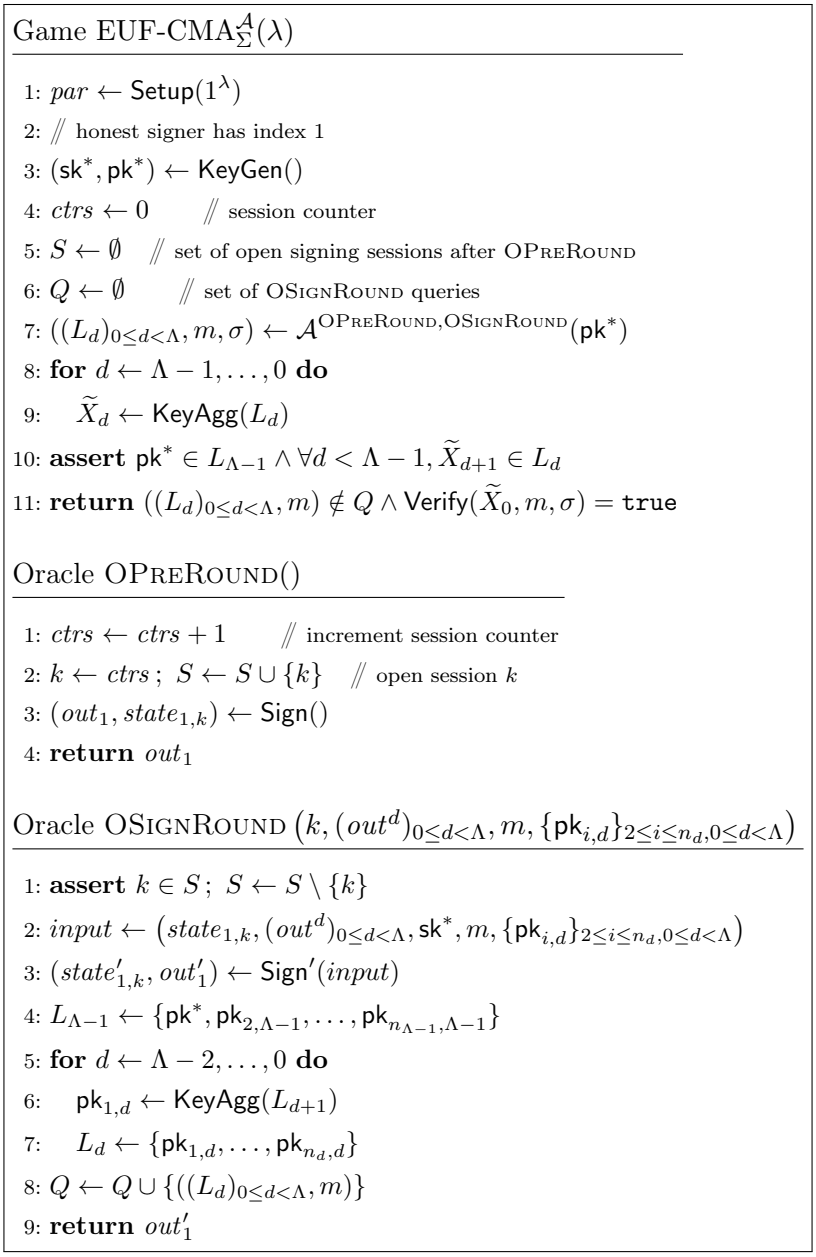

Figure 6: The EUF-CMA security game for a two-round nested multi-signature scheme Σ, adapted from [Koh26].

number of (replicated) shared secrets. Share conversion (introduced in [CDI05]) is then used to turn these replicated secret shares into Shamir secret shares locally without any communication required. In the honest-majority setting, VPSS_1 allows parties to verify function outputs in aggregate by performing verification "in the exponent" and using the fact that an honest majority implies that any honest set of t parties fully determine an interpolated curve of degree $t-1$ so that interpolating a higher degree curve proves dishonest activity has occured.

B Additional Figures

$\mathsf{Setup}(1^\lambda)$

$par \leftarrow (\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$
return par

$\mathsf{KeyGen}(n, t, \mu)$

assert $\mu \geq 2t - 1$
$\mathbf{A} \leftarrow \binom{[n]}{t-1}$
for $\mathbf{a} \in \mathbf{A}$ **do**
  $\phi_{\mathbf{a}} \leftarrow\!\!\$\ \mathbb{Z}_p$ // $\mathsf{sk} = \sum_{\mathbf{a}\in\mathbf{A}} \phi_{\mathbf{a}}$
for $k \in [n]$ **do**
  $\mathsf{sk}_k \leftarrow \{(\mathbf{a}, \phi_{\mathbf{a}})\}_{\mathbf{a}\in\mathbf{A}_k}$
return $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ // $|\mathsf{sk}_k| = \binom{n-1}{t-1}$

$\mathsf{Gen}(k, \mathsf{sk}_k, w)$

$\{(\mathbf{a}, \phi_{\mathbf{a}})\}_{\mathbf{a}\in\mathbf{A}_k} \leftarrow \mathsf{sk}_k$
$d_k \leftarrow (d_{k,\mathbf{a}})_{\mathbf{a}\in\mathbf{A}_k} \leftarrow (\mathsf{H}_{\mathrm{prf}}(\phi_{\mathbf{a}}, w))_{\mathbf{a}\in\mathbf{A}_k}$
$D_k \leftarrow (D_{k,\mathbf{a}})_{\mathbf{a}\in\mathbf{A}_k} \leftarrow (g^{d_{k,\mathbf{a}}})_{\mathbf{a}\in\mathbf{A}_k}$
return (d_k, D_k)

$\mathsf{VPSS}_0.\mathsf{Verify}(t, \mu, C, \{D_j\}_{j\in C})$

assert $|C| \geq \mu$
for $j \in C$ **do**
  $(D_{j,\mathbf{a}})_{\mathbf{a}\in\mathbf{A}_j} \leftarrow D_j$
if $\forall \mathbf{a} \in \mathbf{A}, \left|\{D_{j,\mathbf{a}}\}_{j\in C\setminus\mathbf{a}}\right| = 1$ **then**
  return 1
else return 0

$\mathsf{Agg}(t, \mu, C, \{D_j\}_{j\in C})$

for $\mathbf{a} \in \mathbf{A}$ **do**
  $\{D'_{\mathbf{a}}\} \leftarrow \{D_{j,\mathbf{a}}\}_{j\in C\setminus\mathbf{a}}$
return $D \leftarrow \prod_{\mathbf{a}\in\mathbf{A}} D'_{\mathbf{a}}$

$\mathsf{Recover}(t, \mu, C, \{d_j\}_{j\in C})$

assert $|C| \geq \mu \wedge C \subseteq [n]$
for $j \in C$ **do**
  $(d_{j,\mathbf{a}})_{\mathbf{a}\in\mathbf{A}_j} \leftarrow d_j$
  for $\mathbf{a} \in \mathbf{A}_j$ **do**
    $D_{j,\mathbf{a}} \leftarrow g^{d_{j,\mathbf{a}}}$
  $D_j \leftarrow (D_{j,\mathbf{a}})_{\mathbf{a}\in\mathbf{A}_j}$
assert $\mathsf{VPSS}_0.\mathsf{Verify}(t, \mu, C, \{D_j\}_{j\in C})$
for $\mathbf{a} \in \mathbf{A}$ **do**
  $\{d'_{\mathbf{a}}\} \leftarrow \{d_{j,\mathbf{a}}\}_{j\in C\setminus\mathbf{a}}$
$d \leftarrow \sum_{\mathbf{a}\in\mathbf{A}} d'_{\mathbf{a}}\,;\ D \leftarrow g^d$
return (d, D)

Figure 7: The protocol VPSS_0, a variant of VPSS_1 without share conversion. Notation for subsets is as in Figure 8.

$\mathsf{Setup}(1^\lambda)$

1: $par \leftarrow (\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$
2: **return** par

$\mathsf{KeyGen}(n, t, \mu)$

1: **assert** $\mu \geq 2t - 1$
2: $\mathbf{A} \leftarrow \binom{[n]}{t-1}$
3: **for** $\mathbf{a} \in \mathbf{A}$ **do**
4: $\quad \phi_{\mathbf{a}} \leftarrow\!\$\ \mathbb{Z}_p \quad /\!/\ \mathsf{sk} = \sum_{\mathbf{a}\in\mathbf{A}} \phi_{\mathbf{a}}$
5: **for** $k \in [n]$ **do**
6: $\quad \mathsf{sk}_k \leftarrow \{(\mathbf{a}, \phi_{\mathbf{a}})\}_{\mathbf{a}\in\mathbf{A}_k}$
7: **return** $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n) \quad /\!/\ |\mathsf{sk}_k| = \binom{n-1}{t-1}$

$\mathsf{Gen}(k, \mathsf{sk}_k, w)$

1: $\{(\mathbf{a}, \phi_{\mathbf{a}})\}_{\mathbf{a}\in\mathbf{A}_k} \leftarrow \mathsf{sk}_k$
2: **for** $\mathbf{a} \in \mathbf{A}_k$ **do**
3: $\quad L'_{\mathbf{a}}(x) \leftarrow \prod_{j\in\mathbf{a}} \frac{j-x}{j}$
4: $/\!/$ the values $L'_{\mathbf{a}}(k)$ can be precomputed
5: $d_k \leftarrow \sum_{\mathbf{a}\in\mathbf{A}_k} \mathsf{H}_{\mathrm{prf}}(\phi_{\mathbf{a}}, w) \cdot L'_{\mathbf{a}}(k)$
6: $D_k \leftarrow g^{d_k}$
7: **return** (d_k, D_k)

$\mathsf{VPSS}_1.\mathsf{Verify}(t, \mu, C, \{D_j\}_{j\in C})$

1: **assert** $|C| \geq \mu$
2: **for** $i \leftarrow t, \ldots, |C| - 1$ **do**
3: $\quad$ **for** $j \in C$ **do**
4: $\quad\quad L_j(x) \leftarrow \prod_{k\in C\setminus\{j\}} \frac{x-k}{j-k}$
5: $\quad\quad \lambda_{j,i} \leftarrow [x^i](L_j(x))$
6: $\quad B_i \leftarrow \prod_{j\in C} D_j^{\lambda_{j,i}}$
7: $\quad$ **if** $B_i \neq 1_{\mathbb{G}}$ **then return** 0
8: **return** 1

$\mathsf{Agg}(t, \mu, C, \{D_j\}_{j\in C})$

1: **for** $j \in C$ **do**
2: $\quad \lambda_j \leftarrow \prod_{k\in C\setminus\{j\}} \frac{k}{k-j}$
3: **return** $D \leftarrow \prod_{j\in C} D_j^{\lambda_j}$

$\mathsf{Recover}(t, \mu, C, \{d_j\}_{j\in C})$

1: **assert** $|C| \geq \mu \wedge C \subseteq [n]$
2: **for** $j \in C$ **do**
3: $\quad D_j \leftarrow g^{d_j}$
4: $\quad \lambda_j \leftarrow \prod_{k\in C\setminus\{j\}} \frac{k}{k-j}$
5: **assert** $\mathsf{VPSS}_1.\mathsf{Verify}(t, \mu, C, \{D_j\}_{j\in C})$
6: $d \leftarrow \sum_{j\in C} d_j \cdot \lambda_j\ ;\ D \leftarrow g^d$
7: **return** (d, D)

Figure 8: The protocol VPSS_1 from Arctic [KG25]. We write $\mathbf{A} = \binom{[n]}{t-1}$ for the family of all $(t-1)$-subsets of $[n]$ and $\mathbf{A}_k = \{\mathbf{a} \in \mathbf{A} : k \notin \mathbf{a}\}$ for the subsets whose shares member k holds.

$\mathsf{Setup}(1^\lambda)$

$par \leftarrow (\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$
return par

$\mathsf{PreRound}(k, \mathsf{sk}_k, sid)$

// sid is assumed to have a fixed length
for $i \in [\nu]$ **do**
  $(r_{i,k}, R_{i,k}) \leftarrow \mathsf{VPSS}_0.\mathsf{Gen}(k, \mathsf{sk}_k, i \| sid)$
return $(R_{1,k}, \ldots, R_{\nu,k})$

$\mathsf{KeyGen}(k, n, t, A)$

assert $n \geq 2t-1$
$\mathsf{sk}_k \leftarrow \mathsf{VPSS}_0.\mathsf{KeyGen}(n, t, 2t-1)[k]$
$(x_k, X_k) \leftarrow \mathsf{VPSS}_0.\mathsf{Gen}(k, \mathsf{sk}_k, w_0)$
$(C, \{X_j\}_{j \in C}) \leftarrow A(X_k)$
assert $\mathsf{VPSS}_0.\mathsf{Verify}(t, 2t-1, C, \{X_j\}_{j \in C}) = 1$
$X \leftarrow \mathsf{VPSS}_0.\mathsf{Agg}(t, 2t-1, C, \{X_j\}_{j \in C})$
return (X, sk_k)

$\mathsf{PreAgg}(\mathsf{pk}, C, (R_{i,j})_{i \in [\nu], j \in C})$

for $i \in [\nu]$ **do**
  assert $\mathsf{VPSS}_0.\mathsf{Verify}(t, 2t-1, C, \{R_{i,j}\}_{j \in C}) = 1$
  $R'_i \leftarrow \mathsf{VPSS}_0.\mathsf{Agg}(t, 2t-1, C, \{R_{i,j}\}_{j \in C})$
$b_1 \leftarrow \mathsf{H}_{\mathrm{non}}(\mathsf{pk}, (R'_1, \ldots, R'_\nu))$
for $i \in [\nu]$ **do**
  $R_i \leftarrow (R'_i)^{b_1^{i-1}}$
return $(R_1, \ldots, R_\nu)$

$\mathsf{SignAgg}(R, C, \{s_k\}_{k \in C})$

// R can be computed from protocol
// messages as in SignRound
for $k \in C$ **do**
  $(s_{k,\mathbf{a}})_{\mathbf{a} \in \mathbf{A}_k} \leftarrow s_k$
for $\mathbf{a} \in \mathbf{A}$ **do**
  $\{s'_{\mathbf{a}}\} \leftarrow \{s_{k,\mathbf{a}}\}_{k \in C \setminus \mathbf{a}}$
$s \leftarrow \sum_{\mathbf{a} \in \mathbf{A}} s'_{\mathbf{a}}$
return (R, s)

$\mathsf{SignRound}(k, \mathsf{sk}_k, \mathsf{pk}, sid, m, (R'_i)_{i \in [\nu]}, \overline{j}, \{\overline{\mathsf{pk}}_i\}_{i \in [\overline{n}] \setminus \{\overline{j}\}}, (\overline{R}_i)_{i \in [\nu]})$

$(x_{k,\mathbf{a}}, X_{k,\mathbf{a}})_{\mathbf{a} \in \mathbf{A}_k} \leftarrow \mathsf{VPSS}_0.\mathsf{Gen}(k, \mathsf{sk}_k, w_0)$
for $i \in [\nu]$ **do**
  $(r_{i,k}, R_{i,k}) \leftarrow \mathsf{VPSS}_0.\mathsf{Gen}(k, \mathsf{sk}_k, i \| sid)$
  $(r_{i,k,\mathbf{a}})_{\mathbf{a} \in \mathbf{A}_k} \leftarrow r_{i,k}$
$\overline{\mathsf{pk}}_{\overline{j}} \leftarrow \mathsf{pk}$; $L \leftarrow (\overline{\mathsf{pk}}_j)_{j \in [\overline{n}]}$
$\widetilde{X} \leftarrow \mathsf{KeyAgg}(L)$; $a \leftarrow \mathsf{KeyAggCoef}(L, \mathsf{pk})$
$b_1 \leftarrow \mathsf{H}_{\mathrm{non}}(\mathsf{pk}, (R'_1, \ldots, R'_\nu))$; $b_0 \leftarrow \mathsf{H}_{\overline{\mathrm{non}}}(\widetilde{X}, (\overline{R}_1, \ldots, \overline{R}_\nu), m)$
$\breve{b} \leftarrow b_1 \cdot b_0$; $R \leftarrow \prod_{i \in [\nu]} \overline{R}_i^{b_0^{i-1}}$; $c \leftarrow \mathsf{H}_{\mathrm{sig}}(\widetilde{X}, R, m)$
for $\mathbf{a} \in \mathbf{A}_k$ **do**
  $s_{k,\mathbf{a}} \leftarrow \sum_{i \in [\nu]} r_{i,k,\mathbf{a}} \breve{b}^{i-1} + c \cdot a \cdot x_{k,\mathbf{a}}$
return $s_k \leftarrow (s_{k,\mathbf{a}})_{\mathbf{a} \in \mathbf{A}_k}$

Figure 9: The protocol $\mathsf{Iceberg}_{\mathrm{base}}[\mathsf{GrGen}, \nu, t]$. Changes compared to $\mathsf{Iceberg}[\mathsf{GrGen}, \nu, t]$ are highlighted. Notation for subsets is as in Figure 8.

$\mathsf{Setup}(1^\lambda)$

1: $(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$
2: Select four hash functions
3: $\mathsf{H}_{\mathrm{agg}}, \mathsf{H}_{\mathrm{non}}, \mathsf{H}_{\overline{\mathrm{non}}}, \mathsf{H}_{\mathrm{sig}} : \{0,1\}^* \to \mathbb{Z}_p$
4: $par \leftarrow$
5: $\quad ((\mathbb{G}, p, g), \mathsf{H}_{\mathrm{agg}}, \mathsf{H}_{\mathrm{non}}, \mathsf{H}_{\overline{\mathrm{non}}}, \mathsf{H}_{\mathrm{sig}})$
6: **return** par

$\mathsf{KeyGen}()$

1: $x \leftarrow\!\!\$\ \mathbb{Z}_p\,;\ X \leftarrow g^x$
2: $\mathsf{sk} \leftarrow x\,;\ \mathsf{pk} \leftarrow X$
3: **return** $(\mathsf{sk}, \mathsf{pk})$

$\mathsf{KeyAggCoef}(L, X_i)$

1: **return** $\mathsf{H}_{\mathrm{agg}}(L, X_i)$

$\mathsf{KeyAgg}(L)$

1: $\{X_1, \ldots, X_n\} \leftarrow L$
2: **for** $i \leftarrow 1 \ldots n$ **do**
3: $\quad a_i \leftarrow \mathsf{KeyAggCoef}(L, X_i)$
4: **return** $\widetilde{X} \leftarrow \prod_{i=1}^n X_i^{a_i}$

$\mathsf{Verify}(\widetilde{X}, m, \sigma)$

1: $(R, s) \leftarrow \sigma$
2: $c \leftarrow \mathsf{H}_{\mathrm{sig}}(\widetilde{X}, R, m)$
3: **return** $(g^s = R\widetilde{X}^c)$

$\mathsf{Sign}()$

1: // Local signer has index 1.
2: **for** $j \leftarrow 1 \ldots \nu$ **do**
3: $\quad r_{1,j} \leftarrow\!\!\$\ \mathbb{Z}_p\,;\ R_{1,j} \leftarrow g^{r_{1,j}}$
4: $out_1 \leftarrow (R_{1,1}, \ldots, R_{1,\nu})$
5: $state_1 \leftarrow (r_{1,1}, \ldots, r_{1,\nu})$
6: **return** $(out_1, state_1)$

$\mathsf{SignAgg}(out_1, \ldots, out_n)$

1: **for** $i \leftarrow 1 \ldots n$ **do**
2: $\quad (R_{i,1}, \ldots, R_{i,\nu}) \leftarrow out_i$
3: **for** $j \leftarrow 1 \ldots \nu$ **do**
4: $\quad R_j \leftarrow \prod_{i=1}^n R_{i,j}$
5: **return** $(R_1, \ldots, R_\nu)$

$\mathsf{SignAggExt}(out, \widetilde{X})$

1: $(R'_1, \ldots, R'_\nu) \leftarrow out$
2: $b \leftarrow \mathsf{H}_{\mathrm{non}}(\widetilde{X}, (R'_1, \ldots, R'_\nu))$
3: **for** $j \leftarrow 1 \ldots \nu$ **do**
4: $\quad R_j \leftarrow (R'_j)^{b^{j-1}}$
5: **return** $(R_1, \ldots, R_\nu)$

$\mathsf{Sign}'(state_1, (out^d)_{0 \le d < \Lambda}, \mathsf{sk}_1, m, \{\mathsf{pk}_{i,d}\}_{2 \le i \le n_d, 0 \le d < \Lambda})$

1: // Sign' must be called at most once per $state_1$.
2: $(r_{1,1}, \ldots, r_{1,\nu}) \leftarrow state_1\,;\ \mathsf{pk}_{1,\Lambda-1} \leftarrow g^{\mathsf{sk}_1}$
3: $L_{\Lambda-1} \leftarrow \{\mathsf{pk}_{1,\Lambda-1}, \ldots, \mathsf{pk}_{n_{\Lambda-1},\Lambda-1}\}$
4: $a_{1,\Lambda-1} \leftarrow \mathsf{KeyAggCoef}(L_{\Lambda-1}, \mathsf{pk}_{1,\Lambda-1})$
5: $\mathsf{pk}_{1,\Lambda-2} \leftarrow \mathsf{KeyAgg}(L_{\Lambda-1})$
6: **for** $d \leftarrow \Lambda - 2, \ldots, 0$ **do**
7: $\quad b_{d+1} \leftarrow \mathsf{H}_{\mathrm{non}}(\mathsf{pk}_{1,d}, out^{d+1})$
8: $\quad L_d \leftarrow \{\mathsf{pk}_{1,d}, \ldots, \mathsf{pk}_{n_d,d}\}$
9: $\quad a_{1,d} \leftarrow \mathsf{KeyAggCoef}(L_d, \mathsf{pk}_{1,d})$
10: $\quad \mathsf{pk}_{1,d-1} \leftarrow \mathsf{KeyAgg}(L_d)$
11: $\widetilde{X} \leftarrow \mathsf{pk}_{1,-1}\,;\ (R_{1,0}, \ldots, R_{\nu,0}) \leftarrow out^0$
12: $b_0 \leftarrow \mathsf{H}_{\overline{\mathrm{non}}}(\widetilde{X}, (R_{1,0}, \ldots, R_{\nu,0}), m)$
13: $R \leftarrow \prod_{j=1}^\nu R_{j,0}^{b_0^{j-1}}\ ;\ \check{b} \leftarrow \prod_{\ell=0}^{\Lambda-1} b_\ell$
14: $c \leftarrow \mathsf{H}_{\mathrm{sig}}(\widetilde{X}, R, m)\,;\ \check{c} \leftarrow c \prod_{\ell=0}^{\Lambda-1} a_{1,\ell}$
15: $s_1 \leftarrow \check{c}\mathsf{sk}_1 + \sum_{j=1}^\nu r_{1,j}\check{b}^{j-1} \bmod p$
16: $state'_1 \leftarrow R\,;\ out'_1 \leftarrow s_1$
17: **return** $(state'_1, out'_1)$

$\mathsf{SignAgg}'(out'_1, \ldots, out'_n, state'_1)$

1: $R \leftarrow state'_1$
2: $(s_1, \ldots, s_n) \leftarrow (out'_1, \ldots, out'_n)$
3: $s \leftarrow \sum_{i=1}^n s_i \bmod p$
4: **return** $\sigma \leftarrow (R, s)$

Figure 10: The nested multi-signature scheme $\mathsf{NestedMuSig2}[\mathsf{GrGen}, \nu]$ [Koh26]. Public parameters par returned by Setup are implicitly given as input to all other algorithms.