One-Step Schnorr Threshold Identification

Foteinos Mergoupis-Anagnou

GRNET, Athens, Greece, fmerg@grnet.gr

Abstract. Threshold cryptographic primitives have not been widely adopted beyond the closed committee model, presumably due to state-synchronization overhead and complex certification processes for the shareholders. These are both aspects of their overreliance on infrastructure, an assumption that is usually glossed over in their design. Certifying individual identities is at odds with proactive sharing, just as interacting upon every session conflicts with open or cross-domain environments; indeed, in such settings, distributed identity is more naturally modelled as a fluid group of shareholders that need not interact continuously and can reshare occasionally without need for external entities to take notice. In this work, we investigate a plausible approach to threshold identification that achieves non-interactivity and non-reliance on advertised shares by having the provers attach them to ordinary Schnorr proofs on the fly. Given a Shamir (n, t)-shared secret x, the resulting protocol allows any $t^* \geq t$ (but no less) shareholders to prove over designated communication channels that their keys interpolate to x. Provers do not engage in multi-party computations, sending their packets to the verifier asynchronously; conversely, verifiers need only know the combined public key $y \equiv g^x$, without need to pre-validate and register the individual member identities. The protocol is intended for use in permissionless or unmanaged meshes that lack both overlay networks and trust infrastructure or governance, a use case space that has received little attention. No verifiable setup is required beyond distributing x according to Shamir's sharing (or equivalent distributed key generation scheme) and advertising its public counterpart; in particular, the protocol is intended to be secure against impersonation attacks without relying on the auditability of any advertised shares. We provide evidence that this has good chances to hold true by giving a formal security proof in the random oracle model under the discrete-logarithm (DL) hardness assumption.

1 Introduction

Distributing a signature scheme is usually achieved through a trusted combiner that operates as group proxy. Upon every session, the shareholders engage in the multi-party computation of a commitment, which they use to generate ordinary signatures with their respective secret keys. After aggregating these packets, the proxy combines them into an ordinary signature and forwards it to the verifier, who proceeds to verification like in the non-distributed case. The idea stems from [43] and has been elaborated in the proof-of-knowledge context by [26], who apply it to distribute protocols as general as [28]. By distributing the prover of the Schnorr protocol *and* requiring the verifier to play the role of combiner, we get a real-time threshold identification scheme.

While it does not require pre-validation of public shares, this protocol is highly interactive on the provers' side due to the multi-party computation of the commitment, usually some distributed key generation (DKG) scheme (cf. [15], [14]). This is a bandwidth greedy and

memory intensive operation with increased message and space complexity, heavily relying on stable and optimized connections among the provers. Standardized techniques for reducing bandwidth usage or the number of rounds (e.g., a FROST-like proving phase, cf. [27]) require infrastructure support and extended governance. While this may not be an issue for closed committees, it makes interactivity prohibitive in open or unmanaged meshes, or under volatile intra-group network connections. In such settings, the shareholders probably cannot —or should not need to— coordinate unconditionally, since they have limited means of continuously discovering each other and no managing authority ensures that they are simultaneously online. Intra-group interaction should only be an option for occasionally synchronizing states per the specific needs of a use case rather than a baseline operational requirement; instead, asynchronicity should be thought of per se as an operational advantage. Requiring the provers to be simultaneously online and state-synchronized is a constraint that might be antithetical or irrelevant to the use case, just as joining a session should not require to participate in its initiation and bind to the same commitment as everybody.

Non-interactivity can be trivially achieved if the public shares are advertised and trusted as a fixed sharing of the group public key. In this case, the provers send ordinary Schnorr proofs to the verifier, who validates them against the respective public shares and accepts only if their number is at least equal to the threshold parameter. This scheme is asynchronous and requires no infrastructure support on the provers' side; however, it requires the verifier to have pre-validated and registered the correct public shares in order to reliably verify the individual proof packets. This evidently implies the need for an auditable setup phase attesting to the consistency of a fixed set of shares, otherwise the protocol becomes susceptible to rogue-key attacks. If reliance on authorities is not desired, auditability becomes equivalent to persistent bulletin boards for advertising the transcript of the key setup process, allowing verifiers to anytime validate the claimed public shares against the group public key. Beyond scalability issues, security implications and the infrastructural costs of governance, such solutions require specifications for interoperability and have to address the continuous administrative burden of proactive sharing, which would require updating the boards and repeating validations upon every epoch from scratch. Finally, this scheme is essentially an access control mechanism for onboarding named shareholders through individual authentication —a problem much narrower than identifying a fluid group of shareholders in the permissionless setting. Routinely reducing individual accountability to rigid identities conflicts with the long-term security precautions of threshold cryptography, thereby confining its applications to ephemeral or siloed environments; indeed, as long as universal share certification remains impractical, proactive sharing suggests that distributed identity in open or cross-domain settings is more naturally modelled as a group that reshares internally, while its combined public key remains invariant and external actors need not take notice.

One could circumvent pre-validation by having the provers attach their public shares to the respective proofs on the fly; in this case, the verifier accepts only if every proof verifies and the attached elements interpolate to the group public key. Security reduces intuitively to the knowledge property of the Schnorr protocol. Every prover "should know" the logarithm of its attached element because the respective proof verifies; since the attached shares interpolate to the group public key, their logarithms interpolate to its secret counterpart and the provers constitute thus a qualified coalition. The scheme is non-interactive and non-reliant on advertised shares, but the knowledge argument is not as sound as it might seem; in fact, it is susceptible to a birthday attack whose cost decreases with the size of the impersonated coalition. Hashing every attached share along with the respective commitment restores the knowledge argument and excludes this attack, yielding the protocol proposed in this work.

1.1 Contribution

This discussion indicates a tension between non-interactivity on the provers' side and non-reliance on the public shares; eliminating infrastructure on one side requires its existence and maintenance on the other.[1] In this work, we propose a Schnorr-based threshold identification protocol that achieves both properties[2] by having the provers attach their public shares to ordinary proofs, each bound into the respective challenge, rather than advertise them in bulletin boards or have them certified. The protocol is intended for use cases where both the administrative overhead of certifying public shares and the engineering investment for reducing per-session interactivity are cost-ineffective or prohibitive; this includes edge cases where asynchronous action or operability under absent intra-group connectivity is prioritized and even advertising individual keys is impractical, covering cross-domain scenarios where provers may operate on platforms alien to the verifier's ecosystem. No auditable key setup is required beyond sharing a secret key according to some Shamir-based scheme (no matter which) and correctly advertising its public counterpart; in particular, the protocol is intended to be secure against impersonation attacks in the share-free model. We provide evidence that this has good chances to hold true by giving a formal security proof in the random oracle model under the DL hardness assumption; the proof extends to adversaries who observe transcripts of sessions other than the attacked one, with respect to a variant that mitigates replay attacks and is the one intended for deployment (cf. Section 3.1.1).

The construction combines standard ingredients, namely Schnorr proofs with Fiat-Shamir transform in its strong variant and the interpolation of public shares in the exponent; to the best of our knowledge, it has not been analysed in the share-free setting. Its security proof is likewise standard, extracting the logarithms of all attached shares by forking at multiple points (cf. [3]). We also analyse its compact variant, which dispenses with the attached shares by interpolating the proofs directly (cf. Section 3.1.2); this variant saves one group element per prover, but is subject to a generalized birthday attack whose cost decreases with the number of components, so that it should be used only under a small bound enforced by the verifier. It should be stressed, however, that both security results are asymptotic; moreover, that of the compact variant is established for every fixed bound on the number of components that the verifier accepts per proof, which the main protocol does not require.

1.2 Organization

In Section 2, we investigate the overlooked area of real-time thresholds along with the subtleties of their application. In Section 3, we introduce the protocol, examine its properties and make some security considerations, along with presenting its session-bound and compact variants (cf. Sections 3.1.1 and 3.1.2). In Section 4, we stipulate the relevant threat model and derive the corresponding attack games. We do not formulate zero-knowledge notions

[1] Proofs of partial knowledge [13] and threshold ring signatures [12] do not resolve this tension either; they prove knowledge of t among n independent secret keys against public keys that the verifier must trust and register, while provers holding distinct keys must agree on their individual challenges before responding.

[2] Bilinear threshold protocols (namely threshold BLS, cf. [11]) attain non-interactivity and share-freeness as well, and further compactness, the partial signatures interpolating into an ordinary signature that is verified against the combined public key alone; moreover, their security reduction requires no rewinding. However, they rest on gap Diffie-Hellman assumptions in pairing-friendly groups and are thereby incompatible with the widely deployed DL-based infrastructure; the key sizes of these groups had to be revised upwards after advances in the number field sieve (cf. [4]), and their operations are costlier on both the provers' and the verifier's side, verification further involving a fixed number of pairings. The present work seeks a solution within the pure discrete-logarithm setting, i.e., over any group of prime order where DL is hard, at the cost of proofs linear in the number of provers and of a reduction by forking, which is not tight (cf. Remark 5.20).

for the threshold setting, focusing directly on security against impersonation attacks; we also consider adversaries who observe transcripts of other sessions, with respect to the session-bound variant of the protocol. In Section 5, we provide a formal security proof in the random oracle model under the DL hardness assumption, extracting the logarithms of the attached shares one at a time; for the compact variant, the logarithms of the commitments have to be eliminated jointly (cf. Appendix D).

2 Survey on real-time threshold identification

Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order q, such that the discrete-logarithm (DL) hardness assumption holds true. Let a secret key $x \in \mathbb{Z}_q$ be distributed as $x_1, \ldots, x_n$ among a group of shareholders by means of Shamir's (n, t)-sharing, $1 \leq t \leq n$, or equivalent DKG scheme (cf. Section 2.1.1). That is, it should not matter if x is shared by a dealer or implicitly defined in a decentralized fashion; it suffices that the output of the distribution is computationally indistinguishable from that of Shamir's secret sharing. We are interested in a protocol that allows any $t^* \geq t$ (but no less) shareholders to dynamically prove that they belong to the group entity represented by $y \equiv g^x$. More accurately, the involved parties should be able to collectively prove in a real-time session bound to designated communication channels separately opened to the verifier (cf. Remark 2.1) that their secret shares combine to x in the sense of interpolation (cf. Remark B.3).

In this section, we establish a conceptual framework for analysing the tradeoffs of real-time thresholds, arguing that native and system-agnostic schemes should be share-free and non-interactive. As already hinted, the necessity of share-freeness follows from the impracticality of trusted share governance and is inherent to the permissionless setting, i.e., the natural setting of identification. This setting implies that a particular denial-of-service is endemic to share-free thresholds, even if these are interactive (e.g., adapted Schnorr threshold signatures); in fact, non-interactive thresholds have better chances to be resilient against this type of attack exactly due to their asynchronous nature. Finally, we observe that both interactivity and share-reliance can be overcome by having the provers attach their public shares to asynchronously generated proofs, provided that the shares are bound into the respective challenges (cf. Section 2.9.4).

Remark 2.1. (*Security against impersonation attacks*) The constraint of separate channels is essential. Whether or not they need to collaborate, the provers finally have to send their proof packets through the designated channels that they have separately opened to the verifier. This ensures that the verifier interacts with the actual group members that appear live over the network (i.e., at most through generic relay nodes) and not with a delegated group proxy that could instead present a combined signature bound to a session nonce (cf. Section 2.4). Security against impersonation attacks means that any probabilistic polynomial-time adversary controlling up to $t - 1$ shareholders remains unable to execute the protocol and cause the verifier to accept with non-negligible advantage (cf. Section 4.1).

2.1 Group constitution

The elementary way to create a (n, t)-threshold group, $1 \leq t \leq n$, is when a single dealer distributes a pre-existing key by means of Shamir's secret sharing (cf. Appendix B), i.e., a centralized operation of the form

$$(x_1, \ldots, x_n; y) \leftarrow D(g, x)\,. \tag{2.2}$$

This corresponds to a scenario where the key holder splits x across many devices for the purpose of distributed custody, or in order to secure it against loss or a single point of compromise; more generally, it indicates a redundancy or decentralized access mechanism, with the level of trust upon the dealer depending on its exact role within an organization. In case of limited trust, the shareholders must verify that their received shares are consistent, i.e., that they constitute a (n, t)-sharing of $x = \log y$. This is attained by extending raw sharing into a *verifiable* secret sharing (VSS) scheme ([17], [35]), i.e., by having the dealer commit to the interpolating polynomial and broadcast the commitments, incurring $O(n)$ inbound bandwidth and $O(nt)$ local verification costs per shareholder.

2.1.1 Distributed key generation (DKG)

In several cases, the group secret key must be implicitly defined in a decentralized fashion; i.e., it should not be known to any single dealer (or reconstructible at any single location), with the shareholders contributing to it equally. This is attained through the homomorphic property of Shamir's secret sharing (cf. Remark B.6). Every group participant splits some uniformly random secret across its peers (including itself) and sums the shares received from the others, with the local summation process yielding its final secret share; after all broadcasts have been completed, the peers are able to infer the same group public key y, but x remains completely unknown to everybody. In case of limited trust, these sharings must be VSS-backed, so that everyone can verify the correctness of their received packets. The resulting scheme is called a (n, t)-*distributed key generation* (DKG) scheme and is equivalent to Shamir's sharing in the sense that the probability distribution of its output shares is computationally indistinguishable from the output distribution of (2.2) with $x \leftarrow_{\$} \mathbb{Z}_q$.

Unoptimized DKG's ([18], [40]) have inherently high message and space complexity ($O(n^2)$ per node) since every party must broadcast its own polynomial commitments while downloading those of its peers. This incurs measurable latency and is prone to packet loss or disruptions, making them practical only for small n's. However, recent optimizations suggest adoption paths for moderate to large systems (cf. [24], [15], [14], [21], [44]; an overview can be found in [1]). These include reducing bandwidth usage or mutual verification costs by order of magnitude, or reducing the original message complexity from quadratic to almost linear, but come at the cost of sophisticated engineering and ad hoc tuning (hierarchical or topological designs, asynchronous overlay networks, parallel connections and batching, etc.). In this regard, [15] introduces a formal engineering framework that streamlines the optimization process, with its subsequent improvement [14] reducing total bandwidth consumption by 80% (under the same cubic complexity); however, the engineering effort for maintaining a secure and low-latency consensus network remains substantial (including dynamic participation management to ensure that a two-thirds majority of honest nodes remain available). This must be particularly taken into consideration when DKG's are employed to repeatedly generate a combined commitment per session (cf. Section 2.3).

2.2 Reliance on public shares

A distributed protocol is said to be secure in the *plain public-key* model (cf. [7]) if the public shares $y_i \equiv g^{x_i}$, $1 \leq i \leq n$, need not be certified in any sense; specifically, the shareholders need not undergo validation processes for their secret keys at registration because the protocol is immune to rogue-key attacks by design. In the extreme, the public shares may not even be advertised to the outer world; in this case, the only static input to verification is the group public key y, a property that will be referred to as *share-freeness*.

2.2.1 Universal share certification

Strong reliance on the public shares means that the verifiers have to trust and register them; consequently, in a threshold system, the certification process must generally attest their consistency, i.e., that they constitute a sharing of the group public key. How this is achieved depends on the use case and the exact distribution scheme. In a closed committee, where public auditability does not matter or the role of verifier is reserved for specific shareholders, it suffices that the public shares be advertised and trusted within the group; certification amounts to the setup phase DKG (cf. Section 2.1.1), which allows the peers to one-off validate each other's key, and the relevant infrastructure is confined to the local registries where each node maintains the public keys of its peers. In more transparent settings, where verifiers reside outside the group, the distribution process has to be publicly auditable, which comes at the cost of extended trust infrastructure. Even under centralized sharing, where the shares are essentially controlled by a dealer, the latter must employ some *publicly* verifiable secret sharing (PVSS) scheme ([42], [40]) and have its transcript persistently stored in a bulletin board that makes it accessible to any future verifier or monitor; specifically, external entities must be able to download ($O(n)$) and verify ($O(n^2)$) this transcript in order to trust and register the advertised shares.[3] The administrative burden remains the same in the canonical peer-to-peer case, where a PVSS-backed DKG replaces centralized sharing; in this case, the transcript consists of the transcripts produced by the totality of shareholders, which yields increased space complexity ($O(n^2)$) and a primary bottleneck.[4]

Share certification has not found its way to industry[5] even for the case of small n's. The reason seems to lie beyond transcript size, asymptotic verification costs and the engineering investment required from auditors. Any effort to standardize a "public-share infrastructure" would need to modify the Web PKI standards (e.g., by specifying a group-centric CSR model and extending the existing certificate formats) or the web-of-trust conventions. Even if backwards compatibility were not required, it would still need to rely on persistent and trusted bulletin boards, thereby increasing its governance costs and attack surface, and further address the administrative burden of proactive sharing, since every epoch would entail updating the boards and validating the refreshed shares from scratch.

2.2.2 Share-freeness

We have remarked that share-freeness is a rather radical way (cf. Section 2.2) to achieve security in the plain public-key model. Indeed, the verifier of a distributed protocol can strongly rely on advertised shares and yet these need not be certified; the typical example is the Bellare-Neven multi-signature scheme [7], where security against rogue-key attacks is achieved without requiring the shareholders to prove knowledge of their secret keys during the key setup phase. However, if some threshold condition is to be further satisfied (i.e., the protocol is based on Shamir's sharing), the registered shares must constitute the currently

[3] Recent work in [2] reduces the persistent storage requirement by nearly 40% and the asymptotic costs of verification to $O(n)$ without essential overhead.

[4] Lightweight verification techniques can theoretically mitigate this but introduce significant and context-dependent tradeoffs. KZG commitments (cf. [25]) reduce transcript size to constant but require a trusted setup within the bilinear setting; similarly, SNARKs (cf. [20]) achieve constant time within the bilinear framework while also requiring engineering investment for circuit design. Optimistic verification achieves constant costs but sacrifices universality by delegating trust to a designated verifier cohort, further necessitating incentive and slashing mechanisms.

[5] No existing or emerging applications seem to be reported except for a few ledger-related proposals and digital voting prototypes. These are mostly irrelevant to a standardization effort; moreover, this topic is not explicitly addressed in the NIST 2025 MPTC Project [34].

active sharing of the group public key. Since fabricating consistent public shares for any given public key is trivial, a verifier that relies on registered shares cannot tell the active sharing from a fabricated one without consulting a trusted bulletin board that attests to it. That is, for threshold protocols, reliance on public shares cannot circumvent certification in the sense of Section 2.2.1; to the extent to which this remains impractical beyond the closed committee model, share-freeness emerges as the one plausible alternative for system-agnostic threshold identification protocols. Observe that share-freeness does not preclude the provers from using their public shares upon every session, e.g., by attaching them on the fly to proofs of knowledge checked against y alone (cf. Section 2.9.4).

2.3 Interactivity

The recurring network overhead of per-session interactivity is often the critical bottleneck in threshold protocols, hindering them from scaling to large n's and arbitrary t's. Note that simply establishing secure channels between peers can itself be costly and fragile, while the latency from multi-round hops among globally distributed nodes can accumulate rapidly before the cost of cryptographic operations is even considered.

2.3.1 Compactness

Interactivity of Schnorr-based protocols arises from the need to generate a combined commitment. This serves a double purpose; it ensures that the threshold condition is satisfied by having all provers bind to it uniformly and it allows their packets to be compactly aggregated into an ordinary signature that can be efficiently stored and transferred. The mechanics are outlined in Section 2.8.1.

The commitment can be generated as the public output of a DKG (cf. Section 2.1.1), which is a bandwidth greedy and memory intensive operation that does not scale without tuning. While a system of nodes may be able to sustain one-off or occasional DKG's (e.g., for the purpose of generating a combined public key or refreshing shares during stable connectivity windows), continuously depending upon it might be infeasible; the simplest reason is the engineering effort required to maintain a secure and low-latency overlay network. Moreover, despite attaining moderate to high scalability, practical DKG's still face barriers that currently hinder them from becoming network and device agnostic solutions; for example, the state-of-the-art [14] reports 3.32 MB of outbound traffic per node ($O(n^2)$) and 12.48 seconds of average execution time already at $n = 64$, where nodes are globally distributed lightweight servers operating under realistic but relatively stable network conditions. These benchmarks suggest considerable delays and packet loss in a volatile mesh of memory-constrained devices.

In practice, the commitment is generated as in the signing phase of the FROST protocol ([27]), which achieves linear bandwidth per node and reduces the number of rounds to one or two, depending on whether preprocessing is employed or not. Benchmarks of a production-grade implementation indicate that the protocol remains computationally efficient for several hundreds of nodes.[6] The single round is attained by means of a preprocessing stage, where every node generates nonce pairs and registers the respective commitments in the aggregator's datastore, allowing subsequent proving phases to begin without preparatory rounds. This imposes an upfront operational overhead of synchronization and potentially a continuous storage burden per node, since peers have to securely retain their precomputed nonces; furthermore, system availability depends on this inventory, with the proving functionality

[6] [19] reports practical efficiency at $n = 1000$ and $t = 667$, measured on a Linux-capable machine with a consumer-grade high-performant processor.

being suspended in case of nonce depletion until the next epoch. While standardized, widely implemented and arguably the closest to non-interactive, FROST requires a fixed signer set that is online at signing time (cf. Section 2.3.2); it thus faces challenges in volatile or unmanaged settings, where the scheduling of precomputations and periodically synchronizing the aggregator may be impossible or cost-ineffective. ROAST [38] lifts the fixed signer set by running concurrent FROST sessions under a coordinator until some qualified subset responds; however, the signers still respond to the coordinator in real time, and robustness rests on identifiable aborts, i.e., on verifying the individual signature shares against the public shares (cf. Section 2.2).

2.3.2 Asynchronicity – Incremental verification

The above discussion implies that commitment generation can be a difficult task that requires context-specific recipes. However, the compactness constraint may not be crucial in most identification settings, where the benefits of non-interactivity can outweigh those of producing constant-size audit trails; in this case, the primary justification for interactivity reduces to the threshold constraint, and we are thus motivated to ask if this can be achieved in non-trivial ways without intra-group interaction. Indeed, non-interactive protocols are more likely to be robust; in the extreme, they remain applicable when provers are compelled to operate under volatile or absent intra-group connectivity. Irrespective of this, asynchronicity of action among the provers should be regarded per se as an operational advantage. After group constitution, the shareholders may be unable to discover each other, e.g., because no traceable channels persist among them or intra-group connectivity is intermittent due to uncontrolled factors. Even if communication remains possible, an application may require the provers to operate without being simultaneously online for the purpose of interaction; interaction among the provers should thus remain an option, invoked at will for synchronizing states per the specific use case, rather than a baseline operational requirement.

Asynchronicity of action means that the proving session must be non-interactive on the provers' side, so that their individual interactions with the verifier are independent of each other (unless the latter is able to interactively coordinate them); specifically, the verifier's interaction with the j-th shareholder should not depend on any packet received from the i-th shareholder, $i \neq j$. Note that this requirement facilitates *incremental verification*, i.e., the capability of a prover to attach itself to a session without having the already identified coalition rerun the proving phase in order to state-synchronize with them.

2.4 Real-time verification

Despite being routinely employed as real-time identification protocols, signature schemes per se do not require the signer to communicate with the verifier in real-time; the signature verifier needs only ensure that the claimed holder of a secret key has undeniably approved a specific message, no matter if the signature is sent by the signer live through an open channel, carried by designated entities or persistently stored in a ledger. Only when the verifier interacts with the signer in real-time, asking it to sign a nonce of its choice, can a signature scheme serve the purpose of identification over a designated communication channel. Observe that real-time interaction need not be direct, i.e., the proof may be forwarded to the verifier by a sequence of proxies, which are in turn responsible for relaying back the response of the verifier. For non-distributed protocols, trusting these proxies amounts to trusting them as generic network relay nodes, i.e., that they do not obstruct communication

intentionally and they cannot tamper with the transmitted content (e.g., because SSL/TLS is enforced between network hops).

2.4.1 Untrusted group proxies – Direct interaction

The above remarks carry over to the distributed setting with some nuance. First, the objective of threshold identification is to establish trust against a live coalition of unknown entities that appear across designated communication channels; a threshold signing primitive can only be employed in this context if the verifier asks the provers to sign a nonce of its choice, thereby binding the signature to the current session. Second, threshold primitives usually involve a combiner, who operates as group proxy by aggregating the individual packets into a compact signature and forwarding it to the verifier. However, this gateway is not a generic relay node since it runs ad hoc cryptographic software that is deployed by trusted intermediaries or internally maintained by dedicated group staff; in either case, trusting this service is a highly non-trivial assumption. As with any relay, the concern is not forgery but that the verifier's session terminates at the gateway rather than at the provers themselves. In order to remove this assumption, the verifier must interact with the individual provers directly, i.e., at most through generic relay nodes; specifically, whenever employing [43] or FROST [27] for real-time identification purposes, the role of combiner must *necessarily* be reserved for the verifier (cf. Section 2.8.1). Given this constraint, the particular performance benefit that arises in threshold signing primitives by separating the combiner from the verifier (constant-size signature and constant-time verification) is irrelevant for real-time thresholds, since the verifier has to aggregate and process $O(k)$ distinct packets anyway ($k = |Q|$).

More generally, in order for real-time thresholds over designated communication channels to make sense, the provers must send their cryptographic packets over those same separate channels that have been directly opened to the verifier (i.e., at most through generic relay nodes). Even if their packets are the aggregable product of some intra-group multi-party computation, no special gateway node can generally be assumed to forward their aggregation to the outer world; this would at best identify that gateway as a delegated proxy acting on behalf of a qualified coalition, not the live coalition of shareholders itself.

2.5 The permissionless setting

Recall that the verifier must interact with the provers directly (cf. Section 2.4.1) and without generally knowing their individual identities (cf. Section 2.2.2). Even if the provers present their claimed public keys, the verifier can at best validate them on the fly (e.g., by requiring individual proofs of knowledge); whether these keys constitute the currently active sharing of the group public key is something that the verifier has generally no way to verify, because universal share certification is not an option (cf. Section 2.2.1). More generally, outside the closed committee model, an authority cannot associate any claimed identity with an actual shareholder unless it has itself taken part in the group setup phase as a peer (or dealer). Consequently, for open decentralized systems that keep reliance on authorities to a limited scope, or cross-domain settings where the provers operate outside the verifier's ecosystem, it makes sense to assume that the provers appear completely unvetted, i.e., that the verifier has zero prior knowledge on the entities it interacts with. Contrary to signing schemes, where the shareholders can operate in a permissioned setting that outputs a static artifact, the natural model for identification is a permissionless network where entities appear without accreditation and open unsolicited connections as self-proclaimed shareholders.

2.5.1 Session poisoning

The permissionless model implies that share-free threshold protocols are inherently vulnerable to a stealthy denial-of-service attack at the application layer, obstructing a qualified coalition $Q \subset \{1, \ldots, n\}$, $|Q| \geq t$, of legitimate shareholders from having their ensemble identified. An attacker posing as the j-th shareholder submits a structurally valid but semantically incorrect proof packet to the verifier. In the absence of universal share certification mechanisms (cf. Section 2.2.1), the verifier has no authoritative reference to the public share y_j; in particular, it has no reliable way to detect the malicious packet by pre-validating it against y_j and thus exclude it from verification. The coalition $Q^* = Q \cup \{j\}$ fails to identify and consequently service is denied to Q.

Interactive fault isolation Assuming that intra-group communication is state-synchronized with the verifier (so that the attacker cannot make unsolicited submissions without having talked to the other provers) and the public shares are known among the shareholders for mutual verification purposes, an honest majority can detect the malicious actor during the proving round and notify the verifier to deny service to j separately, without disrupting the session for the other provers. However, this requires a robust consensus and notification mechanism while also relying on subtle security assumptions regarding the majority of honest peers; indeed, maintaining a secure consensus network presupposes active participation management and has itself a notoriously broad attack surface. Finally, such solutions would fail to detect an actual but malevolent shareholder and are obviously non-applicable in non-interactive contexts.

Reactive fault detection Asynchronicity facilitates a potentially better way to deny service to j separately by delegating detection duties to the verifier. Suppose that the verifier has identified some baseline coalition Q and applies incremental verification (cf. Section 2.3.2); by asynchronously onboarding j to the current session, the verifier can reliably infer that this packet causes the verification condition to fail and thus drop the respective connection. This countermeasure makes non-interactive protocols immune to session poisoning after a baseline coalition has been identified, with its feasibility depending on how fast the verifier detects such a coalition within a pool of aggregated proof packets. Obviously, bulk verification should here be as efficient as possible since it may need to be repeated multiple times until a baseline coalition is found. This is trivial if the malicious packets are few, in which case the protocol is practically immune from the outset; however, in case of flooding, uncritically exhausting combinations becomes itself an attack vector because it drains the server resources and precipitates a full denial-of-service. To avoid this, reactive fault detection should employ backtracking algorithms that reduce the possibilities of combinatorial explosion and be combined with rate limiting, CPU throttling or other system-level techniques that ensure abortion if some resource usage threshold is critically exceeded.

2.6 Accountability

Session poisoning is an aspect of non-accountability in the permissionless setting. Accountability measures the ability of a system to detect which participants behave maliciously during a protocol round. This matters in use cases where responsibility must be attributable to named actors (e.g., financial consortia or hierarchical committees) or details on malicious activity must be gathered for the purpose of forensic investigations. Malicious actors can be shareholders who deviate from the protocol or intruders who try to impersonate legitimate

shareholders; in either case, from the cryptographic viewpoint, an actor should be referred to with the claimed public key which interested parties use to validate its activity. This evidently requires that the public shares be to a certain degree known and certified, which deviates from the pure share-free model.

2.6.1 Intra-group vs extra-group

The auditability scope relates directly to the desired accountability model. In the intra-group model of early rejects, i.e., if the protocol is interactive and the provers are responsible for detecting cheating peers through fault isolation (cf. Section 2.5.1), it suffices that the public shares be advertised and trusted within the group; for threshold systems, trust is established during the setup phase DKG (cf. Section 2.1.1), where public shares are mutually verified on the fly, and infrastructure is confined to the local registries where each node maintains the currently active public keys of its peers. In the extra-group model, i.e., if accurate blaming has to be accommodated by the verifier, the public shares have to be publicly auditable in the long run; for threshold systems in non-closed settings, this amounts to universal share certification in the sense of Section 2.2.1. The recurring network overhead of early rejects is replaced by the infrastructural overhead of trusted bulletin boards.

Extra-group accountability is only feasible for closed systems that need not interoperate with external auditors or can tolerate authorities. In the first case, PVSS bulletin boards become practical (provided that n does not grow excessively and the group of shareholders remains relatively static) because the system does not need to comply with external standards. In the second case, the public shares are trusted because they are advertised by a trusted proxy that is widely recognized as such and entitled to perform this action. Systems that prioritize decentralization will naturally opt for PVSS-based solutions; however, for open systems, such solutions amount to universal share certification, which we have seen to be impractical (cf. Section 2.2.1). In other words, extra-group accountability is unattainable in open decentralized settings due to the impracticality of universal share certification. At the same time, these are the environments par excellence where extra-group accountability is irrelevant; indeed, the permissionless setting (cf. Section 2.5) implies that externally attributing responsibility by reference to claimed public keys is pointless, since actors are anyway free to claim identities without vetting.

For open systems, intra-group accountability remains as the only practical approach to accountability and thereby an argument in favour of interactive protocols. Nevertheless, the operational benefits of asynchronicity most probably outweigh those of early rejects. Indeed, intra-group accountability justifies interactivity only insofar as it mitigates session poisoning through interactive fault isolation, a stateful and fragile consensus mechanism with broad attack surface and strong infrastructural prerequisites.

2.7 Efficiency considerations

The wireless transmission of a single bit requires more energy than a 64-bit (or 32-bit) CPU instruction by orders of magnitude (cf. [5]); that is, for low-power provers that perform few exponentiations per round, packet size can be a critical factor regarding energy consumption on their side. Nevertheless, the number of transmitted packets tends to dominate due to the overhead of the radio transition states, implying that message complexity has probably greater impact than bandwidth usage. For constrained meshes, this speaks strongly in favour of asynchronous schemes that eliminate intra-group interactivity altogether.

On the other side, a verifier that performs multiple exponentiations per session is bound by computation rather than by the size of the incoming packets, so that verification time is what should be optimized (cf. Section 3.2). Inbound traffic becomes a concern only in high-throughput scenarios, where a server-class verifier handles multiple sessions concurrently and its network interface, or the relay nodes forwarding the packets, approach saturation; in such cases, the compact variant of the proposed protocol (cf. Section 3.1.2) saves one group element per prover, at the cost of concrete security against large coalitions, which confines it to small groups (cf. Remark 3.28).

2.8 Existing approaches and tradeoffs

In this section, we examine closely the existing paradigms. We denote $k = |Q|$.

2.8.1 Combined commitment approach

The Schnorr signature is a proof-of-knowledge with a message baked into the hash function, suggesting a threshold identification protocol in the lines of [43]. The coalition of shareholders $Q \subset \{1, \dots, n\}$, $|Q| \geq t$, execute some DKG-like process to jointly generate a (n, t)-reconstructible commitment

$$u \equiv g^r, \quad r = \sum_{i \in Q} \lambda_i r_i,$$

where $\{\lambda_i\}_{i \in Q}$ are the interpolation coefficients and the nonce shares $\{r_i\}_{i \in Q}$ are kept private. Next, they generate proofs for their respective keys with a common challenge, i.e.,

$$(u_i, s_i), \quad s_i \leftarrow r_i + c x_i, \quad u_i \leftarrow g^{r_i}, \quad c \leftarrow H(u),$$

and send them to the verifier; after aggregating (u_i, s_i), $i \in Q$, the verifier computes

$$s \leftarrow \sum_{i \in Q} \lambda_i s_i, \quad u \leftarrow \prod_{i \in Q} u_i^{\lambda_i}, \quad c \leftarrow H(u)$$

and accepts only if the aggregate proof (u, s) satisfies $g^s = y^c u$. Indeed, since $|Q| \geq t$, double application of Shamir's interpolation formula (cf. Remark B.3) yields

$$g^s = \prod_{i \in Q} g^{\lambda_i s_i} = \prod_{i \in Q} g^{c \lambda_i x_i} \prod_{i \in Q} g^{\lambda_i r_i} = \Big(\prod_{i \in Q} y_i^{\lambda_i} \Big)^c \prod_{i \in Q} u_i^{\lambda_i} = y^c u.$$

The protocol is share-free and interactive, producing a compact audit trail (u, s). Verification requires $k + 2$ exponentiations and $O(k^2)$ field multiplications for the computation of λ_i's. Inbound traffic grows linearly with respect to k; this is because the verifier must also be combiner according to the requirement of direct interaction (cf. Section 2.4.1). Evidently, incremental verification is impossible because the session is bound to the combined commitment u that must be used by all provers. Efficiency depends on the exact scheme employed for generating u, i.e., some asynchronous DKG (cf. [15], [14]) or FROST-like process; an overview of the tradeoffs can be found in Section 2.3.1. FROST is particularly apt due

to its linear scaling, provided that the infrastructural cost of aggregators (along with the administrative burden of periodically synchronizing them) is not an issue.

The protocol facilitates interactive fault isolation against session poisoning. Specifically, provided that the provers know and trust each other's public key, they broadcast their proofs for mutual verification before sending them to the verifier; in this case, an honest majority agree on which packets are bogus and notify the verifier to reject the respective connections. We have stressed that maintaining a robust and secure consensus mechanism (cf. Section 2.5.1) can easily be infeasible, inadequate or cost-ineffective, while the overhead from broadcasting or mutual verifications can obviously nullify the particular performance advantage of FROST. Delegating pre-validation to the combiner (as is the case with threshold signatures) is meaningless because the latter coincides with the verifier; the verifier would need to know and trust the individual public keys in advance, in which case real-time interaction among the provers becomes redundant (cf. Section 2.8.2).

2.8.2 Trivially decoupled approach

Non-interactivity can be trivially achieved if the public shares $y_i \equiv g^{x_i}$, $1 \leq i \leq n$, are advertised and trusted by the verifier. In this case, the coalition of shareholders $Q \subset \{1, \ldots, n\}$ can generate independent Schnorr proofs for their respective secrets, i.e.,

$$(u_i, s_i), \quad s_i \leftarrow r_i + c_i x_i, \quad c_i \leftarrow H(u_i), \quad u_i \leftarrow g^{r_i}, \quad r_i \leftarrow_{\$} \mathbb{Z}_q, \quad i \in Q$$

and send them to the verifier, who accepts only if the condition

$$|Q| \geq t \ \wedge \ g^{s_i} = y_i^{c_i} u_i, \, \forall i \in Q, \quad c_i \equiv H(u_i),$$

is satisfied. This scheme is asynchronous but share-reliant; in fact, its security relies entirely on the assumption that $y_1, \ldots, y_n$ constitute the *currently* active (n, t)-sharing of y. Attesting to their consistency is a non-trivial task, whose cryptographic solution remains impractical in open or cross-domain systems (cf. Section 2.2.1). In fact, this artificial protocol is appropriate only for closed committees, where verifiers are group participants and have thereby validated the public shares during the key setup phase, or other closed managed environments that tolerate authorities and can afford to rely on certified updates. It is essentially an access control mechanism for onboarding the shareholders to a permissioned environment, allowing them to perform further distributed operations in an individually accountable fashion (e.g., threshold signing). Security is bounded by how reliably the verifiers' registries are updated upon resharing; even revoking a single share requires system-wide administrative action.

2.9 Completely decoupled schemes

Below we proceed to schemes that are both share-free and non-interactive; the public shares are not advertised, but —explicitly or implicitly— attached to the proofs upon every session. The first one attaches the shares explicitly (cf. Section 2.9.1), with extraction (in the sense of special soundness) failing at the level of the individual components. The attached shares are then redundant, the scheme being equivalent to interpolating the proofs directly (cf. Section 2.9.2); however, this is susceptible to a generalized birthday attack whose cost decreases rapidly with the size of the impersonated coalition (cf. Section 2.9.3). Binding the attached shares into the challenges removes the freedom exploited by this attack (cf. Section 2.9.4); the generalized birthday attack is excluded and security holds irrespective of size (cf. Section 5).

2.9.1 Attached public shares

The shareholders $Q \subset \{1, \ldots, n\}$ generate

$$(u_i, s_i, y_i), \quad y_i \leftarrow g^{x_i}, \quad s_i \leftarrow r_i + c_i x_i, \quad c_i \leftarrow H(u_i), \quad u_i \leftarrow g^{r_i}, \quad r_i \leftarrow_{\$} \mathbb{Z}_q$$

respectively and send them to the verifier, who accepts only if the condition

$$y = \prod_{i \in Q} y_i^{\lambda_i} \ \wedge \ g^{s_i} = y_i^{c_i} u_i, \, \forall i \in Q \,, \quad c_i \equiv H(u_i) \tag{2.3}$$

is satisfied. Every prover "should know" the logarithm of its attached element because the respective proof verifies; since the attached elements interpolate to y, their logarithms interpolate to x.

This scheme is share-free and non-interactive; for $n = 1$, it yields the ordinary Schnorr protocol (cf. Appendix A), where the attached element is y itself, fixed before any challenge is issued. For $|Q| \geq 2$, however, the interpolation condition constrains the attached elements only jointly, with every challenge depending on its own commitment alone; the extraction argument of the non-distributed case does not apply, and nothing obviously prevents the impersonator from fixing the attached elements after the respective challenges. That is, it may fabricate u_i's (possibly of unknown logarithm) along with the s_i's, query c_i's properly and only then compute

$$y_i = \sqrt[c_i]{g^{s_i}/u_i}, \quad i \in Q, \quad c_i \equiv H(u_i) \,, \tag{2.4}$$

so that the per-component part of condition (2.3) holds without knowledge of $\log y_i$'s. Such a practical attack exists indeed (cf. Section 2.9.3), with its cost decreasing rapidly against $|Q|$, which is arbitrarily chosen by the adversary; the more the components, the wider the room for satisfying the verification condition by search. To see why ordinary fork and extraction fails to exclude it, consider that rerunning the impersonator up to the hash query for u_i yields a second proof with the same commitment and a different challenge c_i^*, which may however attach a different element y_i^*; that is, subtracting the two conditions, we get

$$s_i - s_i^* = c_i \log y_i - c_i^* \log y_i^* \,, \tag{2.5}$$

which, for $|Q| \geq 2$, determines neither logarithm and prevents thus extraction at the level of the individual components.

2.9.2 Direct proof interpolation

For a valid proof, substituting (2.4) into the interpolation condition of (2.3) eliminates the attached elements and yields

$$y = \prod_{i \in Q} \Big(\frac{g^{s_i}}{u_i} \Big)^{\lambda_i / c_i} ; \tag{2.6}$$

conversely, every collection $\{(u_i, s_i)\}_{i \in Q}$ satisfying (2.6) satisfies (2.3) once the elements (2.4) are attached. That is, the attached elements are redundant; the shareholders may as well

send ordinary Schnorr proofs $\{(u_i, s_i)\}_{i\in Q}$ with the verifier checking (2.6) directly. The two schemes are equivalent; a forgery against either scheme yields a forgery against the other, so that the obstacle to per-component extraction carries over.

Elimination of unknown logarithms is nevertheless possible through nested forking (cf. Appendix D.3), since (2.6) is a single linear relation in x and these logarithms, so that both schemes turn out to be formally "secure" for every fixed bound on the indices (cf. Theorem D.25). It should be stressed again, however, that this guarantee is asymptotic; the reduction loss grows rapidly against $|Q|$ and consequently fails to exclude attacks whose cost decreases with the size of the impersonated coalition (cf. Section 2.9.3).

2.9.3 Generalized birthday attack

The above equivalent protocols admit an attack whose cost decreases with $|Q|$ rapidly and which requires no knowledge of shares on behalf of an adversary $\mathcal{A}$. Letting $\mathcal{A}$ fix an index set Q of its choice, and thereby the interpolation coefficients $\{\lambda_i\}_{i\in Q}$, consider the following attack scheme. Assume that $\mathcal{A}$ is somehow able to fabricate

$$u_i = g^{s_i} y^{e_i}, \quad s_i, e_i \in \mathbb{Z}_q,\ i \in Q\,, \tag{2.7}$$

and sets $y_i = y^{-e_i/c_i}$ (cf. (2.4)), $c_i \equiv H(g^{s_i} y^{e_i})$, so that the right-hand side of (2.6) becomes

$$\prod_{i\in Q} \left(y^{-e_i}\right)^{\lambda_i/c_i} = \exp\Big(y, -\sum_{i\in Q} \lambda_i\, e_i/c_i\Big)\,.$$

That is, the interpolation condition reads $\exp\big(y, 1 + \sum_{i\in Q} \lambda_i\, e_i/c_i\big) = 1$, which —assuming that $x \neq 0$ and thus y is of order q— is equivalent to

$$\sum_{i\in Q} \lambda_i \frac{e_i}{c_i} = -1 \pmod q\,. \tag{2.8}$$

This equation involves neither x nor any share; its parameters are known to the adversary, with the c_i's determined after the exponents of (2.7) have been fixed and the s_i's chosen arbitrarily. That is, $\mathcal{A}$ succeeds in its impersonation attempt if it can fabricate e_i's that satisfy (2.8). Below we show that this is feasible for unbounded coalitions, in the sense that 128-bit security is violated for sufficiently large $|Q|$'s.

Solving (2.8) is instance of the $|Q|$-sum problem over $\mathbb{Z}_q$ with lists of uniformly random elements: since every c_i depends on u_i alone, the summands can be generated separately; the impersonator may thus prepare a list of candidates per $i \in Q$, each entry costing one group operation (e.g., incrementing s_i) and one hash query, and search for one entry per list such that their sum equals -1. For $|Q| = 1$, there is a single list and c_i must hit $-e_i$, which takes about q hash queries; for $|Q| \geq 2$, however, Wagner's algorithm [45] solves the problem with $2^{\lfloor \log_2 |Q| \rfloor}$ lists (the remaining components being fixed arbitrarily) in time roughly

$$2^{\lfloor \log_2 |Q| \rfloor} \cdot q^{1/(1+\lfloor \log_2 |Q| \rfloor)} \tag{2.9}$$

and memory comparable to the second factor. For $|Q| \leq 3$, this is no faster than solving DL. However, this cost falls rapidly as $|Q|$ increases, with any acceptable level of security violated

if no appropriate bound is enforced by the verifier; e.g., over a 256-bit group, impersonating 64 shareholders takes about 2^{43} group operations and terabytes of memory. Since $|Q|$ is chosen by the adversary while n and t are not known to the verifier, the latter can only limit it by enforcing a uniform bound on the size of admissible coalitions (cf. Remark 3.28); specifically, both schemes require q to grow with that bound so that the appropriate level of security is preserved (cf. Remark 3.30).

2.9.4 Binding the attached shares

In both schemes, the above attack exploits the relative freedom of choosing the attached elements after the challenges. This freedom is removed by setting $c_i \leftarrow H(y_i, u_i)$ for some hash function $H : \mathbb{G} \times \mathbb{G} \to \mathbb{Z}_q$ (the strong variant of the Fiat-Shamir transform, cf. [10]). Since y_i is now part of the hash input, rerunning the impersonator up to the hash query for (y_i, u_i) yields a second proof with the same attached element, whence (2.5) becomes

$$s_i - s_i^* = (c_i - c_i^*) \log y_i \tag{2.10}$$

and extractability is restored at the level of individual components. That is, forking separately at each of them yields the combined secret key and security reduces to DL hardness at a loss polynomial in $|Q|$ (cf. Theorem 5.17); in particular, no bound on the indices is required. It should be stressed, however, that this reduction too is asymptotic.

The generalized birthday attack is excluded for the same reason. Let the impersonator fabricate u_i and y_i from g and y with exponents of its choice, y having exponent e_i in u_i (as in (2.7)) and e'_i in y_i. The per-component condition of (2.3) then requires knowledge of $\log(y_i^{c_i} u_i)$, in which x enters with coefficient $e_i + c_i e'_i$; an impersonator who does not know x must thus ensure $e_i + c_i e'_i = 0$, $i \in Q$, while the interpolation condition likewise requires $\sum_{i \in Q} \lambda_i e'_i = 1$. Binding y_i into c_i changes only whether e'_i is fixed after or before c_i, and thus whether the challenges enter the interpolation condition or not. In the scheme of Section 2.9.1, every e'_i is set to $-e_i/c_i$ once c_i is known, so that the per-component conditions hold at no cost and the interpolation condition becomes (2.8), a single relation in which all challenges may compensate for one another; this is what the generalized birthday attack exploits. Here every e'_i is fixed beforehand, y_i being part of the hash input; the interpolation condition thus involves no challenge, every challenge being confined to its own per-component condition. Since the interpolation condition forces $e'_i \neq 0$ for some $i \in Q$, c_i must then hit $-e_i/e'_i$ on its own, which takes about q hash queries, as in the non-distributed case. In short, no challenge can compensate for another; this excludes the above attack, though not every conceivable one.

3 One-Step Schnorr Threshold (OSST) Identification

Fix a cyclic group $\mathbb{G} = \langle g \rangle$ of prime order q and a hash function $H : \mathbb{G} \times \mathbb{G} \to \mathbb{Z}_q$. Shamir's sharing is denoted by D (cf. Appendix B) with its parameters inferred from context.

3.1 The OSST protocol

The proposed scheme is that of Section 2.9.4; every shareholder attaches its public share to an ordinary proof and binds it into the challenge, while the verifier, who knows only y, checks that the attached shares interpolate to it. Given integers (n, t) such that $1 \leq t \leq n < q$, we formally define it as follows.

Protocol 3.1. ($\mathsf{OSST}_{\mathbb{G},n,t}$ – *One-Step Schnorr* (n,t)*-Threshold*)

- *Setup.* Some uniformly random secret is distributed among a group of shareholders by means of Shamir's (n,t)-sharing, i.e.,

$$(x_1,\dots,x_n;\, y) \leftarrow D(g,x),\;\; x \leftarrow_{\$} \mathbb{Z}_q\,.$$

  The threshold parameter, the number of shareholders and the public shares need not be advertised; the group needs only advertise the combined public key y.

- *Proving phase.* A coalition of shareholders $Q \subset \{1,\dots,n\}$ generate

$$(y_i,u_i,s_i),\;\; y_i \leftarrow g^{x_i},\;\; s_i \leftarrow r_i + c_i x_i,\;\; c_i \leftarrow H(y_i,u_i),\;\; u_i \leftarrow g^{r_i},\;\; r_i \leftarrow_{\$} \mathbb{Z}_q \tag{3.2}$$

  respectively for $i \in Q$ and send them to the verifier.

- *Verification.* The verifier discards every component whose index lies outside $\{1,\dots,q-1\}$. After aggregating (y_i,u_i,s_i), $i \in Q$, it accepts only if

$$y = \prod_{i\in Q} y_i^{\lambda_i} \;\wedge\; g^{s_i} = y_i^{c_i} u_i,\; \forall i \in Q,\;\; c_i \equiv H(y_i,u_i), \tag{3.3}$$

  where $\{\lambda_i\}_{i\in Q}$ are the Lagrange interpolation coefficients for Q.

Remark 3.4. (*Fluid group of shareholders*) The attached elements are not checked against any registry; the verifier only ensures that they interpolate to y and that every prover knows the logarithm of its own (provided that the protocol is secure against impersonation attacks; cf. Section 4). Consequently, provers are free to use any arbitrary sharing of $x = \log y$, be it "revoked", "active" or with a different threshold. That is, identification is not bound to a specific sharing, since the verifier cannot know the public shares in advance and keep track of the sharing epochs. The onboarded entities are not committed to their attached shares, except maybe for the purpose of session-scoped accountability (cf. Section 3.3.3).

Remark 3.5. (*Session binding*) As it stands, the protocol is susceptible to replay (cf. Section 3.4.1) and should not be deployed as such. The variant intended for deployment is the session-bound one (cf. Section 3.1.1), where the verifier communicates a session identifier (or "nonce") to the provers for binding it into their respective challenges.

Remark 3.6. (*Interactive variant*) The provers send their public shares and commitments to the verifier, who lazily samples challenges by maintaining a lookup table Map per session. More accurately, the i-th prover samples $r_i \leftarrow_{\$} \mathbb{Z}_q$ and sends y_i along with $u_i \leftarrow g^{r_i}$ to the verifier. If $(y_i,u_i) \notin \mathrm{Dom}(\mathsf{Map})$, the verifier samples $c_i \leftarrow_{\$} \mathbb{Z}_q$ and caches $\mathsf{Map}[y_i,u_i] \leftarrow c_i$; in either case, it sends $c_i \leftarrow \mathsf{Map}[y_i,u_i]$ to the prover, who responds with $s_i \leftarrow r_i + c_i x_i$. After aggregating s_i, $i \in Q$, the verifier checks that condition (3.3) holds true using the previously cached challenges c_i, $i \in Q$.

Below we proceed to the formal definitions.

Definition 3.7. A *proof* is a collection $\sigma = \{(y_i,u_i,s_i)\}_{i\in Q}$ of the form

$$(y_i,u_i,s_i) \in \mathbb{G}\times\mathbb{G}\times\mathbb{Z}_q,\; \forall i \in Q \;\wedge\; Q \subset \{1,\dots,q-1\}.$$

Given σ as above, we refer to $\{y_i\}_{i\in Q}$ as its *attached shares*, to $\{u_i\}_{i\in Q}$ as its *commitments* and to $\{s_i\}_{i\in Q}$ as its *responses*; we also say that σ is *based on* its commitments. The *proof space* is the set of proofs and will be denoted by Σ.

The proper execution of the protocol is modelled as follows.

Definition 3.8. The proof $\{(y_i, u_i, s_i)\}_{i\in Q} \in \Sigma$, $Q \subset \{1, \ldots, n\}$, is called *canonical with respect to* a sharing $(x_1, \ldots, x_n;\, y) \in D_x$ (cf. Def. B.2) if

$$y_i = g^{x_i}, \quad s_i = r_i + c_i\, x_i, \quad c_i \equiv H(y_i, u_i), \quad r_i \equiv \log u_i, \quad \forall i \in Q, \tag{3.9}$$

i.e., its components are Schnorr proofs for the respective shares, attached with the latter. Evidently, a canonical proof is determined by its commitments and the sharing.

Definition 3.10. The *one-step Schnorr* (n, t)*-threshold verifier* is the deterministic algorithm $\mathcal{V} : \mathbb{G} \times \Sigma \to \{\mathsf{true}, \mathsf{false}\}$ invoked as

$$\big(y, \{(y_i, u_i, s_i)\}_{i\in Q}\big) \quad \mapsto \quad y = \prod_{i\in Q} y_i^{\lambda_i} \;\wedge\; g^{s_i} = y_i^{c_i} u_i,\ \forall i \in Q\,,$$

where $c_i \equiv H(y_i, u_i)$ and $\{\lambda_i\}_{i\in Q}$ are the Lagrange interpolation coefficients for Q (cf. (3.3)).

Definition 3.11. A proof $\sigma \in \Sigma$ is called *valid against* $y \in \mathbb{G}$ if $\mathcal{V}(y, \sigma) = \mathsf{true}$.

Proposition 3.12. (*Validity conditions*) *A proof* $\{(y_i, u_i, s_i)\}_{i\in Q} \in \Sigma$ *is valid against* $y = g^x$ *if and only if*

$$s_i = r_i + c_i \log y_i,\ \forall i \in Q \;\wedge\; x = \sum_{i\in Q} \lambda_i \log y_i\,, \quad r_i \equiv \log u_i, \quad c_i \equiv H(y_i, u_i)\,. \tag{3.13}$$

Proof. Follows directly by equating exponents. □

3.1.1 Session-bound variant

Below we examine the variant intended for deployment (cf. Remark 3.5), which binds the challenges to the session. Given a space $\mathcal{N}$ of session identifiers and a hash function $H : \mathbb{G} \times \mathbb{G} \times \mathcal{N} \to \mathbb{Z}_q$, the *session-bound* variant of OSST is obtained by computing the challenges as $c_i \leftarrow H(y_i, u_i, \mathsf{sid})$ in both the proving phase and verification (cf. Protocol 3.1), where $\mathsf{sid} \in \mathcal{N}$ is chosen by the verifier upon opening the session (e.g., as a uniformly random nonce) and communicated to the provers; we denote the respective verifier by $\mathcal{V}_{\mathsf{sid}}$. Note that the provers still operate in one step and independently of each other, since sid is fixed before they commit.

Remark 3.14. (*Reused commitments*) By (3.13), a shareholder who uses the same commitment under two distinct challenges leaks its share, i.e., $x_i = (s_i - s_i^*)/(c_i - c_i^*)$. This occurs whenever a commitment is reused in the session-bound or the interactive variant (cf. Remark 3.6), where the challenge changes from session to session; just like in the non-distributed case, commitments should never be reused.

3.1.2 Compact variant

Below we examine the scheme of Section 2.9.2, which operates on structurally ordinary Schnorr proofs; every shareholder runs the prover of the non-distributed protocol (cf. Appendix A), which may be of interest wherever that prover is already deployed or not easily replaceable. This saves one group element per prover and requires $|Q|+2$ exponentiations (cf. Section 3.2). It should be stressed, however, that these savings come at the cost of the generalized birthday attack of Section 2.9.3. The scheme is strictly confined to small groups since it enforces a uniform bound on $|Q|$ (cf. Remark 3.28) for the given security parameter (cf. Remark 3.30); beyond that, Protocol 3.1 should be used. Given integers (n, t) such that $1 \le t \le n \le \bar{n} < q$, where $\bar{n}$ is a public uniform bound on n's, and a hash function $H : \mathbb{G} \to \mathbb{Z}_q$, we define the following scheme[7] by completing the exponents in (2.6).

Protocol 3.15. ($\mathsf{OSST}^{\mathsf{c}}_{\mathbb{G},n,t}$ – *Compact One-Step Schnorr* (n,t)*-Threshold*)

- *Setup.* Some uniformly random secret is distributed among a group of shareholders by means of Shamir's (n, t)-sharing, i.e.,

$$(x_1, \ldots, x_n;\, y) \leftarrow D(g, x),\;\; x \leftarrow_{\$} \mathbb{Z}_q\,.$$

  The threshold parameter, the number of shareholders and the public shares need not be advertised; the group needs only advertise the combined public key y.

- *Proving phase.* A coalition of shareholders $Q \subset \{1, \ldots, n\}$ generate

$$(u_i, s_i), \quad s_i \leftarrow r_i + c_i\, x_i, \quad c_i \leftarrow H(u_i), \quad u_i \leftarrow g^{r_i}, \quad r_i \leftarrow_{\$} \mathbb{Z}_q \tag{3.16}$$

  respectively for $i \in Q$ and send them to the verifier.

- *Verification.* The verifier discards every component whose index lies outside $\{1, \ldots, \bar{n}\}$ or has already been received. After aggregating (u_i, s_i), $i \in Q$, it accepts only if

$$\exp\Big(g, \sum_{i\in Q} \mu_i s_i\Big) = y^{\bar{c}} \prod_{i\in Q} u_i^{\mu_i}, \tag{3.17}$$

  where

$$\bar{c} \leftarrow \prod_{i\in Q} c_i, \quad \mu_i \leftarrow \lambda_i \prod_{j\in Q\setminus\{i\}} c_j, \quad c_i \leftarrow H(u_i) \tag{3.18}$$

  and $\{\lambda_i\}_{i\in Q}$ are the Lagrange interpolation coefficients for Q.

Remark 3.19. (*Validity condition*) By equating exponents, (3.17) translates to

$$\sum_{i\in Q} \mu_i\,(s_i - r_i) = \bar{c}\cdot x\,, \quad r_i \equiv \log u_i\,. \tag{3.20}$$

Definition 3.21. The *weights* and *normalizer* of a collection $\{u_i\}_{i\in Q} \subset \mathbb{G}$, $Q \subset \{1, \ldots, \bar{n}\}$, are $\{\mu_i\}_{i\in Q}$ and $\bar{c}$ respectively as defined in (3.18).

[7] An earlier version of this work presented this variant under the name OSST.

Remark 3.22. (*Relation between weights and the normalizer*) Let $\{\mu_i\}_{i\in Q}$, $\bar{c}$ be the weights and the normalizer of a collection $\{u_i\}_{i\in Q}$ respectively. Then

$$c_i\mu_i = \lambda_i\bar{c}, \quad \forall i \in Q\,. \tag{3.23}$$

Remark 3.24. (*Correctness*) Let $y_i = g^{x_i}$, $1 \le i \le n$ be the public shares. Applying (3.23), the left-hand side of condition (3.17) transforms as

$$\prod_{i\in Q} g^{s_i\mu_i} = \prod_{i\in Q} g^{\mu_i c_i x_i} g^{\mu_i r_i} = \prod_{i\in Q} g^{x_i\lambda_i\bar{c}} \prod_{i\in Q} g^{r_i\mu_i} = \Big(\prod_{i\in Q} y_i^{\lambda_i}\Big)^{\bar{c}} \prod_{i\in Q} u_i^{\mu_i},$$

which yields the right-hand side if and —with overwhelming probability— only if $|Q| \ge t$. In other words, the proper execution of the protocol verifies if and —with overwhelming probability— only if at least t shareholders are involved.

Remark 3.25. (*Public-share inference*) Although not advertised, the involved public shares can be trivially inferred as

$$y_i = \sqrt[c_i]{g^{s_i}/u_i}, \quad i \in Q, \tag{3.26}$$

since $\{(u_i, s_i)\}_{i\in Q}$ are just ordinary Schnorr proofs for the corresponding secrets (cf. (3.16)). Remark 3.4 applies likewise, with the inferred elements in place of the attached ones.

Remark 3.27. (*Zero challenges*) If $c_{i_0} = 0$ for a single $i_0 \in Q$, then $\bar{c} = 0$ and $\mu_i = 0$ for $i \ne i_0$, so that condition (3.17) reduces to $g^{s_{i_0}} = u_{i_0}$; if two or more challenges vanish, then all weights vanish and every proof based on these commitments is valid. In either case, validity depends neither on y nor on the other components. Since H is a hash function, zero challenges occur with negligible probability in the bitlength of q; the verifier may nevertheless discard every component with $c_i = 0$.

Remark 3.28. (*Small bound on the indices*) The compact variant should not be used unless the verifier enforces a small bound $\bar{n}$; otherwise, valid proofs can be fabricated without any share at a cost far below that of solving DL. The reason is that condition (3.17) is a single relation across all components, while every challenge depends only on its own commitment. Indeed, an impersonator who fabricates $u_i = g^{s_i}y^{e_i}$ with exponents of its choice, $i \in Q$, obtains a valid proof if and only if condition (2.8) holds (provided that $x \ne 0$) and finds such exponents by means of Wagner's algorithm, every candidate summand costing one group operation and one hash query (cf. Section 2.9.3). More components make the search cheaper, and their number is limited by $\bar{n}$ alone; indeed, the verifier knows neither n nor t, so that the impersonator may employ any $2^a \le \bar{n}$ distinct indices. With $a \equiv \lfloor \log_2 \bar{n} \rfloor$, a valid proof is fabricated in time roughly

$$2^a \cdot q^{1/(a+1)} \tag{3.29}$$

and memory of about $q^{1/(a+1)}$ list entries. For $\bar{n} \le 3$, this is no faster than solving DL; beyond that, the exponent decreases with $\bar{n}$; e.g., over a 256-bit group, the attack takes about 2^{87}, 2^{67}, 2^{55}, 2^{48} and 2^{43} operations for $\bar{n}$ up to 7, 15, 31, 63 and 127 respectively, and bottoms out at about 2^{31} for $\bar{n} \ge 2^{15}$ (the optimal a being about $\sqrt{\log_2 q}$).

Remark 3.30. (*Choice of the group*) For the attack of Remark 3.28 to cost at least 2^ς operations, the group should satisfy

$$\log_2 q \;\ge\; \max\{\,2\varsigma,\; (a+1)(\varsigma - a)\,\}; \tag{3.31}$$

e.g., for $\varsigma = 128$, a 256-bit group suffices only for $\bar{n} \leq 3$, while $\bar{n}$ up to 7, 15 and 127 requires about 380, 500 and 850 bits respectively. These are estimates for Wagner's algorithm with constants neglected and memory unlimited, i.e., upper bounds on security rather than guarantees; refinements of the algorithm for numbers of lists other than powers of two (cf. [30]) may lower them somewhat, while an impersonator with less memory than above has to repeat the search and pays accordingly more. Session binding (cf. Section 3.1.1) leaves them unaffected, since it merely requires the search to be completed before the session expires; halving the timeout gains only a single bit.

The formal definitions regarding condition (3.17) are given in Appendix D.

3.2 Practical efficiency

The provers remain unaffected by network overhead because the proving phase is non-interactive. That is, apart from the inherent delays of asynchronicity and packet transmission, system latency depends entirely on the performance of the verifier. Verification amounts to k ordinary Schnorr verifications and one interpolation in the exponent, $k \equiv |Q|$, i.e., to well-studied and widely deployed operations. Specifically, the per-component conditions of (3.3) require $2k$ exponentiations, while the interpolation condition requires k exponentiations and is directly amenable to multi-exponentiation (cf. [31], [36]). The interpolation coefficients require $O(k^2)$ field multiplications (cf. (B.5)), while the modular inversions involved can reduce to a single one by simultaneous inversion (cf. [32]). By contrast, the compact variant requires $k + 2$ exponentiations, all but one of them within a single multi-exponentiation (cf. condition (3.17)); however, this is to its advantage for small groups only (cf. Section 3.1.2).

3.2.1 Implementation challenges

No peer-to-peer software is required because the provers operate asynchronously; similarly, no low-latency overlay network needs to be maintained between them because the protocol is non-interactive. The implementation effort concentrates on the verifier, i.e., the secure software required for verifying condition (3.3), and its associated optimizations. While well-studied and widely implemented, multi-exponentiation algorithms require profiling and tuning, so that the tradeoff between window size and runtime remains effective under the specific hardware constraints of the verifier (including the capacity of its CPU cache). This is a local optimization problem, potentially addressable by vertical scaling, whose solution does not require governance and is more tractable than globally tuning an overlay network. Beyond algebraic optimizations, OSST verifiers should be thought of as state machines that have to address several interconnected issues related to session management. These include the exact aggregation model for the incoming components, the scheduling of incremental verification (cf. Section 3.3.1) and their defense mechanisms against session poisoning (cf. Section 3.4.2); the exact decisions and configuration belong obviously to the specific use case.

3.3 Operational considerations

Below we examine OSST from the operability viewpoint. In Section 3.3.1 we show how it achieves incremental verification during its stable phase. In Section 3.3.2 we show how that same mechanism fixes its verification time, allowing for predictable latencies after a baseline coalition has been detected. In Section 3.3.3 we describe how the protocol decouples accountability from authentication, effectively serving itself as a share attestation mechanism.

3.3.1 Incremental verification

Being non-interactive on the provers' side, the protocol facilitates incremental verification (cf. Section 2.3.2) allowing the provers to join and leave dynamically. Concretely, every component is checked upon arrival against the respective condition of (3.3) and discarded if it does not verify. If the verifier retains the attached shares $\{y_i\}_{i\in Q}$ of an already identified coalition Q, any shareholder $j \notin Q$ can attach itself asynchronously to the session without requiring Q to rerun the proving phase; the verifier checks the component (y_j, u_j, s_j) alone and recomputes the interpolation condition for the augmented set $Q^* \equiv Q \cup \{j\}$, in order to ensure that y_j is consistent with the sharing in use. In this context, Q will be referred to as a *baseline coalition*. Once a baseline coalition has been detected, the session enters its stable phase, where a joining shareholder cannot affect the components already accepted (cf. Section 3.4.2). It should be stressed, however, that accepting (y_j, u_j, s_j) against a retained baseline lies outside the security notions of Section 4, which assume that no honest shareholder participates in the attacked session.

Various strategies are here possible for reducing the costs of repeated verification. For example, instead of computing the interpolation coefficients of Q^* from scratch, the verifier can retain those of Q and update them as

$$\lambda_i \leftarrow \lambda_i\, j\,(j-i)^{-1}, \quad i \in Q, \qquad \lambda_j \leftarrow \prod_{i\in Q} i\,(i-j)^{-1}$$

(cf. (B.5)); this requires $O(k)$ field operations and a single modular inversion instead of $O(k^2)$ field operations (cf. Section 3.2), a gain that becomes significant for large k's. Similarly, since the retained shares serve as the bases of the interpolation condition in every iteration, multi-exponentiation algorithms with per-base precomputation (cf. [31]) are particularly tailored to the needs of incremental verification; the precomputed elements of a share are computed once and reused for as long as it is retained.

3.3.2 Fixed-time verification

The verifier response time is generally unpredictable due to the asynchronous nature of the protocol. Specifically, since individual provers operate at their own pace, the verifier has to defer its responses until a baseline coalition is detected among the continuously aggregated components. If no session timeout is applied and a baseline coalition never identifies, the individual provers have to await indefinitely.

The situation is different after some baseline coalition Q has been detected. During this phase, when a shareholder sends its proof to the verifier, the latter's response time is determined by the time required for processing it in the backend; no random costs occur from asynchronicity and —aside from packet transmission— latency is that of incremental verification (cf. Section 3.3.1). This seemingly depends on $k = |Q|$, meaning that response time grows with the number of onboarded entities if the baseline coalition is incrementally augmented with them. However, we can avoid this by verifying incoming components against a fixed baseline coalition of minimal size, i.e., $|Q| = t$ (cf. Remark B.3); in this case, verification time becomes a system-wide constant and protocol efficiency stabilizes around its value. Latency does not reveal the number of onboarded shareholders.

3.3.3 Session-scoped accountability – Combiner as-a-service

Even if individual identities remain opaque in the output, threshold operations typically unfold in a permissioned environment where the shareholders participate accountably. In a signature scheme, the combiner validates the partial signatures before forwarding their combination to the outer world; similarly, during threshold decryption, the combiner validates each partial decryptor against the respective share in order to ensure its consistency with the ciphertext. In general, the combiner must trust the public shares in advance; since the received packets typically contain an embedded proof-of-knowledge for the corresponding secret share, these simultaneously serve as means to authenticate each shareholder individually, thereby granting them access to the permissioned environment where the combiner offers its services. This is an implicit application of the trivially decoupled protocol described in Section 2.8.2.

In closed committees (i.e., if the combiner is a shareholder), trust is established during group constitution (cf. Section 2.1.1); this is the case of a financial consortium that maintains a combiner for outputting transaction signatures. If the combiner is external to the group, it must still belong to a closed organization where trusted entities attest to the correctness of the advertised shares, while the system itself must be ephemeral to avoid the complications of proactive sharing. This is the case of an election server that combines partial decryptors received from the board of trustees to decrypt the collected votes. In open or cross-domain settings, neither of these approaches is practical and the feasibility of an external combiner API is bounded by the impracticality of universal share certification (cf. Section 2.2.1); the combiner cannot be deployed as-a-service unless able to enforce its custom certification policies on every registered group and further monitor the lifecycle of its individual member identities. Interoperability decreases while the attack surface becomes unmanageable.

Nevertheless, onboarding a coalition of shareholders to a permissioned environment and holding them accountable for their activity does not necessarily require authenticating them individually against already trusted shares; while a trusted share mechanism ensures both at once, the two can probably be decoupled, especially if accountability is a session-scoped requirement. Indeed, onboarding is the objective of any real-time threshold identification protocol, while accountability follows automatically if the involved public shares are — implicitly or explicitly— attached to its transcript.

Let y be a group public key that has been registered on a combiner service $\mathcal{C}$ (e.g., a signature gateway or delegated decryption service). When a coalition of shareholders request access to it, $\mathcal{C}$ executes OSST in the role of verifier in order to identify their ensemble over the respective communication channels and thus onboard them to its permissioned environment. Before the session, $\mathcal{C}$ does not know the onboarded entities' public shares and consequently cannot hold them accountable for their actions. However, $\mathcal{C}$ receives the shares attached to the respective components, each along with a proof of knowledge for its logarithm (cf. Remark 3.4), and can use them to validate any subsequent activity during the current session; in the compact variant, the shares are inferred in the form (3.26) instead. It does not need to know and trust their fixed shares in advance. It should be stressed, however, that these are the shares declared by the coalition for the session and need not belong to any fixed sharing (cf. Remark 3.4); accountability is thus scoped to the session and does not bind the onboarded entities to persistent identities.

This is how combiners can operate in open systems without relying on pre-validations. The consortium of signers does not need to maintain an internal combiner for constructing their threshold signature; they can use an external API, provided that their group public key has been registered on it. They may not even need to maintain a peer-to-peer network if the combiner can operate as a relay node between them. Conversely, the combiner does not

need to certify the consistency of any individual member identities or monitor their refresh; it needs only trust the entity that registers the group public key, once and upon registration (e.g., because it pays for the service). It can be deployed conveniently as-a-service.

3.4 Security considerations

Since OSST decouples into independent Schnorr proofs, any security precaution regarding the latter applies to the former. For example, the protocol remains susceptible to side-channel attacks if the random number generator leaks linear relations between the discrete logarithms of distinct commitments or partial information about them (cf. the lattice attacks on partially known nonces of [22, 33]). We here focus on high-level attacks that arise particularly in the distributed setting. The concurrent-session attacks against two-round Schnorr multi-signatures [16, 9] do not apply (cf. Corollary 5.24) because they presuppose a common challenge derived from an aggregate commitment; it should be stressed, however, that the decoupled nature of challenges enables precisely the generalized birthday attack against large coalitions in the compact variant (cf. Remark 3.28).

3.4.1 Generalized replay attack

OSST is as much susceptible to replay attacks as the ordinary Schnorr protocol; a man-in-the-middle can passively capture and replay the components sent by a coalition of provers in order to impersonate them anytime. The captured components can only be replayed as they are, since every response is determined by the respective attached share and commitment (cf. (3.3)). In the compact variant, by contrast, the attacker has $|Q| - 1 \geq t - 1$ degrees of freedom in modifying $\{s_i\}_i$ while keeping $\{u_i\}_i$ fixed. Let $\{(u_i, s_i)\}_{i\in Q}$ be a valid proof of the compact variant with $|Q| \geq t$ and weights $\{\mu_i\}_{i\in Q}$. For fixed $j \in Q$, pick arbitrary $\{\delta_i\}_{i\in J} \subset \mathbb{Z}_q$ for $J = Q \setminus \{j\}$ and define

$$\delta_j = -\frac{1}{\mu_j} \sum_{i\in J} \mu_i \delta_i \,.$$

Then $\{(u_i, s_i^*)\}_{i\in Q}$ with $s_i^* = s_i + \delta_i$ remains valid because

$$\sum_{i\in Q} \mu_i s_i^* = \sum_{i\in Q} \mu_i s_i + \Big(\mu_j \delta_j + \sum_{i\in J} \mu_i \delta_i\Big) = \sum_{i\in Q} \mu_i s_i$$

while condition (3.17) involves the responses only through this sum. This generalizes the replay attack of the non-distributed case $n = t = 1$, where the intercepted proof can only be replayed as is. In both protocols, replay attacks are mitigated by the same method as in the non-distributed case, i.e., by baking a nonce chosen by the verifier per session into the challenges (cf. the session-bound variant of Section 3.1.1, and Corollaries 5.24 and D.27).

3.4.2 Session poisoning – Reactive fault detection

Like all share-free thresholds, OSST is susceptible to the DoS attack of Section 2.5.1, obstructing a legitimate coalition $Q \subset \{1, \ldots, n\}$, $|Q| \geq t$, from being identified. A component with arbitrary elements is of no use to the attacker, since it does not verify and is discarded upon arrival (cf. Section 3.3.1). However, the attacker may submit

$$(y_j, u_j, s_j),\ y_j \leftarrow g^{x_j^*},\ x_j^* \leftarrow_{\$} \mathbb{Z}_q,\ j \notin Q$$

with an ordinary Schnorr proof for y_j of its choice. This component verifies, but its attached element is not a public share; the coalition $Q^* = Q \cup \{j\}$ thus fails the interpolation condition of (3.3) and, in case of bulk verification, service is denied to Q. That is, the per-component conditions filter out malformed components but cannot tell a fabricated element from a public share (cf. Remark 3.4). In the compact variant, no component can be checked individually, so that even a pair of arbitrary elements (u_j, s_j) suffices. Nevertheless, reactive fault detection facilitates immunity to poisoning after a baseline coalition has been identified; consequently, it is the verifier's priority to detect such a coalition as early as possible within a pool of aggregated components. In Protocol 3.1, this search involves the attached elements alone, the per-component conditions being checked only once. The exact scheduling belongs to the specific use case and the hardware capabilities of the verifier; we here remark that it can benefit greatly from the per-base precomputation of the attached elements (cf. Section 3.3.1) and the parallel evaluation of the interpolation condition across separate threads. In any case, uncritically exhausting combinations can drain the verifier's resources in case of flooding and must be prevented via rate limiting or CPU throttling.

4 Security notions

In this section, we derive and stipulate the OSST threat model. Attack games are formulated in the asymptotic framework and negligibility is meant with respect to the bitlength of the group order. In Section 4.1, we outline security against impersonation and relate it to a pragmatic design requirement, namely a distributed knowledge property. We further argue that this property can be derived under DL hardness by fork and extraction, applied separately to every component of the fabricated proof (cf. Section 5). The working model for impersonation attacks is defined in Section 4.2. In Section 4.2.1, we extend it to adversaries who observe transcripts of other sessions, regarding the session-bound variant of the protocol (cf. Section 3.1.1). The respective notions for the compact variant are given in Appendix D.2.

4.1 Threat model

The current standard for identification protocols is security against active impersonation attacks. This refers to adversaries who not only intercept multiple identification sessions before mounting their actual attack, but also engage in them actively by impersonating a legitimate verifier (e.g., by cloning a website in order to collect information about the user's credentials); after several such interactions, the adversary switches roles and attempts to identify itself against some verifier as a legitimate user. We will not pursue this path, focusing on passive adversaries who potentially exert some control over the network (cf. Section 4.2.1). Once the relation between threshold and ordinary identification has been clarified from the security viewpoint, security against active adversaries can probably be established by techniques similar to those of [8], albeit under the stronger OMDL hardness assumption (cf. Appendix A). A weaker form of interaction, namely the observation of transcripts in sessions of the adversary's choice, is considered in Section 4.2.1 for the session-bound variant of the protocol (cf. Section 3.1.1).

Aside from observed transcripts (cf. Remark 4.4), the OSST threat model consists of an adversary $\mathcal{A}$ who knows up to $t-1$ secret shares and attempts to fabricate a proof $\{(y_i, u_i, s_i)\}_{i \in Q}$, $Q \subset \{1, \ldots, q-1\}$, that is valid against the combined public key, i.e.,

impersonate the group under indices of its choice. We confine ourselves to the static model, assuming that the attacker decides from the outset the set of corrupted parties. This is no restriction as long as $\mathcal{A}$ observes no transcripts, since any $t-1$ shares are uniformly distributed and independent of the combined public key (cf. Remark 4.3). Adaptive corruption during the key setup phase belongs to the threat model of the particular use case. Likewise, binding the session to the designated communication channels (cf. Remark 2.1) is a matter of deployment, as in the non-distributed case, and is not modelled by the attack games below. We could further assume that $\mathcal{A}$ knows the public shares; indeed, although not advertised, these are attached to every proof and thus revealed by any intercepted session. The security proof of Section 5.5 remains intact, since $\mathcal{A}$ can anyway compute them efficiently (cf. Remark 5.19).

Let $(x_1, \ldots, x_n;\, y)$ be the given (n,t)-sharing of some secret key x and $\mathcal{A}$ be an adversary who knows the shares $\{x_i\}_{i\in J}$ with $|J| \leq t-1$. Let $Q \subset \{1, \ldots, q-1\}$ be the indices of a purportedly valid proof $\{(y_i, u_i, s_i)\}_{i\in Q}$ fabricated by $\mathcal{A}$. Note that no restriction is imposed on Q; in particular, $\mathcal{A}$ may use less than t indices (e.g., $Q \subset J$), arbitrarily many indices of uncorrupted shareholders or indices that correspond to no shareholder at all. Moreover, $\mathcal{A}$ is free to fabricate all attached shares and commitments, including those of the corrupted shareholders; that is, the attached elements need not be the public shares of the given sharing (cf. Remark 3.4). We conclude with the following attack model.

Game 4.1. ($\mathsf{OSST}^{\mathbf{imp}}_{\mathbb{G},n,t}$ – *Impersonation*) Given adversary $\mathcal{A}$ with challenger $\mathcal{C}$,

- *Attack statement and corruption*:
  - $\mathcal{A}$ chooses $J \subset \{1, \ldots, n\}$ with $|J| \leq t-1$.
  - $\mathcal{A}$ chooses $Q \subset \{1, \ldots, q-1\}$ and sends it along with J to its challenger $\mathcal{C}$.
  - $\mathcal{C}$ runs $(x_1, \ldots, x_n;\, y) \leftarrow D(g, x)$, $x \leftarrow_{\$} \mathbb{Z}_q$, and sends $y, \{x_i\}_{i\in J}$ to $\mathcal{A}$.
- *Impersonation*: $\mathcal{A}$ outputs a proof of the form $\sigma = \{(y_i, u_i, s_i)\}_{i\in Q}$.

$\mathcal{A}$ wins if $\mathcal{V}(y, \sigma) = \mathsf{true}$.

Note that successful impersonation is equivalent to condition (3.13) for σ. Security against impersonation should mean that every polynomial-time adversary $\mathcal{A}$ wins the above game with at most negligible advantage. The core requirement behind is that if $\mathcal{A}$ outputs a valid proof $\{(y_i, u_i, s_i)\}_{i\in Q}$ then it should necessarily know x_j for some $j \notin J$; i.e., impersonation should be infeasible without knowledge of at least t secret shares. We will refer to this requirement as the OSST *knowledge property*. Since $\{x_i\}_{i\in J}$ along with either x or any x_j, $j \notin J$, uniquely determine the sharing (cf. Appendix B), the knowledge property equivalently requires that $\mathcal{A}$ know the combined secret x.

This is an instance of a more general pattern. Security notions for Schnorr-based protocols are typically intermediated by some knowledge property, which codifies a pragmatic design requirement. Note that the above knowledge property cannot be formally reduced to ordinary Schnorr in a straight-line fashion because the attached shares are fabricated by the adversary along with the commitments. Consequently, we have to derive it by means of fork and extraction in the random oracle model; specifically, an appropriate extraction procedure should itself be reproducible within any forkable impersonation attack. In this case, assuming $\mathcal{A}$ wins with some non-negligible advantage, one has to construct a DL-adversary $\mathcal{A}^*$ who perfectly simulates $\mathcal{C}$ while also having forkable black-box access to $\mathcal{A}$; upon successful completion of $\mathcal{A}$'s first attack, $\mathcal{A}^*$ should be able to fork $\mathcal{A}$ at some special points, let it complete further successful attacks with non-negligible advantage and finally solve DL by extracting the desired logarithm from the resulting partially differing transcripts.

Every component of a valid proof is an ordinary Schnorr proof for the logarithm of its attached share, while the attached shares interpolate to y (cf. Proposition 3.12); that is, $x = \sum_{i \in Q} \lambda_i \log y_i$ and the knowledge property follows if the logarithm of every attached share can be extracted. A single fork yields one such logarithm by the usual extraction formula and leaves the remaining ones undetermined. However, this obstacle pertains to the extraction procedure rather than the protocol itself. Forking a single attack separately at the oracle query of each component, and repeating every fork up to a prescribed number of times until it reproduces the component of its own position, yields the logarithms of all attached shares, so that x is finally obtained by interpolation (cf. Section 5); for $|Q| = 1$, a single fork may be used instead, as for the non-distributed protocol. In view of Game 4.1, this means that OSST security can be reduced to DL hardness alone, without restricting the above attack model, albeit at the cost of a running time that is inversely proportional to the adversarial advantage (cf. Remark 5.20). Note also that the corrupted shares play no role in extraction; the threshold enters the argument only through the simulation of $\mathcal{A}$'s view, which is possible because any $t - 1$ shares are independent of x (cf. Remark B.7).

4.2 Security against impersonation attacks

We adopt Game 4.1 as working model for impersonation attacks. The probability that $\mathcal{A}$ wins this game is denoted by

$$\mathsf{Adv}^{\mathsf{OSST}}_{\mathbb{G},n,t}[\mathcal{A}]. \tag{4.2}$$

OSST *security against impersonation attacks* (*over* $\mathbb{G}$) means that (4.2) is negligible for every choice of n and t and every polynomial-time adversary $\mathcal{A}$.

Remark 4.3. (*Adversarial components*) Since J and Q are stated before $\mathcal{A}$ receives any input, $\mathcal{A}$ is a pair of non-interactive algorithms $(\mathcal{A}_0, \bar{\mathcal{A}})$ of the form

$$(J, Q) \leftarrow \mathcal{A}_0(\cdot), \quad \{(y_i, u_i, s_i)\}_{i \in Q} \leftarrow \bar{\mathcal{A}}(y, \{x_i\}_{i \in J}),$$

where $\bar{\mathcal{A}}$ consumes $\mathcal{A}$'s view during the impersonation phase in order to fabricate $\{y_i\}_{i \in Q}$, $\{u_i\}_{i \in Q}$ and $\{s_i\}_{i \in Q}$ such that $\{(y_i, u_i, s_i)\}_{i \in Q}$ is a valid proof. Note that, when using $\mathcal{A}$ in a black-box reduction, the logarithms of the attached shares y_i, $i \in Q$, cannot be assumed to be known (not even for $i \in J$, since y_i need not equal g^{x_i}) and we will have to extract them explicitly. Note also that $\mathcal{A}_0$ is not prevented from querying the random oracle; consequently, Q (and thereby the Lagrange coefficients λ_i, $i \in Q$) may depend on hash values (cf. Section 5.1). Stating J and Q in advance is no restriction. The reduction of Section 5 reads Q off the fabricated proof and hands out the shares of J from uniformly sampled scalars, independently of y (cf. Lemma 5.1); consequently, the proof of Theorem 5.17 carries over verbatim to an adversary who corrupts up to $t - 1$ shareholders one at a time after receiving y and chooses Q upon output.

Remark 4.4. (*No observed transcripts*) In Game 4.1, $\mathcal{A}$ observes no proofs produced by the honest shareholders in previous sessions; that is, the attack model is passive in the most restrictive sense. It should be stressed that this genuinely restricts the attack model, since a real adversary has most probably intercepted earlier sessions before mounting its actual attack. This is the replay attack of Section 3.4.1 at the level of a single component; the corresponding extension of the threat model is developed in Section 4.2.1.

4.2.1 Observed transcripts

We proceed to the extension of the threat model suggested in Remark 4.4 for the session-bound variant of the protocol (cf. Section 3.1.1). An honest execution of the proving phase by the k-th shareholder in session sid is modelled by the oracle

$$\mathcal{O}^{\mathsf{prove}}(k, \mathsf{sid}) \;=\; (y_k, u, s), \quad y_k \equiv g^{x_k}, \quad s \leftarrow r + H(y_k, u, \mathsf{sid})\, x_k, \quad u \leftarrow g^r, \quad r \leftarrow_{\$} \mathbb{Z}_q\,,$$

whose output we refer to as an *honest component*.

Game 4.5. ($\mathsf{OSST}^{\mathbf{tr}}_{\mathbb{G},n,t}$ – *Impersonation with observed transcripts*) Given adversary $\mathcal{A}$ with challenger $\mathcal{C}$,

- *Attack statement and corruption*: as in Game 4.1.
- *Observation*: $\mathcal{A}$ may query $\mathcal{O}^{\mathsf{prove}}(k, \mathsf{sid})$ for any $k \in \{1, \ldots, n\} \setminus J$ and $\mathsf{sid} \in \mathcal{N}$ of its choice.
- *Impersonation*: $\mathcal{C}$ samples $\mathsf{sid}^* \leftarrow_{\$} \mathcal{N}$ and sends it to $\mathcal{A}$, who may continue querying $\mathcal{O}^{\mathsf{prove}}$ and finally outputs a proof $\sigma = \{(y_i, u_i, s_i)\}_{i \in Q}$.

$\mathcal{A}$ wins if $\mathcal{V}_{\mathsf{sid}^*}(y, \sigma) = \mathsf{true}$ and no query to $\mathcal{O}^{\mathsf{prove}}$ has been issued on sid^*.

That is, before and during its impersonation attempt, $\mathcal{A}$ may observe honest executions of the proving phase by any uncorrupted shareholder in sessions of its own choice, while the attacked session is opened by an honest verifier. Observe that $\mathcal{A}$ may as well play the verifier in the observed sessions, since in the session-bound variant the only influence of the verifier upon the provers is the choice of sid. If $\mathcal{A}$ issues no queries to $\mathcal{O}^{\mathsf{prove}}$, the game essentially reduces to Game 4.1, $H(\cdot, \cdot, \mathsf{sid}^*)$ being itself a random oracle in the random oracle model. The probability that $\mathcal{A}$ wins Game 4.5 is denoted by

$$\mathsf{Adv}^{\mathsf{OSST},\mathbf{tr}}_{\mathbb{G},n,t}[\mathcal{A}]. \tag{4.6}$$

Security of session-bound OSST *against impersonation attacks with observed transcripts* (*over* $\mathbb{G}$) means that (4.6) is negligible for every choice of n and t and every polynomial-time adversary $\mathcal{A}$ (cf. Section 5.5.1).

5 Security proof

In this section, we prove OSST security against impersonation attacks (cf. Game 4.1) under DL hardness in the random oracle model. The rationale behind the attack model is developed throughout Section 4.1. Our reduction extracts the logarithm of every attached share by the usual extraction formula of the Schnorr protocol and obtains the combined secret x by interpolation (cf. Proposition 3.12). Since all attached shares are fabricated by $\mathcal{A}$, a single fork yields only one of these logarithms (cf. Remark 4.3); instead of assuming the remaining ones known to the reduction, we extract them by forking a single execution separately at each oracle query for the respective components. Since a fork need only reproduce the component submitted at its own position, it is repeated up to a prescribed number of times until it does so, along the lines of the generalized forking lemma of [3]; we state the counterpart of the latter for the Local Forking Lemma of [6], in which a fork re-samples the hash at its own position alone. The attack simulator is presented in Section 5.1, along with the simulation

of the attacked sharing; validity is expressed in terms of linear relations over the simulated oracle queries in Section 5.2. The multiple local fork is defined and bounded in Section 5.3 and the resulting extraction procedure is given in Section 5.4. The main theorem follows in Section 5.5; in Section 5.5.1, we extend it to the session-bound variant in the presence of observed transcripts (cf. Game 4.5), by simulating the honest components without knowledge of the respective shares. For the compact variant, every fork must reproduce the indices of the fabricated proof along with the positions of all its components; the forks are nested instead and the logarithms of the commitments are eliminated jointly (cf. Appendix D.3).

5.1 Simulated impersonation attack

We define an algorithm $\mathcal{S}[\mathcal{A}]$ that simulates Game 4.1 for an adversary $\mathcal{A}$, provided that H is modelled as a random oracle. Roughly speaking, $\mathcal{S}[\mathcal{A}]$ accepts the ingredients of $\mathcal{A}$'s view and outputs a proof along with its indices and the oracle queries at which its components were submitted. The simulation is perfect in the sense that, if the algorithm is fed with uniform input, the probability of success equals the adversarial advantage of winning the real attack game. We first observe that the attacked sharing need not be simulated explicitly.

Lemma 5.1. (*Sharing simulation*) *Let* $J \subset \{1, \ldots, n\}$ *with* $|J| \leq t-1$. *The probability distribution*

$$\{(y, \{x_i\}_{i\in J}) : y \leftarrow_\$ \mathbb{G}, \quad x_i \leftarrow_\$ \mathbb{Z}_q, \ i \in J\}$$

is identical to $\{(y, \{x_i\}_{i\in J}) : (x_1, \ldots, x_n; y) \leftarrow D(g, x), \ x \leftarrow_\$ \mathbb{Z}_q\}$.

Proof. Fix $x \in \mathbb{Z}_q$ and write $f(X) = x + \sum_{j=1}^{t-1} a_j X^j$ for the polynomial of Definition B.1, whose coefficients $a_1, \ldots, a_{t-1}$ are uniform. The linear map $(a_1, \ldots, a_{t-1}) \mapsto (f(i) - x)_{i\in J}$ is onto, since its matrix $(i^j)_{i\in J, 1\leq j\leq t-1}$ has full row rank, the indices being distinct, non-zero and at most $t-1$ in number. Consequently, $\{x_i\}_{i\in J}$ is uniformly distributed and independent of x (cf. Remark B.7). The claim follows because $y = g^x$ is uniform for $x \leftarrow_\$ \mathbb{Z}_q$. □

That is, the challenger of Game 4.1 can be simulated by forwarding a uniform public key along with uniform and independent shares, without reference to the secret x or the remaining shares.[8] More accurately, given a black-box adversary $\mathcal{A} = (\mathcal{A}_0, \bar{\mathcal{A}})$ to game $\mathsf{OSST}^{\mathbf{imp}}_{\mathbb{G},n,t}$ (cf. Remark 4.3), the desired simulation is a wrapper of $\mathcal{A}$ of the form

$$(\pi, \sigma) \leftarrow \mathcal{S}[\mathcal{A}](y^*, \omega; h_1, \ldots, h_m), \quad \omega = (\theta_{\mathcal{A}}, x_1^*, \ldots, x_n^*),$$

arranging the input values appropriately in order to emulate $\mathcal{A}$'s execution environment. The exact procedure is shown in Algorithm 1. The value y^* is the attacked public key, $\theta_{\mathcal{A}}$ are the random coins of $\mathcal{A}$ and $x_1^*, \ldots, x_n^*$ are scalars from which the shares known to $\mathcal{A}$ are taken; that is, once $\mathcal{A}$ has stated J, the virtual challenger forwards $x_i = x_i^*$, $i \in J$. Next, the embedded replica of $\mathcal{A}$ consumes its view in order to fabricate a purportedly valid proof σ with the difference that, instead of invoking H directly, it queries an embedded hash oracle $\mathcal{O}^{\mathsf{hash}}$ with tape $h_1, \ldots, h_m$ (cf. Appendix C). If needed, $\mathcal{A}$ may be reprogrammed without loss of generality to query hashes for all output components, so that the verifier issues no fresh queries and m bounds the number of distinct queries issued during the simulation. An OSST-verifier $\mathcal{V}$ is included with access to the same hash oracle in order to verify the output.

[8] Observe that this is the only point where the corruption bound $|J| \leq t-1$ enters the security proof (cf. Section 4.1); for $|J| \geq t$, the shares would determine x anyway.

Algorithm 1 $\mathcal{S}[\mathcal{A}](y^*, \omega; h_1, \ldots, h_m)$

1: $h \leftarrow [h_1, \ldots, h_m],\ \nu \leftarrow 1,\ \mathsf{Map}_1 = \varnothing,\ \mathsf{Map}_2 = \varnothing$ // *Setup* $\mathcal{O}^{\mathsf{hash}}$
2: $(J, Q) \leftarrow \mathcal{A}_0(\cdot)$ // *Attack statement*
3: $y \leftarrow y^*,\ x_i \leftarrow x_i^*,\ i \in J$ // *Corruption*
4: $\{(y_i, u_i, s_i)\}_{i \in Q} \leftarrow \bar{\mathcal{A}}(y, \{x_i\}_{i \in J})$ // *Impersonation*
5: $res \leftarrow \mathcal{V}(y, \{(y_i, u_i, s_i)\}_{i \in Q})$
6: **for** $i \in Q$
7: $\quad c_i \leftarrow \mathsf{Map}_1[y_i, u_i],\quad k_i \leftarrow \mathsf{Map}_2[y_i, u_i]$
8: $\sigma \leftarrow \{(y_i, u_i, c_i, s_i)\}_{i \in Q},\quad \psi \leftarrow \{k_i\}_{i \in Q}$
9: **if** $res = \mathsf{true}$
10: $\quad$ **return** $((Q, \psi), \sigma)$
11: **return** $(\perp, \sigma)$

Concretely, the hash oracle $\mathcal{O}^{\mathsf{hash}}$ is first initialized with tape $[h_1, \ldots, h_m]$ and empty lookup tables. After $\mathcal{A}$ makes its attack statement (step 2), the virtual challenger forwards $y \leftarrow y^*$ along with the shares $\{x_i\}_{i \in J}$ (step 3) and $\mathcal{A}$ proceeds to impersonation (step 4). Note that $\mathcal{A}_0$ may itself query the oracle (cf. Remark 4.3), so that J and Q may depend on the tape and need not remain the same in an execution that is forked with the same random coins at the position of such a query. Neither needs to be reproduced: J plays no role in extraction, while a forked execution is only required to reproduce the pair of its own fork position (cf. Section 5.4). Observe that the random oracle queries are implicitly issued at steps 2 and 4, including the challenges $c_i \leftarrow \mathcal{O}^{\mathsf{hash}}(y_i, u_i)$, $i \in Q$, which are collected and attached to the fabricated proof for later usage (steps 6-8). We also locate the positions $k_i \in \{1, \ldots, m\}$ for which $c_i = h[k_i]$ (step 7), indicating that the pair (y_i, u_i) was submitted at the k_i-th distinct query; note that $k_i = k_l$ whenever $(y_i, u_i) = (y_l, u_l)$. If $\mathcal{A}$ succeeds in its impersonation attempt, the simulation returns as main output the indices of the proof along with these positions, i.e., $\pi = (Q, \psi)$ with $\psi \equiv \{k_i\}_{i \in Q}$, and as side output the proof annotated with its challenges, which we still denote by σ; otherwise, it returns $(\perp, \sigma)$ to indicate failure.

Remark 5.2. (*Coins as input*) The random coins $\theta_{\mathcal{A}}$ form part of the input, so that any two executions of $\mathcal{S}[\mathcal{A}]$ with the same y^* and ω run $\mathcal{A}$ with the same coins and the same candidate shares. Consequently, $\mathcal{S}[\mathcal{A}]$ is a deterministic algorithm with (y^*, ω) in the role of the random coins and the tape $h_1, \ldots, h_m$ as its remaining input, as required for forking (cf. Section 5.3).

Remark 5.3. (*Queries as functions of the tape prefix*) Since every element is fabricated before being submitted to the oracle, the element submitted at the k-th distinct query in an execution of $\mathcal{S}[\mathcal{A}]$ is determined by y^*, ω and the preceding hashes $h_1, \ldots, h_{k-1}$. In particular, two executions with the same y^* and ω whose tapes coincide at all positions below k submit the same elements at all positions up to k. This is the only property of $\mathcal{A}$ that our extraction procedure will require (cf. Section 5.4).

Proposition 5.4. ($\mathsf{OSST}^{\mathbf{imp}}_{\mathbb{G},n,t}$ – *Perfect simulation*) *If H is modelled as a random oracle, then $\mathcal{S}[\mathcal{A}]$ simulates Game 4.1 perfectly if invoked with uniformly random input, in the sense that the real and simulated game executions are indistinguishable from the adversary's*

viewpoint. More accurately, the probability distribution of $\mathcal{A}$'s view in the real attack game is identical to the distribution of $\mathcal{A}$'s view in the invocation

$$\begin{aligned} (\pi, \sigma) \leftarrow \mathcal{S}[\mathcal{A}](y^*, \omega; h_1, \ldots, h_m), \quad \omega = (\theta_{\mathcal{A}}, x_1^*, \ldots, x_n^*), \\ y^* \leftarrow_{\$} \mathbb{G}, \quad x_i^* \leftarrow_{\$} \mathbb{Z}_q,\ 1 \le i \le n, \quad h_\nu \leftarrow_{\$} \mathbb{Z}_q,\ 1 \le \nu \le m, \end{aligned} \tag{5.5}$$

where $\theta_{\mathcal{A}}$ is distributed as the coins of $\mathcal{A}$. In the context of this simulation, if $\bar{\mathcal{A}}$ succeeds in its impersonation attempt, i.e., $\pi \neq \perp$, the input value h_{k_i} maps to $c_i \equiv H(y_i, u_i)$ of the real game execution for every $i \in Q$.

Proof. In the real game execution, the adversarial view consists of the random coins of $\mathcal{A}$, the combined public key y along with the compromised secret shares $\{x_i\}_{i \in J}$, and the hashes queried during the attack. The coins are identically distributed by definition, while the hashes are perfectly simulated due to Lemma C.1. Given J, the challenger samples the sharing independently of the coins and the hashes; in the simulated execution, y^* and $\{x_i^*\}_{i \in J}$ are similarly independent of $\theta_{\mathcal{A}}$ and the tape, irrespective of how J depends on the latter. The public key and the shares are thus perfectly simulated due to Lemma 5.1. The fact that $h_{k_i} = h[k_i]$ corresponds to $H(y_i, u_i)$ follows directly from step 7, Algorithm 1. □

We denote by $\mathsf{acc}(\mathcal{S}[\mathcal{A}])$ the probability that the embedded adversary $\bar{\mathcal{A}}$ succeeds in the context of a simulated impersonation attack when run with uniformly random input (cf. Section 5.3). This relates to the real adversarial advantage as follows.

Corollary 5.6. *Given an adversary $\mathcal{A}$ for Game 4.1, there holds*

$$\mathsf{acc}(\mathcal{S}[\mathcal{A}]) \;=\; \mathsf{Adv}^{\mathsf{OSST}}_{\mathbb{G},n,t}[\mathcal{A}]. \tag{5.7}$$

Proof. Direct consequence of Proposition 5.4. □

5.2 Validity as linear relations

We proceed to express the validity of a simulated proof in terms of the oracle queries of Section 5.1. Given an execution of $\mathcal{S}[\mathcal{A}]$, denote by $v_k \equiv (Y_k, U_k)$ the pair submitted at the k-th distinct oracle query, $1 \le k \le m$, and let Q, $\sigma = \{(y_i, u_i, c_i, s_i)\}_{i \in Q}$ and $\psi = \{k_i\}_{i \in Q}$ be as computed at steps 2-8 of Algorithm 1, irrespective of whether $\mathcal{A}$ succeeds; then $y_i = Y_{k_i}$, $u_i = U_{k_i}$ and $c_i = h_{k_i}$ for every $i \in Q$ (step 7). We define

$$P(\psi) \equiv \{\, k_i : i \in Q \,\}, \quad \Lambda_k \equiv \sum_{i \in Q:\, k_i = k} \lambda_i, \quad k \in P(\psi), \tag{5.8}$$

where $\{\lambda_i\}_{i \in Q}$ are the interpolation coefficients for Q. That is, $P(\psi)$ collects the distinct positions at which the components were submitted, so that $|P(\psi)| \le |Q|$ with strict inequality if some pair is repeated, while Λ_k accumulates the coefficients of the components submitted at position k. Both depend only on Q and ψ, i.e., on the main output in case of success.

Proposition 5.9. (*Validity as linear relations*) *Given an execution of $\mathcal{S}[\mathcal{A}]$, let Q, $\sigma = \{(y_i, u_i, c_i, s_i)\}_{i \in Q}$ and $\psi = \{k_i\}_{i \in Q}$ be as above. Then $\{(y_i, u_i, s_i)\}_{i \in Q}$ is valid against $y = g^x$ if and only if*

$$x \;=\; \sum_{k \in P(\psi)} \Lambda_k \log Y_k \;\;\wedge\;\; s_i = \log U_{k_i} + h_{k_i} \log Y_{k_i},\ \forall i \in Q. \tag{5.10}$$

Proof. This is condition (3.13) with $y_i = Y_{k_i}$, $u_i = U_{k_i}$ and $c_i = h_{k_i}$ (cf. Proposition 3.12); the summands $\lambda_i \log y_i$ with $k_i = k$ combine to $\Lambda_k \log Y_k$. □

Observe that x enters only through the first relation, which involves no challenge and holds irrespective of $|Q|$ (including the case $|Q| < t$), while the shares $\{x_i\}_{i \in J}$ do not appear at all (cf. Section 4.1). Every other relation is the validity condition of the ordinary Schnorr protocol for Y_{k_i}, in the unknowns $\log U_{k_i}$ and $\log Y_{k_i}$ of its own position; that is, the positions are related to each other only through the first relation. Consequently, no assumption on the challenges is needed. For $|Q| = 1$, the relations reduce to $y_i = y$ and $s_i = \log u_i + c_i x$, i.e., the validity condition of the ordinary Schnorr protocol for y. If two executions submit the same pair at some position k, each with a component based on it, and receive distinct challenges there, then subtracting the respective relations eliminates $\log U_k$ and determines $\log Y_k$, irrespective of their indices and of the remaining positions. This is the usual extraction formula and the elementary step of the extraction procedure of Section 5.4.

5.3 Multiple local fork

The Local Forking Lemma of [6] is a variant of the Forking Lemma of [37], in the general form of [7], that reprograms the oracle at a single location rather than at every location after the fork; it forks a process once, at the oracle query indicated by its main output. Our extraction procedure requires forking a single execution at several positions (namely, those of all components of the fabricated proof), where every fork need only reproduce the component submitted at its own point. A single trial per position does not suffice: all forks would have to succeed at once, with a probability that is bounded only by a product over the positions and thus decays exponentially in their number. Forking of this kind is the subject of the generalized forking lemma of [3], which repeats every fork up to a prescribed number of times, trading running time for probability of success, and was introduced for extracting the witnesses of several Fiat-Shamir proofs from a single adversary; OSST components are proofs of this kind (cf. Section 2.9.4). We state its local counterpart, in which a fork re-samples the hash at its own position alone. Consider a deterministic algorithm of the form

$$\mathcal{S} : \Theta \times C^m \to (\Pi \cup \{\perp\}) \times T, \quad (\pi, \tau) \leftarrow \mathcal{S}(\theta; h_1, \ldots, h_m), \tag{5.11}$$

where Π is a set equipped with a map P that assigns to every $\pi \in \Pi$ a set $P(\pi) \subset \{1, \ldots, m\}$ of at most ℓ elements. Typically, $\mathcal{S}$ runs internally a hash oracle with tape $h_1, \ldots, h_m$ (cf. Appendix C), so that h_k is the hash returned at its k-th distinct oracle query; however, only the dependence of the outputs on the coins and the tape will matter below. Think of θ as the random coins, π as the *main* output and τ as the *side* output; $\pi = \perp$ is interpreted as failure, while $\pi \in \Pi$ designates the elements of $P(\pi)$ as fork positions. We denote by

$$\mathsf{acc}(\mathcal{S}) = Pr[\pi \neq \perp : \left\{ \begin{array}{c} (\pi, \tau) \leftarrow \mathcal{S}(\theta; h_1, \ldots, h_m), \quad \theta \leftarrow_\$ \Theta \\ h_\nu \leftarrow_\$ C, \quad 1 \leq \nu \leq m \end{array} \right\}]$$

the probability that $\mathcal{S}$ accepts taken over the possible coins and oracle outputs.

Definition 5.12. The *multiple local fork of $\mathcal{S}$ with $\rho \geq 1$ trials* is Algorithm 2.

Algorithm 2 $\mathsf{MultiFork}_{\mathcal{S},\rho}(\theta)$

1: $h_\nu \leftarrow_\$ C, \quad 1 \le \nu \le m$
2: $(\pi, \tau) \leftarrow \mathcal{S}(\theta; h_1, \ldots, h_m)$
3: **if** $\pi = \bot$
4: **return** $(\bot, \bot)$
5: **for** $k \in P(\pi)$
6: $\tau_k^* \leftarrow \bot$
7: **for** up to ρ times, while $\tau_k^* = \bot$
8: $h_k^* \leftarrow_\$ C$
9: $(\pi^*, \tau^*) \leftarrow \mathcal{S}(\theta; h_1, \ldots, h_{k-1}, h_k^*, h_{k+1}, \ldots, h_m)$
10: **if** $\pi^* \neq \bot \ \wedge \ k \in P(\pi^*) \ \wedge \ h_k^* \neq h_k$
11: $\tau_k^* \leftarrow \tau^*$
12: **if** $\tau_k^* = \bot$
13: **return** $(\bot, \bot)$
14: **return** $(\pi, (\tau, \{\tau_k^*\}_{k \in P(\pi)}))$

That is, if $\mathcal{S}$ accepts with main output π, then for every position $k \in P(\pi)$ the algorithm repeats its execution with the same input except for the k-th hash, which is re-sampled uniformly (steps 8-9); otherwise it aborts. A repeated execution is retained if $\mathcal{S}$ accepts again with k among the positions designated by its *own* main output, under the constraint $h_k^* \neq h_k$ (step 10); otherwise it is discarded and the fork at k is tried anew, up to ρ times. The algorithm returns the main output of the first execution along with its side output and those of the retained executions, one per position; if some position exhausts its trials, it aborts (step 13). Note that a retained execution is not required to return the main output π, nor do the forks at distinct positions depend on each other. We denote by $\mathsf{mforkAcc}(\mathcal{S}, \rho)$ the probability that the algorithm does not abort, taken over $\theta \leftarrow_\$ \Theta$ and the hashes it samples. The probability that the multiple local fork does not abort is bounded as follows.

Lemma 5.13. (*Multiple local fork*) *For every deterministic algorithm $\mathcal{S}$ as in* (5.11)*, every $4m/|C| \le \varepsilon \le 1$ and every $\rho \ge (4m/\varepsilon)\ln(4\ell/\varepsilon)$,*

$$\mathsf{mforkAcc}(\mathcal{S}, \rho) \ \ge \ \mathsf{acc}(\mathcal{S}) \ - \ \frac{3}{4}\,\varepsilon\,. \tag{5.14}$$

Proof. Let $\gamma \equiv \varepsilon/(2m)$. Given $1 \le k \le m$, θ and $\{h_\nu\}_{\nu \neq k}$, denote by f_k the probability over $h_k \leftarrow_\$ C$ that $\mathcal{S}$ accepts with $k \in P(\pi)$, and call position k *light* if $f_k < \gamma$. Since lightness does not depend on h_k, $\mathcal{S}$ accepts with $k \in P(\pi)$ while k is light with probability less than γ; consequently, the first execution accepts with no light position in $P(\pi)$ with probability greater than $\mathsf{acc}(\mathcal{S}) - m\gamma = \mathsf{acc}(\mathcal{S}) - \varepsilon/2$. In that case, fix $k \in P(\pi)$. Every trial at k shares θ and $\{h_\nu\}_{\nu \neq k}$ with the first execution, while h_k^* is sampled independently; it is thus retained with probability at least $f_k - 1/|C| \ge \gamma/2$, where $1/|C| \le \gamma/2$ by the range of ε. Since the trials are independent, all of them are discarded with probability at most $(1-\gamma/2)^\rho \le e^{-\rho\varepsilon/(4m)} \le \varepsilon/(4\ell)$, and the claim follows by the union bound over the at most ℓ positions of $P(\pi)$. □

5.4 Extraction

We proceed to fork $\mathcal{S}[\mathcal{A}]$ at every position of $P(\psi)$ and to extract x from the resulting executions by means of Proposition 5.9. Let $\ell \geq 1$ be an upper bound on the number of components that $\mathcal{A}$ fabricates, so that $|P(\psi)| \leq |Q| \leq \ell$ in every execution; then $\mathcal{S}[\mathcal{A}]$ is of the form (5.11) with $P(\pi) \equiv P(\psi)$ for $\pi = (Q, \psi)$ (cf. (5.8) and Remark 5.2). An execution and its fork at some position of $P(\psi)$ submit the same pair there, since all pairs up to the fork position remain intact (cf. Remark 5.3), while the pairs submitted afterwards and the indices of the proof may change. Consequently, the two fulfil the premises of the elementary step of Section 5.2, provided that the forked execution succeeds with this position among its own, i.e., with a component based on the same pair. The multiple local fork of $\mathcal{S}[\mathcal{A}]$ provides one such execution per position. Given a non-aborting execution of $\mathsf{MultiFork}_{\mathcal{S}[\mathcal{A}],\rho}$ with main output $\pi = (Q, \psi)$, denote by L_0 the first execution of $\mathcal{S}[\mathcal{A}]$ and by L_k, $k \in P(\psi)$, the execution retained at position k (cf. Algorithm 2, step 11).

The procedure repeats the subtraction outlined after Proposition 5.9, once per position of $P(\psi)$; it is shown in Algorithm 3, where $(\tau, \{\tau_k^*\}_{k \in P(\psi)})$ is the side output of the multiple local fork. The algorithm reads the proof of L_0 (step 1) and visits the positions of its components (step 3). At every position, it picks a component of L_0 submitted there (step 4) and the component of L_k based on the same pair (step 5), extracts the logarithm of the attached share (step 6) and accumulates it with the respective coefficient.

Algorithm 3 $\mathsf{Ext}_\pi(\tau, \{\tau_k^*\}_{k \in P(\psi)})$

1: Parse $\{(y_i, u_i, c_i, s_i)\}_{i \in Q} \leftarrow \tau$
2: $z \leftarrow 0$
3: **for** $k \in P(\psi)$
4: Choose $j \in Q$ with $k_j = k$
5: Parse $\{(y_i^*, u_i^*, c_i^*, s_i^*)\}_{i \in Q^*} \leftarrow \tau_k^*$, choose $j^* \in Q^*$ with $(y_{j^*}^*, u_{j^*}^*) = (y_j, u_j)$
6: $z \leftarrow z + \Lambda_k \dfrac{s_j - s_{j^*}^*}{c_j - c_{j^*}^*}$
7: **return** z

Proposition 5.15. (*Multiple local fork and extraction*) *Let $\pi = (Q, \psi)$ and $(\tau, \{\tau_k^*\}_{k \in P(\psi)})$ be the main and side output of a non-aborting execution of $\mathsf{MultiFork}_{\mathcal{S}[\mathcal{A}],\rho}$ with $y^* = g^x$. Then $\mathsf{Ext}_\pi(\tau, \{\tau_k^*\}_{k \in P(\psi)}) = x$. More accurately, for every $k \in P(\psi)$, the quotient computed at step 6 equals $\log Y_k$, where (Y_k, U_k) is the pair submitted by L_0 at position k.*

Proof. Fix $k \in P(\psi)$ and let j be the index chosen at step 4, so that $(y_j, u_j) = (Y_k, U_k)$. The executions L_0 and L_k share y^* and ω, and their tapes coincide except at position k, where the two hashes are distinct (cf. Algorithm 2, steps 9-10); by Remark 5.3, L_k also submits (Y_k, U_k) at position k. Since k is among the positions of L_k, some component of its proof is based on this pair, so that the index j^* of step 5 exists; its component was submitted at k, whichever j^* is chosen, because every pair is submitted at a single position. Both proofs being valid against y^*, Proposition 5.9 yields

$$s_j = \log U_k + c_j \log Y_k\,, \quad s_{j^*}^* = \log U_k + c_{j^*}^* \log Y_k\,,$$

where $c_j \neq c^*_{j^*}$ are the challenges received at position k. Subtracting them determines $\log Y_k$ as in step 6. Consequently, the algorithm returns $\sum_{k \in P(\psi)} \Lambda_k \log Y_k$, which equals x by the first relation of (5.10) for L_0. □

Observe that the extracted values are the true logarithms of the shares attached in L_0, however these are fabricated. That is, the combined secret is extracted from any impersonator whose multiple local fork does not abort, which is the knowledge property of Section 4.1 in the random oracle model. For $|Q| = 1$, which covers the non-distributed case, the procedure reduces to the usual extraction formula $x = (s - s^*)/(c - c^*)$ of the Schnorr protocol, where (y, u, c, s), (y, u, c^*, s^*) are the two forked transcripts.

5.5 Main theorem

Recall that DL hardness in $\mathbb{G}$ is the assumption that computing the logarithm of a uniformly random group element is infeasible (cf. Appendix A). The probability that an algorithm $\mathcal{A}^*$ outputs $\log y$ when given $y \leftarrow_\$ \mathbb{G}$ is denoted by

$$\mathsf{Adv}^{\mathsf{DL}}_{\mathbb{G}}[\mathcal{A}^*], \tag{5.16}$$

so that DL hardness means that (5.16) is negligible for every polynomial-time $\mathcal{A}^*$.

Theorem 5.17. (OSST – *Security against impersonation attacks*) *Given* $1 \leq t \leq n < q$, $\mathsf{OSST}_{\mathbb{G},n,t}$ (*cf. Protocol* 3.1) *is secure against impersonation attacks under* DL *hardness in the random oracle model. In particular, if* H *is modelled as a random oracle, for every polynomial-time adversary* $\mathcal{A}$ *that attacks Game* 4.1*, fabricates proofs with at most* ℓ *components and issues at most* q_{ro} *distinct queries to the random oracle, and for every* $4m/q \leq \varepsilon \leq 1$*, there exists an adversary* $\mathcal{A}^*$ *whose running time is about* $1 + \ell\rho$ *times that of* $\mathcal{A}$*, with*

$$\mathsf{Adv}^{\mathsf{DL}}_{\mathbb{G}}[\mathcal{A}^*] \;\geq\; \mathsf{Adv}^{\mathsf{OSST}}_{\mathbb{G},n,t}[\mathcal{A}] \;-\; \frac{3}{4}\,\varepsilon, \tag{5.18}$$

where $m = q_{\mathsf{ro}} + \ell$ *and* $\rho = \lceil (4m/\varepsilon) \ln(4\ell/\varepsilon) \rceil$.

Proof. We construct $\mathcal{A}^*$ as a wrapper of the multiple local fork of $\mathcal{S}[\mathcal{A}]$ with ρ trials (cf. Definition 5.12), followed by extraction (cf. Algorithm 3). If needed, $\mathcal{A}$ is first reprogrammed to query hashes for all the components it outputs (cf. Section 5.1), at the cost of at most $|Q| \leq \ell$ additional queries; the simulation then issues at most $m = q_{\mathsf{ro}} + \ell$ distinct queries, which we take as the length of the tape.

When given w by its challenger, $\mathcal{A}^*$ operates as shown in Algorithm 4. In steps 1-2, $\mathcal{A}^*$ prepares the coins of the simulation, embedding its own challenge as the attacked public key and sampling the remaining ones uniformly; since w is uniform, they are distributed exactly as in (5.5), while the tape and the re-sampled hashes are drawn by the multiple local fork itself (cf. Algorithm 2, steps 1 and 8). Observe that $\mathcal{A}^*$ aborts exactly when the multiple local fork aborts (step 5); otherwise, Proposition 5.15 ensures that step 6 yields $\log y^* = \log w$. Consequently,

$$\mathsf{Adv}^{\mathsf{DL}}_{\mathbb{G}}[\mathcal{A}^*] \;=\; \mathsf{mforkAcc}(\mathcal{S}[\mathcal{A}], \rho) \;\geq\; \mathsf{acc}(\mathcal{S}[\mathcal{A}]) \;-\; \frac{3}{4}\,\varepsilon \;=\; \mathsf{Adv}^{\mathsf{OSST}}_{\mathbb{G},n,t}[\mathcal{A}] \;-\; \frac{3}{4}\,\varepsilon$$

by Lemma 5.13 with $C = \mathbb{Z}_q$ and Corollary 5.6. Regarding running time, $\mathcal{A}^*$ executes $\mathcal{S}[\mathcal{A}]$ at most $1 + \ell\rho$ times, along with polynomial-time bookkeeping and the at most ℓ field

divisions of Algorithm 3. Finally, let $\mathcal{A}$ be polynomial-time, so that ℓ and m are polynomial in the bitlength of q, and assume that its advantage is non-negligible, i.e., at least $1/p$ for some polynomial p and infinitely many bitlengths. For $\varepsilon = 1/p$, which satisfies $\varepsilon \geq 4m/q$ for all but finitely many of them, ρ is polynomial; consequently, $\mathcal{A}^*$ is polynomial-time and its advantage is at least $1/(4p)$ for infinitely many bitlengths, contradicting DL hardness. □

Algorithm 4 $\mathcal{A}^*(w)$

Set $y^* \leftarrow w$
Set $\omega \leftarrow (\theta_{\mathcal{A}}, x_1^*, \ldots, x_n^*)$, where $\theta_{\mathcal{A}}$ are fresh coins for $\mathcal{A}$ and $x_i^* \leftarrow_{\$} \mathbb{Z}_q$, $1 \leq i \leq n$
$(\pi, (\tau, \{\tau_k^*\}_{k \in P(\psi)})) \leftarrow \mathsf{MultiFork}_{\mathcal{S}[\mathcal{A}],\rho}(y^*, \omega)$
if $\pi = \perp$
    return $\perp$
return $\mathsf{Ext}_\pi(\tau, \{\tau_k^*\}_{k \in P(\psi)})$

Remark 5.19. (*Leaked public shares*) The black-box adversary of Game 4.1 is not assumed to know $y_i = g^{x_i}$, $1 \leq i \leq n$, since OSST public shares are not advertised. However, a real-life attacker most probably knows them before mounting its actual attack. They are attached to every canonical proof, so that a man-in-the-middle who intercepts a single identification session reads them off (cf. Remark 4.4 regarding the transcripts themselves); or, if the shareholders store $\{y_i\}_{i=1}^n$ for mutual verification purposes, an attacker who compromises a shareholder's device can read them immediately if not erased. The result of Theorem 5.17 remains intact. Indeed, an adversary with $|J| < t-1$ is no stronger than one that corrupts $t-1$ shareholders and ignores the surplus shares; for $|J| = t-1$, the embedded adversary $\bar{\mathcal{A}}$ can anyway compute $\{y_k\}_{k \notin J}$ in the form

$$y_k \leftarrow \sqrt[\lambda_k^k]{y \Big(\prod_{i \in J} y_i^{\lambda_i^k} \Big)^{-1}}, \quad k \in \{1, \ldots, n\} \setminus J,$$

where $\{\lambda_i^k\}_i$ are the interpolation coefficients for $J \cup \{k\}$ (cf. Remark B.3). In particular, an adversary who controls devices in the non-erasure model has no greater advantage than one who only steals keys.

Remark 5.20. (*Concrete security*) Theorem 5.17 is an asymptotic statement and should be read as such. The reduction preserves the advantage of $\mathcal{A}$ up to the term $3\varepsilon/4$ and pays for it in running time: $\mathcal{A}$ is executed about $(4\ell m/\varepsilon)\ln(4\ell/\varepsilon)$ times, which is polynomial in the number of components and in $1/\varepsilon$. It should be stressed, however, that the reduction is not tight. The bound (5.18) is void for $\varepsilon \geq 4\mathsf{Adv}/3$, where Adv is the advantage of $\mathcal{A}$; that is, the number of trials is inversely proportional to a lower bound on the advantage, which must be available in advance. For $\ell = 1$, which covers the non-distributed case, a single trial may be used instead, Algorithm 2 then reducing to the local fork of [6], whose bound yields the usual bound $\mathsf{Adv}^2/m - \mathsf{Adv}/q$ of the Schnorr protocol with two executions. The nested fork, whose loss grows doubly exponentially with ℓ, is needed by the compact variant alone (cf. Theorem D.25).

5.5.1 Observed transcripts

We extend the proof of Theorem 5.17 to Game 4.5, i.e., to the session-bound variant and adversaries who issue at most q_{pr} queries to $\mathcal{O}^{\mathsf{prove}}$ (cf. Section 4.2.1). Two modifications of the simulation are needed; the hash oracle is keyed by triples (y, u, sid), and the honest components must be produced without any knowledge of shares. The rest of the argument carries over verbatim, since Lemma 5.13 is generic and Sections 5.2 and 5.4 rely on $\mathcal{A}$ only through Proposition 5.9.

The simulator must answer every query $\mathcal{O}^{\mathsf{prove}}(k, \mathsf{sid})$, $k \notin J$, with a component (y_k, u, s) such that $g^s = y_k^c u$ for $c = H(y_k, u, \mathsf{sid})$. The public share y_k of k is computable: let $J^* \supset J$ with $|J^*| = t - 1$ be obtained from J by adding the smallest indices outside J, and $y_i \equiv g^{x_i^*}$, $i \in J^*$; then $y_k = g^{x_k^*}$ if $k \in J^*$, and otherwise y_k is obtained by interpolation over $J^* \cup \{k\}$ (cf. Remark 5.19). The share x_k, however, depends on the unknown $\log y^*$ for $k \notin J^*$; the simulator therefore produces every component without its share, by reversing the order of the Schnorr protocol: it samples s and c from its coins, sets $u \leftarrow g^s y_k^{-c}$, so that the above condition holds, and *programs* $H(y_k, u, \mathsf{sid}) := c$, i.e., inserts c into Map_1 without drawing from the tape. If (y_k, u, sid) has already been queried, its hash cannot be programmed and the simulation aborts. The exact procedure is shown in Algorithm 5.

Algorithm 5 $\widetilde{\mathcal{O}}^{\mathsf{prove}}(k, \mathsf{sid})$

1: Read the next pair $(s, c) = (\tilde{s}_j, \tilde{c}_j)$ from $\omega^{\mathbf{tr}}$
2: $y_k \leftarrow g^{x_k^*}$ if $k \in J^*$, else $y_k \leftarrow \sqrt[\lambda_k^k]{y \left(\prod_{i \in J^*} y_i^{\lambda_i^k}\right)^{-1}}$ // *cf. Remark 5.19*
3: $u \leftarrow g^s y_k^{-c}$
4: **if** $(y_k, u, \mathsf{sid}) \in \mathrm{Dom}(\mathsf{Map}_1)$
5: abort the simulation with main output $\perp$ // *Programming collision*
6: $\mathsf{Map}_1[(y_k, u, \mathsf{sid})] \leftarrow c$ // *No tape position*
7: **return** (y_k, u, s)

Denote by $\mathcal{S}^{\mathbf{tr}}[\mathcal{A}](y^*, \omega^{\mathbf{tr}}; h_1, \ldots, h_m)$ the simulator of Algorithm 1 with these modifications, whose coins are

$$\omega^{\mathbf{tr}} = \big(\theta_{\mathcal{A}}, x_1^*, \ldots, x_n^*, \mathsf{sid}^*, (\tilde{s}_j, \tilde{c}_j)_{1 \le j \le q_{\mathsf{pr}}}\big), \quad \mathsf{sid}^* \leftarrow_{\$} \mathcal{N}, \quad \tilde{s}_j, \tilde{c}_j \leftarrow_{\$} \mathbb{Z}_q, \tag{5.21}$$

the remaining components being distributed as in (5.5); the j-th query to $\mathcal{O}^{\mathsf{prove}}$ is answered by Algorithm 5 with the pair $(\tilde{s}_j, \tilde{c}_j)$, the verifier is $\mathcal{V}_{\mathsf{sid}^*}$, and the simulation also fails if a component has been issued on sid^*. Programmed answers consume no tape positions; in a successful execution, no component has been issued on sid^*, so that every query of the form (y, u, sid^*) is answered from the tape. In particular, the components of a valid proof occupy tape positions as before, including honest components (y_k, u_k) copied from other sessions, which receive fresh challenges under sid^*; these are forked like any fabricated component, the extracted logarithm being the share x_k. Remark 5.3 remains valid, where positions refer to the queries answered from the tape, programmed answers being determined by y^*, $\omega^{\mathbf{tr}}$ and the preceding hashes; consequently, the multiple local fork of $\mathcal{S}^{\mathbf{tr}}[\mathcal{A}]$ and the extraction of Section 5.4 apply without change.

Proposition 5.22. ($\mathsf{OSST}^{\mathbf{tr}}_{\mathbb{G},n,t}$ – *Simulation up to programming collisions*) *If H is modelled as a random oracle, then for every adversary $\mathcal{A}$ for Game 4.5 issuing at most q_{pr} queries to $\mathcal{O}^{\mathsf{prove}}$,*

$$\left|\,\mathsf{acc}(\mathcal{S}^{\mathbf{tr}}[\mathcal{A}]) - \mathsf{Adv}^{\mathsf{OSST},\mathbf{tr}}_{\mathbb{G},n,t}[\mathcal{A}]\,\right| \;\leq\; \frac{q_{\mathsf{pr}}\,(m+q_{\mathsf{pr}})}{q}\,. \tag{5.23}$$

Proof. Call *collision* the event that, upon some query $\mathcal{O}^{\mathsf{prove}}(k,\mathsf{sid})$, the triple (y_k,u,sid) with the returned commitment u has already been submitted to the oracle; this event is defined in both the real and the simulated game execution, and only in the latter does it cause an abort (cf. Algorithm 5, steps 4-5). We show that the two executions are identically distributed from $\mathcal{A}$'s viewpoint as long as no collision occurs; consequently, the collision has the same probability in both, and the probabilities that $\mathcal{A}$ wins differ by at most this probability. The coins of $\mathcal{A}$, the public key, the shares and the hashes drawn from the tape are distributed as in the real execution (cf. Proposition 5.4), and sid^* is uniform in both. The public share y_k is the same function of y and $\{x_i\}_{i\in J^*}$ in both executions, namely g^{x_k} if $k\in J^*$ and the interpolation in the exponent over $J^*\cup\{k\}$ otherwise (cf. Remark 5.19), where $(y,\{x_i\}_{i\in J^*})$ is distributed identically in both by Lemma 5.1. Consider now the j-th query $\mathcal{O}^{\mathsf{prove}}(k,\mathsf{sid})$, conditioned on $\mathcal{A}$'s view so far. In the real execution, $u=g^r$ is uniform and independent of the view; if (y_k,u,sid) is fresh, $c=H(y_k,u,\mathsf{sid})$ is uniform and independent of u and the view, while $s=r+cx_k$ is determined by u and c. In the simulated execution, $\tilde{s}_j$ and $\tilde{c}_j$ are uniform and independent of the view, so that $u=g^{\tilde{s}_j}y_k^{-\tilde{c}_j}$ is uniform, $c=\tilde{c}_j$ is uniform and independent of u, and $s=\tilde{s}_j=\log u+cx_k$ is determined by u and c in the same way. In both executions, later queries on (y_k,u,sid) return c, and the win condition is the same function of the view. Finally, at the j-th query the oracle has answered at most $m+j-1$ distinct triples, at most m from the tape and $j-1$ by programming, while u is uniform and independent of them; the collision thus occurs at this query with probability at most $(m+q_{\mathsf{pr}})/q$, and (5.23) follows by the union bound over the q_{pr} queries. □

The security of the session-bound variant follows as in the main theorem.

Corollary 5.24. (OSST – *Security with observed transcripts*) *Given $1\leq t\leq n<q$, the session-bound variant of $\mathsf{OSST}_{\mathbb{G},n,t}$ is secure against impersonation attacks with observed transcripts under* DL *hardness in the random oracle model. In particular, if H is modelled as a random oracle, for every polynomial-time adversary $\mathcal{A}$ that attacks Game 4.5, fabricates proofs with at most ℓ components, issues at most q_{ro} distinct queries to the random oracle and at most q_{pr} queries to $\mathcal{O}^{\mathsf{prove}}$, and for every $4m/q\leq\varepsilon\leq 1$, there exists an adversary $\mathcal{A}^*$ whose running time is about $1+\ell\,\rho$ times that of $\mathcal{A}$, with*

$$\mathsf{Adv}^{\mathsf{DL}}_{\mathbb{G}}[\mathcal{A}^*] \;\geq\; \mathsf{Adv}^{\mathsf{OSST},\mathbf{tr}}_{\mathbb{G},n,t}[\mathcal{A}] \;-\; \frac{q_{\mathsf{pr}}\,(m+q_{\mathsf{pr}})}{q} \;-\; \frac{3}{4}\,\varepsilon\,, \tag{5.25}$$

where $m=q_{\mathsf{ro}}+\ell$ and $\rho=\lceil(4m/\varepsilon)\ln(4\ell/\varepsilon)\rceil$.

Proof. As for Theorem 5.17, with $\mathcal{S}^{\mathbf{tr}}[\mathcal{A}]$ in place of $\mathcal{S}[\mathcal{A}]$ and Proposition 5.22 in place of Corollary 5.6; the coins $\omega^{\mathbf{tr}}$ are sampled as in (5.21). The running time additionally includes the simulation of at most q_{pr} components per execution of $\mathcal{S}^{\mathbf{tr}}[\mathcal{A}]$, each of which costs $O(t)$ exponentiations. Finally, since q_{pr} and m are polynomial in the bitlength of q for polynomial-time $\mathcal{A}$, the term $q_{\mathsf{pr}}(m+q_{\mathsf{pr}})/q$ is negligible, and the asymptotic conclusion of Theorem 5.17 carries over. □

Several questions are left open. Both results are asymptotic statements; the reduction is not tight, with its running time inversely proportional to a lower bound on the adversarial

advantage (cf. Remark 5.20). Security against adaptive corruption in the presence of observed transcripts lies outside the present model; we expect it to be provable by similar means, but do not pursue it here. Similarly, active impersonation attacks in the sense of [8] pertain to the interactive variant of the protocol (cf. Remark 3.6) and are not considered.

6 Conclusion

We have investigated the overlooked area of real-time threshold identification, arguing that threshold signing primitives are not native solutions unless adapted to the permissionless setting, where verifiers must play the role of combiner. In this context, we have stressed that practical schemes for open or cross-domain settings should not rely on advertised shares because these are generally impractical to attest, certify and monitor, while interaction among the provers appears as an inconvenience —if not an operational obstacle— without essential gains for identification. In particular, we have argued that share-free and non-interactive schemes emerge as the most native and system-agnostic option for unmanaged meshes, treating this "niche" as canonical while threshold cryptography has implicitly focused on the closed committee model under idealized assumptions regarding its infrastructure.

We have introduced OSST as a concrete and lightweight instance of such a protocol in the pure DL-setting, proving its security against impersonation attacks in the random oracle model. Specifically, we have seen that a formal proof is possible without the assumption of trusted shares and for an adversary who corrupts up to $t-1$ shareholders, statically or adaptively, chooses the indices of its proof and fabricates all of its components, the attached shares being extracted one at a time by forking. The result extends to adversaries who observe transcripts of other sessions, provided that the challenges are bound to them. The compact variant, which dispenses with the attached shares, is proved secure by eliminating the logarithms of the commitments jointly; it should be stressed, however, that large coalitions are inherently easier to impersonate in this variant by means of a generalized birthday attack, so that it is confined to small groups (cf. Appendix D). Attacks in which honest shareholders participate in the attacked session, adaptive corruption in the presence of observed transcripts and active attacks against the interactive variant remain open, while both results are asymptotic and neither reduction is tight; that of the compact variant further requires a fixed bound on the number of components, with a loss that grows doubly exponentially with it.

A Schnorr identification

Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of order q. A secret key is a scalar $x \in \mathbb{Z}_q$ and $y = g^x$ is its public counterpart; equivalently, $x = \log y$, where the logarithm is taken with respect to g. *Discrete-logarithm hardness* (DL) for the system under consideration is the assumption that computing the logarithm of a uniformly random group element is infeasible for every existing computer. More accurately, given $y \leftarrow_{\$} \mathbb{G}$, every probabilistic polynomial-time algorithm should compute $x \equiv \log y$ with at most negligible chance of success, where negligibility is determined by the current level of disposable computational power in the security parameter.

The *Schnorr protocol* [39] is a sigma protocol allowing the holder of $y = g^x$ to prove knowledge of x without revealing any information about the latter; its non-interactive (NIZK) variant is obtained through the Fiat-Shamir transform (see below). The prover samples $r \leftarrow_{\$} \mathbb{Z}_q$, stores it for the current session and sends the commitment $u \leftarrow g^r$ to the verifier.

The verifier samples a challenge $c \leftarrow_{\$} \mathbb{Z}_q$, caches it for the current session and sends it to the prover. Finally, the prover responds with $s = r + cx$ and the verifier accepts only if

$$g^s = y^c u$$

is satisfied; indeed, $g^s = g^{r+cx} = g^{xc} g^r = y^c u$.

Given a hash function $H : \mathbb{G} \to \mathbb{Z}_q$, the protocol can be made non-interactive through the Fiat-Shamir transform; the prover computes $c \leftarrow H(u)$ and sends (u, s) to the verifier, who retrieves c from u and proceeds to verification. This is preferable due to greater robustness and lower bandwidth. Intuitively, the hash function injects sufficiently high entropy across every bit of its output, so that assigning $c \leftarrow H(u)$ becomes indistinguishable from uniformly sampling c. The current mainstream formalizes this fact in the random oracle model (ROM).

The protocol identifies any procedure that "sees" the logarithm of a given group element. Security against impersonation attacks means that it is infeasible to forge a valid proof for a group element without knowing its logarithm; in the proof-of-knowledge framework, this is formalized as witness extractability under the DL hardness assumption. The protocol is meaningful only under DL hardness, which in turn enables a security proof in the ROM.[9]

B Shamir's Secret Sharing

We here summarize the basic facts regarding the sharing scheme introduced by [41]. Given a cyclic group $\mathbb{G}$ of prime order q, we denote by $\left(\mathbb{Z}_q\right)_{t-1}[X]$ the ring of polynomials with degree at most $t-1$ over $\mathbb{Z}_q$, $t \geq 1$.

Definition B.1. The *Shamir (n,t)-sharing scheme*, $1 \leq t \leq n < q$, is the probabilistic algorithm $(x_1, \ldots, x_n;\, y) \leftarrow D(g, x)$, $(g, x) \in \mathbb{G} \times \mathbb{Z}_q$, defined as

$$y \leftarrow g^x, \quad x_i \leftarrow f(i), \quad 1 \leq i \leq n, \quad f \leftarrow_{\$} \left(\mathbb{Z}_q\right)_{t-1}[X] \text{ with } f(0) = x\,.$$

Definition B.2. For fixed $g \in \mathbb{G}$, the space of Shamir (n,t)-sharings of x is denoted by

$$D_x = \left\{(x_1, \ldots, x_n;\, y) \mid (x_1, \ldots, x_n;\, y) \leftarrow D(g, x)\right\}.$$

Remark B.3. (*Interpolation formula*) Given $(x_1, \ldots, x_n;\, y) \leftarrow D(g, x)$, the condition

$$x = \sum_{i \in Q} \lambda_i x_i \iff |Q| \geq t \iff y = \prod_{i \in Q} y_i^{\lambda_i}, \tag{B.4}$$

holds true with overwhelming probability, where $\{\lambda_i\}_{i \in Q}$ are the Lagrange interpolation coefficients for Q, i.e.,

$$\lambda_i = \prod_{j \in Q \setminus \{i\}} j\,(j-i)^{-1}, \quad i \in Q, \tag{B.5}$$

and $y_i \equiv g^{x_i}, 1 \leq i \leq n$, are the public shares.

[9] The most far-reaching result in this direction is security against active and concurrent impersonation attacks under one-more discrete logarithm (OMDL) hardness; cf. Theorem 2, [8].

Remark B.6. (*Homomorphic property*) By the properties of polynomial evaluation, adding

$$(x_1, \ldots, x_n; y) \in D_x,\ (x_1^*, \ldots, x_n^*; y^*) \in D_{x^*}$$

componentwise yields a (n, t)-sharing of $x + x^*$, i.e.,

$$(x_1 + x_1^*, \ldots, x_n + x_n^*; yy^*) \in D_{x+x^*}$$

Remark B.7. (*Semantic security*) Given $g \in \mathbb{G}$, for any two secrets $x, x^* \in \mathbb{Z}_q$ and index set $J \subset \{1, \ldots, n\}$ with $|J| \leq t - 1$, the probability distributions

$$\{\{x_i\}_{i\in J} : (x_1, \ldots, x_n;\ y) \leftarrow D(g, x)\}, \quad \{\{x_i\}_{i\in J} : (x_1, \ldots, x_n;\ y) \leftarrow D(g, x^*)\}$$

are identical.

C Hash oracles

When querying a random oracle H, we usually need to keep track of distinct oracle queries. To this end, given a finite tape of preallocated hashes $h = [h_1 \ldots, h_m]$ and initially empty lookup tables Map_1 and Map_2, we define the *hash oracle* $\mathcal{O}^{\mathsf{hash}}$ to be Algorithm 6. This functions like a random oracle with lookup table Map_1, but draws hashes from h rather than sampling them uniformly; upon inserting a fresh hash to Map_1, it takes care to update the tape pointer so that the next predefined hash be available on demand and insert the current pointer to Map_2 so that the respective query can be tracked by order. Evidently, if $h_1, \ldots, h_m$ are uniformly sampled from the codomain of H, then $\mathcal{O}^{\mathsf{hash}}$ cannot be distinguished from H. We state this remark formally for the sake of reference.

Lemma C.1. (*Random oracle simulation*) *Given a random oracle H with codomain C and initially empty lookup table* Map*, initialize a hash oracle $\mathcal{O}^{\mathsf{hash}}$ over a tape h of size $m \geq 1$ and empty lookup tables* Map_1 *and* Map_2*. For any distinct queries $u_1, \ldots, u_m$, each possibly depending on the answers to the previous ones, the probability distribution*

$$\{(c_1, \ldots, c_m) :\ c_\nu \leftarrow \mathcal{O}^{\mathsf{hash}}(u_\nu),\ h[\nu] \leftarrow_{\$} C,\ \nu = 1, \ldots, m\}$$

is identical to the distribution $\{(c_1, \ldots, c_m) : c_\nu \leftarrow H(u_\nu),\ \nu = 1, \ldots, m\}$.

Proof. Follows directly from the definition of $\mathcal{O}^{\mathsf{hash}}$, Algorithm 6. □

Algorithm 6 $\mathcal{O}^{\mathsf{hash}}(u)$

if $u \notin \mathrm{Dom}(\mathsf{Map}_1)$
  $c \leftarrow h[\nu]$
  $\mathsf{Map}_1[u] \leftarrow c$
  $\mathsf{Map}_2[u] \leftarrow \nu$
  $\nu \leftarrow \nu + 1$
else
  $c \leftarrow \mathsf{Map}_1[u]$
return c

D Analysis of the compact variant

In this section, we examine Protocol 3.15. In Section D.1, we characterize the proofs that satisfy its verification condition (3.17). In Section D.2, we transpose the security notions of Section 4. In Section D.3, we prove the variant secure by adapting the reduction of Section 5.

D.1 Validity conditions

Throughout this appendix, proofs, canonical proofs and validity refer to the compact variant.

Definition D.1. A *proof* is a collection $\sigma = \{(u_i, s_i)\}_{i\in Q}$ of the form

$$(u_i, s_i) \in \mathbb{G} \times \mathbb{Z}_q, \forall i \in Q \ \wedge \ Q \subset \{1, \dots, \bar{n}\}.$$

Given σ as above, we refer to $\{u_i\}_{i\in Q}$ as its *commitments* and say that σ is *based on* them. We also refer to $\{s_i\}_{i\in Q}$ as the *responses*. The *weights* and *normalizer of* σ are the weights and normalizer of its commitments (cf. Def. 3.21) respectively. The *proof space* is the set of proofs and will be denoted by Σ. We also denote by

$$\Sigma\big[\{u_i\}_{i\in Q}\big]$$

the subspace of proofs that are based on $\{u_i\}_{i\in Q}$.

The proper execution of the protocol is modelled as follows.

Definition D.2. The proof $\{(u_i, s_i)\}_{i\in Q} \in \Sigma$, $Q \subset \{1, \dots, n\}$, is called *canonical with respect to* a sharing $(x_1, \dots, x_n;\, y) \in D_x$ (cf. Def. B.2) if

$$s_i = r_i + c_i x_i, \quad c_i \equiv H(u_i), \quad r_i \equiv \log u_i, \quad \forall i \in Q, \tag{D.3}$$

i.e., its components are Schnorr proofs for the respective shares. Evidently, for $Q \subset \{1, \dots, n\}$, the subspace $\Sigma\big[\{u_i\}_{i\in Q}\big]$ contains a unique canonical representative with respect to every fixed sharing.

Definition D.4. The *compact one-step Schnorr* (n, t)*-threshold verifier* is the deterministic algorithm $\mathcal{V}^{\mathsf{c}} : \mathbb{G} \times \Sigma \to \{\mathsf{true}, \mathsf{false}\}$ invoked as

$$\big(y, \{(u_i, s_i)\}_{i\in Q}\big) \quad \mapsto \quad \exp\Big(g, \sum_{i\in Q} \mu_i s_i\Big) = y^{\bar{c}} \prod_{i\in Q} u_i^{\mu_i},$$

where $\{\mu_i\}_{i\in Q}$, $\bar{c}$ are the weights and the normalizer of $\{(u_i, s_i)\}_{i\in Q}$ respectively.

Definition D.5. A proof $\sigma \in \Sigma$ is called *valid against* $y \in \mathbb{G}$ if $\mathcal{V}^{\mathsf{c}}(y, \sigma) = \mathsf{true}$.

$\mathsf{OSST}^{\mathsf{c}}$ correctness (cf. Rem. 3.24) means that a proof canonical with respect to $(x_1, \dots, x_n;\, y)$ is valid against y if and —with overwhelming probability— only if $|Q| \geq t$. Whenever a proof is related to a given sharing $(x_1, \dots, x_n;\, y)$ in the sequel, $Q \subset \{1, \dots, n\}$ is understood.

The subspace of proofs that are based on $\{u_i\}_{i\in Q}$ and valid against y will be denoted by

$$\Sigma_y\big[\{u_i\}_{i\in Q}\big].$$

If $|Q| \geq t$, then this space contains a unique canonical representative with respect to every fixed sharing of $x = \log y$. In the non-distributed case $n = 1$, the subspace $\Sigma_y\big[\{u_i\}_{i\in Q}\big]$ is a singleton due to the "unique responses" property of the Schnorr protocol.[10] In general, however, this space has $|Q| - 1$ degrees of freedom (cf. Remark D.9), so that most valid proofs are not canonical.

Proposition D.6. (*Validity conditions*) *Let $x \in \mathbb{Z}_q$, $y = g^x$ and $\sigma = \{(u_i, s_i)\}_{i\in Q}$ with $c_i \equiv H(u_i) \neq 0$ for all $i \in Q$. Then σ is valid against y if and only if*

$$\frac{1}{\bar{c}} \sum_{i\in Q} \mu_i(s_i - r_i) = x, \quad r_i \equiv \log u_i\,, \tag{D.7}$$

where $\{\mu_i\}_{i\in Q}$, $\bar{c}$ are the weights and normalizer of σ respectively, or, equivalently,

$$x = \sum_{i\in Q} \lambda_i \frac{s_i - r_i}{c_i}, \tag{D.8}$$

where $\{\lambda_i\}_{i\in Q}$ are the interpolation coefficients for Q.

Proof. Since $c_i \neq 0$ for all $i \in Q$, we have $\bar{c} \neq 0$ and the validity condition (3.20) transforms to (D.7). This further reformulates to (D.8) by means of (3.23). □

Since H is a hash function, the hypothesis holds with overwhelming probability in the bitlength of q (cf. Remark 3.27). The statement holds true irrespective of size (including the case $|Q| < t$), establishing a relation between x and any proof that happens to be valid against y. Both (D.7) and (D.8) generalize the relation of a secret key to a valid non-distributed proof; it should be stressed however that (D.8) does not imply $x_i = (s_i - r_i)/c_i$ and it can hold true without σ being canonical. Indeed (cf. Remark D.9), valid proofs are not confined to those properly generated and a relevant security notion should ensure that such proofs are infeasible to compute without knowledge of at least t secret shares (cf. Section 4).

Remark D.9. (*Validity and degrees of freedom*) Fix commitments $\{u_i\}_{i\in Q} \subset \mathbb{G}$ with non-zero challenges. Choose $J \subset Q$ with $|J| = |Q| - 1$, pick arbitrary $\{s_i\}_{i\in J} \subset \mathbb{Z}_q$ and set

$$s_j \;=\; r_j + \frac{1}{\mu_j}\Big(\bar{c}\cdot x \;-\; \sum_{i\in J} \mu_i\,(s_i - r_i)\Big)$$

where $\{\,j\,\} = Q \setminus J$ and $r_i \equiv \log u_i$, $i \in Q$. By Proposition D.6, every proof that is based on $\{u_i\}_{i\in Q}$ and valid against $y \equiv g^x$ is obtained in this way from exactly one choice of $\{s_i\}_{i\in J}$. That is, the subspace $\Sigma_y\big[\{u_i\}_{i\in Q}\big]$ has $|Q| - 1$ degrees of freedom in the sense that every proof is uniquely determined by the choice of $\{s_i\}_{i\in J}$ and $J \subset Q$ with $|J| = |Q| - 1$. Observe that this process has no utility for an impersonating adversary.

D.2 Security notions

The security notions of Section 4 carry over to the compact variant. More accurately, Game 4.1 is read with $\mathcal{A}$ choosing $Q \subset \{1, \ldots, \bar{n}\}$, outputting a proof $\sigma = \{(u_i, s_i)\}_{i\in Q}$ and winning if $\mathcal{V}^{\mathsf{c}}(y, \sigma) = \mathsf{true}$, which, unless some challenge vanishes, is equivalent to

[10] For every u there exists a unique s such that the proof (u, s) is valid against $y = g^x$, namely $s = r + H(u)\,x$ with $r \equiv \log u$.

condition (D.7) for σ; Game 4.5 is read likewise, with $H : \mathbb{G} \times \mathcal{N} \to \mathbb{Z}_q$, the challenges computed as $c_i \leftarrow H(u_i, \mathsf{sid})$, the respective verifier denoted by $\mathcal{V}^{\mathsf{c}}_{\mathsf{sid}}$ and the oracle returning (u, s) with $s \leftarrow r + H(u, \mathsf{sid})\, x_k$. The respective winning probabilities are denoted by

$$\mathsf{Adv}^{\mathsf{OSST^c}}_{\mathbb{G},n,t}[\mathcal{A}], \quad \mathsf{Adv}^{\mathsf{OSST^c},\mathbf{tr}}_{\mathbb{G},n,t}[\mathcal{A}] \tag{D.10}$$

and security means that they are negligible for every choice of n, t and $\bar{n}$ and every polynomial-time adversary $\mathcal{A}$.

The knowledge property (cf. Section 4.1) is derived differently, since the components carry no attached shares. Observe that condition (D.8) holds true irrespective of size and does not involve $\{x_i\}_{i\in J}$ at all; that is, a valid proof is in effect a Schnorr proof for the combined secret x, distributed over the $|Q|$ fabricated commitments, and a single fork leaves the logarithms of the remaining commitments undetermined. Forking $\mathcal{A}$ iteratively, once at each oracle query for these commitments, yields a system of relations that successively eliminates their unknown logarithms, so that x can finally be extracted (cf. Section D.3). Consequently, the security of the compact variant can be reduced to DL hardness alone (cf. Theorem D.25, and Corollary D.27 regarding observed transcripts), albeit at the cost of a reduction loss that grows rapidly with $|Q|$. This loss is not merely an artifact of the proof technique; proofs with many components are inherently easier to fabricate by means of the generalized birthday attack of Remark 3.28, which requires no knowledge of shares at all.

D.3 Security proof

We prove the compact variant secure in the sense of Section D.2 along the lines of Section 5. The simulation of Section 5.1 carries over with the hash oracle queried on commitments; that is, the respective simulator, still denoted by $\mathcal{S}[\mathcal{A}]$, is Algorithm 1 with $\mathcal{A}$ outputting $\{(u_i, s_i)\}_{i\in Q}$, the verifier $\mathcal{V}^{\mathsf{c}}$, the lookup tables read at u_i and $\sigma \leftarrow \{(u_i, c_i, s_i)\}_{i\in Q}$, while Remarks 5.2 and 5.3, Proposition 5.4 and Corollary 5.6 hold likewise, with $H(u_i)$ and $\mathsf{Adv}^{\mathsf{OSST^c}}$ in place of $H(y_i, u_i)$ and $\mathsf{Adv}^{\mathsf{OSST}}$. The differences lie in the validity relation and, consequently, in the fork and the extraction procedure. The validity relation is atomic across all components, so that every forked execution must reproduce Q and the positions of all components, not merely the element of its own position (cf. Sections D.3.1 and D.3.2). Given an execution of the simulator, denote by v_k the element submitted at the k-th distinct oracle query, $1 \le k \le m$, and let Q, $\sigma = \{(u_i, c_i, s_i)\}_{i\in Q}$ and $\psi = \{k_i\}_{i\in Q}$ be as computed at steps 2-8 of Algorithm 1, with $P(\psi)$ and Λ_k as in (5.8); then $u_i = v_{k_i}$ and $c_i = h_{k_i}$ for every $i \in Q$.

Proposition D.11. (*Validity as a linear relation*) *Given an execution of $\mathcal{S}[\mathcal{A}]$, let Q, $\sigma = \{(u_i, c_i, s_i)\}_{i\in Q}$ and $\psi = \{k_i\}_{i\in Q}$ be as above, with $c_i \neq 0$ for all $i \in Q$. Then $\{(u_i, s_i)\}_{i\in Q}$ is valid against $y = g^x$ if and only if*

$$x + \sum_{k\in P(\psi)} \Lambda_k \frac{\log v_k}{h_k} = K_\sigma, \quad K_\sigma \equiv \sum_{i\in Q} \lambda_i \frac{s_i}{c_i}. \tag{D.12}$$

Proof. By (3.20), the proof is valid against y if and only if $\sum_{i\in Q} \mu_i(s_i - r_i) = \bar{c} \cdot x$, where $r_i \equiv \log u_i$. Since $c_i \neq 0$ for all $i \in Q$, we have $\bar{c} \neq 0$ and, by (3.23), $\mu_i/\bar{c} = \lambda_i/c_i$; consequently, validity is equivalent to (D.8) (cf. Proposition D.6), i.e., $x + \sum_{i\in Q} \lambda_i r_i/c_i = K_\sigma$. Since $r_i = \log v_{k_i}$ and $c_i = h_{k_i}$, the summands with $k_i = k$ combine to $\Lambda_k \log v_k/h_k$. □

This is condition (D.8) rewritten over the oracle queries. Observe that x appears with coefficient 1 irrespective of $|Q|$ (including the case $|Q| < t$), while the shares $\{x_i\}_{i\in J}$ do not appear at all (cf. Section 4.1). The right-hand side K_σ is computable from σ and Q, so that the only unknowns are x and the logarithms $\log v_k$, $k \in P(\psi)$, which enter linearly with coefficients Λ_k/h_k determined by the main output and the tape. In particular, two executions with the same main output yield relations with the same positions and the same Λ_k's. Repeated commitments are submitted at a single position and their coefficients add up in Λ_k; it may even happen that $\Lambda_k = 0$, in which case $\log v_k$ is absent from the relation. Note also that no assumption is made on how $\mathcal{A}$ fabricates its commitments; e.g., if $v_k = g^b y^e$, then $\log v_k = b + e\,x$ and (D.12) still holds as an identity among the true values, so that solving it never requires the unknowns to be independent. For $|Q| = 1$, it reduces to $s_i = \log u_i + c_i\,x$, i.e., the validity condition of the ordinary Schnorr protocol for y. Finally, if two executions yield relations with the same main output and the same elements at all positions in $P(\psi)$, whose challenges differ only at a single position k, then subtracting them eliminates x along with all other unknowns and determines $\log v_k$ (provided that $\Lambda_k \neq 0$). This is the elementary step of the elimination procedure of Algorithm 8.

Remark D.13. (*Zero challenges*) The assumption $c_i \neq 0$, $i \in Q$, cannot be dispensed with; if some challenge vanishes, $\mathcal{A}$ can complete a valid proof irrespective of y, which carries no information about x (cf. Remark 3.27). In the context of the simulation, such proofs occur only if $h_k = 0$ for some k, i.e., with probability at most m/q over a uniformly random tape; we will account for this possibility in Theorem D.25.

D.3.1 Generalized Local Forking Lemma

The Local Forking Lemma of [6] forks a process at the oracle query indicated by its main output, which is a single index $k \in \{1, \ldots, m\}$. The elimination procedure will require forking at positions that are determined by a more elaborate main output (namely, the indices of the fabricated proof along with the positions of all its components), which every fork must reproduce, and repeating this recursively; the multiple local fork of Section 5.3, whose forks are independent of each other, does not provide such executions. We thus generalize the lemma to main outputs ranging over an arbitrary finite set, each of which designates its own fork position. Nested applications of the forking lemma are known from the security proof of MuSig, which forks twice in this fashion [29]; a forking lemma for two nested positions is stated by [23]. Consider a deterministic algorithm $\mathcal{S}$ of the form (5.11), where Π is now a finite set equipped with a map $\kappa : \Pi \to \{1, \ldots, m\}$ in place of P; that is, $\pi \in \Pi$ designates the single position $\kappa(\pi)$ of the tape as fork position (the $\kappa(\pi)$-th distinct oracle query in the typical case), the acceptance probability $\mathsf{acc}(\mathcal{S})$ being defined as in Section 5.3.

Definition D.14. The *local fork of $\mathcal{S}$ with respect to κ* is Algorithm 7.

That is, if $\mathcal{S}$ accepts with main output π, then the algorithm repeats its execution with the same input except for the $\kappa(\pi)$-th hash, which is re-sampled uniformly (step 5); otherwise it aborts. When the second execution completes, the algorithm checks whether $\mathcal{S}$ accepts again with the *same* main output under the constraint $h_k^* \neq h_k$ (step 7) and returns the main output along with both side outputs; otherwise, it aborts by returning $\perp$. Note that the fork position is determined by the first execution only; the requirement $\pi^* = \pi$ ensures that it is also the one designated by the second. We denote by $\mathsf{forkAcc}(\mathcal{S}, \kappa)$ the probability that the algorithm does not abort taken over the random coins of $\mathcal{S}$.

Algorithm 7 $\mathsf{LocalFork}_{\mathcal{S},\kappa}(\theta)$

$h_\nu \leftarrow_\$ C, \quad 1 \le \nu \le m$
$(\pi, \tau) \leftarrow \mathcal{S}(\theta; h_1, \ldots, h_m)$
if $\pi = \bot$
    return $(\bot, \bot)$
$k \leftarrow \kappa(\pi), \quad h_k^* \leftarrow_\$ C$
$(\pi^*, \tau^*) \leftarrow \mathcal{S}(\theta; h_1, \ldots, h_{k-1}, h_k^*, h_{k+1}, \ldots, h_m)$
if $\pi^* = \pi \ \wedge \ h_k \neq h_k^*$
    return $(\pi, (\tau, \tau^*))$
return $(\bot, \bot)$

The local fork can itself be forked, which will allow us to apply it recursively (cf. Section D.3.2). Indeed, if the tape $h_1, \ldots, h_m$ is passed as input instead of being sampled at step 1 and the re-sampled hash h_k^* is regarded as part of the coins along with θ, then $\mathsf{LocalFork}_{\mathcal{S},\kappa}$ is itself a deterministic algorithm of the form (5.11) with main output π and side output (τ, τ^*), whose acceptance probability is $\mathsf{forkAcc}(\mathcal{S}, \kappa)$ (although not of the typical kind, since it runs $\mathcal{S}$ twice on different tapes). The local forking lemma generalizes as follows.

Lemma D.15. (*Generalized Local Forking Lemma*) *For every deterministic algorithm $\mathcal{S}$ as in (5.11) with Π finite and every map $\kappa : \Pi \to \{1, \ldots, m\}$,*

$$\mathsf{forkAcc}(\mathcal{S}, \kappa) \ \ge \ \frac{1}{|\Pi|}\, \mathsf{acc}(\mathcal{S})^2 \ - \ \frac{1}{|C|}\, \mathsf{acc}(\mathcal{S}). \tag{D.16}$$

Proof. Fix $\pi \in \Pi$ and let $k = \kappa(\pi)$. Given θ and the hashes h_ν, $\nu \neq k$, denote by f_π the probability over $h_k \leftarrow_\$ C$ that $\mathcal{S}$ outputs π as main output. Averaging over θ and $\{h_\nu\}_{\nu \neq k}$, the mean a_π of f_π is the probability that $\mathcal{S}$ accepts with main output π; since the events of accepting with distinct main outputs are disjoint, there holds $\sum_{\pi \in \Pi} a_\pi = \mathsf{acc}(\mathcal{S})$. Now observe that $\mathsf{LocalFork}_{\mathcal{S},\kappa}$ returns main output π if and only if both executions output π and $h_k^* \neq h_k$. The two executions share θ and $\{h_\nu\}_{\nu \neq k}$, while h_k and h_k^* are sampled independently; consequently, both output π with probability equal to the mean of f_π^2, which is at least a_π^2 (the mean of a square being at least the square of the mean). Requiring $h_k^* \neq h_k$ removes at most the probability that the first execution outputs π and $h_k^* = h_k$, i.e., at most $a_\pi / |C|$. Summing over the disjoint events for distinct π, we obtain

$$\mathsf{forkAcc}(\mathcal{S}, \kappa) \ \ge \ \sum_{\pi \in \Pi} a_\pi^2 \ - \ \frac{1}{|C|} \sum_{\pi \in \Pi} a_\pi \ \ge \ \frac{1}{|\Pi|} \Big(\sum_{\pi \in \Pi} a_\pi \Big)^2 \ - \ \frac{1}{|C|}\, \mathsf{acc}(\mathcal{S}),$$

where the last step follows from the Cauchy-Schwarz inequality. □

D.3.2 Nested fork and elimination

We proceed to fork the simulated impersonation attack iteratively, once at each position of $P(\psi)$, and to eliminate the logarithms of the commitments from the resulting relations (D.12).

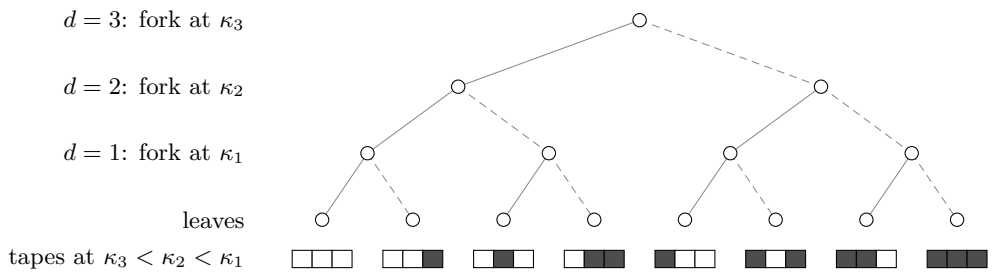

Figure 1: The nested fork $\mathcal{S}^{(\ell)}$ for $\ell = |P(\psi)| = 3$, with $\kappa_d \equiv \kappa_d(\pi)$. Every node at level d runs its left child on its own tape and its right child on the same tape with position κ_d re-sampled (dashed); the cells below a leaf mark the positions at which its tape differs from that of the leftmost leaf. The elimination of Algorithm 8 combines all leaves, level by level.

Let $\ell \geq 1$ be an upper bound on the number of components of the proofs that $\mathcal{A}$ fabricates, so that $|Q| \leq \ell \leq \bar{n}$ in every execution; the main outputs of $\mathcal{S}[\mathcal{A}]$ then range over

$$\Pi_\ell \equiv \big\{ (Q, \{k_i\}_{i \in Q}) : Q \subset \{1, \ldots, \bar{n}\},\ |Q| \leq \ell,\ k_i \in \{1, \ldots, m\} \big\},\quad |\Pi_\ell| \leq 2^{\bar{n}} m^\ell, \tag{D.17}$$

since there are at most $2^{\bar{n}}$ sets of indices and ψ assigns a position to each of their at most ℓ elements. Given $\pi = (Q, \psi) \in \Pi_\ell$ and $1 \leq d \leq \ell$, we denote by $\kappa_d(\pi)$ the d-th largest element of $P(\psi)$ if $d \leq |P(\psi)|$, and set $\kappa_d(\pi) = m$ otherwise. That is, the fork positions are those at which the components were submitted, processed in decreasing order; levels beyond $|P(\psi)|$ fork at the last tape position and serve only to make the depth of the construction independent of the execution.

The order of the fork positions is essential. Forking at some position changes the challenge returned there and thereby, possibly, every element that $\mathcal{A}$ submits afterwards, while earlier elements remain intact (cf. Remark 5.3); two executions that differ at position k thus share the logarithms below k but not necessarily above it. Consequently, the positions above k must be eliminated within each of the two executions separately before these are compared at k, which is achieved by forking them at inner levels (cf. Figure 1).

Definition D.18. The *nested fork of* $\mathcal{S}[\mathcal{A}]$ is the algorithm $\mathcal{S}^{(\ell)}$ defined recursively by

$$\mathcal{S}^{(0)} \equiv \mathcal{S}[\mathcal{A}], \quad \mathcal{S}^{(d)} \equiv \mathsf{LocalFork}_{\mathcal{S}^{(d-1)}, \kappa_d}, \quad 1 \leq d \leq \ell,$$

where each $\mathcal{S}^{(d)}$ is regarded as a deterministic algorithm of the form (5.11) with main outputs in Π_ℓ (cf. Section D.3.1).

The coins of $\mathcal{S}^{(d)}$ consist of (y^*, ω) (cf. Remark 5.2) along with the hashes re-sampled at levels $1, \ldots, d$ (cf. Algorithm 7, step 5). Unfolding the recursion, a non-aborting execution of $\mathcal{S}^{(\ell)}$ consists of 2^ℓ executions of $\mathcal{S}[\mathcal{A}]$, the *leaves* of a binary tree of depth ℓ; every node at level $d \geq 1$ corresponds to an execution of $\mathsf{LocalFork}_{\mathcal{S}^{(d-1)}, \kappa_d}$, whose two children are executions of $\mathcal{S}^{(d-1)}$ on tapes that differ at position $\kappa_d(\pi)$. The side output of $\mathcal{S}^{(\ell)}$ is accordingly a nested pair of depth ℓ whose entries are the side outputs of the leaves (cf. Figure 1).

Lemma D.19. (*Leaves of the nested fork*) *Let $\pi = (Q, \psi)$ be the main output of a non-aborting execution of $\mathcal{S}^{(\ell)}$. Then every leaf has main output π; in particular, $P(\psi)$ and Λ_k, $k \in P(\psi)$, are common to all leaves.*

Proof. By construction, $\mathsf{LocalFork}_{\mathcal{S}^{(d-1)},\kappa_d}$ returns π only if both executions of $\mathcal{S}^{(d-1)}$ return π (cf. Algorithm 7, step 7); by induction on d, every node and thus every leaf has main output π. The second part follows because $P(\psi)$ and the Λ_k's depend only on π (cf. (5.8)). □

The probability that the nested fork does not abort is bounded as follows.

Corollary D.20. (*Nested fork bound*) *For every adversary $\mathcal{A}$ for Game 4.1 as read in Section D.2, fabricating proofs with at most ℓ components,*

$$\mathsf{acc}(\mathcal{S}^{(\ell)}) \;\geq\; \frac{\mathsf{acc}(\mathcal{S}[\mathcal{A}])^{2^\ell}}{|\Pi_\ell|^{2^\ell-1}} \;-\; \frac{2^\ell-1}{q}. \tag{D.21}$$

Proof. Denote $a_d = \mathsf{acc}(\mathcal{S}^{(d)})$, $M = |\Pi_\ell|$, $X_d = a_0^{2^d}/M^{2^d-1}$. Since $\mathcal{S}^{(d)}$ is the fork of $\mathcal{S}^{(d-1)}$ with respect to κ_d, Lemma D.15 yields $a_d \geq a_{d-1}^2/M - 1/q$. We show $a_d \geq X_d - (2^d-1)/q$ by induction on d. The case $d = 0$ is trivial. If $X_{d-1} \leq (2^{d-1}-1)/q$, then $X_d \leq X_{d-1} < (2^d-1)/q$ and there is nothing to prove. Otherwise, the induction hypothesis bounds a_{d-1} from below by a positive value; squaring it and using $X_{d-1} \leq 1$ yields $a_{d-1}^2 \geq X_{d-1}^2 - 2(2^{d-1}-1)/q$, so that $a_d \geq X_{d-1}^2/M - (2^d-2)/q - 1/q = X_d - (2^d-1)/q$. □

The leaves below a node are related as follows.

Lemma D.22. (*Invariants of the nested fork*) *Let $\pi = (Q, \psi)$ be the main output of a non-aborting execution of $\mathcal{S}^{(\ell)}$. For every node at level $1 \leq d \leq |P(\psi)|$,*

(i) all leaves below it submit the same element v_k at every position $k \leq \kappa_d(\pi)$ and receive the same challenge h_k at every position $k < \kappa_d(\pi)$;

(ii) all leaves below its left child receive the same challenge at position $\kappa_d(\pi)$, and so do all leaves below its right child, the two challenges being distinct.

Proof. (i) Fix a node at level $1 \leq d \leq |P(\psi)|$. Its leaves are executions of $\mathcal{S}[\mathcal{A}]$ with the same y^* and ω, whose tapes coincide except at the positions where the node and its descendants fork, i.e., at $\kappa_{d'}(\pi)$, $d' \leq d$; indeed, forks at higher levels act on the input of the node, which is common to all of its leaves, while every fork below it occurs at position $\kappa_{d'}$ of the main output of its first execution, which is π (cf. Lemma D.19). Since $d' \leq d \leq |P(\psi)|$, these positions are the d largest elements of $P(\psi)$ and thus lie at or above $\kappa_d(\pi)$; consequently, the tapes of all leaves coincide at every position $k < \kappa_d(\pi)$. By Remark 5.3, the leaves submit the same elements at all positions $k \leq \kappa_d(\pi)$. (ii) The children of the node run on tapes that differ at position $k = \kappa_d(\pi)$, where they carry distinct hashes (cf. Algorithm 7, step 7). For $d = 1$, the children are the leaves themselves. For $d \geq 2$, every child is a node at level $d-1$, whose leaves run on tapes that coincide with its own at every position below $\kappa_{d-1}(\pi)$ (cf. the proof of (i)), in particular at k. □

We now proceed to the elimination procedure, which generalizes the subtraction outlined after Proposition D.11. Given $\pi = (Q, \psi)$ and $0 \leq d \leq \ell$, denote by $P_d(\psi)$ the set obtained from $P(\psi)$ by removing its d largest elements (so that $P_0(\psi) = P(\psi)$ and $P_d(\psi) = \varnothing$ for $d \geq |P(\psi)|$). The procedure is shown in Algorithm 8, where τ_N is the side output of a node at level d, i.e., the proof σ of a leaf if $d = 0$ and the pair (τ, τ^*) of side outputs of its children otherwise (cf. Algorithm 7). At the leaves, it computes the right-hand side of (D.12) (steps 1-3). At every other node, it combines the values returned for its children (step 5) by eliminating the unknown logarithm at the fork position of the node (steps 9-11), unless there is nothing to eliminate (steps 7-8). The challenges at position k can be read from any leaf below the respective child, namely as c_i for some $i \in Q$ with $k_i = k$ (cf. Lemma D.22, (ii)).

Algorithm 8 $\mathsf{Elim}_\pi(d, \tau_N)$

1: **if** $d = 0$
2: Parse $\{(u_i, c_i, s_i)\}_{i \in Q} \leftarrow \tau_N$
3: **return** $\sum_{i \in Q} \lambda_i \, s_i / c_i$ // K_σ, *cf.* (D.12)
4: Parse $(\tau, \tau^*) \leftarrow \tau_N$
5: $K \leftarrow \mathsf{Elim}_\pi(d-1, \tau), \quad K^* \leftarrow \mathsf{Elim}_\pi(d-1, \tau^*)$
6: $k \leftarrow \kappa_d(\pi)$
7: **if** $d > |P(\psi)| \ \vee \ \Lambda_k = 0$
8: **return** K // *Nothing to eliminate*
9: $h_k, h_k^* \leftarrow$ challenges at position k in the leftmost leaves of τ, τ^*
10: $z \leftarrow \dfrac{K - K^*}{\Lambda_k \, (1/h_k - 1/h_k^*)}$ // $z = \log v_k$
11: **return** $K - \Lambda_k \, z / h_k$

Proposition D.23. (*Nested fork and extraction*) *Let $\pi = (Q, \psi)$ and τ be the main and side output of a non-aborting execution of $\mathcal{S}^{(\ell)}$ with $y^* = g^x$, such that all challenges of its leaves are non-zero. Then $\mathsf{Elim}_\pi(\ell, \tau) = x$. More accurately, for every node at level d with side output τ_N, the value $K_N \equiv \mathsf{Elim}_\pi(d, \tau_N)$ satisfies*

$$x \ + \sum_{k \in P_d(\psi)} \Lambda_k \frac{\log v_k}{h_k} \ = \ K_N \, , \tag{D.24}$$

where v_k and h_k, $k \in P_d(\psi)$, are the elements and challenges common to the leaves of the node.

Proof. Note first that the elements and challenges in question are indeed common to the leaves of the node: for $d = 0$ the node is a single leaf; for $1 \le d \le |P(\psi)|$, every $k \in P_d(\psi)$ satisfies $k < \kappa_d(\pi)$ and Lemma D.22, (i), applies; for $d > |P(\psi)|$ the sum is empty. We proceed by induction on d. For $d = 0$, the node is a leaf whose proof is valid against y^* with non-zero challenges, so that (D.24) is condition (D.12) (cf. Proposition D.11). Let $d \ge 1$ and K, K^* be the values returned for the two children, which satisfy (D.24) for $P_{d-1}(\psi)$ by the induction hypothesis. If $d > |P(\psi)|$, then $P_{d-1}(\psi) = P_d(\psi) = \varnothing$ and $K = x$. Otherwise, $P_{d-1}(\psi) = P_d(\psi) \cup \{k\}$ with $k = \kappa_d(\pi)$, and the two relations involve the same elements $v_{k'}$ at all positions $k' \le k$ and the same challenges at all positions $k' < k$ (cf. Lemma D.22, (i)), while their challenges h_k, h_k^* at position k are distinct (cf. Lemma D.22, (ii)). Subtracting them, we get

$$\Lambda_k \Big(\frac{1}{h_k} - \frac{1}{h_k^*} \Big) \log v_k \ = \ K - K^* .$$

If $\Lambda_k = 0$, the term at position k is absent from the relation of the first child, which is thus (D.24) with $K_N = K$. Otherwise, the above determines $z = \log v_k$ (step 10); substituting it into the relation of the first child yields (D.24) with $K_N = K - \Lambda_k z / h_k$ (step 11). For $d = \ell$, we have $P_\ell(\psi) = \varnothing$ since $|P(\psi)| \le |Q| \le \ell$, so that $K_N = x$. □

Observe that the logarithms computed along the way are the true logarithms of the fabricated commitments, even if these are internally structured (e.g., built from y or from each other); the system of relations is linear and solved exactly (cf. the remarks after Proposition D.11). For the non-distributed case $n = \bar{n} = 1$, there is a single position to eliminate and, by (D.12), the procedure reduces to the usual extraction formula $x = (s - s^*)/(c - c^*)$ of the ordinary Schnorr protocol, where (u, c, s), (u, c^*, s^*) are the two forked transcripts.

Theorem D.25. ($\mathsf{OSST}^{\mathsf{c}}$ – *Security against impersonation attacks*) *Given* $1 \le t \le n \le \bar{n} < q$, $\mathsf{OSST}^{\mathsf{c}}_{\mathbb{G},n,t}$ (*cf. Protocol* 3.15) *is secure against impersonation attacks under* DL *hardness in the random oracle model, for a verifier that enforces the bound* $\bar{n}$ *on the indices. In particular, if* H *is modelled as a random oracle, for every polynomial-time adversary* $\mathcal{A}$ *that attacks Game* 4.1 *as read in Section* D.2*, fabricates proofs with at most* ℓ *components and issues at most* q_{ro} *distinct queries to the random oracle, there exists an adversary* $\mathcal{A}^*$ *whose running time is about* 2^ℓ *times that of* $\mathcal{A}$*, with*

$$\mathsf{Adv}^{\mathsf{DL}}_{\mathbb{G}}[\mathcal{A}^*] \;\ge\; \frac{\mathsf{Adv}^{\mathsf{OSST}^{\mathsf{c}}}_{\mathbb{G},n,t}[\mathcal{A}]^{2^\ell}}{|\Pi_\ell|^{2^\ell - 1}} \;-\; \frac{m + \ell + 2^\ell - 1}{q}, \tag{D.26}$$

where $m = q_{\mathsf{ro}} + \ell$ *and* $|\Pi_\ell| \le 2^{\bar{n}} m^\ell$ (*cf.* (D.17)).

Proof. We construct $\mathcal{A}^*$ as a wrapper of the nested fork $\mathcal{S}^{(\ell)}$ (cf. Definition D.18), followed by elimination (cf. Algorithm 8). If needed, $\mathcal{A}$ is first reprogrammed to query hashes for all the components it outputs (cf. Section 5.1), at the cost of at most $|Q| \le \ell$ additional queries; the simulation then issues at most $m = q_{\mathsf{ro}} + \ell$ distinct queries, which we take as the length of the tape.

When given w by its challenger, $\mathcal{A}^*$ operates as shown in Algorithm 9. In steps 1-3, $\mathcal{A}^*$ prepares the input of the nested fork, embedding its own challenge as the attacked public key and sampling the remaining input uniformly; since w is uniform, the input is distributed as in (5.5), along with the uniform values $h_{m+1}, \ldots, h_{m+\ell}$, which serve as the hashes re-sampled at levels $1, \ldots, \ell$. The hash re-sampled at level d forms part of the coins of $\mathcal{S}^{(d)}$ (cf. Definition D.18) and is thus shared by both executions of $\mathcal{S}^{(d)}$ at every node of higher level; consequently, the challenges of all leaves are drawn from the $m + \ell$ uniformly random values $h_1, \ldots, h_{m+\ell}$, and one of them vanishes with probability at most $(m + \ell)/q$. Observe that $\mathcal{A}^*$ aborts exactly when the nested fork aborts or some leaf carries a zero challenge (step 5); otherwise, Proposition D.23 ensures that step 7 yields $\log y^* = \log w$. Consequently,

$$\mathsf{Adv}^{\mathsf{DL}}_{\mathbb{G}}[\mathcal{A}^*] \;\ge\; \mathsf{acc}(\mathcal{S}^{(\ell)}) \;-\; \frac{m + \ell}{q} \;\ge\; \frac{\mathsf{Adv}^{\mathsf{OSST}^{\mathsf{c}}}_{\mathbb{G},n,t}[\mathcal{A}]^{2^\ell}}{|\Pi_\ell|^{2^\ell - 1}} \;-\; \frac{m + \ell + 2^\ell - 1}{q}$$

by Corollary D.20 and Corollary 5.6. Regarding running time, $\mathcal{A}^*$ executes $\mathcal{S}[\mathcal{A}]$ at most 2^ℓ times, along with polynomial-time bookkeeping and the $2^\ell - 1$ invocations of Algorithm 8 at the inner nodes, each of which costs a constant number of field operations apart from computing K_σ at the leaves. Finally, for fixed $\bar{n}$ we have $\ell \le \bar{n}$, so that 2^ℓ is a constant and $|\Pi_\ell| \le 2^{\bar{n}} m^\ell$ is polynomial in the bitlength of q; consequently, $\mathcal{A}^*$ is polynomial-time and its advantage is non-negligible whenever that of $\mathcal{A}$ is. □

Algorithm 9 $\mathcal{A}^*(w)$

Set $y^* \leftarrow w$
Set $\omega \leftarrow (\theta_{\mathcal{A}}, x_1^*, \ldots, x_n^*)$, where $\theta_{\mathcal{A}}$ are fresh coins for $\mathcal{A}$ and $x_i^* \leftarrow_{\$} \mathbb{Z}_q$, $1 \le i \le n$
Set $h_\nu \leftarrow_{\$} \mathbb{Z}_q$, $1 \le \nu \le m + \ell$
$(\pi, \tau) \leftarrow \mathcal{S}^{(\ell)}(y^*, \omega, h_{m+1}, \ldots, h_{m+\ell}; h_1, \ldots, h_m)$ // *Nested fork*
if $\pi = \perp$ or some leaf of τ carries a zero challenge
    return $\perp$
return $\mathsf{Elim}_\pi(\ell, \tau)$

For $\ell = \bar{n} = 1$, i.e., in the non-distributed case, $|\Pi_1| = m + 1$ and (D.26) reads $\mathsf{Adv}^2/(m+1) - (m+2)/q$, essentially the usual bound of the Schnorr protocol (cf. Remark 5.20).

Corollary D.27. ($\mathsf{OSST}^{\mathsf{c}}$ – *Security with observed transcripts*) *Given $1 \le t \le n \le \bar{n} < q$, the session-bound compact variant is secure against impersonation attacks with observed transcripts under* DL *hardness in the random oracle model, for a verifier that enforces the bound $\bar{n}$ on the indices. In particular, if H is modelled as a random oracle, for every polynomial-time adversary $\mathcal{A}$ that attacks Game 4.5 as read in Section D.2, fabricates proofs with at most ℓ components, issues at most q_{ro} distinct queries to the random oracle and at most q_{pr} queries to $\mathcal{O}^{\mathsf{prove}}$, there exists an adversary $\mathcal{A}^*$ whose running time is about 2^ℓ times that of $\mathcal{A}$, with*

$$\mathsf{Adv}^{\mathsf{DL}}_{\mathbb{G}}[\mathcal{A}^*] \;\ge\; \frac{\max\left\{0,\; \mathsf{Adv}^{\mathsf{OSST}^{\mathsf{c}},\mathbf{tr}}_{\mathbb{G},n,t}[\mathcal{A}] - q_{\mathsf{pr}}(m+q_{\mathsf{pr}})/q\right\}^{2^\ell}}{|\Pi_\ell|^{2^\ell-1}} \;-\; \frac{m+\ell+2^\ell-1}{q}, \tag{D.28}$$

where $m = q_{\mathsf{ro}} + \ell$ and $|\Pi_\ell| \le 2^{\bar{n}} m^\ell$.

Proof. As for Theorem D.25, with $\mathcal{S}^{\mathbf{tr}}[\mathcal{A}]$ in place of $\mathcal{S}[\mathcal{A}]$ and Proposition 5.22 in place of Corollary 5.6; the coins $\omega^{\mathbf{tr}}$ are sampled as in (5.21), the simulated prover programs $H(u, \mathsf{sid}) := c$ and returns (u, s), and Proposition 5.22 holds verbatim. The bound of Corollary D.20 is increasing in $\mathsf{acc}(\mathcal{S}^{\mathbf{tr}}[\mathcal{A}])$ and thus remains valid if the latter is replaced by its non-negative lower bound $\max\{0, \mathsf{Adv} - q_{\mathsf{pr}}(m+q_{\mathsf{pr}})/q\}$ (cf. (5.23)). Honest commitments copied from other sessions receive fresh challenges under sid^* and are eliminated like any fabricated commitment.[11] The running time additionally includes the simulation of at most q_{pr} components per execution of $\mathcal{S}^{\mathbf{tr}}[\mathcal{A}]$, each of which costs $O(t)$ exponentiations. Finally, since q_{pr} and m are polynomial in the bitlength of q for polynomial-time $\mathcal{A}$, the term $q_{\mathsf{pr}}(m+q_{\mathsf{pr}})/q$ is negligible, and the asymptotic conclusion of Theorem D.25 carries over. □

References

[1] Renas Bacho and Alireza Kavousi. Sok: Dlog-based distributed key generation. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 614–632, 2025.

[11] Substituting their logarithms by means of the honest responses would make the coefficient of x depend on the forked challenges, so that the differences of Algorithm 8 would no longer eliminate it.

[2] Karim Baghery, Noah Knapen, Georgio Nicolas, and Mahdi Rahimi. Pre-constructed publicly verifiable secret sharing and applications. In *Applied Cryptography and Network Security (ACNS) 2025*, volume 15825 of *Lecture Notes in Computer Science*, pages 89–119. Springer, 2025.

[3] Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. Multisignatures Secure under the Discrete Logarithm Assumption and a Generalized Forking Lemma. In *ACM Conference on Computer and Communications Security (CCS) 2008*, pages 449–458. ACM, 2008.

[4] Razvan Barbulescu and Sylvain Duquesne. Updating Key Size Estimations for Pairings. *Journal of Cryptology*, 32(4):1298–1336, 2019.

[5] K. Barr and K. Asanovic. Energy aware lossless data compression. In *MobiSys*. ACM, 2003.

[6] Mihir Bellare, Wei Dai, and Lucy Li. The Local Forking Lemma and its Application to Deterministic Encryption. In *Advances in Cryptology – ASIACRYPT 2019*, volume 11923 of *Lecture Notes in Computer Science*, pages 607–636. Springer, 2019.

[7] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-Key model and a general forking lemma. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006*, pages 390–399. ACM, 2006.

[8] Mihir Bellare and Adriana Palacio. GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and Concurrent Attacks. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 2002.

[9] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)Security of ROS. In *Advances in Cryptology – EUROCRYPT 2021*, volume 12696 of *Lecture Notes in Computer Science*, pages 33–53. Springer, 2021.

[10] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 626–643. Springer, 2012.

[11] Alexandra Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *Public Key Cryptography – PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2003.

[12] Emmanuel Bresson, Jacques Stern, and Michael Szydlo. Threshold Ring Signatures and Applications to Ad-hoc Groups. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 465–480. Springer, 2002.

[13] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In Yvo G. Desmedt, editor, *Advances in Cryptology – CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 1994.

[14] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, 2023.

[15] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2518–2534. IEEE, 2022.

[16] 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 (SP)*, pages 1084–1101. IEEE, 2019.

[17] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438, 1987.

[18] 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, 2007.

[19] Conrado Gouvêa. Frost performance. https://zfnd.org/frost-performance, 2023. Accessed: 2025-09-11.

[20] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology - EUROCRYPT 2016*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

[21] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Aggregatable distributed key generation. In *Advances in Cryptology – EUROCRYPT 2021*, volume 12696 of *Lecture Notes in Computer Science*, pages 147–176. Springer, 2021.

[22] Nick A. Howgrave-Graham and Nigel P. Smart. Lattice Attacks on Digital Signature Schemes. *Designs, Codes and Cryptography*, 23(3):283–290, 2001.

[23] Shaoquan Jiang, Dima Alhadidi, and Hamid Fazli Khojir. Key-and-Signature Compact Multi-Signatures for Blockchain: A Compiler With Realizations. *IEEE Transactions on Dependable and Secure Computing*, 22(1):579–596, 2025.

[24] Aniket Kate, Yizhou Huang, and Ian Goldberg. Distributed key generation in the wild. Cryptology ePrint Archive, Paper 2012/377, 2012.

[25] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology - ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.

[26] Marcel Keller, Gert Læssøe Mikkelsen, and Andy Rupp. Efficient Threshold Zero-Knowledge with Applications to User-Centric Protocols. In *Information Theoretic Security (ICITS) 2012*, volume 7412 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 2012.

[27] Chelsea Komlo and Ian Goldberg. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. In *Selected Areas in Cryptography (SAC) 2020*, volume 12804 of *Lecture Notes in Computer Science*, pages 34–65. Springer, 2020.

[28] Ueli M. Maurer. Unifying Zero-Knowledge Proofs of Knowledge. In Bart Preneel, editor, *Progress in Cryptology - AFRICACRYPT 2009, Second International Conference on Cryptology in Africa, Gammarth, Tunisia, June 21-25, 2009. Proceedings*, volume 5580 of *Lecture Notes in Computer Science*, pages 272–286. Springer, 2009.

[29] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr Multi-Signatures with Applications to Bitcoin. *Designs, Codes and Cryptography*, 87(9):2139–2164, 2019.

[30] Lorenz Minder and Alistair Sinclair. The Extended k-tree Algorithm. *Journal of Cryptology*, 25(2):349–382, 2012.

[31] Bodo Möller. Algorithms for Multi-exponentiation. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography*, volume 2259 of *Lecture Notes in Computer Science*, pages 165–180, Berlin, Heidelberg, 2001. Springer.

[32] Peter L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48(177):243–264, 1987.

[33] Phong Q. Nguyen and Igor E. Shparlinski. The Insecurity of the Digital Signature Algorithm with Partially Known Nonces. *Journal of Cryptology*, 15(3):151–176, 2002.

[34] Computer Security Resource Center NIST. Multi-party threshold cryptography. https://csrc.nist.gov/projects/threshold-cryptography, 2025. Accessed: 2025-09-11.

[35] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*, 1991.

[36] Nicholas Pippenger. On the Evaluation of Powers and Monomials. *SIAM Journal on Computing*, 9(2):230–250, May 1980.

[37] David Pointcheval and Jacques Stern. Security Proofs for Signature Schemes. In *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398. Springer, 1996.

[38] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: Robust Asynchronous Schnorr Threshold Signatures. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2551–2564. ACM, 2022.

[39] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[40] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Annual International Cryptology Conference*, 1999.

[41] Adi Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.

[42] Markus Stadler. Publicly verifiable secret sharing. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 190–199. Springer-Verlag, 5 1996.

[43] Douglas R. Stinson and Reto Strobl. Provably Secure Distributed Schnorr Signatures and a (t, n) Threshold Scheme for Implicit Certificates. In Vijay Varadharajan and Yi Mu, editors, *Information Security and Privacy*, pages 417–434, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[44] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan Gueta, and Srinivas Devadas. Towards scalable threshold cryptosystems. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 877–893. IEEE, 2020.

[45] David Wagner. A Generalized Birthday Problem. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–304. Springer, 2002.