SCHNORR THRESHOLD IDENTIFICATION

AUTHOR NAME

Abstract. We provide evidence that the protocol has good chances to be secure by demonstrating a formal security proof in the random oracle model under one-more discrete-logarithm (OMDL) hardness.

1. Introduction

native threshold protocol as much as the ordinary protocol

2. The problem of distributed schnorr identification

Fix a cyclic group $\mathbb{G} = \langle g \rangle$ of order q. By secret key is meant a scalar $x \in \mathbb{Z}_q$, with the group element $y = g^x$ being its public counterpart. Equivalently, $x = \log y$, where the logarithm is taken with respect to the generator g of the underlying group structure. *Discrete-logarithm hardness* (DL) for the system under consideration is the assumption that computing the logarithm of a uniformly random element is infeasible for every existing computer. More accurately, given $y \leftarrow_{\$} \mathbb{G}$, every probabilistic polynomial-time algorithm sould compute $x \leftarrow \log y$ correctly with at most negligible chance of success, where negligibility is determined by the current level of disposable computational power. In more applied terms, it should be impossible to infer a truly random key from its public counterpart within reasonable time. Industrial public-key cryptography relies increasingly more on the plausibility of this assumption over carefully chosen groups, a trend that will only be challenged with the advent of practical quantum computing.

The *Schnorr identification protocol* is a non-interactive zero-knowledge (NIZK) scheme, allowing the key holder of $y = g^x$ to prove knowledge of x without revealing any information about the latter. Its interactive variance is a sigma protocol. The prover samples $r \leftarrow_{\$} \mathbb{Z}_q$, stores it for the current session and sends the commitment $u = g^r$ 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 the relation

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

is satisfied; indeed, $g^s = g^{r+cx} = (g^x)^c g^r = y^c u$. In the presence of a secure hash function $H : \mathbb{G} \to \mathbb{Z}_q$, the protocol can be made non-interactive (1-step) by means of the Fiat-Shamir transform. Namely, the prover precomputes $c \leftarrow H(u)$ and sends (u, s) to the verifier, who retrieves c from u and proceeds to verification. Non-interactiveness is most often preferrable due to greater robustness, reduced throughput and non-reliance on real-time communication. The intuition is that a secure hash function injects sufficiently high entropy across every bit of its output, so that assigning $c \leftarrow H(u)$ becomes indistinguishable from sampling c uniformly at random. The current mainstream formalizes this fact within the *random oracle model* (ROM).

The Schnorr protocol is an "identification" scheme because it verifies the holder of a key against its public identity in a sound and complete fashion. Note that key semantics are not essential; the protocol identifies any procedure that "sees" the secret logarithm of some given group element. It is namely a zero-knowledge (ZK) proof-of-knowledge primitive for the discrete logarithm and as such applicable in any context where this knowledge arises as crucial (e.g., in plain ElGamal encryption, where a proof for the involved randomness may be attached in order to derive a non-malleable ciphertext); most notably, it is the building block for proving knowledge of witnesses to arbitrary linear relations, common instances of which are the Chaum-Pedersen protocol for Diffie-Hellman tuples and the Okamoto protocol for representations. From this perspective, security against impersonation attacks amounts to the infeasibility of fabricating a proof that verifies against a group element without knowing its logarithm; in the proof-of-knowledge terminology, this is called witness extractability under discrete-logarithm hardness. Whether extraction is explicitly performed by rewinding or is absorbed within a stronger hardness assumption is a matter of taste and strategy; in any case, the protocol becomes meaningful only in the font of discrete-logarithm hardness, which in turn guarantees that the protocol is secure.[1]

Suppose that some secret x with public counterpart y is distributed as $x_1, \ldots, x_n$ among n shareholders by means of Shamir's (n,t)-sharing or any equivalent distributed key generation (DKG) protocol. That is, it shouldn't matter if x is shared by a dealer or is implicitly defined in a fully decentralized fashion without being locally reconstructible; it only suffices that the output of the distribution scheme is indistinguishable from that of Shamir's (n,t)-sharing. We are interested in a protocol that allows any $t^* \geq t$ (but no less) shareholders to collectively prove that they belong to the group represented by y in a zero-knowledge fashion. More accurately, involved shareholders should be able to prove that their individual shares combine to x by means of Shamir's reconstruction formula (cf. Remark A.4) without revealing any information beyond that; evidently, this reduces to ordinary Schnorr identification for the edge case $t = 1$. Security against impersonation attacks should roughly mean that any efficient adversary controlling up to $t-1$ shareholders remains unable to fabricate a proof that verifies with non-negligible chances. We obviously ask for the protocol to be as less interactive as possible; in the ideal case, involved shareholders should not need to engage in any communication rounds, performing instead single and independent requests to the verifier.

There is a seemingly trivial solution to this problem involving the public keys $y_i = g^{x_i}$, $1 \leq i \leq n$ of the shareholders. Let $Q \subset \{1, \ldots, n\}$ be a coalition of shareholders that engage in an identification session with a verifier. The i-th party generates an ordinary Schnorr proof for its respective share, i.e.,

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

and sends it to the verifier. After aggregating the proof packets (u_i, s_i), $i \in Q$, the verifier computes $c_i \leftarrow H(u_i)$, $i \in Q$ and accepts only if the relation

[1] The most far-reaching result of this kind is that Schnorr identification is secure against active and concurrent impersonation attacks under "one-more" discrecte logarithm (OMDL) hardness; cf. Theorem 2, [9].

$$y = \prod_{i \in Q} y_i^{\lambda_i} \;\wedge\; g^{s_i} = y_i^{c_i} u_i,\; \forall i \in Q \tag{2.1}$$

is satisfied, where $\{\lambda_i\}_{i \in Q}$ are the Lagrange interpolation coefficients for Q. We will refer to this as the *trivial approach.* That is, the verifier identifies the coalition only if the involved shareholders prove knowledge of their respective keys individually and their shares combine to the group public key in the sense of interpolation. Indeed, the latter is satisfied only if $|Q| \geq t$ (as desired), while the first is a common method for hardening distributed schemes against malleability, namely, requiring from the shareholders to present decoupled proofs for contextual statements that verify against their respective shares.[2] Note also that (2.1) is an overkill. If the public shares $y_1, \ldots, y_n$ are advertised as such anyway (i.e., the verifier registers them as the fixed sharing of y), the identifying condition reduces to

$$|Q| \geq t \;\wedge\; g^{s_i} = y_i^{c_i} u_i,\; \forall i \in Q; \tag{2.2}$$

even if the threshold parameter does not become known along with the public shares, the verifier can trivially infer t from $y_1, \ldots, y_n$ and y and make use of it in subsequent protocol rounds.

Evidently, this protocol is secure if $y_1, \ldots, y_n$ is a (n, t)-sharing of y; however, in the absence of side input on the verifier's side, the protocol is secure *only* under the latter assumption. Indeed, as illustrated by (2.2), the bulk of the work has been absorbed in the registration of the public shares during the setup phase; the verifier accepts only because the registered keys have already been trusted. Equivalently, the security analysis reduces trivially to the thread model for the initial setup. The problem is that, in real life, taking a trusted multi-key setup for granted is itself the opposite of trivial, since the possibilities regarding roles, trust assumptions and network operations compose a rather broad attack surface. In particular, since proof verification is entirely dependent on the registered public shares, the protocol remains susceptible to to all sorts of malicious acting during the setup phase. For example, a proxy that is responsible for advertising the public shares may tamper with a certain subset of them, causing denial-of-service whenever the respective shareholders engage in a session. Or, worse, the proxy colludes with a malicious registry delivering the group public key and mounts the following rogue-key attack. The proxy retains $y_1, \ldots, y_{t-1}$, replaces y_t with some y_t^* for which it knows the discrete logarithm and instructs the registry to deliver

$$y^* = (y_t^*)^{\lambda_t} \prod_{i=1}^{t-1} y_i^{\lambda_i}$$

in place of y, where $\{\lambda_i\}_i$ are the Lagrange interpolation coefficients for $\{1, \ldots, t\}$. By reverse-engineering Shamir reconstruction, the proxy tunes y_i^*, $t < i \leq n$ so that $y_1, \ldots, y_{t-1}, y_t^*, \ldots, y_n^*$ becomes a (n, t)-sharing of y^* and sends it to the verifier; in the sequel, it can jump in and impersonate the t-th shareholder whenever the shareholders $1, \ldots, t-1$ engage in a session. While ad hoc threat models can obviously be formulated for specific use cases (e.g., for a restricted pool of shareholders that undergo a well defined certification process), this discussion implies

[2]Compare the decoupled Chaum-Pedersen proofs in a threshold decryption scheme, used to ensure validity of the partial decryptors with respect to the ciphertext.

that an ecompassing threat model is impossible to pin down. In the terminology of Bellare and Neven [12], the protocol has no chance to be provably secure in the *plain public-key* model.

One can alleviate the situation by working in the *knowledge of secret key* (KOSK) model and design a zero-knowledge based registration process, capable of detecting rogue-key attempts by means of purely cryptographic techniques. In this case, a security analysis becomes more tractable because a threat model for the initial setup can be formulated in terms of well defined malicious acts (even though dependence on the setup may introduce unexpected complexities to a formal security proof). However, an operation of this kind is not obvious to scale, since it should ensure that the public shares are compatible, and its repetition can become cumbersome if the shareholders are not static. More importantly, the KOSK assumption narrows the applicability of the protocol severely, since any verifiable registration process is an infrastructure plugin which real-world vendors may be unable or unwilling to interoperate with.

These are standard arguments speaking in favour of secure multi-party protocols that are agnostic with respect to their setup internals. For threshold identification, this requirement is further hardened as *non-reliance on the public shares*, meaning that the verification operation should only involve the group public key while the protocol remains secure in the plain model. In fact, this is a sine qua non if the desired protocol is to be applicable in a high-stressed and flexible decentralized setting (cf. Section 2.1). In the extreme, the public shares should not even leak from the protocol transcripts (cf. Section 2.3).

2.1. **Distributed provers in decentralized networks.** We contended above that non-reliance on the public shares is a necessary condition for pragmatic security in the plain public-key model, where no concrete certification process for the shareholders' keys can be assumed to be available or known. Existence of such processes usually indicates a relatively limited number of shareholders that either belong to an organization or participate in a supposedly decentralized network, where group creation is subject to uniform rules and monitored. In such settings, the role of verifier is usually reserved for a bunch of authorized servers or dedicated nodes, responsible for managing some protected resource or performing specific acts in the context of a broader protocol. The trivial approach with an ad hoc initial setup, supported by the relevant infrastructure and accompanying attack model, can be a decent solution for these cases, at least insofar as the group of shareholders remains relatively static and general interoperability is not a hard requirement.

We are interested for use cases beyond that, where even maintaining the public shares, let alone trusting them, can cause severe headache. As a direct consequence, note that non-reliance on the public shares yields an almost plug-and-play solution for already advertised identities. Specifically, if the verification operation involves only the group public key, existing Schnorr verifiers need only extend their software in order to support the distributed case, without adding new infrastrucure for the registration of newly advertised shares. Pre-existing keys can be silently shared at the verifiers' complete negligence.

We address distributed identification in a fairly unconditioned context, with as few assumptions as possible beyond a generic synchronous network with reliable packet delivery. The only assumption about Shamir-backed group creation should be that the public key of the combined prover be advertised in some "trusted" fashion; no group relevant infrastructure is assumed to be available after that phase, e.g., proxies, registries for individual member identities, storages for pre-processed quantities, aggregators interacting with involved provers, or even traceable communication channels among them. A group might be created on the fly and its public key be the only remnant of that ceremony (except for the secrets held by its members in private); after dissolution of the setup channels, its members may have no way to coordinate or identify with each other, be unwilling to do so or be compelled to act independently. This is even more important if the desired primitive is to be applicable as distributed proof-of-secret in a secure multiparty computation; in this case, combined keys do not usually outlive protocol rounds and any relevant infrastructure is a costly overkill (if not at all prohibited).

Absence of group infrastructure implies that the shareholders should in general be able to act without coordination during the proving phase. In particular, they should not even need to communicate, meaning that the protocol has to be non-interactive on the provers' side. This introduces a high degree of asynchronicity which in turn imposes constraints on the interaction of the combined prover with the verifier. In particular, a verifier's response to the j-th shareholder should not be dependent on any request from the i-th shareholder for $i \neq j$. Equivalently, the proving session should decouple into independent interactions between the verifier and the individual shareholders; after aggregating the decoupled results of these interactions, the verifier combines them and proceeds to the final verification step. Ideally, the decoupled interactions should themselves be non-interactive, i.e., consist of a single communication step in which individual shareholders send their proof packets to the verifier respectively.

Note that the trivial approach satisfies these requirements automatically. Indeed, it presupposes no coordination or infrastructure to be available during the proving phase; infrastructure is only needed during the setup in order to accomodate the verifiers' trust on the public shares. To see why this is still inadequate, consider a *truly* decentralized network where qualified verifiers can be user devices or application servers alike. That is, verification may occur massively on a multitude of devices that need not be aware of the individual shareholders' identities for every group they come accross in the network. It simply cannot be expected from a mobile device to maintain the public shares of every group it might want to verify, let alone engage in a certification process in order to register them. Note also that fetching the public shares from remote registries on demand in not generally an option; even if maintaining these registries were possible, trusting them is a highly non-trivial assumption (comparable to the KOSK assumption) while depending on them significantly affects network latency and robustness.

Further efficiency considerations are here in order, since decentralized verifiers do not necessarily possess the capabilities of a fine-tuned application server. It is well known that the wireless transmission of a single bit consumes more power than a 64-bit (or 32-bit) instruction by orders of magnitude. Consequently, reduced throughout is most probably the dominating factor with respect to overall energy consumption; from the individual verifier's perspective, this amounts to longer batterly life if it runs on a wireless device. Reduced throughput is even more important in order to avoid network congestion; indeed, if the network is truly decentralized, proof packets have to be massively trasmitted through a bunch of dedicated relay nodes in order to be verified at arbitrary locations. The problem remains to design a non-interactive threshold protocol with reduced throughput that is non-reliant on public shares, infrastructure-free on both sides, secure in the plain public-key model, and as such applicable in truly decentralized settings.

2.2. **Non-working solutions.** We informally discuss two plausible approaches to the problem of distributed Schnorr identification and show how they fail to meet the requirements of Section 2.1. Both are within the plain public-key model.

2.2.1. *Uniform challenge approach.* Recall that Schnorr signature is Schnorr identification with a message baked into the hash function. This suggests an approach similar to the threshold signature scheme introduced by Stinson and Strobl [8]. Indeed, this requires no infrastructure extensions on the verifier's side, since verification involves only the group public key. Let $x_1, \ldots, x_n$ be a (n, t)-sharing of some secret x with public counterpart y. A coalition of shareholders $Q \subset \{1, \ldots, n\}$, $|Q| \geq t$, execute an equivalent DKG protocol to generate

$$r = \sum_{i \in Q} \lambda_i r_i,$$

where $\{\lambda_i\}_{i \in Q}$ are the interpolation coefficients for Q, holding the respective r_i's in private. r need not be reconstructible at any single location but involved parties should be able to verifiably infer the distributed commitment $u = g^r$ at the end of the process. Next, the involved parties generate Schnorr proofs for their keys using the respective commitment share and uniform challenge $H(u)$, 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

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

and accepts only if the relation $g^s = y^c u$ is satisfied. Indeed, since $|Q| \geq t$, twofold application of Shamir's reconstruction formula yields

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

Verification requires $|Q| + 2$ exponentations in total while the inbound throughput grows linearly with respect to $|Q|$; for example, taking $\mathbb{G}$ to be the elliptic curve P256 and assuming that 128-bit challenges is sufficient to work with, the total throughput is $512\,|Q|$ bits per protocol round. If desired, we can offload the computation of s and u to a semi-trusted combiner, who acts as group proxy by forwarding (u, s) to the verifier; in this case, the verifier coincides with that of the non-distributed case and its performance is not dependent on the number of involved shareholders. This is usually the case for Schnorr threshold signatures.

We will refer to the above protocol as the *uniform challenge* approach. Its security reduces most probably to that of the non-distributed case by carefully adapting the arguments of [8] to the identification setting. It relies on the fact that this "is" ordinary Schnorr identification if we treat the involved shareholders as a single entity; indeed, the combined proof (u, s) lies in the domain of proofs that could be generated for the combined secret x directly. This commutation is crucial for a threshold signature scheme, where the final object may need to remain available in the long run and consequently use minimum space, but not for an identification protocol, where no persistent data need be generated. In fact, it countervenes the requirement of reduced interactiveness, as it is achieved through a highly interactive computation for generating g^r. This evidently introduces significant network overhead upon every protocol round and presupposes that some infrastructure is constantly available to the group of shareholders. Consequently, despite the efficiency of its verifier, the uniform challenge approach does not meet the requirements of Section 2.1 on the provers' side. The state-of-the-art for managing the network overhead of Schnorr threshold signatures is the FROST protocol introduced by Komlo and Goldberg [4] and its variants. This reduces the number of communication rounds to one or two by means of a semi-trusted aggregator, depending on whether a preprocessing stage takes place during the initial setup; however, it comes at the cost of more infrastructure (and consequently more trust assumptions) which makes it even more prohibiting for a flexible identification setting.

2.2.2. *Completely decoupled approach.* This is a variance of the trivial approach, where the public shares are not advertized during the setup phase. Instead, they are attached to the Schnorr proofs that are generated by the respective shareholders. Given a (n, t)-sharing $x_1, \ldots, x_n$ of some secret x with public counterpart $y = g^x$, a coalition of shareholders $Q \subset \{1, \ldots, n\}$ generate

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

respectively and send them to the verifier. After aggregating (y_i, u_i, s_i), $i \in Q$, the verifier computes $c_i \leftarrow H(u_i)$ and accepts only if condition (2.1) is satisfied. We will refer to this protocol as the *completely decoupled* approach. Intuitively, its security reduces to the proof-of-knowledge property of the Schnorr protocol. Every shareholder "should know" the discrete logarithm of the attached share because the respective proof verifies; since the attached shares combine to the group public key, their respective logarithms comprise a sharing of x and the shareholders belong to the group represented by y.

The protocol is non-reliant on public shares and infrastructure-free on both sides. However, this is achieved at the cost of $3\,|Q|$ exponentiations per verification and at least 33% increased throughout as compared to the trivial approach. While this regression might be insignificant for a central identifying server with even several thousands of users, it can badly affect latency under the energy intensive and highly congested network conditions of true decentralization (cf. Section 2.1).

Moreover, the security argument is not as strong as it might seem. The Schnorr proof-of-knowledge property is not concretely defined in terms of an attack game and it basically means extractability under discrete logarithm hardness. Consequently, the security of the threshold protocol cannot be formally reduced to it in the strict sense; by saying that it "reduces to" it, what we actually mean is that some relevant extraction procedure should itself be reproducible in the context of a black-box threshold attack. This does not seem to be possible. Consider an adversary $\mathcal{A}$ who knows $\{x_i\}_{i \in J}$ for some $J \subset \{1, \ldots, n\}$ with $|J| = t - 1$ and wants to impersonate $Q = J \cup \{j\}$ with $j \notin J$ against an interactive verifier. One possible attack pattern could be using the shares that are known to the adversary directly. That is, for every $i \in J$, $\mathcal{A}$ sends

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

to the verifier, who samples $c_i \leftarrow_{\$} \mathbb{Z}_q$ for u_i and sends it to $\mathcal{A}$; in turn, $\mathcal{A}$ responds with $s_i \leftarrow r_i + c_i x_i$. Next, $\mathcal{A}$ fabricates u_j accoridng to some black-box strategy and sends (y_j, u_j), $y_j \leftarrow g^{x_j}$ to the verifier, who samples $c_j \leftarrow_{\$} \mathbb{Z}_q$ for u_j and sends it to $\mathcal{A}$. Finally, $\mathcal{A}$ fabricates s_j according to some black-box strategy and sends it to the verifier. Evidently, extraction of x_j is here possible by rewinding $\mathcal{A}$ exactly before sending (y_j, u_j) to the verifier, provided that it succeeds in generating a valid transcript. This is ordinary extraction embedded into a decoupled threshold attack pattern. However, a black-box threshold attack is far from confined to this pattern. Since y admits many possible sharings, $\mathcal{A}$ could possibly fabricate attachments that do not coincide with the fixed public shares and still combine to the group public key; in this case, rewinding $\mathcal{A}$ at any location would fail to yield x_j. We would need the decoupled proofs to be further junct into a single operation in order for the extraction of x_j to be possible by rewinding.

2.3. **Anonymous identification.** Consider an anonymous network that is moreover decentralized in the sense of Section 2.1. That is, fully qualified nodes can be user devices or application servers alike, with the extra assumption that they do not posess traceable identities. Certain nodes may do have some kind of traceable identity for discovery reasons (e.g., relay nodes similar to those of a Tor network), but these need not be used other than that.

Nodes may create permanent or ephemeral cryptographic identities for the purpose of engaging in verifiable operations; as usual, these are keys over a discrete-logarithm based cryptosystem $\mathbb{G} = \langle g \rangle$ that is agreed upon for the whole network. No assumptions are made about registries and lifetimes. A node may advertise a public key only to nodes of its choice, broadcast it to the network, use it for a restricted number of protocol rounds and then discard it, or fix it for signing and receiving messages in the eternity. Since the network is anonymous, a public key cannot be physically associated with the node holding its private counterpart. Consequently, a node is free to maintain multiple public keys which, by virtue of anonymity, cannot be linked unless their holder whishes so; for example, a node y can claim y^* by generating a Schnorr proof for the latter.

We further assume that group constitution is possible by means of Shamir's sharing or its DKG variants. We do not care how nodes discover each other for this purpose, e.g., through some physical identification process or referring to pre-existing cryptographic identities. The public shares may remain private to the group, who only advertises the combined public key; under circumstances, the group participants may not even learn each other's identity at the end of the process. We are interested for a threshold identification protocol as outlined in Section 2.1 under the extra constraint that the individual identities do not leak during a protocol round. Specifically, any coalition of shareholders should be able to act on behalf of the group without their public shares being implicitly inferred from the protocol transcript. The i-th shareholder would then be able to use its secret share x_i in irrelevant contexts without risk of revealing that $y_i = g^{x_i}$ is a group member identity; in particular, a curious verifier who happens to observe that usage should be unable to link y_i with the i-th shareholder.

Despite non-reliance on public shares, both protocols of Section 2.2 fail to be anonymous. In the uniform challenge approach, the public shares are inferred as

$$y_i \leftarrow \sqrt[c_i]{\frac{u_i}{g^{s_i}}}, \quad c_i \equiv H(u_i)$$

from the respective proof packets. Anonymity seems even more impossible in the completely decoupled approach, since the shares are explicitly attached to the proof packets. It can however be easily achieved with a tweak. During the setup phase, the zero scalar is distributed as $\eta_1, \ldots, \eta_n$ among the shareholders by means of Shamir (n, t)-sharing or equivalent DKG protocol. By the homomorphic property of sharings (cf. Remark A.2), $x_1 + \eta_1, \ldots, x_n + \eta_n$ remains a (n, t)-sharing of x. This suggests that the shareholders can use their respective zero share in order to blind their identity whenever involved in a round. Specifically, a coalition of shareholders $Q \subset \{1, \ldots, n\}$ generate

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

respectively and send them to the verifier. After aggregating (θ_i, u_i, s_i), $i \in Q$, the verifier computes $c_i \leftarrow H(u_i)$ and accepts only if the relation

$$y = \prod_{i \in Q} \theta_i^{\lambda_i} \ \wedge \ g^{s_i} = \theta_i^{c_i} u_i, \, \forall i \in Q$$

is satisfied. Evidently, this is the completely decoupled protocol with the difference that the involved parties attach the shares $\{\theta_i\}_{i \in Q}$ of y instead of their actual identities $\{y_i\}_{i \in Q}$. Anonymity follows because it is infeasible to compute any y_i's from $\{\theta_i\}_{i \in Q}$, provided that the blinding factors remain private to the shareholders. We will apply this technique to anonymize the protocol of the next section.

3. One-round schnorr threshold identification

Throughout this section we fix a cyclic group $\mathbb{G} = \langle g \rangle$ of order q where the discrete-logarithm problem is hard and a secure hash function $H : \mathbb{G} \to \mathbb{Z}_q$. The Shamir's (n, t)-sharing is denoted by D (cf. Section A), with the particular values of its parameters inferred always from context.

3.1. **Basic protocol** – ORST.

Protocol 3.1. (ORST – *One-Round Schnorr* (n, t)*-Threshold Identification*) Given integers $1 \leq t \leq n < q$,

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

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

  The threshold parameter and the public shares may remain private to the group, who needs only advertise the combined public key y.

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

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

  to the verifier and hold r_i in private, $i \in Q$.
- *Verification.* After aggregating (u_i, s_i), $i \in Q$ the verifier accepts only if

$$\exp\Big(g, \sum_{i \in Q} \mu_i s_i\Big) = y^{\bar{c}} \prod_{i \in Q} u_i^{\mu_i}, \tag{3.2}$$

  where

$$\bar{c} \leftarrow \prod_{i \in Q} c_i, \quad \mu_i \leftarrow \lambda_i \prod_{j \in Q \setminus \{i\}} c_j, \quad c_i \leftarrow H(u_i) \tag{3.3}$$

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

Remark 3.4. (*Validity condition*) By equating exponents, (3.2) translates to

$$\sum_{i \in Q} \mu_i (s_i - r_i) = \bar{c} \cdot x, \tag{3.5}$$

where $r_i \equiv \log u_i$. Observe that this contains no reference to shares.

Remark 3.6. (*Correctness of* ORST) Let $y_i \equiv g^{x_i},\ 1 \leq i \leq n$ be the public shares. The left-hand side of (3.2) transforms as

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

which yields the right-hand side if (and only if) $|Q| \geq t$. In other words, the proper execution of the protocol verifies only if at least t shareholders are involved.

Remark 3.7. (*Interactive aspect*) We may regard H as approximating a random oracle insofar as predicting any of its output bits remains infeasible; in the limit, involved shareholders submit their commitments to an external oracle that is accessible to the verifier. Or, equivalently, they send their commitments directly to the verifier, who lazily samples challenges by maintaining an associative array Map, so that the protocol becomes interactive. Specifically, the i-th shareholder, $i \in Q$, samples $r_i \leftarrow_{\$} \mathbb{Z}_q$ and sends $u_i \leftarrow g^{r_i}$ to the verifier, who samples $c_i \leftarrow_{\$} \mathbb{Z}_q$ if $u \notin \mathrm{Dom}(\mathsf{Map})$, caches $\mathsf{Map}[u_i] \leftarrow c_i$ and sends c_i to the i-th shareholder; otherwise, it sends $c_i \leftarrow \mathsf{Map}[u_i]$. The i-th shareholder responds with $s_i \leftarrow r_i + c_i x_i$. After aggregating s_i, $i \in Q$, the verifier checks that condition (3.2) is satisfied using the previously cached challenges $c_i = \mathsf{Map}[u_i]$, $i \in Q$.

The sharing scheme of the setup may be replaced with any DKG whose output distribution is indistinguishable from that of Shamir's secret sharing. That is, it shouldn't matter if x has been shared by a dealer or it cannot be reconstructed at a single location; it only suffices that the correct public key $y = g^x$ be advertised at the end of the process. Note that DKG's usually assume a bound $0 \leq l < t$ on the number of corrupt parties in order for the distribution to be secure. Since a threshold threat model allows the corruption of up to $t - 1$ parties, plugging a DKG is indifferent only if the corruption takes place after the completion of key distribution. This is in accordance with the requirement that ORST should be agnostic with respect to its setup internals, meaning that a separate security analysis should be dedicated per use case to it. TODO

During the proving phase, the involved shareholders generate ordinary Schnorr proofs for their respective shares and send them to the verifier asynchronously. No agreement or coordination is assumed between them. Order coordination or some session timeout may be enforced by the verifier for the needs of a specific use case, but this should not affect the threat model essentially because the local computations remain decoupled. The protocol is non-interactive in a strict sense ("one-round"), since it requires no communication rounds among the shareholders. Specifically, ORST meets the requirements of Section 2.1 on the provers' side.

Verification efficiency is comparable to that of the uniform challenge approach (without proxy), even if at the cost of more numerical operations. It requires $|Q|+2$ exponentiations while its throughput grows linearly with respect to the number of involved shareholders; for example, if $\mathbb{G}$ is the elliptic curve P256 and assuming that 128-bit challenges is sufficient to work with, the inbound throughput is $512\,|Q|$ bits per protocol round. Since the operation per se involves the group public key alone, no maintentance of the public shares is needed and the protocol meets the requirements of Section 2.1 on the verifier's side. Existing Schnorr verifiers need only extend their software in order to support the distributed case, without adding new infrastructure for the certification and registration of public shares.

Definition 3.8. The *weights of* a collection $\{u_i\}_{i\in Q} \subset \mathbb{G}$, $Q \subset \{1, \ldots, n\}$, are the scalars $\{\mu_i\}_{i\in Q}$ defined as

$$\mu_i = \lambda_i \prod_{j\in Q\setminus\{i\}} c_j \tag{3.9}$$

where $c_i \equiv H(u_i)$, $i \in Q$ and $\{\lambda_i\}_{i\in Q}$ are the interpolation coefficients for Q. Under the same notation, the *normalizer of* $\{u_i\}_{i\in Q}$ is the scalar

$$\bar{c} = \prod_{i\in Q} c_i \tag{3.10}$$

That is, ORST-verification juncts the decoupled Schnorr proofs through the weights and normalizer of the points $\{u_i\}_{i\in Q}$, which the shareholders respectively commit to. We have already used (cf. Remark 3.6) the following fundamental remark.

Remark 3.11. (*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

$$c_i\mu_i = \lambda_i\bar{c}, \quad \forall i \in Q \tag{3.12}$$

Definition 3.13. By *proof* is meant a collection $\sigma = \{(u_i, s_i)\}_{i\in Q}$ of the form

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

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* resp. *normalizer of* σ are the weights resp. normalizer of its commitments. The *proof space* is the set of proofs and will be denoted by Σ. We also denote by

$$\Sigma\big[\{u_i\}_{i\in Q}\big] = \{\,\sigma \in \Sigma : \exists \{s_i\}_{i\in Q} \text{ such that } \sigma = \{(u_i, s_i)\}_{i\in Q}\,\} \tag{3.14}$$

the subspace of proofs that are based on $\{u_i\}_{i\in Q}$. Evidently, proofs in this subspace share the same weights and normalizer.

Note that the above proof definition does not impose constraints on the relation between the commitments and the responses. Specifically, it does not confine to properly generated transcipts; it models the verifier's view in the wild and includes anything that malicious provers can possibly fabricate. The proper execution of the protocol is modelled as follows.

Definition 3.15. $\{(u_i, s_i)\}_{i\in Q}$ is called *canonical with respect to* $(x_1, \ldots, x_n;\ y) \in D_x$ if its components are ordinary Schnorr proofs for the respective shares, i.e.,

$$s_i = r_i + c_i\, x_i, \quad c_i \equiv H(u_i), \quad r_i \equiv \log u_i, \quad \forall i \in Q \tag{3.16}$$

By the "unique responses" property of the ordinary Schnorr protocol, $\Sigma\big[\{u_i\}_{i\in Q}\big]$ contains a unique canonical representative with respect to every fixed sharing.

Definition 3.17. The *one-round Schnorr* (n,t)*-threshold verifier* is the deterministic algorithm $\mathcal{V} : \mathbb{G} \times \Sigma \to \{\mathsf{true}, \mathsf{false}\}$ 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 3.18. A proof $\sigma \in \Sigma$ is called *valid against* $y \in \mathbb{G}$ if $\mathcal{V}(y, \sigma) = \mathsf{true}$.

Remark 3.19. ORST correctness (cf. Remark 3.6) means that a proof canonical with respect to $(x_1, \ldots, x_n; y)$ is valid against y if and only if $|Q| \geq t$.

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

$$\Sigma_y\big[\{u_i\}_{i\in Q}\big] = \{\,\sigma \in \Sigma\big[\{u_i\}_{i\in Q}\big] : \mathcal{V}(y, \sigma) = \mathsf{true}\,\}\,. \tag{3.20}$$

If $|Q| \geq t$, then this space contains a unique canonical representative with respect to every fixed sharing of $x = \log y$.

Remark 3.21. In the non-distributed case $t = 1$, the space $\Sigma_y\big[\{u_i\}_{i\in Q}\big]$ is a singleton due to the "unique responses" property of the Schnorr protocol. Specifically, for every commitment u there exists a unique response s such that the proof (u, s) is valid against $y = g^x$, namely $s = r + H(u)\,x$ with $r \equiv \log u$. In general, however, this space inflates dramatically at a rate dependent on $|Q| \geq t$ (cf. Remark 3.36). This implies an abundance of valid proofs that are non-canonical, a fact that every meaningful security notion should be able to trace.

Proposition 3.22. (*Validity conditions*) *Let $x \in \mathbb{Z}_q$, $y = g^x$ and $\sigma = \{(u_i, s_i)\}_{i\in Q}$. Then, with overwhelming probability, σ is valid against y if and only if*

$$\frac{1}{\bar{c}} \sum_{i\in Q} \mu_i (s_i - r_i) = x, \quad r_i \equiv \log u_i\,, \tag{3.23}$$

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

$$x = \sum_{i \in Q} \lambda_i \frac{s_i - r_i}{c_i}, \tag{3.24}$$

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

Proof. Since H is a secure hash function, $\bar{c} \neq 0$ with overwhelming probability in the bitlength of the group order. In this case, the verification condition (3.5) becomes (3.23). It further reformulates to (3.24) by application of (3.12). □

This 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 without reference to sharings. Both (3.23) and (3.24) generalize the relation of a secret to a valid non-distributed proof; it should be stressed however that (3.24) does not imply $x_i = (r_i - s_i)/c_i$ and can hold true without σ being canonical. Indeed (cf. Remark 3.29), valid proofs do not confine to those properly generated in ORST and a relevant security notion should ensure that such proofs are infeasible to compute without knowledge of at least t shares (cf. Section 4). The exact meaning of (3.24) is clarified in Proposition 3.28. A proof should evidently not verify against different public keys. In fact, this key almost over exists and it is unique.

Proposition 3.25. (*Almost every proof is valid against some unique public key*) *Given commitments $\{u_i\}_{i \in Q} \subset \mathbb{G}$, then, with overwhelming probability, every $\sigma \in \Sigma$ based on them is valid against some unique $y \in \mathbb{G}$.*

Proof. Let $\{\mu_i\}_{i \in Q}$, $\bar{c}$ be the weights and the normalizer of $\{u_i\}_{i \in Q}$, which are common to all proofs based on them. Since H is a secure hash function, $\bar{c} \neq 0$ with overwhelming probability in the bitlength of the group order. Consquently, given any proof of the form $\sigma = \{(u_i, s_i)\}_{i \in Q}$, we can define

$$x = \frac{1}{\bar{c}} \sum_{i \in Q} \mu_i \left(s_i - r_i\right),$$

where $r_i \equiv \log u_i$. By Proposition 3.22, σ is valid against $y = g^x$ solely. □

Corollary 3.26. *Given $\{u_i\}_{i \in Q} \subset \mathbb{G}$, then, with overwhelming probability,*

$$\Sigma\big[\{u_i\}_{i \in Q}\big] = \bigcup_{y \in \mathbb{G}} \Sigma_y\big[\{u_i\}_{i \in Q}\big] \tag{3.27}$$

where the right-hand side is a partition, i.e.,

$$\Sigma_y\big[\{u_i\}_{i \in Q}\big] \cap \Sigma_{y^*}\big[\{u_i\}_{i \in Q}\big] = \varnothing \quad \textit{for} \quad y \neq y^*.$$

Proof. Direct consequence of Proposition 3.25 and the definitions. □

Proposition 3.28. (*Almost every proof of size $|Q| = t$ is canonical with respect to some unique sharing*) *Given commitments $\{u_i\}_{i \in Q} \subset \mathbb{G}$ with $|Q| = t$, then, with overwhelming probability, every $\sigma \in \Sigma$ based on them is canonical with respect to some unique sharing $(x_1, \ldots, x_n; y) \in D_x$ of some unique $x \in \mathbb{Z}_q$.*

Proof. Let $\{\mu_i\}_{i \in Q}$ be the weights of $\{u_i\}_{i \in Q}$ and denote $c_i \equiv H(u_i)$. Note that these parameters are common to all proofs based on the given commitments. Since

H is a secure hash function, $c_i \neq 0,\ \forall i \in Q$ with overwhelming probability in the bitlength of the group order. Consequently, given $\sigma = \{(u_i, s_i)\}_{i\in Q}$, we can define

$$x_i = \frac{s_i - r_i}{c_i}, \quad i \in Q$$

where $r_i \equiv \log u_i$. By virtue of interpolation, since $|Q| = t$, the scalars $\{x_i\}_{i\in Q}$ uniquely determine a (n,t)-sharing $(x_1, \ldots, x_n; y)$ of some $x \in \mathbb{Z}_q$, namely

$$x = \sum_{i\in Q} \lambda_i x_i \,.$$

Since $s_i = r_i + c_i\, x_i, \forall i \in Q$, the proof is canonical with respect to this sharing and thus valid against $y = g^x$. The claim follows because a proof can be valid with respect to at most one public key (cf. Prop. 3.25). □

Remark 3.29. (*Validity and degrees of freedom*) Fix commitments $\{u_i\}_{i\in Q} \subset \mathbb{G}$. 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 $r_i \equiv \log u_i$. By Proposition 3.22 (cf. (3.23)), this process exhausts the proofs that are based on $\{u_i\}_{i\in Q}$ and valid against $y \equiv g^x$ bijectively. That is, the subspace $\Sigma_y\big[\{u_i\}_{i\in Q}\big]$ has $|Q| - 1$ degrees of freedom in the sense that every point of it is uniquely determined by the choice of $\{s_i\}_{i\in J}$ and $J \subset Q$ (for $|Q| = 1$ this is the "unique responses" property of ordinary Schnorr identification). Note that this process has no obvious utility for an impersonating adversary, since the adversary would still need to eliminate x in order to compute s_j.

Proposition 3.30. (*Validity against sharing*) *Let $x \in \mathbb{Z}_q$ and $\sigma = \{(u_i, s_i)\}_{i\in Q}$. Given $(x_1, \ldots, x_n; y) \in D_x$, then σ is valid against y if and only if*

$$\sum_{i\in Q} \mu_i\,(s_i - r_i - c_i\, x_i) = \bar{c}\cdot\Big(x - \sum_{i\in Q} \lambda_i x_i\Big), \tag{3.31}$$

Proof. Follows from the verification condition (3.5) by substracting

$$\sum_{i\in Q} \mu_i c_i x_i$$

on both sides and applying (3.12) on the right. □

Remark 3.32. (*Valid proofs with* $|Q| < t$) We know that if a proof of size $< t$ is valid, then it cannot be canonical (cf. Remark 3.6). Such proofs exist in abundance (cf. Remark 3.29) and Prop. 3.30 indicates a way to generate them programmatically. Fix $\{u_i\}_{i\in Q} \subset \mathbb{G}$ and set $s_i = r_i + c_i\, x_i$ where $r_i \equiv \log u_i$, $c_i \equiv H(u_i)$. Pick $S \subset Q$ with $S \neq \varnothing$ and define $s_i^* = s_i$ if $i \notin S$, otherwise

$$s_i^* = s_i + \delta_i, \quad \delta_i \;\equiv\; \frac{1}{\mu_i}\cdot\frac{\bar{c}}{|S|}\cdot\Big(x - \sum_{j\in Q} \lambda_j x_j\Big),$$

where $\bar{c}$ is the normalizer of $\{u_i\}_{i\in Q}$. Consider the proof $\sigma = \{(u_i, s_i)\}_Q$. If $|Q| \geq t$, then $\delta_i = 0, \forall i \in S$ so that σ is canonical and thus automatically valid. If $|Q| < t$, then σ still remains valid because it satisfies (3.31) by design. Note however that this construction has no obvious utility for a malicious coalition $Q \subset \{1, \ldots, n\}$ with $|Q| < t$ since the colluders (or, equivalently, an impersonating adversary who knows $\{x_i\}_{i\in Q}$) would still need to know x in order to adjust the δ_i's appropriately.

Corollary 3.33. (*Validity against sharing,* $|Q| \geq t$) *Let* $x \in \mathbb{Z}_q$, $\sigma = \{(u_i, s_i)\}_{i\in Q}$ *with* $|Q| = t$. *Given* $(x_1, \ldots, x_n; y) \in D_x$, *then* σ *is valid against* y *if and only if*

$$\sum_{i\in Q} \mu_i s_i = \sum_{i\in Q} \mu_i (r_i + c_i x_i), \tag{3.34}$$

where $r_i \equiv \log u_i$, $c_i \equiv H(u_i)$ *and* $\{\mu_i\}_{i\in Q}$ *are the weights of* σ.

Proof. The right-hand side of (3.31) vanishes exactly if $|Q| \geq t$. □

Remark 3.35. ($t-1$ *degrees of freedom*) Fix $\{u_i\}_{i\in Q}$ with $|Q| = t$. Choose $J \subset Q$ with $|J| = t-1$, pick arbitrary $\{\delta_i\}_{i\in J} \subset \mathbb{Z}_q$ and set

$$s_j \ = \ r_j + c_j x_j - \frac{1}{\mu_j} \sum_{i\in J} \mu_i (s_i - r_i - c_i x_i),$$

where $\{j\} = Q \setminus J$ and $r_i \equiv \log u_i$, $c_i \equiv H(u_i)$. By Corollary (3.33), this process exhausts the proofs that are based on $\{u_i\}_{i\in Q}$ and valid against y bijectively. This is a special case of Remark 3.29 with reference to a fixed sharing. Note however that it has no obvious utility for an adversary who knows $\{x_i\}_{i\in J}$ but not x_j.

Remark 3.36. (*Valid proofs with* $|Q| \geq t$) This can be seen from a more general perspective. Consider a canonical proof $\{(u_i, s_i)\}_{i\in Q}$ with $|Q| \geq t$ and let $\{\mu_i\}_{i\in Q}$ be its weights. Pick $j \in Q$ and arbitrary scalars δ_i, $i \in Q \setminus \{j\}$ such that at least one of them is non-zero. If we define

$$\delta_j = -\frac{1}{\mu_j} \sum_{i\neq j} \mu_i \delta_i,$$

then the proof $\{(u_i, s_i + \delta_i)\}_{i\in Q}$ satisfies condition (3.34) by design. Evidently, this process exhausts the valid proofs that are based on $\{u_i\}_{i\in Q}$.

3.2. **Anonymized protocol** – AnORST.

Protocol 3.37. (AnORST – *Anonymous One-Round Schnorr* (n, t)*-Threshold Identification*) Fix a DL-hard group $\mathbb{G} = \langle g \rangle$ of order q and a hash function $H : \mathbb{G} \to \mathbb{Z}_q$.

- *Setup.* Some uniformly random secret is distributed among a group of $n < q$ shareholders by means of Shamir (n, t)-sharing, $1 \leq t \leq n$, i.e.

$$(x_1, \ldots, x_n;\ y) \leftarrow D(x), \quad x \leftarrow_{\$} \mathbb{Z}_q \tag{3.38}$$

  Subsequently, the zero scalar is distributed among the participants by means of Shamir (n, t)-sharing, i.e.,

$$(\eta_1, \ldots, \eta_n;\ 1) \leftarrow D(0) \tag{3.39}$$

The threshold parameter and the public shares may remain private to the group, who needs only advertise the combined public key y.

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

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

to the verifier, $i \in Q$.

- *Verification.* Same as that of ORST.

Remark 3.40. (*Rotations and anonymity*) The mechanics of Shamir rotations underly the design of AnORST. Given a sharing $(x_1, \ldots, x_n\,;\,y)$, the group participants agree on a rotation $(\eta_1, \ldots, \eta_n\,;\,1)$ and subsequently generate canonical proofs with respect to the rotated sharing $(x_1+\eta_1, \ldots, x_n+\eta_n\,;\,y)$. By the correctness of ORST (cf. Remark 3.6), these proofs are ensured to be valid. Anonymity is because the shares $y_i = g^{x_i}$ cannot be feasibly inferred from the rotated shares $y_i^* = g^{x_i+\eta_i}$ that leak with each round. Regarding impersonation, Remark **??** indicates that the security of the anonymous protocol should reduce to that of ORST. We formally perform this reduction (cf. Proposition 4.7) using the indistinguishability statement presented above.

3.3. **Performance considerations.** Let $\mathbf{1} \in \mathbb{G}$ be the neutral group element and α be the bytelength of the group elements.

3.3.1. *Notation.* Given a proof $\{(u_i, s_i)\}_{i \in Q}$, $|Q| = m \geq 1$, we write $Q = \{i_1, \ldots, i_m\}$ for $i_1 < \cdots < i_m$.

Given $i \in Q \subset \{1, \ldots, n\}$, $n \geq 1$, we denote by

$$\lambda \leftarrow \mathsf{Lagrange}(i, Q)$$

the computation of the respective Lagrange interpolation coeffiecient for Q, i.e.,

$\lambda \leftarrow 1$
for $i \in Q$
  for $k \in Q \setminus \{i\}$
    $\lambda \leftarrow k \cdot (k - i)^{-1}$
return λ

It contains $m^2 - m$ multiplications and the same number of inversions.

3.3.2. *Bulk verification.* Assuming that no precomputed tables are available, we can accelerate weight computation using (3.12). The idea is to first compute the normalizer iteratively while inserting hashes at a lookup table for later usage.

$\bar{s} \leftarrow 0, \quad \bar{u} \leftarrow \mathbf{1}, \quad \bar{c} \leftarrow 1, \quad c \leftarrow \varnothing$
for $i \in Q$
  $c_i \leftarrow H(u_i)$
  $\bar{c} \leftarrow c_i \cdot \bar{c}$
  $c[i] \leftarrow c_i$
for $i \in Q$
  $\lambda_i \leftarrow \mathsf{Lagrange}(i, Q)$

 $c_i \leftarrow c[i]$
 $\mu_i \leftarrow \lambda_i \cdot \bar{c} \cdot c_i^{-1}$
 $\bar{s} \leftarrow \bar{s} + \mu_i \cdot s_i$
 $\bar{u} \leftarrow \bar{u} \cdot u_i^{\mu_i}$
$L \leftarrow g^{\bar{s}}$
$R \leftarrow y^{\bar{c}} \cdot \bar{u}$
return $L = R$

It includes $m^3 - m^2 + 5m + 1$ multiplications and $m^3 - m^2 + m$ inversions. In both summations, the $m^3 - m$ part is contributed by the computation of the interpolation coefficients.

3.3.3. *Incremental verification.*

$$Q^* = Q \cup \{j\}, \quad j \notin Q \tag{3.41}$$

$$\mu_i^* = \frac{j}{j - i} \cdot c_j \cdot \mu_i\,, \quad i \in Q^* \setminus \{j\} \tag{3.42}$$

$$\bar{c}^* = c_j \cdot \bar{c} \tag{3.43}$$

$$\mu_j^* = \lambda_j^* \cdot \bar{c} \tag{3.44}$$

$\alpha^* \leftarrow \varnothing, \quad \beta^* \leftarrow \varnothing$
$\bar{c}^* \leftarrow c_j \cdot \bar{c}$
$\lambda_j^* \leftarrow \mathsf{Lagrange}(j, Q^*)$
$\mu_j \leftarrow \lambda_j^* \cdot \bar{c}$
$\alpha_j^* \leftarrow \mu_j \cdot s_j$
$\beta_j^* \leftarrow u_j^{\mu_j}$
$\alpha^*[j] \leftarrow \alpha_j^*$
$\beta^*[j] \leftarrow \beta_j^*$
$\bar{s} \leftarrow \alpha_j^*$
$\bar{u} \leftarrow \beta_j^*$
for $i \in Q$
 $\gamma_i \leftarrow j \cdot (j - i)^{-1} \cdot c_j$
 $\alpha_i^* \leftarrow \gamma_i \cdot \alpha[i]$
 $\beta_i^* \leftarrow \beta[i]^{\gamma_i}$
 $\alpha^*[i] \leftarrow \alpha_i^*$
 $\beta^*[i] \leftarrow \beta_i^*$
 $\bar{s} \leftarrow \bar{s} + \alpha_i^*$
 $\bar{u} \leftarrow \bar{u} \cdot \beta_i^*$
$L \leftarrow g^{\bar{s}}$
$R \leftarrow y^{\bar{c}} \cdot \bar{u}$
if $L \neq R$
 return false

$Q \leftarrow Q^*, \quad \bar{c} \leftarrow \bar{c}^*, \quad \alpha \leftarrow \alpha^*, \quad \beta \leftarrow \beta^*$
return true

For $|Q| = m$, it includes $m^2 + 5m + 4$ multiplications and $m^2 + m$ inversions.
For $|Q^*| = m$, it includes $m^2 + 3m - 1$ multiplications and $m^2 - 1$ inversions.

3.3.4. *Optimized verification.* TODO

3.4. **Security considerations.** Since ORST decouples into concurrent runs of ordinary Schnorr identification, any security precaution for the latter applies also in the threshold case. Most notably, security against side-channel attacks cannot be attained if the random number generator leaks linear relations between the discrete logarithms of distinct commitments. We here focus on high-level attacks that arise particularly in the distributed setting.

3.4.1. *Generalized replay attack.* ORST is as much susceptible to replay attacks as the non-distributed Schnorr protocol. That is, a man-in-the-middle can passively capture and anytime replay the packets sent by a coalition of shareholders in order to impersonate them. Things seem worse because the attacker can transform the intercepted proof almost arbitrarily, provided that the commitments are kept fixed. Let $\{(u_i, s_i)\}_{i \in Q}$, $|Q| \geq t$ be a valid proof with weights $\{\mu_i\}_{i \in Q}$. For fixed $j \in Q$, denote $J = Q \setminus \{j\}$, pick $\{\delta_i\}_{i \in J} \subset \mathbb{Z}_q$ 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 bacause

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

(cf. Cor. 3.33). This essentially exploits the structure described in Remark 3.36. The security of the protocol should ensure that an attacker controlling up to $t-1$ shareholders remains unable to fabricate $\{(u_i, s_i^*)\}_{i \in Q}$ without having first intercepted $\{(u_i, s_i)\}_{i \in Q}$; still, in case of interception, the attacker has a least $t-1$ degrees of freedom in fabricating a valid proof based on the same commitments. This generalizes the replay attack of the non-distributed case $t = 1$, where the intercepted proof can only be replayed as is. It should be nontheless clear that generalized replay attacks are mitigated by the same method as in the non-distributed case, i.e., by baking a nonce into the hash function, capable of maintaining state between the verifier and the involved provers.

3.4.2. *Denial-of-service attack.* TODO

3.4.3. *Anonymizer leaks.* TODO

3.4.4. *Key-recovery attack – interactive aspect.* This practically affects only the interactive version of the protocol (cf. Remark 3.7) under rather special circumstances, but sheds light on its inherent security structure. Similar to the non-distributed case, we employ this attack in order to derive a formal security proof by means of fork and extraction (cf. Section 5.3). Evidently, if a shareholder uses the same commitment in two canonical rounds of the interactive ORST (due to, say, broken random generator) then its secret share leaks trivially. This is the key-recovery attack of the non-distributed case embedded into the threshold setting. We will show that this holds true even for non-canonical executions given a malicious verifier who controls $t-1$ shareholders.

Let $Q = J \cup \{j\}$ be a coalition of shareholders such that $|J| = t-1$, $j \notin J$ and the cluster J is controlled by an adversary $\mathcal{A}^*$. We further assume that $\mathcal{A}^*$ has compromised the identifying server. The shareholders engage in an identification session and generate a transcript $\{(u_i, c_i, s_i)\}_{i \in Q}$ that verifies, so that

$$\sum_{i \in Q} \mu_i (s_i - r_i - c_i x_i) = 0$$

holds true, where $r_i \equiv \log u_i$ and $\{\mu_i\}_{i \in Q}$ are the weights induced by the challenges $\{c_i\}_{i \in Q}$. (cf. Corollary 3.34). They next engage in a second session, where the j-th shareholder reuses the commitment u_j. After receiving u_j on the server's side, $\mathcal{A}^*$ responds with a challenge $c_j^* \neq c_j$ and tunes the rest shareholders to reuse their commitments u_i, $i \in J$ from the previous session. To each of them, the server responds with the same challenge $c_i^* = c_i$ from the previous session. Subsequently, all shareholders compute honestly their respective responses s_i, $i \in Q$ and send them to the server; since the transcript $\{(u_i, c_i^*, s_i^*)\}_{i \in Q}$ verifies, we again have

$$\sum_{i \in Q} \mu_i^* (s_i - r_i - c_i^* x_i) = 0\,,$$

where $\{\mu_i^*\}_{i \in Q}$ are the weights induced by the challenges $\{c_i^*\}_{i \in Q}$. Note that, since $c_i = c_i^*$ for all $i \neq j$, by definition of weights (cf. Definition 3.8) we get $\mu_j = \mu_j^*$. Substracting terms in the above relations and applying this remark, we get

$$\mu_j (s_j - s_j^* - (c_j - c_j^*) x_j) + \sum_{i \in J} (\mu_i (s_i - r_i - c_i x_i) - \mu_i^* (s_i^* - r_i - c_i x_i)) = 0\,.$$

This eliminates r_j, which is the only unknown to the adversary. Since $c_j - c_j^* \neq 0$, $\mathcal{A}^*$ can use this relation to retrieve x_j in the form

$$x_j = \frac{s_j - s_j^*}{c_j - c_j^*} + \frac{1}{\mu_j} \sum_{i \in J} \Big(\mu_i (s_i - r_i - c_i x_i) - \mu_i^* (s_i^* - r_i - c_i x_i) \Big). \tag{3.45}$$

Note finally that since $|Q| = t$, the adversary can use the shares $\{x_i\}_{i \in Q}$ to compute the combined secret key by means of Shamir's reconstruction formula.

4. Security notions

In this section we specify the threat model for the protocols of Section 3. Formulations are given in terms of attack games within the asymptotic paradigm and negligibility is meant with respect to the bitlength of the group order. TODO The relevant formalism for discrete-log based settings is developed in Section B.

4.1. **Threat model.** The current standard for identification protocols is security against active impersonation attacks. This refers to adversaries who not only passively intercept 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 adversaries who only intercept identification sessions and potentially exert some control on the network. Once the relation between threshold and ordinary Schnorr indentification has been clarified from the provability viewpoint, security against active adversaries should be straightforward to establish using techniques similar to those of [9] due to the decoupled structure of the distributed protocol.

4.1.1. *General threat model.* In most general terms, the ORST threat model consists of an adversary $\mathcal{A}$ who compromises the keys of up to $t-1$ shareholders and attempts to impersonate some coalition $Q \subset \{1, \ldots, n\}$, i.e., fabricate a proof of the form $\{(u_i, s_i)\}_{i \in Q}$ that verifies. It does not make sense to consider adaptive adversaries because the proving phase consists of decoupled packets sent to the verifier; since no ephemeral quantities are exchanged between the shareholders, $\mathcal{A}$ has no chance to collect potentially useful information and decide which of them to corrupt adaptively. We can thus confine ourselves to the static model, assuming that the attacker decides from the outset the set of corrupted parties.[3] One could further assume that $\mathcal{A}$ knows the public keys of individual shareholders; however, these are not advertised or used anyhow, although trivially inferred from protocol transcripts. Letting $\mathcal{A}$ know them from the outset corresponds to a scenario where the public shares become advertised within the group for at least some initial timespan (e.g., for recognition and verification purposes) and $\mathcal{A}$ compromises the device of a shareholder who has not yet erased them. Still, the security proof presented in Section 5.4 remains intact under this stronger assumption (cf. Remark TODO), indicating that an adversary who controls devices in the erasure-free model has no greater advantage than an adversary who only steals keys.

Let $(x_1, \ldots, x_n;\, y)$ be the given (n,t)-sharing of some secret x and $\mathcal{A}$ be an adversary who knows the shares $\{x_i\}_{i \in J}$ with $|J| = t-1$. Let $Q \subset \{1, \ldots, n\}$ be the sharholders that fall victim to impersonation under a purportedly valid proof $\{(u_i, s_i)\}_{i \in Q}$ fabricated by $\mathcal{A}$. Without loss of generality, $Q \subset J$ or $J \subset Q$. Indeed, since the black-box adversary knows nothing special about the uncorrupted shareholders, the choice of Q is irrelevant (except for its size) in the extent to which Q intersects with $\{1, \ldots, n\} \setminus J$; the adversary can change Q without this affecting its strategy, provided that $|Q|$ is preserved and its original intersection with J remains included. We conclude with the following attack model.

- *Corruption and attack statement*:
  - $\mathcal{A}$ chooses $J \subset \{1, \ldots, n\}$ with $|J| = t-1$.
  - $\mathcal{A}$ chooses $Q \subset \{1, \ldots, n\}$ with $Q \subset J$ or $J \subset Q$ and sends it along with J to its challenger $\mathcal{C}$.
  - $\mathcal{C}$ runs $(x_1, \ldots, x_n;\, y) \leftarrow D(x),\ x \leftarrow_{\$} \mathbb{Z}_q$ and sends $y, \{x_i\}_{i \in J}$ to $\mathcal{A}$.

[3]Adaptive corruption during the execution of an equivalent DKG protocol in place of Shamir's sharing falls under the static model. TODO

- *Impersonation*: $\mathcal{A}$ outputs a proof of the form $\sigma = \{(u_i, s_i)\}_{i \in Q}$.

$\mathcal{A}$ wins if σ is valid against y.

Note that successful impersonation is equivalent to condition

$$\frac{1}{\bar{c}} \sum_{i \in Q} \mu_i \left(s_i - r_i\right) = x\,, \tag{4.1}$$

where $r_i \equiv \log u_i$, $i \in Q$ and $\{\mu_i\}_{i \in Q}$, $\bar{c}$ are the weights and the normalizer of the output proof σ (cf. Proposition 3.22).

ORST security against impersonation attacks should mean that every polynomial-time adversary wins the above game with at most negligible chances. The pragmatic requirement behind is that, if $\mathcal{A}$ outputs a proof $\{(u_i, s_i)\}_{i \in Q}$ that verifies, then it should necessarily know x_j for some $j \notin J$ (or, equivalently, x). In other words, impersonation should be infeasible without knowledge of at least t secret shares. We will informally refer to this design requirement as the *knowledge property.*

This is instance of a more general pattern. Security notions for Schnorr-based protocols are naturally intermediated by some contextual knowledge property like above, which is the pragmatic security requirement per se. Consequently, a security reduction should –most probably– proceed by recovery of some contextual key in case of successful attack; by *knowing* this key, the adversary solves the DL or related hard problem, meaning that it cannot have succeded (with non-negligible chances) in its attack. Key-recovery proceeds as follows. By rewinding the successful adversary at some special point and let it repeat successfully its attack, the rewinder *extracts* the key from the partially differing transcripts of the two successful attacks. Security reduces to the hardness assumption only because extraction yields the solution to the underlying hard problem under any circumstances that allow rewinding; if extraction is not possible under certain circumstances, security is –most probably– unprovable without further assumptions. It should be stressed that extractability is only a way to formalize the knowledge property in a security reduction argument; in particular, the knowledge property can hold true and pragmatically guaruantee security without security being provable under the fixed hardness assumption.

Regarding ORST, the relevant extraction formula for x_j is evidently equation (3.45) from the key-recovery attack of Section 3.4.4. We will see (cf. Section 5.3) that this formula is insufficient under certain circumstances, indicating that ORST security is –most probably– unprovable under DL hardness alone. We would need extra assumptions in order for security to become provable or, what is equivalent, reduce it to an easier problem than DL.

4.1.2. *Extractability and* $(t-1)$-OMDL *hardness.* Consider the following situation with $Q = J \cup \{j\}$, $|J| = t - 1$. The shareholders $i \in J$ engage in a session with a verifier and an adversary $\mathcal{A}$ who knows $\{x_i\}_{i \in J}$ intercepts their communication; in particular, $\mathcal{A}$ captures the transcript u_i, c_i, s_i, $i \in J$. Subsequently, $\mathcal{A}$ fabricates a commitment u_j according to some-black-box strategy and sends it to the verifier, who responds with a challenge c_j. Finally, $\mathcal{A}$ fabricates a response s_j according to some black-box strategy and sends it to the verifier. If the protocol is secure, the transcript u_i, c_i, s_i, $i \in Q$ should verify with at most negligible chances, even though $\mathcal{A}$ knows $x_i, u_i, c_i, s_i,\ i \in J$ while fabricating u_j and s_j. Pragmatically, if the transcript verifies, then $\mathcal{A}$ must necessarily know x_j.

Extraction is here concretely isntantiated as follows. Let $\mathcal{A}^*$ be an algorithm with rewindable black-box access to $\mathcal{A}$. If $\mathcal{A}$ fabricates $u_j,\ s_j$ such that the transcript $u_i, c_i, s_i,\ i \in Q$ verifies with probability ε, $\mathcal{A}^*$ rewinds $\mathcal{A}$ exactly before sending u_j and lets it complete a second impersonation attempt. By the forking lemma (cf. [12], or the reset lemma [9]), $\mathcal{A}$ succeeds again with probability $\varepsilon^* \approx \varepsilon^2$. Let $u_i^*, c_i^*, s_i^*,\ i \in Q$ be the trancript of the second impesonation attack; since rewinding occurs at the last commitment, we have

$$(u_i^* = u_i \ \wedge \ c_i^* = c_i \ \wedge \ s_i^* = s_i)\,\forall i \in J \ \wedge \ u_j^* = u_j \ \wedge \ c_j^* \neq c_j \tag{4.2}$$

where the last condition holds with overwhelming probability true. Extractability means that $\mathcal{A}^*$ *should* be able to recover x_j from $x_i, u_i, c_i, s_i,\ i \in J$ and u_j, c_j^*, s_j^*. We contend that this is impossible *without* further assumptions.

Condition (4.2) allows us to apply the computations of Section 3.4.4 in the present context, meaning that the relevant extraction formula is the same as (3.45). In particular, x_j is extractable in the form

$$x_j \ = \ \frac{s_j - s_j^*}{c_j - c_j^*} \ + \ \frac{1}{\mu_j} \sum_{i \in J} (\mu_i - \mu_i^*)(s_i - r_i - c_i x_i)\,, \quad r_i \equiv \log u_i\,, \tag{4.3}$$

where we have further applied the fact $s_i^* = s_i, \forall i \in J$. However, $\mathcal{A}^*$ cannot know the logarithms $r_i,\ i \in J$ of the intercepted commitments and consequently cannot use this formula to recover x_j. Extraction is not possible in the present context.

Since we have found circumstances where extraction cannot complete without further assumptions, security is –most probably– unprovable without further assumptions as well. In particular, ORST security against impersonation attacks is –most probably– *not* reducible to discrete-logarithm (DL) hardness alone. However, $\mathcal{A}^*$ would indeed be able to extract x_j if it could somehow see the logarithms of the intercepted commitments, e.g., if it controlled the respective shareholders' devices, or could at least subvert their random number generator or other security sensitive module. We can codify such scenarios under the assumption that $\mathcal{A}^*$ is allowed to issue at most $t-1$ queries to a hypothetical discrete-logarithm oracle; extraction of x_j by means of (4.3) would then become automatically possible, suggesting that security against impersonation attacks might as well be reducible to the $(t-1)$-OMDL hardness assumption (cf. Section 4.3). Note that $(t-1)$-OMDL is indeed an easier problem than DL, potentially yielding the extra assumptions needed for security provability.

4.1.3. *Implications of* $(t-1)$-OMDL *hardness.* The black-box pattern of the previous section is rather restricted as compared to the general attack model for ORST (cf. Section 4.1.1). This is because $\mathcal{A}$ does not influence the compromised shareholders, mounting its actual attack only after collecting their transcripts; in particular, the fabrication of u_j and s_j is deferred until after $u_i, s_i,\ i \in J$ have resolved, in order for extractability failure to be demonstrated in the simplest possible way (i.e., rewinding instead of some more complicated forking). At the same moment, the unprovability argument indicates in which way security becomes provable under the stronger $(t-1)$-OMDL hardness assumption. Specifically, in order to be able to apply extraction, it seems that $\mathcal{A}^*$ should at least reproduce the conditions of Remark

3.4.4 and let $\mathcal{A}$ operate in that context. While this remains more restricted than the general attack model of Section 4.1.1, it generalizes by far the restricted attack pattern of Section 5.3, allowing $\mathcal{A}$ to fabricate u_j before the rest shareholders send their commitments and moreover control their responses. We propose the following preliminary attack model for $\mathcal{A}$, representing a tradeoff between the general attack model and security provability under the $(t-1)$-OMDL hardness assumption.

- *Corruption and attack statement*:
  - $\mathcal{A}$ chooses $J \subset \{1, \dots, n\}$ with $|J| = t - 1$ and sends it along with some $Q \subset \{1, \dots, n\}$ to its challenger $\mathcal{C}$.
  - $\mathcal{A}$ chooses $Q \subset \{1, \dots, n\}$ with $Q \subset J$ or $J \subset Q$ and sends it along with J to its challenger $\mathcal{C}$.
  - $\mathcal{C}$ runs $(x_1, \dots, x_n; y) \leftarrow D(x),\ x \leftarrow_{\$} \mathbb{Z}_q$ and sends $y, \{x_i\}_{i \in J}$ to $\mathcal{A}$.
- *Interception*: $\mathcal{C}$ samples $u_i \leftarrow_{\$} \mathbb{G},\ i \in J$ and sends $\{u_i\}_{i \in J}$ to $\mathcal{A}$.
- *Impersonation*: $\mathcal{A}$ outputs a proof of the form $\{(u_i, s_i)\}_{i \in Q}$.

$\mathcal{A}$ wins if $\{(u_i, s_i)\}_{i \in Q}$ verifies against y.

ORST is meant to be secure against impersonation attacks if every polynomial-time adversary wins this game with at most negligible advantage.

Observe that no restrictions are yet imposed on $|Q|$, allowing the possibility that $\mathcal{A}$ fabricates a valid proof with less than t components. This is plausible because proofs of this kind are known to exist in abundance (cf. Remarks 3.29 and 3.32). However, under the $(t-1)$-OMDL hardness assumption, fabricating such proofs can be ruled out as infeasible. Let $\mathcal{A}$ be a polynomial-time adversary that wins with $Q \subset J$ (i.e., $|Q| < t$). Let $J = \{i_1, \dots i_{t-1}\}$ with $i_1 < \cdots < i_{t-1}$, so that $Q = \{i_1, \dots, i_m\}$ for some $m \in \{1, \dots, t-1\}$. We will construct a $(t-1)$-OMDL adversary $\mathcal{A}^*$ as a wrapper of $\mathcal{A}$ with equal time complexity and advantage. When given $w_0, \dots, w_{t-1}$ by its challenger (cf. Game 4.12), $\mathcal{A}^*$ sets $y \leftarrow w_0$ and $u_{i_k} \leftarrow w_k,\ 1 \le k \le t-1$. Next, $\mathcal{A}^*$ samples $x_i \leftarrow_{\$} \mathbb{Z}_q,\ i \in J$ and forwards $y,\ \{x_i\}_{i \in J}$ to $\mathcal{A}$. By the security property of Shamir's sharing (cf. Remark A.7), y and $\{x_i\}_{i \in J}$ determine some (n, t)-sharing $x_1, \dots, x_n$ of $x \equiv \log y$ indistinguishably, meaning that $\mathcal{A}^*$ simulates perfectly the corruption phase of the game. Subsequently, $\mathcal{A}^*$ sends $\{u_i\}_{i \in Q}$ to $\mathcal{A}$ and lets it proceed to impersonation; since its challenger has been perfectly simulated, $\mathcal{A}$ outputs a valid proof $\{(u_i, s_i)\}_{i \in Q}$ with probability equal to its advantage. Condition (4.1) is then satisfied and $\mathcal{A}^*$ can recover x by issuing at most $t-1$ discrete-logarithm queries. More accurately, $\mathcal{A}^*$ queries

$$r_{i_k} \leftarrow \mathcal{O}^{\mathsf{dlog}}_{\mathbb{G}}(u_{i_k}), \quad 1 \le k \le t-1$$

and uses $r_{i_k},\ 1 \le k \le m$ to compute x. Finally, $\mathcal{A}^*$ sets $z_0 \leftarrow x$, and $z_k \leftarrow r_{i_k}$ for $1 \le k \le t-1$, and wins by forwarding $z_0, \dots, z_{t-1}$ to its challenger. In short, if $\mathcal{A}$ succeeds with non-negligible probability, then $\mathcal{A}^*$ solves the $(t-1)$-OMDL problem with equal advantage, which contradicts the hardness assumption.

This straight-line reduction shows that we can restrict attention to the case $J \subsetneq Q$ (i.e. $|Q| \ge t$), excluding the possibility of impersonating less than t shareholders (without knowledge of at least t secret shares). This is evidently a security requirement for the ORST protocol, captured (as should) by the $(t-1)$-OMDL hardness assumption. On the other hand, TODO

4.2. **Security against impersonation attacks.** TODO

4.2.1. ORST – *Basic protocol.* TODO We conclude with the following model for impersonation attacks against the ORST protocol. Given a cyclic group $\mathbb{G} = \langle g \rangle$ of order q and integers $1 \leq t \leq n < q$, we define the impersonation attack against ORST to be the following interactive game.

Game 4.4. ($\mathsf{ORST}_{\mathbb{G},n,t}$ – *impersonation*) Given adversary $\mathcal{A}$ and challenger $\mathcal{C}$,

- *Corruption and attack statement*:
  - $\mathcal{A}$ chooses $J \subset \{1, \ldots, n\}$ with $|J| = t - 1$ and sends it along with some $j \in \{1, \ldots, n\} \setminus J$ to the challenger. Denote $Q = J \cup \{j\}$.
  - $\mathcal{C}$ runs $(x_1, \ldots, x_n; y) \leftarrow D(x),\ x \leftarrow_\$ \mathbb{Z}_q$ and sends $y, \{x_i\}_{i \in J}$ to $\mathcal{A}$.
- *Interception*: $\mathcal{C}$ samples $u_i \leftarrow_\$ \mathbb{G},\ i \in J$ and sends $\{u_i\}_{i \in J}$ to $\mathcal{A}$.
- *Impersonation*: $\mathcal{A}$ outputs a proof of the form $\sigma = \{(u_i, s_i)\}_{i \in Q}$.

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

The probability that $\mathcal{A}$ wins Game 4.4 is denoted by

$$\mathsf{Adv}^{\mathsf{ORST}}_{\mathbb{G},n,t}\,[\mathcal{A}]. \tag{4.5}$$

ORST *security in* $\mathbb{G}$ is the assumption that (4.13) is negligible for every choice of the parameters n, t and every polynomial-time adversary $\mathcal{A}$.

By the indistinguishability property of Shamir's sharing (cf. Remark A.7), the parameters j and J (and consequently Q) may be regarded as fixed for $\mathcal{A}$. This yields a non-interactive algorithm of the form

$$\{(u_i, s_i)\}_{i \in Q} \leftarrow \bar{\mathcal{A}}(y, \{x_i\}_{i \in J}, \{u_i\}_{i \in J}) \tag{4.6}$$

that consumes $\mathcal{A}$'s view during the impersonation phase in order to fabricate a commitment u_j and responses $\{s_i\}_{i \in Q}$ such that the output $\{(u_i, s_i)\}_{i \in Q}$ verifies. It should be stressed that, when using $\mathcal{A}$ in a black-box reduction, the logarithm of the fabricated commitment u_j cannot be assumed to be known and will normally have to be eliminated from our computations.

4.2.2. AnORST – *Anonymized protocol.* Aside from privacy considerations (cf. Section TODO), the threat model for the anonymized protocol is the same except that the adversary compromises also the blinding factors of the corrupted shareholders. Given a cyclic group $\mathbb{G} = \langle g \rangle$ of order q and integers $1 \leq t \leq n < q$, we define the impersonation attack against AnORST to be the following interactive game.

Game 4.7. ($\mathsf{AnORST}_{\mathbb{G},n,t}$ – *impersonation*) Given adversary $\mathcal{A}$ and challenger $\mathcal{C}$,

- *Corruption*:
  - $\mathcal{A}$ chooses $J \subset \{1, \ldots, n\}$ with $|J| = t - 1$ and sends it along with some $j \in \{1, \ldots, n\} \setminus J$ to the challenger. Denote $Q = J \cup \{j\}$.
  - $\mathcal{C}$ runs $(x_1, \ldots, x_n; y) \leftarrow D(x),\ x \leftarrow_\$ \mathbb{Z}_q$ and sends $y, \{x_i\}_{i \in J}$ to $\mathcal{A}$.
  - $\mathcal{C}$ runs $(\eta_1, \ldots, \eta_n; 1) \leftarrow D(0)$ and sends $\{\eta_i\}_{i \in J}$ to $\mathcal{A}$.
- *Interception*: $\mathcal{C}$ samples $u_i \leftarrow_\$ \mathbb{G},\ i \in J$ and sends $\{u_i\}_{i \in J}$ to $\mathcal{A}$.
- *Impersonation*: $\mathcal{A}$ outputs a proof of the form $\sigma = \{(u_i, s_i)\}_{i \in Q}$.

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

The probability that $\mathcal{A}$ wins game (4.7) is denoted by

$$\mathsf{Adv}^{\mathsf{AnORST}}_{\mathbb{G},n,t}[\mathcal{A}]. \tag{4.8}$$

AnORST *security in* $\mathbb{G}$ is the assumption that (4.13) is negligible for every choice of the parameters n, t and every polynomial-time adversary $\mathcal{A}$.

It should be clear that an adversary of Game 4.7 can be easily transformed to an adversary of Game 4.4 with equal time complexity and advantage. The formal straight-line reduction requires the following claim, essentially asserting that the security of Sharmir's sharing is preserved under the action of blinding factors.

Lemma 4.9. *The probability distribution*

$$\{(x_1 + \eta_1, \ldots, x_n + \eta_n\,; y) : \begin{Bmatrix} (\eta_1, \ldots, \eta_n; 1) \leftarrow D(0), (x_1, \ldots, x_n; y) \leftarrow D(x) \\ x \leftarrow_\$ \mathbb{Z}_q \end{Bmatrix}\}$$

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

Proof. $\{\alpha + \beta : \alpha, \beta \leftarrow_\$ \mathbb{Z}_q\}$ is identical to the uniform distribution over $\mathbb{Z}_q$. The claim follows from the homomorphic property (cf. Remark A.2), applying this fact to the coefficients of the interpolating polynomials (cf. Definition A.1). □

Proposition 4.10. *If* ORST *is secure, then* AnORST *is secure. In particular, given a cylic group* $\mathbb{G} = \langle g \rangle$ *of order* q *and integers* $1 \le t \le n < q$*, for every polynomial-time adversary* $\mathcal{A}$ *attacking* $\mathsf{ORST}_{\mathbb{G},n,t}$ *there exists a polynomial-time adversary* $\mathcal{A}^*$ *attacking* $\mathsf{AnORST}_{\mathbb{G},n,t}$ *such that*

$$\mathsf{Adv}^{\mathsf{AnORST}}_{\mathbb{G},n,t}[\mathcal{A}^*] = \mathsf{Adv}^{\mathsf{ORST}}_{\mathbb{G},n,t}[\mathcal{A}] \tag{4.11}$$

Proof. We construct $\mathcal{A}^*$ as a wrapper of $\mathcal{A}$ that simulates the latter's challenger. When given j and J by $\mathcal{A}$ (step 1, Game 4.4), $\mathcal{A}^*$ forwards them to its own challenger and receives $y, \{x_i\}_{i \in J}, \{\eta_i\}_{i \in J}$ (steps 1-3, Game 4.7). Next, $\mathcal{A}^*$ sends

$$y,\ \{x_i + \eta_i\}_{i \in J}$$

to $\mathcal{A}$. By Lemma 4.9, $\mathcal{A}^*$ perfectly simulates the corruption phase of Game 4.4, so that $\mathcal{A}$ outputs a valid proof σ with probability equal to its advantage. $\mathcal{A}^*$ wins by forwarding σ to its own challenger. □

4.3. **One-more discrete-logarithm (OMDL) hardness.** The OMDL family is a parametrized extension of the discrete-logarithm (DL) problem, yielding a sequence of increasingly easier related problems. It was introduced in [10] as a way to capture protocol design properties that pragmatically guarantee security under the DL hardness assumption while probably being themselves irreducible to the latter. Note that reducing security to an easier problem, i.e., a stronger hardness assumption, leads usually to assumptions regarding the adversarial strategy, i.e., narrowing attention to a more restricted class of attacks. Under these assumptions, a formal security proof may become possible, while a proof under the broader hardness assumption may not exist; since the solvability of the harder problem implies solvability of the easier one, the restricted security proof provides evidence that security holds true under the broader hardness assumption. This is namely a tradeoff between the plausibility of the broader hardness assumption and having a concrete security proof, which may practically be impossible under the former. The tradeoff is ofter made in the context of secure multi-party protocols, where the adversary has extended control on its execution environment through the corrupted shareholders.

By *discrete-logarithm oracle* for a cyclic group $\mathbb{G} = \langle g \rangle$ with order q is meant a procedure of the form $z \leftarrow \mathcal{O}^{\mathsf{dlog}}_{\mathbb{G}}(u)$, $u \in \mathbb{G}$, such that $u = g^z$. Given $\mathcal{O}^{\mathsf{dlog}}_{\mathbb{G}}$ and integer $t \geq 1$, the one-more discrete-logarithm game is defined as follows.

Game 4.12. ($\mathsf{OMDL}_{\mathbb{G},\,t-1}$ *problem*) Given an adversary $\mathcal{A}$ with challenger $\mathcal{C}$,

- $\mathcal{C}$ samples $z_i \leftarrow_{\$} \mathbb{Z}_q,\ 0 \leq i \leq t-1$
- $\mathcal{C}$ computes $w_i \leftarrow g^{z_i},\ 0 \leq i \leq t-1$ and sends $w_0, w_1, \dots, w_{t-1}$ to $\mathcal{A}$
- $\mathcal{A}$ outputs $z^*_0, z^*_1, \dots, z^*_{t-1}$

$\mathcal{A}$ wins if $\bigwedge_{i=0}^{t-1} z_i = z^*_i$ and it has issued at most $t-1$ queries to $\mathcal{O}^{\mathsf{dlog}}_{\mathbb{G}}$.

The probability that $\mathcal{A}$ wins game $\mathsf{OMDL}_{\mathbb{G},\,t-1}$ is denoted by

$$\mathsf{Adv}^{\mathsf{OMDL}}_{\mathbb{G},\,t-1}[\mathcal{A}]. \tag{4.13}$$

$(t-1)$-OMDL *hardness in* $\mathbb{G}$ is the assumption that (4.13) is negligible for every polynomial-time $\mathcal{A}$. Evidently, 0-OMDL is equivalent to DL.

If an adversary solves $\mathsf{ORST}_{\mathbb{G},\,t-1}$ for $t < t^*$, then it trivially solves $\mathsf{ORST}_{\mathbb{G},\,t^*-1}$, that is, (t^*-1)-ODML is a stronger assumption than $(t-1)$-OMDL. In particular, $(t-1)$-OMDL, $t > 1$ is a stronger assumption than DL.

5. Security proof

TODO [rephrasings, references] In this section we prove the security of ORST and AnORST against impersonation attacks (cf. Section 4) under one-more discrete-logarithm hardness (OMDL) in the random oracle model (ROM). Recall that security is defined against adversaries who compromise the keys of up to $t-1$ shareholders and try to impersonate any t parties including them after intercepting the commitments sent by the compromised parties (cf. Game 4.4). By Proposition 4.10, it suffices to prove the security of ORST, i.e., that every polynomial-time adversary $\mathcal{A}$ wins $\mathsf{ORST}_{n,t}(\kappa)$ (cf. Game 4.4) with only negligible advantage against the security parameter κ. We do so by reducing the hardness of the concrete game $\mathsf{ORST}_{\mathbb{G},n,t}$ to the hardness of $\mathsf{OMDL}_{\mathbb{G},t-1}$ (cf. Section 4.3).

The reduction generalizes the standard rewinding argument of the non-distributed case into the general threshold setting in order to extract the unknown share x_j. This is a non-trivial generalization. As it turns out (cf. Section 5.3), the extractor comes with a residue, essentially stemming from the $t-1$ degrees of freedom in perturbing a valid proof with fixed commitments (cf. Remark (**??**)); it becomes useful only if the logarithms of the intercepted commitments are known to the adversary, which is the reason for reducing to $(t-1)$-OMDL hardness. Moreover, extraction presupposes the fixture of the challenges $c_i,\ i \neq j$, which makes the rewinding argument non-applicable in the general threshold setting. We bypass this obstcle by utilizing the Local Forking Lemma (cf. Section A.0.2) and ensuring that the conditions for its applicability are satisfied. The attack simulator is presented in Section 5.2; it becomes useful when combined with the lemma presented in Section 5.1.2, used to embed game $\mathsf{OMDL}_{\mathbb{G},t-1}$ into the attack simulator.

5.1. **Preliminaries.** We present in detail the elementary components of the attack simulator (cf. Section 5.2) and the final reduction argument (cf. Section 5.4).

5.1.1. *Index mapping formalism.* Given $J \subset \{1,\dots,n\}$ with $|J| = m \geq 1$, we can represent any collection $\alpha_1,\dots,\alpha_m$ in the form $\{\beta\}_{i\in J}$ by applying the Zip operator shown in Figure 1. It should be clear that Zip preserves the uniform sampling. We state this remark formally for the sake of reference.

Lemma 5.1. Given $J \subset \{1,\dots,n\}$ with $|J| = m \geq 1$, the probability distribution

$$\{\,\mathsf{Zip}(\alpha_1,\dots,\alpha_m;\,J\,) : \alpha_i \leftarrow_\$ A,\ i = 1,\dots,m\}$$

is identical to $\{\{\beta_i\}_{i\in J} : \beta_i \leftarrow_\$ A,\ i \in J\}$.

Proof. Follows directly from the definition of Zip. $\square$

Algorithm 1 $\mathsf{Zip}(\alpha_1,\dots,\alpha_m;\,J\,)$

$\{i_1,\dots,i_m\} \leftarrow J$ with $i_1 < \dots < i_m$
for $k = 1,\dots,m$
  $\beta_{i_k} \leftarrow \alpha_k$
return $\{\beta_i\}_{i\in J}$

Algorithm 2 $\mathsf{Zip}^{-1}(\{\beta_i\}_{i\in J})$

$\{i_1,\dots,i_m\} \leftarrow J$ with $i_1 < \dots < i_m$
for $k = 1,\dots,m$
  $\alpha_k \leftarrow \beta_{i_k}$
return $(\alpha_1,\dots,\alpha_m)$

FIGURE 1. Zip operator and its inverse

5.1.2. *Shamir sharing simulation.* Given $t-1$ predefined shares $\{x_i\}_{i\in J} \subset \mathbb{Z}_q$ and $j \in \{1,\dots,n\} \setminus J$, we can embed any $y^* \in \mathbb{G}$ as the j-th public share of a (n,t)-sharing interpolating them. Specifically, we define

$$(\{x_i\}_{i\in J};\,y\,) \leftarrow \mathcal{D}(\,y^*,j\,,\{x_i\}_{i\in J}\,) \tag{5.2}$$

as deterministic procedure shown in Figure 2.

Lemma 5.3. *Let $J \subset \{1,\dots,n\}$ with $|J| = t-1$ and $j \notin J$. Given* (5.2),

$$y^* = g^{x_j}$$

where $(x_1,\dots,x_n;y)$ is the (n,t)-sharing uniquely determined by $\{x_i\}_{i\in J}$, y. Moreover, the probability distribution

$$\{\,(\{x_i\}_{i\in J};\,y\,) : \begin{Bmatrix} (\{x_i\}_{i\in J};\,y) \leftarrow \mathcal{D}(\,y^*,j\,,\{x_i\}_{i\in J}) \\ y^* \leftarrow_\$ \mathbb{G},\quad x_i \leftarrow_\$ \mathbb{Z}_q,\ i \in J \end{Bmatrix}\}$$

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

Proof. By the elementary properties of interpolation, $\{x_i\}_{i\in J}$ and y^* uniquely determine a sharing $(x_1,\ldots,x_n;y)$ such that $y^*=g^{x_j}$, with the outputs of $\mathcal{D}$ mapping to those of D bijectively. Equality of distributions follows from the security of the Shamir's sharing (cf. Remark A.7). $\square$

Algorithm 3 $\mathcal{D}(y^*,j,\{x_i\}_{i\in J})$

1: Set $y_j \leftarrow y^*$
2: Set $y_i \leftarrow g^{x_i},\ i\in J$
3: $Q\leftarrow J\cup\{j\}$
4: $y\leftarrow \prod_{i\in Q} y_i^{\lambda_i}$
5: **return** $\{x_i\}_{i\in J}, y$

FIGURE 2. Shamir sharing simulation

5.2. **Simulated impersonation attack.** In this section, we define an algorithm that simulates Game 4.4 if the hash function H is modelled as a random oracle. Roughly speaking, it accepts as input the view of the adversary and outputs the fabricated proof along with the hashes of the commitments and an index indicating successful or failed impersonation. The simulation should be perfect in the sense that, if the algorithm runs with uniformly random input, the probability of success should equal the adversarial advantage of winning the real attack game.

Consider an adversary $\mathcal{A}$ that chooses $J\subset\{1,\ldots,n\}$ with $|J|=t-1$ and $j\in\{1,\ldots,n\}\setminus J$ in the corruption phase of Game 4.4. The desired simulation will be designed as an execution environment for an embedded replica of the adversarial component $\bar{\mathcal{A}}$, namely an algorithm of the form

$$(k,\{(u_i,c_i,s_i)\}_{i\in Q})\leftarrow \mathcal{S}[\mathcal{A}](y^*,\{x_i\}_{i\in J},\{u_i\}_{i\in J};h_1,\ldots,h_m)$$

where $Q\equiv J\cup\{j\}$. The pair $y^*,\{x_i\}_{i\in J}$ will uniquely determine the initial (n,t)-sharing with $\{x_i\}_{i\in J}$ utilized as the "corrupted" secret shares. Similarly, the collection $\{u_i\}_{i\in J}$ will be utilized as the "intercepted" commitments. These will be consumed by the embedded instance of the adversary with the difference that, instead of invoking directly the random oracle, $\bar{\mathcal{A}}$ queries an embedded hash-oracle $\mathcal{O}^{\mathsf{hash}}$ that simulates H (cf. Section A.0.1). The values $h_1,\ldots,h_m$ will be the distinct "hashes" returned by the oracle in respective order. If needed, $\mathcal{A}$ may be re-programmed without loss of generality to query hashes for all the commitments it sees (including the fabricated commitment u_j), in which case we may also need to increase m. Finally, an ORST-verifier $\mathcal{V}$ will be included with access to the same hash-oracle for the purpose of verifying the outputs of the embedded adversary. The algorithm is formally defined in Figure 3.

Algorithm 4 $\mathcal{S}[\mathcal{A}](y^*, \{x_i\}_{i \in J}, \{u_i\}_{i \in J};\, h_1, \dots, h_m)$

$h \leftarrow [h_1, \dots, h_m],\ \nu \leftarrow 1,\ \mathsf{Map}_1 = \varnothing,\ \mathsf{Map}_2 = \varnothing$ // *Setup* $\mathcal{O}^{\mathsf{hash}}$

$(\{x_i\}_{i \in J}, y) \leftarrow \mathcal{D}(y^*, \{x_i\}_{i \in J})$ // *Sharing simulation*

$\{(u_i, s_i)\}_{i \in Q} \leftarrow \bar{\mathcal{A}}(y, \{x_i\}_{i \in J}, \{u_i\}_{i \in J})$ // *Impersonation attempt*

$res \leftarrow \mathcal{V}(y, \{(u_i, s_i)\}_{i \in Q})$

for $i \in Q$

  $c_i \leftarrow \mathsf{Map}_1[u_i]$

$k \leftarrow \mathsf{Map}_2[u_j]$

$\sigma \leftarrow \{(u_i, c_i, s_i)\}_{i \in Q}$

if $res = \mathsf{true}$

  return (k, σ)

return $(0, \sigma)$

FIGURE 3. Simulated impersonation attack, $\mathcal{S}[\mathcal{A}]$

The hash-oracle is first initialized with $h_1, \dots, h_m$ allocated on its hash-tape and its lookup-tables set to empty (cf. Section A.0.1). Afterwards, the virtual challenger generates the unique sharing $x_j = \log y^*$ with unknown secret share x_j (cf. Section 5.1.2). The shares $\{x_i\}_{i \in J}$, the combined public key y and the commitments $\{u_i\}_{i \in J}$ are then fed to component $\bar{\mathcal{A}}$ of $\mathcal{A}$ (cf. Section B.1), which fabricates a proof of the form $\{(u_i, s_i)\}_{i \in Q}$ with $Q = J \cup \{j\}$. Note that the distinct oracle queries are all issued during the latter impersonation attempt, including the "challenges" $c_i \leftarrow \mathcal{O}^{\mathsf{hash}}(u_i)$, $i \in Q$ which are collected and attached to the fabricated proof for later usage. We moreover locate the unique index $k \in \{1, \dots, m\}$ for which $c_j = h[k]$, indicating that the hash of the fabricated commitment u_j corresponds to the k-th distinct oracle query. If $\bar{\mathcal{A}}$ succeeds, the simulation returns (k, σ); otherwise, it returns $(0, \sigma)$ to indicate that the impersonation was "failed".

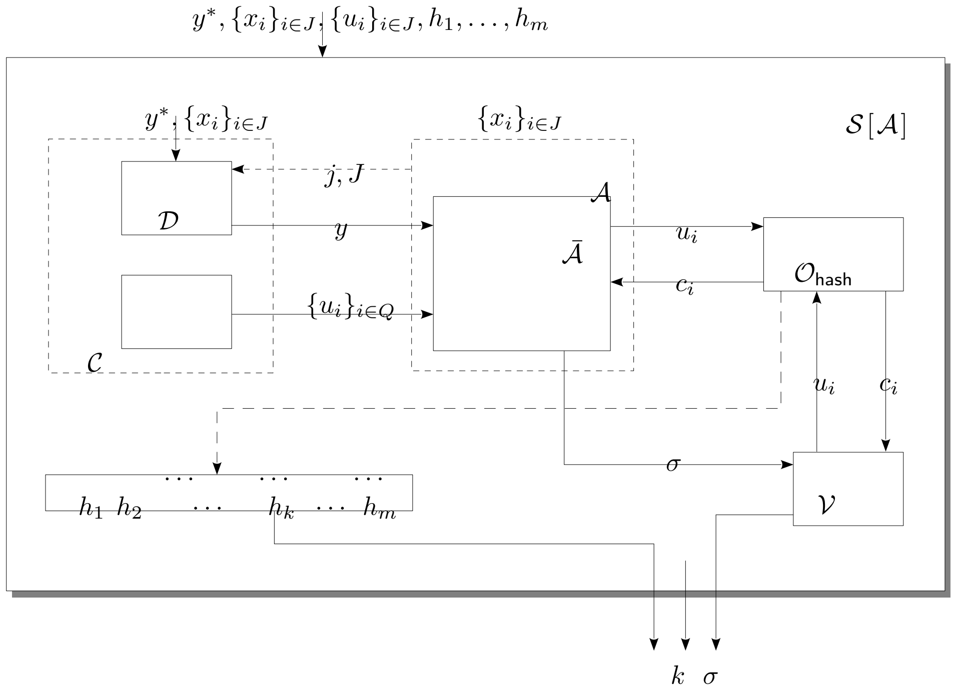

FIGURE 4. Impersonation attack simulator

Proposition 5.4. ($\mathsf{ORST}_{\mathbb{G},n,t}$–simulation) *If H is modelled as a random oracle, $\mathcal{S}[\mathcal{A}]$ simulates Game 4.4 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 $\bar{\mathcal{A}}$'s input in an invocation*

$$\begin{aligned}(k,\sigma) \leftarrow \mathcal{S}[\mathcal{A}](y^*, \{x_i\}_{i\in J}, \{u_i\}_{i\in J};\, h_1,\dots,h_m),\\ y^* \leftarrow_\$ \mathbb{G}, \quad x_i \leftarrow_\$ \mathbb{Z}_q, \quad u_i \leftarrow_\$ \mathbb{G}, \quad i\in J,\\ h_\nu \leftarrow_\$ \mathbb{Z}_q,\ 1\le \nu \le m.\end{aligned} \tag{5.5}$$

In the context of this simulation, if $\bar{\mathcal{A}}$ succeeds in its impersonation attempt, i.e., $k \ge 1$, the input value h_k maps to $c_j \equiv H(u_j)$ of the real game execution.

Proof. In the real game execution, the adversarial view consists of the corrupted shares $\{x_i\}_{i\in J}$ along with the combined public key y, the intercepted commitments $\{u_i\}_{i\in Q}$ and the hashes queried during the impersonation phase. These three blocks are independent from each other; clearly, the first and second blocks are independent, while the third block is independent from the others because H is a random oracle. It thus suffices to check that the respective blocks have identical distributions in the simulated execution (5.5). The first block is perfectly simulated due to Lemma 5.3. The second block is also perfectly simulated because $\{u_i\}_{i \in Q}$ is in either case uniformly sampled. Finally, the third block is perfectly simulated because the predefined hashes $h_1, \dots, h_m$'s are uniformly sampled (cf. Section A.0.1). The fact that $h_k = h[k]$ corresponds to $H(u_j)$ in the real game execution follows directly from the definition of k. ☐

We are interested in the probability that the simulation does not "fail" when run with iniformly random input, i.e.,

$$\mathsf{acc}(\mathcal{S}[\mathcal{A}]) = Pr[\, k \geq 1 : \left\{ \begin{array}{c} (k, \sigma) \leftarrow \mathcal{S}[\mathcal{A}](y^*, \{x_i\}_{i \in J}, \{u_i\}_{i \in Q};\, h_1, \dots, h_m) \\ y^* \leftarrow_\$ \mathbb{G}, \quad x_i \leftarrow_\$ \mathbb{Z}_q, \quad u_i \leftarrow_\$ \mathbb{G}, \quad i \in J, \\ h_\nu \leftarrow_\$ \mathbb{Z}_q,\ 1 \leq \nu \leq m \end{array} \right\}].$$

By definition of $\mathcal{S}[\mathcal{A}]$, this is equal to the probability that the embedded adversary succees in its impersonation attempt when run with uniformly random input. We conclude with the following statement regarding the real attack adversary.

Corollary 5.6. *Given any adversary $\mathcal{A}$ for Game 4.4,*

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

Proof. Direct consequence of Proposition 5.4. ☐

5.3. **Local fork and extraction.** In the non-distributed case $t = 1$ (i.e., $J = \varnothing$), $\mathcal{S}[\mathcal{A}]$ yields the execution environment of the rewindable adversary $\mathcal{A}$ that is usually employed to reduce the security of ordinary Schnorr identification to discrete-logarithm hardness in the random oracle model. This reduction adapts the bad randomness attack against the non-interactive version of the protocol in order to recover the key by rewinding. In particular, an adversary $\mathcal{A}^*$ attacking the discrete-logarithm of some uniformly random y^* embeds $\mathcal{A}$ in a simulated impersonation attack with public key $y = y^*$ and lets it fabricate a proof (u, s). The impersonation is successful only if $s = x + r\,c$, where $x = \log y$ and $r = \log u$ with $c \leftarrow H(u)$. Note that $\mathcal{A}^*$ cannot extract x at this point because the logarithm r cannot be assumed to be known; specifically, $\mathcal{A}$'s black-box strategy for fabricating u cannot be assumed to include seeing r in the process. However, $\mathcal{A}^*$ can rewind the simulation and have $\mathcal{A}$ reuse u in order to eliminate r. More accurately, $\mathcal{A}^*$ resets the simulation (including the internal state of both $\mathcal{A}$ and H) exactly before $\mathcal{A}$ submits u to H and then lets $\mathcal{A}$ fabricate a proof of the form (u, s^*). In the process of its second attempt, $\mathcal{A}$ queries $c^* \leftarrow H(u)$ where $c \neq c^*$ with overwhelming probability. In case of success, i.e., if $s^* = r + c^* x$, $\mathcal{A}^*$ substracts responses and extracts the secret key in the form

$$x = \frac{s - s^*}{c - c^*}\,. \tag{5.8}$$

Since $x = \log y^*$ by choice of y, $\mathcal{A}^*$ forwards x to its own challenger and wins the discrete-logarithm game. By the forking lemma (cf. [12] or Reset Lemma, [9]), if $\mathcal{A}$ succeeds in its first impersonation attempt with probability ε, then the probability ε^* that it succeeds in its second attempt is lower-bounded as

$$\varepsilon \leq \frac{1}{q} + \sqrt{\varepsilon^*}, \tag{5.9}$$

where q stands for the group order. Consequently, if $\mathcal{A}$ breaks Schnorr identification with non-negligible advantage, $\mathcal{A}^*$ breaks the discrete-logarithm with non-negligible advantage as well.

In the general case $t \geq 1$, the unknown to be extracted is $x_j = \log y_j$. As it turns out (cf. Prop. 5.12), in order to derive an extraction formula that generalizes (5.8) we would need to repeat the impersonation attack with *all* hashes fixed except for $c_j \leftarrow H(u_j)$. This disallows the rewinding argument from being effective in the general threshold setting; indeed, resetting the attack before $\mathcal{A}$ queries $c_j \leftarrow H(u_j)$ does not guarantee that subsequent oracle responses remain the same and, in fact, they differ almost certainly. In other words, the forking lemma would only be applicable if $c_j \leftarrow H(u_j)$ were the last distict oracle query, but this need not hold true for the embedded adversary $\bar{\mathcal{A}}$.[4] However, when invoking $\mathcal{S}[\mathcal{A}]$ we predefine the hashes $h_1, \ldots, h_m$ returned by the embedded oracle in respective order. This allows us to re-execute the simulation keeping all hashes fixed except for (with overwhelming probability) $c_j = h_k$; in this case, the Local Forking Lemma (cf. Lemma A.13) becomes, provided that the embedded adversary is a deterministic algorithm. With this in mind, we define

$$(b, \mathsf{out}) \leftarrow \mathsf{LocalFork}_{\mathcal{S}[\mathcal{A}]}(y^*, \{x_i\}_{i \in J}, \{u_i\}_{i \in J})$$

as the following computation.

Algorithm 5 $\mathsf{LocalFork}_{\mathcal{S}[\mathcal{A}]}(y^*, \{x_i\}_{i \in J}, \{u_i\}_{i \in J})$

$h_\nu \leftarrow_\$ \mathbb{Z}_q, \quad 1 \leq \nu \leq m$
$(k, \sigma) \leftarrow \mathcal{S}[\mathcal{A}](\theta\,;\, h_1, \ldots, h_m), \quad \theta \equiv (y^*, \{x_i\}_{i \in J}, \{u_i\}_{i \in J})$
if $k = 0$
  return $(0, \perp)$
$h_k^* \leftarrow_\$ \mathbb{Z}_q$
$(k^*, \sigma^*) \leftarrow \mathcal{S}[\mathcal{A}](\theta\,;\, h_1, \ldots, h_{k-1}, h_k^*, h_{k+1}, \ldots h_m)$
if $k^* = k \;\wedge\; h_k \neq h_k^*$
  return $(1, (\sigma, \sigma^*))$
return $(0, \perp)$

FIGURE 5. $\mathsf{LocalFork}_{\mathcal{S}[\mathcal{A}]}$

[4] Inspecting the validity condition (**??**), one could possibly argue without loss of generality that $c_j \leftarrow H(u_j)$ is the last query issued by $\mathcal{A}$ to the random oracle and then apply a rewinding argument. This is because –heuristically– it makes no sense for $\mathcal{A}$ to start fabricating the s_i's before the totality of μ_i's have first stabilized, and consequently the totality of c_i's have resolved. Indeed, the very last oracle query injects perfect entropy into equation (**??**), forcing $\mathcal{A}$ to recalibrate the s_i's irrespective of any previous computations in order to preserve it; and, since H is a random oracle, it makes no difference if $\mathcal{A}$ defers the query $c_j \leftarrow H(u_j)$ to be last. We will not pursue the heuristic reasoning, contenting ourselves with the formalism of local forking.

That is, if the embedded adversary $\bar{\mathcal{A}}$ succeeds in its first impersonation attempt, the simulation is re-executed with the same input values except for the k-th predefined "hash", which is afresh uniformly sampled as h_k^*; otherwise, the fork aborts. When the second simulation completes, the fork checks if $\bar{\mathcal{A}}$ succeeds again under the condition $h_k^* \neq h_k$; in this case, it returns the outputs σ and σ^* of both simulations; otherwise, it aborts. Note that $\mathsf{LocalFork}_{\mathcal{S}[\mathcal{A}]}$ is the local fork of $\mathcal{S}[\mathcal{A}]$ in the sense of Section A.0.2. We are interested in the probability that it does not abort when invoked with uniformly random input, i.e.,

$$\mathsf{forkAcc}(\mathcal{S}[\bar{\mathcal{A}}]) \;=\; Pr\,[\,b \neq 0 : \left\{ \begin{array}{c} (b, \mathsf{out}) \leftarrow \mathsf{LocalFork}_{\mathcal{S}[\bar{\mathcal{A}}]}(y^*, \{x_i\}_{i\in J}, \{u_i\}_{i\in J}) \\ y^* \leftarrow_{\$} \mathbb{G}, \quad x_i \leftarrow_{\$} \mathbb{Z}_q, \quad u_i \leftarrow_{\$} \mathbb{G}, \quad i \in J \end{array} \right\}]$$

Evidently, this is equal to the probability that the embedded adversary succeds both in its first and forked impersonation attempt when invoked with the same random coins. We deduce the following bound.

Corollary 5.10. *Given any deterministic adversary $\mathcal{A}$ for Game 4.4,*

$$\mathsf{forkAcc}(\mathcal{S}[\mathcal{A}]) \;\geq\; \frac{1}{q_{\mathsf{ro}} + 1}\, \mathsf{Adv}^{\mathsf{ORST}}_{\mathbb{G},n,t}[\mathcal{A}]^{\,2} \tag{5.11}$$

where $q_{\mathsf{ro}} + 1$ is the number of distinct queries issued by $\mathcal{A}$ to the random oracle.

Proof. Direct consequence of Corollary 5.6 and the Local Forking Lemma. □

The importance of this inequality lies in the fact that it represents a lower bound to the probability of extracting x_j that will be used in our final reduction argument (cf. Section 5.4). This is because, in case of repeated success, the outputs of the first and the forked simulation can be correlated as follows.

Proposition 5.12. (ORST – Extractability) *Let $\mathcal{A}$ be a deterministic adversary of Game 4.4 selecting $J \subset \{1, \ldots, n\}$ with $|J| = t - 1$ and $j \in \{1, \ldots, n\} \setminus J$ during the corruption phase. Denote $Q = J \cup \{j\}$. Given a non-aborting execution*

$$(1, (\sigma, \sigma^*)) \leftarrow \mathsf{LocalFork}_{\mathcal{S}[\mathcal{A}]}(y^*, \{x_i\}_{i\in J}, \{u_i\}_{i\in J}),$$

then σ, σ^* are of the form

$$\{(u_i, c_i, s_i)\}_{i\in Q},\; \{(u_i, c_i^*, s_i^*)\}_{i\in Q} \tag{5.13}$$

respectively and subject to the contraint

$$c_j \neq c_j^* \;\wedge\; c_i = c_i^*,\; \forall i \in Q \setminus \{j\}. \tag{5.14}$$

Moreover, the extraction formula

$$x_j \;=\; \frac{s_j - s_j^*}{c_j - c_j^*} \;+\; \frac{1}{\mu_j} \sum_{i\in J} (\mu_i \delta_i - \mu_i^* \delta_i^*), \quad x_j \equiv \log y^* \tag{5.15}$$

holds true, where $\{\mu_i\}_{i\in Q}$, $\{\mu_i^*\}_{i\in Q}$ are the weights and $\{\delta_i\}_{i\in Q}$, $\{\delta_i^*\}_{i\in Q}$ are the local deviations of σ and σ^* respectively.

Proof. Denoting $\sigma = \{(u_i, c_i, s_i)\}_{i\in Q}$, $\sigma^* = \{u_i^*, c_i^*, s_i^*\}_{i\in Q}$ so that $u_i^* = u_i$ for $i \neq j$, we need to show that $u_j^* = u_j$ as well. Recall that u_j resp. u_j^* is fabricated by $\bar{\mathcal{A}}$ during the first and forked impersonation attempt respectively. Up to that moment, the adversary cannot have queried the hash-oracle for the j-th commitment. Since $\bar{\mathcal{A}}$ is deterministic and its execution environment is the same in both executions except only for the k-th distinct hash, we conclude that $u_j = u_j^*$. The constraint regarding the challenges follows because

$$c_j^* = h_k^* \neq h_k = c_j$$

and the rest challenges are drawn from $\{h_\nu\}_{\nu\neq k}$. We proceed to derive the extraction formula. Since $c_i = c_i^*$ for $i \neq j$, by definition of weights we get

$$\mu_j = \mu_j^* \tag{5.16}$$

(cf. Def. 3.8). Expanding local deviations, Corollary 3.33 implies

$$\sum_{i\in Q} \mu_i(s_i - r_i - c_i x_i) = 0 \quad \text{and} \quad \sum_{i\in Q} \mu_i^*(s_i^* - r_i - c_i^* x_i) = 0.$$

Substracting terms and applying (5.16) yields

$$\mu_j(s_j - s_j^* - (c_j - c_j^*)x_j) \;+ \sum_{i\in Q\setminus\{j\}} (\mu_i\delta_i - \mu_i^*\delta_i^*) = 0.$$

Extraction follows because $c_j - c_j^* \neq 0$. □

Remark 5.17. It should be stressed that, despite eliminating $r_j = \log u_j$, the extraction formula still contains the logarithms of the intercepted commitments $\{u_i\}_{i\neq j}$. Indeed, expanding local deviations, (5.15) reformulates as

$$x_j \;=\; \frac{s_j - s_j^*}{c_j - c_j^*} + \frac{1}{\mu_j}\sum_{i\in J}\Big(\mu_i(s_i - r_i - c_i x_i) - \mu_i^*(s_i^* - r_i - c_i x_i)\Big) \tag{5.18}$$

and is thus useful only if the logarithms $r_i = \log u_i$ are known to the adversary. For our purposes, this knowledge is obtained by submitting the intercepted commitments to the dlog-oracle (cf. Section 4.3), which is the reason why ORST security reduces to $(t-1)$-OMDL hardness (cf. Theorem 5.19).

5.4. **Main theorems.** TODO: Refer to the appendix when needed: deterministic operators without narrowing the threat model

Theorem 5.19. (ORST – *Security against impersonation attack*) ORST *is secure against impersonation attacks under one-more discrete-logarithm hardness in the random oracle model. In particular, if H is modelled as a random oracle, for every polynomial-time adversary $\mathcal{A}$ attacking Game* 4.4 *there exists a polynomial-time adversary $\mathcal{A}^*$ solving the* $(t-1)$-OMDL *problem in $\mathbb{G}$ with advantage*

$$\mathsf{Adv}^{\mathsf{OMDL}}_{\mathbb{G},t-1}[\mathcal{A}^*] \;\geq\; \frac{1}{q_{\mathsf{ro}}+1}\,\mathsf{Adv}^{\mathsf{ORST}}_{\mathbb{G},n,t}[\mathcal{A}]^2 \tag{5.20}$$

where $q_{\mathsf{ro}} + 1$ is the number of distinct queries issued by $\mathcal{A}$ to the random oracle.

Proof. We will wrap $\mathcal{A}$ with an execution environment $\mathcal{S}$ that simulates Game 4.4 under the control of $\mathcal{A}^*$; by forking $\mathcal{S}$ at a certain location, $\mathcal{A}^*$ should be able to win Game 4.12 with a certain advantage. Specifically, if $m = q_{\mathsf{ro}} + 1$ is the number of queries issued by $\mathcal{A}$ to H, we will define $\mathcal{S}$ as an algorithm of the form $\square$

Algorithm 6 $\mathcal{A}^*(w_0, w_1, \ldots, w_{t-1})$

Set $y^* \leftarrow w_0$

Set $\{u_i\}_{i\in J} \leftarrow \mathsf{Zip}(w_1, \ldots, w_{t-1}; J)$

Sample $x_i \leftarrow_\$ \mathbb{Z}_q,\ i \in J$

$(b, \mathsf{out}) \leftarrow \mathsf{LocalFork}_{\mathcal{S}}(y^*, \{x_i\}_{i\in J}, \{u_i\}_{i\in J})$

if $b = 0$

  return $\perp$

Parse $(\sigma, \sigma^*) \leftarrow \mathsf{out}$

Parse $\{(u_i, c_i, s_i)\}_{i\in Q} \leftarrow \sigma,\ \{(u_i, c_i^*, s_i^*)\}_{i\in Q} \leftarrow \sigma^*$

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

Compute

$$\mu_j \leftarrow \lambda_j \prod_{i\in J} c_i$$

for $i \in J$,

$$\mu_i \leftarrow \lambda_i \prod_{k\in Q\setminus\{i\}} c_k, \qquad \mu_i^* \leftarrow \lambda_i \prod_{k\in Q\setminus\{i\}} c_k^*$$

Query $r_i \leftarrow \mathcal{O}^{\mathsf{dlog}}_{\mathbb{G}}(u_i),\ i \in J$ // $t-1$ *queries*

Compute

$$x_j \leftarrow \frac{s_j - s_j^*}{c_j - c_j^*} + \frac{1}{\mu_j} \sum_{i\in J} \Big(\mu_i(s_i - r_i - c_i x_i) - \mu_i^*(s_i^* - r_i - c_i x_i)\Big)$$

Set $z_0 \leftarrow x_j$

Set $(z_1, \ldots, z_{t-1}) \leftarrow \mathsf{Zip}^{-1}(\{r_i\}_{i\in J})$

return $z_0, z_1, \ldots, z_{t-1}$

6. Applications

6.1. Optimized threshold decryption.

Appendix A. Shamir's secret sharing

We gather here for reference some elementary facts regarding the celebrated scheme introduced by Shamir [1]. Let $\big(\mathbb{A}\big)_m[X]$ be the ring of polynomials of degree at most

$m > 1$ over a ring $\mathbb{A}$. Fix a cyclic group $G = \langle g \rangle$ of prime order q. The fact that $\mathbb{A} = \mathbb{Z}_q$ is a field facilitates Lagrange interpolation and consequently the following sharing scheme.

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

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

We will denote the set of possible Shamir (n,t)-sharing outputs for x as

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

Remark A.2. (*Homomorphic property of Shamir's sharing*) By the properties of polynomial evaluation, the componentwise addition of

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

yields a (n,t)-sharing of $x + x^*$, i.e., $(x_1 + x_1^*, \ldots, x_n + x_n^*;\ y \cdot y^*) \in D_{x+x^*}$.

Remark A.3. (*Reconstruction formula*) By linearity of interpolation, the equality

$$x = \sum_{i \in Q} \lambda_i x_i \tag{A.4}$$

holds true if and only if $|Q| \geq t$, where $\{\lambda_i\}_{i \in Q}$ are the interpolation coefficients for $Q \subset \{1, \ldots, n\}$. Applying exponentiation, this translates to

$$y = \prod_{i \in Q} y_i^{\lambda_i}, \tag{A.5}$$

where $y_i \equiv g^{x_i},\ 1 \leq i \leq n$.

Remark A.6. ($t-1$ *degrees of freedom*) Given x, for every $J \subset \{1, \ldots, n\}$ with $|J| = t-1$ and every $\{x_i^*\}_{i \in J} \subset \mathbb{Z}_q$, there exists a unique $(x_1, \ldots, x_n;\ y) \in D_x$ such that $x_i = x_i^*,\ \forall i \in J$.

Remark A.7. (*Security of Shamir's secret sharing*) Given $x,\ x^*$ and $J \subset \{1, \ldots, n\}$ with $|J| = t - 1$, the probability distribution

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

is indistinguishable from the distribution

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

Algorithm 8 $\mathcal{O}^{\mathsf{hash}}(u)$

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

FIGURE 7. Hash oracle

A.0.1. *Hash oracles.* Recall that a random oracle H with codomain C is a randomized process operating on top of a lookup table Map as follows.

Algorithm 7 $H(u)$

if $u \notin \mathrm{Dom}(\mathsf{Map})$
    $c \leftarrow_{\$} C$
    $\mathsf{Map}[u] = c$
else
    $c \leftarrow \mathsf{Map}[u]$
return c

FIGURE 6. Random oracle

When querying H in the context of a simulation (cf. Section 5.2), we usually need to keep track of distinct oracle queries. We do so by means of a component with read access to a tape of preallocated "hashes", which is indistinguishable from a random oracle when this tape is filled up with uniformly random values. Specifically, given a finite tape $h = [h_1, \ldots, h_m]$ and initially empty lookup tables Map_1 and Map_2, we define the *hash-oracle* $\mathcal{O}^{\mathsf{hash}}$ as the following process.
That is, $\mathcal{O}^{\mathsf{hash}}$ operates like a random oracle with lookup table Map_1 except for it draws "hashes" from the tape h instead of sampling them uniformly; upon inserting a fresh "hash" to Map_1, it takes care to (a) update the tape pointer so that the next predefined "hash" is available on demand and (b) insert the current value of the pointer to Map_2 so that the order of the respective query can be easily tracked. More accurately, the condition $\mathsf{Map}_2[u] = k$ means that u was the k-th distinct element submitted to the hash oracle. Evidently, if $h_1, \ldots, h_n$ are uniformly sampled from the codomain of H, interacting with $\mathcal{O}^{\mathsf{hash}}$ cannot be distinguished from interacting with H. We state this remark formally for the sake of reference.

Lemma A.8. (*Random oracle simulation*) *Given a random oracle H with codomain C and initially empty lookup table, initialize a hash oracle $\mathcal{O}^{\mathsf{hash}}$ over an empty tape h of size $m \geq 1$ and initially empty lookup tables. For any distinct elements $u_1, \ldots, u_m$, 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 hash oracle. □

A.0.2. *Local forking lemma.* The forking lemma was introduced by Pointcheval and Stern ([6], [7]) in order prove the security of various signature schemes in the random oracle model by rewinding. It was further elaborated as "general" forking lemma by Bellare and Neven [12], who abstracted away the signature specific details. The "local" forking lemma is a variance by Bellare, Dai and Li [11] which dispenses with rewinding completely. Namely, it allows to fork a process by reprogramming the oracle at a single location, as opposed to every location after the fork. Consider a deterministic algorithm of the form

$$\mathcal{S} : \Theta \times C^m \rightarrow \{0, 1, \ldots, m\} \times \Sigma, \quad (k, \sigma) \leftarrow \mathcal{S}(\theta; h_1, \ldots, h_m), \tag{A.9}$$

running internally a hash oracle with preallocated "hashes" $h_1, \ldots, h_m$ in respective order (cf. Section A.0.1). Think of θ as the random coins, k as the *main* output and σ as the *side* output. If $k \geq 1$, the main output indicates the special character of the k-th distinct oracle query regarding the side output, while the case $k = 0$ is interpreted as *failure*. We denote by

$$\mathsf{acc}(\mathcal{S}) \;=\; Pr\,[\,k \geq 1 : \left\{ \begin{array}{c} (k, \sigma) \leftarrow \mathcal{S}(\theta; h_1, \ldots, h_m), \quad \theta \leftarrow_\$ \Theta \\ h_\nu \leftarrow_\$ C, \quad 1 \leq \nu \leq m \end{array} \right\}] \tag{A.10}$$

the probability that $\mathcal{S}$ *accepts* (taken over the possible random coins and oracle responses) and reserve a symbol $\perp \notin \Sigma$ to denote *abortion*.

Definition A.11. The *local fork of* $\mathcal{S}$ is the algorithm

$$(b, \mathsf{out}) \leftarrow \mathsf{LocalFork}_{\mathcal{S}}(\theta)$$

defined in TODO.

That is, if $\mathcal{S}$ accepts on its first run, the fork re-executes it with the same input except for the k-th "hash", which is afresh uniformly sampled as h_k^*; otherwise the fork aborts. When the second execution completes, the fork checks if $\mathcal{S}$ accepts again under the constraint $h_k^* \neq h_k$, in which case it returns both side outputs σ and σ^*; otherwise, it aborts. We denote by

$$\mathsf{forkAcc}(\mathcal{S}) \;=\; Pr\,[\,b = 1 : \; (b, \mathsf{out}) \leftarrow \mathsf{LocalFork}_{\mathcal{S}}(\theta), \; \theta \leftarrow_\$ \Theta\,] \tag{A.12}$$

the probability that the local fork does not abort taken over the possible random coins of $\mathcal{S}$. The local forking lemma asserts the following lower bound.

Lemma A.13. (*Local Forking Lemma*, [11]) *For every deterministic algorithm* $\mathcal{S}$ *of the form* (A.9) *issuing* m *distinct queries to a hash oracle with tape* $[h_1, \ldots, h_m]$,

$$\mathsf{forkAcc}(\mathcal{S}) \geq \frac{\mathsf{acc}(\mathcal{S})^2}{m} \tag{A.14}$$

Algorithm 9 $\mathsf{LocalFork}_{\mathcal{S}}(\theta)$

1: $h_\nu \leftarrow_\$ \mathbb{Z}_q, \quad 1 \leq \nu \leq m$
2: $(k, \sigma) \leftarrow \mathcal{S}(\theta\,;\, h_1, \ldots, h_m)$
3: **if** $k = 0$
4:     **return** $(0, \perp)$
5: $h_k^* \leftarrow_\$ \mathbb{Z}_q$
6: $(k^*, \sigma^*) \leftarrow \mathcal{S}(\theta\,;\, h_1, \ldots, h_{k-1}, h_k^*, h_{k+1}, \ldots h_m)$
7: **if** $k^* = k \;\wedge\; h_k \neq h_k^*$
8:     **return** $(1, (\sigma, \sigma^*))$
9: **return** $(0, \perp)$

Appendix B. Security formalism

B.1. **Attack games.** [TODO Abstract formalism not needed] Attack games are loosely defined as interactive procedures where an adversary with limited access to secrets tries to break some protocol that is honestly simulated by a challenger; specifically, the adversary tries to fabricate an output that verifies, causing the protocol execution to finalize without rejection. Adversaries are modelled as probabilistc algorithms of certain time complexity that consume the view generated for them by the challenger. By *view* is here meant the totality of public and secret quantities that the adversary sees in the process, the challenger revealing secrets in accordance with some specified threat model.

We confine ourselves to games of low interactivity, suitable for threat models where the adversary corrupts a group of involved parties from the outset. Much of the following considerations apply to more general settings but we deliberately pin down a definition for the restricted case of interest in order to derive the desired results rapidly.

Definition B.1. By *attack game* is meant a pair $\mathsf{G} = (\mathsf{View}_\mathsf{G}, \mathsf{Verify}_\mathsf{G})$, where View_G is a probabilistic algorithm of the form

$$y, v_1, v_2, \cdots \leftarrow \mathsf{View}_\mathsf{G}(\rho, \alpha),\;\; \rho \in \Theta_\mathsf{G} \tag{B.2}$$

and $\mathsf{Verify}_\mathsf{G}$ is a deterministic algorithm of the form

$$b \leftarrow \mathsf{Verify}_\mathsf{G}(y, \sigma) \tag{B.3}$$

with codomain $\{\mathsf{true}, \mathsf{false}\}$.

An *adversary* attacking G is a pair of probabilistic algorithms $\mathcal{A} = (\mathcal{A}_0, \bar{\mathcal{A}})$, where $\mathcal{A}_0$ generates the setup parameter α and $\bar{\mathcal{A}}$ is capable of consuming the output of View_G in order to fabricate a purportedly valid output σ. Specifically, the game is played as follows. First, $\mathcal{A}$ sends $\alpha \leftarrow \mathcal{A}_0(\cdot)$ to the challenger $\mathcal{C}$, who runs

$$y, v_1, v_2, \dots \leftarrow \mathsf{View}_\mathsf{G}(\rho, \alpha), \quad \rho \leftarrow_\$ \Theta_\mathsf{G}$$

and sends $y, v_1, v_2, \dots$ to $\mathcal{A}$. Then $\mathcal{A}$ fabricates

$$\sigma \leftarrow \bar{\mathcal{A}}(y, v_1, v_2, \dots)$$

and *wins* only if $\mathsf{Verify}_\mathsf{G}(y, \sigma) = \mathsf{true}$.

Definition B.4. The *concrete advantage* of an adversary $\mathcal{A}$ attacking game G is the probability that $\mathcal{A}$ wins taken over the random coins of game execution, i.e.,

$$\mathsf{Adv}^\mathsf{G}[\mathcal{A}] = Pr\left[\, b = \mathsf{true} : \left\{ \begin{array}{c} b \leftarrow \mathsf{Verify}_\mathsf{G}(y, \sigma), \quad \sigma \leftarrow \bar{\mathcal{A}}(y, v_1, v_2, \dots) \\ (y, v_1, v_2, \dots) \leftarrow \mathsf{View}_\mathsf{G}(\rho, \alpha) \\ \rho \leftarrow_\$ \Theta_\mathsf{G}, \quad \alpha \leftarrow \mathcal{A}_0(\cdot) \end{array} \right\} \right] \tag{B.5}$$

The following statement allows to safely assume that, if needed, $\mathcal{A}$ is deterministic in the context of a hardness reduction argument, where its advantage is typically used to lower-bound the probability of success in attacking the target game.[5]

Proposition B.6. *For every adversary $\mathcal{A}$ attacking game G, there exists a deterministic advesary $\mathcal{A}'$ of equal time complexity with at least equal advantage, i.e.,*

$$\mathsf{Adv}^\mathsf{G}[\mathcal{A}'] \geq \mathsf{Adv}^\mathsf{G}[\mathcal{A}]. \tag{B.7}$$

Proof. Consider the residual determinisic strategies $\mathcal{A}_1, \dots, \mathcal{A}_m$ obtained by fixing the possible values of $\mathcal{A}$'s random coins and let p_i be the probability that $\mathcal{A}$ runs $\mathcal{A}_i$, which is itself an adversary for G. By the expression (B.5) of concrete advantage,

$$\begin{aligned} \mathsf{Adv}^\mathsf{G}[\mathcal{A}] &= \sum_{i=1}^m Pr[\,\mathcal{A}_i \text{ wins } \mathsf{G} \;\wedge\; \mathcal{A} \text{ runs } \mathcal{A}_i\,] \\ &= \sum_{i=1}^m Pr[\,\mathcal{A} \text{ runs } \mathcal{A}_i\,] \cdot Pr[\,\mathcal{A}_i \text{ wins } \mathsf{G} \mid \mathcal{A} \text{ runs } \mathcal{A}_i\,] \\ &= \sum_{i=1}^m p_i \cdot \mathsf{Adv}^\mathsf{G}[\mathcal{A}_i] \end{aligned}$$

Since $p_1 + \dots + p_m = 1$, we conclude that $\mathsf{Adv}^\mathsf{G}[\mathcal{A}_i] \geq \mathsf{Adv}^\mathsf{G}[\mathcal{A}]$ for some index $i \in \{1, \dots, m\}$. □

[5] It should be stressed that the following proof does not generalize to arbitrary attack games, which is the actual reason for confining ourselves to Definition B.1. Refer to [13] for a discussion regarding its falseness in a proof-of-knowldge context.

Disregarding the adversarial strategy, attack games of interest unfold agnostically over a cyclic group $\mathbb{G}$ and constants $n, \ldots,$ that together determine view and verification. By abstracting away the specifics, we derive the following notion.

Definition B.8. An *attack game template* is a pair of parametrized algorithms

$$\mathsf{GAME}_{\mathbb{G},n,\ldots} = (\mathsf{View}_{\mathbb{G},n,\ldots}, \mathsf{Verify}_{\mathbb{G},n,\ldots}) \tag{B.9}$$

which for every choice of $\mathbb{G}, n, \ldots$ yields an attack game in the sense of Def. B.1.

If G instantiates $\mathsf{GAME}_{\mathbb{G},n,\ldots}$ for specific choices of $\mathbb{G}, n, \ldots,$ we write

$$\mathsf{G} = \mathsf{GAME}_{\mathbb{G},n,\ldots}$$

to indicate this fact and further denote the concrete adversarial advantage by

$$\mathsf{Adv}^{\mathsf{GAME}}_{\mathbb{G},n,\ldots}[\mathcal{A}] \equiv \mathsf{Adv}^{\mathsf{G}}[\mathcal{A}].$$

It is usually the case that $\mathcal{A}$ operates agnostically on top of $\mathbb{G}, n, \ldots$ as well, i.e., $\mathcal{A}$ is per se a strategy template yielding a concrete attack for every choice of the parameters; if not, we can reprogram $\mathcal{A}$ to trivially "fabricate" some fixed output for those parameter values for which it is not initially defined. Henceforth we assume for convenience that every adversary is templated in this fashion. We will see that this does not affect security definitions.

References

[1] Adi Shamir. How to Share a Secret. Commun. ACM 1979.

[2] Balthazar Bauer, Georg Fuchsbauer, Antoine Plouviez. The One-More Discrete Logarithm Assumption in the Generic Group Model. ASIACRYPT 2021.

[3] Claus-Peter Schnorr. Efficient Signature Generation by Smart Cards. Journal of Cryptogoly 1991, p. 161-174

[4] C. Komlo and I. Goldberg. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. SAC 2020.

[5] David Bernhard, Olivier Pereira, Bogdan Warinschi. How not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. https://eprint.iacr.org/2016/771.pdf.

[6] David Pointcheval, Jacques Stern. Security proofs for Signature Schemes. Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding.

[7] David Pointcheval, Jacques Stern. Security Arguments for Digital Signatures and Blind Signatures. Journal of Cryptology, 2000, p. 361-396

[8] Douglas R. Strinson, Reto Stroble. Provably Secure Distributed Schnorr Signatures and a (t, n) Threshold Scheme for Implicit Certificates. ACISP 2001.

[9] Mihir Bellare, Adriana Palacio. GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and Concurrent Attacks. CRYPTO 2002.

[10] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. Journal of Cryptology, 16(3):185–215, June 2003.

[11] Mihir Bellare, Wei Dai, and Lucy Li. The local forking lemma and its application to deterministic encryption. In S. D. Galbraith and S. Moriai, editors, Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III, volume 11923 of Lecture Notes in Computer Science, pages 607–636. Springer, 2019.

[12] Mihir Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In A. Juels, R. N. Wright, and S. D. C. di Vimercati, editors, Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006, pages 390–399. ACM, 2006.

[13] Mihir Bellare, Oded Goldreich. On Probabilistic versus Deterministic Provers in the Definition of Proofs Of Knowledge. Cryptology ePrint Archive, Paper 2006/359.

[14] M. Bellare. Lecture on NIZKS: A Concrete Security Treatment. August 2021.

[15] R. Gennaro S. Jarecki H. Krawczyk and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In Eurocrypt '99, pages 295–310, 1999.