SHINE: Resilience via Practical Interoperability of Multi-party Schnorr Signature Schemes

Antonin Dufka[1], Vladimir Sedlacek[1,2] and Petr Svenda[1]

[1]*Masaryk University, Czech Republic*

[2]*Université de Picardie Jules Verne, France*

Keywords: Cryptographic Hardware, Multi-party Computation, Nonce Agreement, Schnorr Signatures, Interoperability.

Abstract: Secure multi-party cryptographic protocols divide the secret key among multiple devices and never reconstruct it in a single place. Such a mechanism protects against malware, code vulnerabilities, and backdoors when different implementations and devices are used. Still, a protocol-level issue may result in a compromise, and up until now, it has been unknown how to combine different unmodified multi-party protocols.

We study the interoperability of different multi-party Schnorr signature schemes and classify them based on their approach to the nonce agreement. We identify issues that could hinder in-class interoperability, and we propose a trustless mediator that facilitates interoperability among different classes in certain cases. Besides mitigating the risks, interoperability provides usability and performance benefits, as protocols better suited for special devices can be used together with more general protocols.

We make use of these advantages in our new multi-signature scheme SHINE, which is optimized for resource-limited devices like cryptographic smartcards while being interoperable with popular schemes such as MSDL, MuSig2, or SpeedyMuSig.

1 INTRODUCTION

Since the expiry of Schnorr's patent (Schnorr, 1991b), Schnorr signatures are making a considerable comeback, fueling digital signature specifications like EdDSA (Bernstein et al., 2012) and BIP-Schnorr (Wuille et al., 2020a). EdDSA is being gradually incorporated in many protocols like TLS, SSH, Tor, and WireGuard (IANIX, 2022), and BIP-Schnorr is now being used in Bitcoin (Nakamoto, 2008) as a part of the Taproot consensus upgrade (Wuille et al., 2020b).

The practical problems these applications face have reignited research interest in the area of multi-party Schnorr signatures. New schemes (Alper and Burdges, 2021; Maxwell et al., 2019; Nick et al., 2021; Syta et al., 2016; Crites et al., 2021) improved the practicality of Schnorr multi-signatures, e.g., decreased communication and achieved security in the plain public-key model. However, none of the schemes is suitable for all scenarios, and their approaches often differ in technical or design details, rendering them partly or fully incompatible. This results in having multiple protocols that perform in principle the same task yet cannot work together.

Good protocol interoperability would improve not only usability and performance but also security. A frequent weak point of any real-world cryptographic system is its implementation. Multi-party computation can mitigate the impact of implementation vulnerabilities by having different implementations running on different devices, as long as at least one of them remains secure (Mavroudis et al., 2017). Multi-party scheme interoperability extends this idea further: choosing different protocols limits the threat of common implementation errors in all of them at once.

Unfortunately, it is unknown if and how we can achieve interoperability of different Schnorr-based multi-party schemes. To tackle this problem, we study the scheme differences and classify them by their approach to the nonce agreement – a key component in signature computation. This allows us to investigate possible interoperability across classes and identify obstacles to in-class interoperability. We discover that cross-class interoperability can be in certain cases achieved via an untrusted mediator that translates communication among different protocols.

Based on these new insights, we design a new multi-signature scheme, SHINE, which is interoperable with several other Schnorr-based schemes and is optimized for computationally limited devices.

Dufka, A., Sedlacek, V. and Svenda, P.
SHINE: Resilience via Practical Interoperability of Multi-party Schnorr Signature Schemes.
DOI: 10.5220/0011145600003283
In *Proceedings of the 19th International Conference on Security and Cryptography (SECRYPT 2022)*, pages 305-316
ISBN: 978-989-758-590-6; ISSN: 2184-7711

Main Contributions:

- We classify the current Schnorr-based multi-party schemes by their approach to the nonce agreement, and propose an untrusted mediation layer that bridges class differences to achieve interoperability.
- We design a two-round (one-round plus precomputation) Schnorr multi-signature scheme called SHINE, which targets computationally limited devices like smartcards, and is interoperable with many pre-existing schemes. The scheme features a novel approach to nonce caching that avoids the previous attack on two-round schemes (Drijvers et al., 2019).
- We provide an open-source implementation of SHINE on the JavaCard platform, evaluate its performance on five different smartcard models, and demonstrate its interoperability with other scheme classes via the proposed mediator.

After presenting the relevant background in Section 2, we survey and compare different approaches to the nonce agreement in Section 3. Building upon the lessons learned, we propose our scheme SHINE in Section 4 and draw conclusions in Section 5.

2 BACKGROUND

This section introduces the notation, recalls Schnorr signatures, and presents relevant multi-party schemes.

2.1 Notation

A group description is a triplet $(\mathbb{G}, q, G)$, where q is a λ-bit prime, $\mathbb{G}$ is a cyclic group of order q, and G is a selected generator of $\mathbb{G}$. We use the additive notation for the group operation and denote group elements in upper-case. Conversely, we use lower-case to denote elements of $\mathbb{Z}_q$. Sampling of an element e from non-empty set S is denoted as $e \leftarrow S$. By $\mathcal{A}(x)$, we denote the set of outputs of probabilistic algorithm $\mathcal{A}$ given input x. We reserve n for the number of signing parties. Furthermore, with a secret s, we use the following notation:

- PRF_s – a pseudorandom function seeded with s;
- KDF_s – a key derivation function seeded with s;
- Enc_s – symmetric encryption with a key s;
- Dec_s – symmetric decryption with a key s;
- Com – a commitment function.

2.2 Schnorr Signatures

The Schnorr signature scheme (Schnorr, 1991a) is derived from the Schnorr identification scheme using the Fiat-Shamir transform (Fiat and Shamir, 1986), and it relies on the hardness of the discrete logarithm problem. The scheme outputs efficiently computable and verifiable signatures of short length. It has been proven existentially unforgeable under the chosen message attack in the random oracle model (Pointcheval and Stern, 2000). Various formulations of Schnorr signature schemes have been proposed, but in this paper, we choose the one typically used in recent works (Bernstein et al., 2012; Wuille et al., 2020a), as it supports efficient batch verification and prevents related-key attacks (Morita et al., 2015).

Definition 2.1 (Schnorr Signature)**.** Let $(\mathbb{G}, q, G)$ be a group description and $H : \mathbb{G}^2 \times \mathbb{Z}_q \to \mathbb{Z}_q$ be a hash function. A Schnorr signature of a message $m \in \mathbb{Z}_q$ verifiable with public key $X \in \mathbb{G}$ is a pair $(R, s) \in \mathbb{G} \times \mathbb{Z}_q$ satisfying the verification equation $sG = R + H(R, X, m)X$.

For a random nonce $r \in \mathbb{Z}_q$ and a private key $x \in \mathbb{Z}_q$ such that $xG = X$, a valid Schnorr signature of a message m is $(R, s) = (rG, r + H(R, X, m)x)$.

For fixed $H(R, X, m)$, the signing equation is linear, which is useful for efficient multi-party Schnorr signature schemes. First, all n parties need to agree on a collective nonce R, which is a linear combination of their individual contributions $R_i = r_iG$. Subsequently, they produce signature shares $s_i = r_i + H(R, X, m)x_i$, which are summed up to obtain the resulting signature $s = \sum_{i=1}^{n} s_i$, verifiable under the aggregate public key $X = \sum_{i=1}^{n} X_i$.

The simple multi-signature scheme described in the previous paragraph has a few caveats, which cause it to be insecure in many use-cases, and these issues are addressed by more complex designs. The two main security obstacles are related to the group key aggregation and the nonce agreement, both of which can be attacked to perform a forgery.

The key aggregation is prone to *rogue-key attacks*, where the adversary computes her key as a function of the public keys of other parties and cancels out their contribution. To illustrate the problem, assume the attacker is the first party. She can compute her key as $X_1 = x_1'G - \sum_{i=2}^{n} X_i$ for some $x_1' \in \mathbb{Z}_q$. When this rogue key is combined with the other keys, the resulting aggregate key is $X = x_1'G$, and the attacker can create signatures on behalf of the group.

Rogue-key attacks can be prevented by distributed key generation, which requires fresh key pairs like in Myst (Mavroudis et al., 2017). Alternatively, pre-existing keys can be reused when supplemented by proof of knowledge of their private key, e.g., (Boneh et al., 2018). Another method that supports key reuse is the non-interactive key aggregation method presented in MuSig (Maxwell et al., 2019), which avoids the attack by unpredictably altering the aggregate key whenever any of the inputs changes.

If sequential signing can be enforced, the protocol is secure, and the aggregate nonce does not even need to be computed by the signer, as is the case in CoSi (Syta et al., 2016). However, if the signing instances with the same key can be executed concurrently (e.g., nonce contributions are shared in advance), the Drijvers et al.'s attack can achieve signature forgery (Drijvers et al., 2019). The attack relies on solving an instance of the ROS problem (Schnorr, 2001), which can be solved in subexponential (Wagner, 2002) or polynomial time (Benhamouda et al., 2021), depending on the number of concurrent sessions.

2.3 Current Multi-party Schemes

In this subsection, we list and shortly describe all recent Schnorr-based multi-party schemes that we consider for the interoperability study.

CoSi (Syta et al., 2016) is a two-round Schnorr-based multi-signature scheme designed for high-speed signing by many parties organized into a tree structure. The scheme has been proven secure only for logarithmically many concurrent signing instances in a later work (Drijvers et al., 2019).

Myst (Mavroudis et al., 2017) is a setup of a large number of smartcards interconnected into a grid performing multi-party computations to achieve high guarantees of backdoor tolerance. Myst uses a multi-signature scheme similar to CoSi, optimized for limited devices. One of the optimizations (nonce caching) was found vulnerable to an attack by Drijvers et al. (Drijvers et al., 2019).

MuSig (Maxwell et al., 2019) was originally presented as a two-round scheme that was later found vulnerable by Drijvers et al. (Drijvers et al., 2019). Earlier MSDL (Boneh et al., 2018) used a preliminary commitment round that avoided the problem, and the same approach was also adopted to MuSig, resulting in a three-round concurrently-secure scheme.

The first concurrently-secure two-round multi-signature scheme resulting in standard Schnorr signatures was MuSig-DN (Nick et al., 2020), which avoids the Drijvers et al.'s attack by generating the nonce deterministically. The nonce needs to be supplemented with costly non-interactive zero-knowledge proofs of its correct construction to achieve security.

MuSig2 (Nick et al., 2021) and DWMS (Alper and Burdges, 2021) made advances in secure two-round multi-signature schemes with unlimited concurrency and, independently of each other, introduced a technique preventing the Drijvers et al.'s attack. This approach is much more efficient than deterministic nonce derivation with zero-knowledge proofs but it still presents a significant computation overhead, limiting its usefulness for constrained devices.

Crites et al. (Crites et al., 2021) combined MuSig2 with proofs of possession similar to MSDL to construct the latest Schnorr-based scheme called SpeedyMuSig. This combination brings faster key aggregation to the MuSig2 scheme, resulting in the fastest two-round concurrently-secure Schnorr multi-signature scheme.

FROST (Komlo and Goldberg, 2021) is a threshold signature scheme that is secure for an arbitrary threshold $t \leq n$ and, as such, provides great flexibility to its applications. Its original version was also vulnerable to the Drijvers et al.'s attack, but a later version employed a variation of the technique used in MuSig2 (Nick et al., 2021) and DWMS (Alper and Burdges, 2021) to avoid the issue.

Garillot et al. (Garillot et al., 2021) presented another deterministic scheme secure in the dishonest majority setting that has the benefit of not requiring additional randomness nor state. Its construction is conceptually similar to the MuSig-DN scheme, as it uses deterministic nonce derivation supplemented by non-interactive zero-knowledge proofs. The computation of the proof is more efficient than in MuSig-DN, but the proof size and thus communication requirements were significantly increased.

3 INTEROPERABILITY OF SCHNORR-BASED SCHEMES

The Schnorr-based schemes mentioned in Section 2.3 exhibit different trade-offs. Some schemes are optimized for a low number of communication rounds; others are better suited for limited devices where the computation is costly; some use only standard operations available on legacy systems or can utilize dedicated co-processors, and some need to use non-standard cryptographic primitives. As a result, none of the schemes is ideally suitable for all platforms.

In this section, we attempt to address the problem of scheme heterogeneity. We surveyed current multi-party Schnorr signature designs and classified them based on their approach to the nonce agreement. With this classification, we specify what is required of the schemes from the same class to be compatible with each other. Furthermore, we inspect the differences among the classes and bridge them using an untrusted third party without any changes to the underlying schemes. If this mediation is possible, we call the schemes *interoperable*.

More precisely, we define *interoperability* as the ability of two or more multi-party protocols to execute jointly via an untrusted mediator in a way that results in a valid signature on behalf of all of the parties and none of the parties can distinguish such an execution from the execution with its own instances.

With this definition, the security of interoperability can be reduced to the security of individual schemes. Since all of the considered schemes were proven secure in the dishonest majority setting, their security does not rely on the actions of other participants. In particular, the mediator can be considered as the adversary in the security proofs of the schemes.

Nonce Agreement. The method of nonce agreement is the main part in computing multi-party Schnorr signatures. All signing parties need to contribute to the nonce agreement with their fresh nonce, which they later reflect in signing. After the nonce is known, the signatures can be computed non-interactively.

We have identified four main approaches to the nonce agreement: 1) nonce exchange, 2) nonce commitment, 3) nonce delinearization, and 4) deterministic nonce derivation. These methods differ in the number of communication rounds, computational complexity, and security assumptions. In the following subsections, we analyze the approaches and describe the mediator for each interoperable approach.

3.1 Nonce Exchange

Nonce exchange (NE) features two communication rounds and is the simplest and most efficient approach to nonce agreement. It is used by applications focusing on high performance (Syta et al., 2016; Mavroudis et al., 2017), which utilize that its security does not rely on a specific construction of the aggregate nonce. Thus the signers do not even have to compute the aggregate nonce themselves, allowing further decrease of the computation requirements.

The disadvantage of this approach is that it is secure only when executed sequentially, i.e., no concurrent signing sessions occur[1]. Otherwise, a practical message forgery can be achieved by the Drijvers et al.'s attack (Drijvers et al., 2019).

[1]Or more precisely, only a logarithmic number of concurrent sessions occur.

Principle. Each signer i uniformly samples a random nonce $r_i \leftarrow \mathbb{Z}_q$, computes the corresponding element $R_i = r_i G$, and transmits this element. The elements of all signers are then summed up into the aggregate nonce $R = \sum_{i=1}^{n} R_i$ used in the signing.

Interoperability. Since there are no constraints on the construction of the aggregate nonce, NE schemes are convenient for achieving interoperability with other nonce agreement approaches that are more restrictive but concurrently-secure.

3.2 Nonce Commitment

Nonce commitment (NC) is a three-round approach that has been used by MSDL (Boneh et al., 2018) and MuSig (Maxwell et al., 2019). It provides concurrent security with only a minimal computation overhead over NE. If the additional communication round is not too costly, e.g., the devices are co-located, this approach is also quite efficient and suitable for computationally-limited devices.

Principle. The Drijvers et al.'s attack requires the attacker to be able to choose their nonce depending on the nonces of other parties[2]. This precondition can be broken by a preliminary communication round, in which each signer i first outputs a commitment to its nonce element $\mathsf{Com}(R_i)$, and only after receiving commitments of all other parties reveals the nonce element R_i. The provided nonce elements need to be verified against the commitments, and if an inconsistency is discovered, the protocol must be aborted.

Interoperability. The need for commitment limits interoperability among different instances of NC. To be able to work together, the schemes have to use the same Com function, which is a consequence of the commitment properties, preventing the commitment from being readjusted by a third party.

Nonetheless, NC is interoperable with NE-based schemes via a translation layer that simulates the commitment round on behalf of the NE schemes as follows (see Figure 1): First, the NE schemes share their R_i. The translation layer computes commitments for these elements with an appropriate Com function and simulates the commitment round with NC schemes. Afterward, the NC schemes and the translation layer (on behalf of NE schemes) reveal R_i that successfully verify against the commitments. Hence no party aborts and they all arrive at the same aggregate nonce.

[2]Assuming other parts of the input are already fixed.

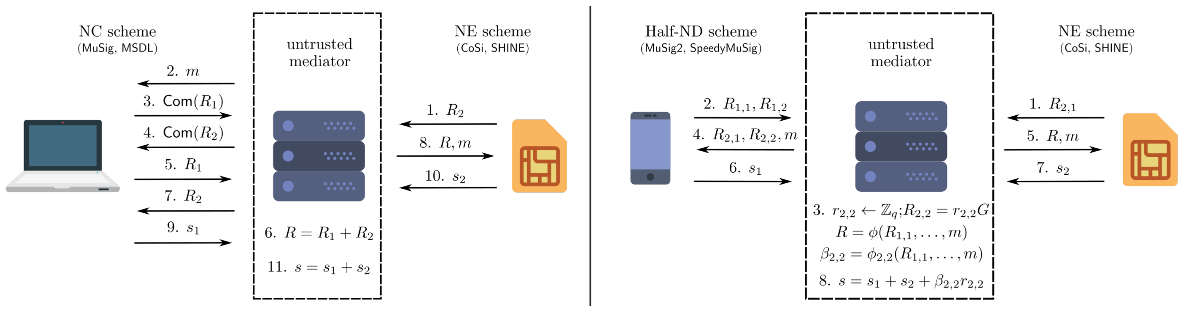

Figure 1: Interoperability mediation between NC and NE schemes (left), and half-ND and NE schemes (right).

3.3 Nonce Delinearization

Nonce delinearization (ND) is the most recent approach to the nonce agreement that is secure under concurrent execution with just two communication rounds, the first of which can be precomputed. The main downside of this approach is its high computational cost, as it requires each signer to generate multiple nonces and to perform multi-scalar multiplication in the second round. Nonetheless, the practical benefits in many applications outweigh the cost, and this technique has been used in the design of the latest schemes (Alper and Burdges, 2021; Komlo and Goldberg, 2021; Nick et al., 2021; Crites et al., 2021).

Principle. Each signer i uniformly samples ν nonces $r_{i,j}$ and reveals the corresponding elements $R_{i,j} = r_{i,j}G$ (possibly in advance). When the message to be signed is known, the nonce elements are used to compute the aggregate nonce

$$R = \sum_{i=1}^{n} \sum_{j=1}^{\nu} \beta_{i,j} R_{i,j},$$

where $\beta_{i,j}$ are *delinearization coefficients*. The delinearization coefficients are non-linearly dependent on all nonce elements and the message via a hash function, which causes the aggregate nonce to change unpredictably whenever any of the inputs changes, and thus thwarts the Drijvers et al.'s attack.

Interoperability. The requirement of specific nonce aggregation based on pre-shared nonce elements and the message limits the interoperability with other instances of ND schemes, as the same coefficients $\beta_{i,j}$ and ν would need to be used. Schemes using, e.g., a different hash function in the coefficient computation, do not arrive at the same aggregate nonce.

Interoperability with NE schemes is a bit more nuanced and cannot be achieved in general. Equations (1) and (2) show a signature by an NE scheme and an ND scheme, respectively ($\nu = 2$ for brevity).

$$s = r_i + ex_i \tag{1}$$

$$s = \beta_{i,1}r_{i,1} + \beta_{i,2}r_{i,2} + ex_i \tag{2}$$

The mediator cannot reconcile the difference between (1) and (2) because it cannot multiply the nonce r_i without also changing the ex_i component. However, if $\beta_{i,1} = 1$, the interoperability is achievable via the following mediation (see Figure 1).

First, all signers begin by sharing their nonce elements. The single nonce element provided by NE schemes is used as their first nonce element, and the mediator computes the other nonce elements instead of the signers. These simulated nonce elements can be sent to ND-based signers, who can now compute the aggregate nonce. The aggregate nonce R is also computed by the mediator who provides it to NE-based signers, which reply with their signatures $s_i = r_i + ex_i$. The mediator then augments the signatures by the simulated nonces $s_i + \beta_{i,2}r_{i,2} + \cdots + \beta_{i,\nu}r_{i,\nu}$, making them compatible with signatures of ND schemes. Finally, the signatures can be combined into a valid signature without any change of the underlying schemes.

Having $\beta_{i,1} = 1$ has been suggested as an optimization of MuSig2 (Nick et al., 2021) that became the default choice in a later revision of the scheme and was since then adopted by other works (Komlo and Goldberg, 2021; Crites et al., 2021). We call this variation, where the first nonce is not multiplied by the coefficient, *half-nonce delinearization*. This variant is not a mere performance optimization (as presented in the original paper), but importantly, it also enables interoperability with NE schemes, as illustrated above. For this reason, we suggest preferring the designs of schemes with half-ND. This choice still allows for adding arbitrarily many nonces to tweak the problem[3] while remaining interoperable with NE schemes and incurring no additional cost to them.

Achieving interoperability is not possible with NC schemes, as those schemes need to receive a commit-

[3] A MuSig2 variant suggested four nonces.

ment to the single nonce produced by each party before revealing their own nonce, but ND schemes use multiple nonces and cannot combine them before all other nonces are known.

3.4 Deterministic Nonce

Deterministic nonce (DN) derivation is another approach to concurrently-secure two-round scheme construction. DN derivation has been used in standard signature schemes, preventing attacks due to biased randomness in the nonce generation. However, it cannot be directly applied to a multi-party setting as a malicious party that diverges from the correct computation could force an honest signer to reuse a nonce, which would result in a key compromise. To deal with this problem, MuSig-DN (Nick et al., 2020) and a scheme by Garrilot et al. (Garillot et al., 2021) use non-interactive zero-knowledge proofs. The disadvantage of this approach is its computational cost, which is the highest among the presented approaches.

Principle. Each signer derives its nonce using a deterministic approach and computes a non-interactive zero-knowledge proof of its correct construction. The proof is output along with the nonce element, and other protocol participants need to verify the proof. If the verification fails, the signing must be aborted.

Interoperability. The need for a zero-knowledge proof severely limits the interoperability of the scheme. Different implementations of DN schemes require a consensus on the used derivation function and the proof construction to work together. Interoperability with schemes using a different approach to the nonce agreement is not possible, as it contradicts the requirement of deterministic nonce derivation.

3.5 Summary

Table 1 displays the interoperability matrix of the nonce agreement approaches. Schemes based on NE stand out among others as the most flexible ones due to their ability to accept an externally provided nonce without any knowledge of its construction and remain secure with a sequential execution. Interactions of NE and ND schemes are interoperable when the half-ND method is used. Even though NE schemes are interoperable with NC schemes and half-ND schemes separately, they cannot be used together since NC schemes and ND schemes are not interoperable. DN is interoperable only with its own instances.

Based on the study of interoperability, we propose to focus on three designs of Schnorr-based schemes:

Table 1: Interoperability of nonce agreement approaches. The icons ✓, ✗, and ∼, denotes possible interoperability always, never and under certain preconditions, respectively.

	NE	NC	ND	DN
NE	✓	✓	∼	✗
NC	✓	∼	✗	✗
ND	∼	✗	∼	✗
DN	✗	✗	✗	∼

1. The sequentially constrained NE schemes for computationally restricted devices, which benefit the most from the efficiency;
2. the half-ND schemes for devices where the computation is not a limiting factor, and that could benefit from interoperability;
3. the DN schemes for the cases where the performance is not an issue, determinism is crucial, and a setup of homogenous instances is guaranteed.

We followed up on our first recommendation and used NE to design SHINE, a multi-signature scheme optimized for cryptographic smartcards and similar resource-constrained devices, which is interoperable with other nonce agreement approaches. We present the design in the next section.

Lastly, it remains to comment that the theoretical interoperability does not automatically imply interoperability of corresponding implementations. For that, the implementations need to use compatible group key aggregation and produce compatible signatures – not only Schnorr signatures but the same instance of Schnorr signatures, e.g., Ed25519 (Bernstein et al., 2012) or BIP-Schnorr (Wuille et al., 2020a).

4 MULTI-SIGNATURE SCHEME SHINE

This section describes SHINE (Smartcard Highly-Interoperable Nonce Encryption scheme) – a multi-signature scheme optimized for computations on cryptographic smartcards while being interoperable with many pre-existing Schnorr multi-signature schemes. The design includes a central party that mediates communication among individual signers. We utilize this central party for the precomputation of inputs, data storage, and also for achieving interoperability, as described in the previous section.

SHINE can create a signature in two communication rounds, the first of which can be securely precomputed. The scheme uses a variant of NE that enables interoperability with all classes except DN. But it also requires sequential execution to be secure, which we enforce by design. Additionally, to avoid randomness generation failure attacks and minimize storage requirements, we derive nonce using a secret pseudorandom function that depends on an internal counter. Security proof is provided in the Appendix.

In the following subsections, we describe the attacker model and the group establishment of SHINE. Next, we introduce the technique of nonce caching with encryption and discuss its differences from plain nonce caching. Finally, in Section 4.4 we describe the signing protocol. The last subsection presents an implementation of SHINE on cryptographic smartcards and an evaluation of its performance.

4.1 Attacker Model

We assume that the attacker is able to control the central party and $n-1$ of the signing parties. Since the central party is under the control of the attacker, it can drop or alter messages, and as a result, cause a denial of service. We also make the standard assumptions that the number of computation steps the attacker can make is bound by a polynomial and that a variant of the discrete logarithm problem is hard.

4.2 Group Establishment

Before SHINE can be used to sign messages, the signing group needs to be established. The signing group consists of a set of signers, who generate their private key shares, compute the group key, and initialize context information. In this process, we assume that the central party included all participants in the key aggregation, i.e., did not ignore or simulate messages by some parties. In practice, the validity of this assumption can be later verified by querying participants for their key contributions by secondary channels.

SHINE can use different key aggregation approaches depending on which schemes it should be interoperable with. As the default option, we have selected proofs of possession like in (Crites et al., 2021), as it allows efficient group key aggregation and low communication. Alternatively, MuSig key aggregation (Maxwell et al., 2019) or some form of distributed key generation can be used instead.

After a new group key is successfully computed, each party i finalizes the group establishment process by initializing its signing context. The signing context consist of a λ-bit secret p_i that is used for deterministic nonce and encryption key derivation, and an increase-only counter c_i that tracks the index of the most recently used nonce. The former value is uniformly sampled, and the latter is initially set to zero. These values are later used in the signing protocol.

4.3 Nonce Caching with Encryption

Nonce caching is a technique used in Myst (Mavroudis et al., 2017) for a setup of smartcards. It optimizes the signing speed by generating nonces in advance, computing the corresponding elements, and storing them at a server. Thus the costly scalar multiplication can be performed during downtime when there are no signing requests, and the precomputed nonce can be provided to signers when needed.

This approach was later shown to be vulnerable to the Drijvers et al.'s attack (Drijvers et al., 2019). Since the properties of nonce caching are beneficial for resource-limited devices like smartcards, we designed a technique called *nonce caching with encryption* or *nonce encryption* for short, which avoids the Drijvers et al.'s attack and still allows for nonce caching with some restrictions.

The vulnerability to the Drijvers et al.'s attack occurs only if the scheme is used concurrently, e.g., an adversary can open multiple signing sessions in parallel or cache multiple nonces before they are used for signing. Nonce encryption leverages this property and enforces sequential execution to ensure at most a single cached nonce is revealed at a time while still being usable for signing.

Nonce encryption consists of two phases – Cache (Figure 2) and Reveal (Figure 3). The smartcard computes the nonce element during the caching phase and sends it encrypted[4] using a fresh key to the central party. The reveal phase is then used to reveal the corresponding decryption key and thus reveal the cached nonce while invalidating previous nonces.

The Cache phase is initiated by the central party, which sends a signing instance identifier j. The smartcard i uses a pseudorandom function, keyed with a secret p_i (generated during the group establishment), to derive a nonce $r_{i,j}$ corresponding to the signing instance j. Then the smartcard computes $R_{i,j} \leftarrow r_{i,j}G$, the demanding operation that would be the bottleneck during the signing.

In the standard nonce caching, $R_{i,j}$ would be simply transmitted to the central party; however, if this step was repeated, it would lead to the vulnerability to the Drijvers et al.'s attack. With nonce encryption, the nonce is not transmitted in plaintext but rather encrypted with symmetric key $k_{i,j}$ derived from the signing instance identifier j using a key derivation function known only to the signer i. The central party stores the received value for future use.

[4]We use encryption only to securely store the cached element on the central party to minimize storage requirements on the signing device (as memory on smartcards is limited). We do not want the signer to bind to the encrypted value.

```
Algorithm 1: Cache
    Input: Session index $j$
    Output: Encrypted nonce $E_{i,j}$
    $r_{i,j} \leftarrow \mathsf{PRF}_{p_i}(j)$
    $R_{i,j} \leftarrow r_{i,j}G$
    $k_{i,j} \leftarrow \mathsf{KDF}_{p_i}(j)$
    $E_{i,j} \leftarrow \mathsf{Enc}_{k_{i,j}}(R_{i,j})$
    return $E_{i,j}$
```

Figure 2: SHINE nonce caching algorithm.

```
Algorithm 2: Reveal
    Input: Session index $j$
    Output: Decryption key $k_{i,j}$
    if $j \geq c_i$ then $c_i \leftarrow j$ fi
    $k_{i,j} \leftarrow \mathsf{KDF}_{p_i}(c_i)$
    return $k_{i,j}$
```

Figure 3: SHINE nonce revealing algorithm.

The Reveal phase uses the smartcard internal increase-only counter c_i to track which nonce elements were already revealed. When prompted by the central party to send $k_{i,j}$ for a signing instance $j \geq c_i$, the smartcard derives $k_{i,j}$ and sends it back, but at the same time increases its inner counter $c_i \leftarrow j$. With the knowledge of $k_{i,j}$, the central party can decrypt the cached nonce $R_{i,j}$, possibly combine it with nonces of other parties, and use it in a subsequent signing.

So far, the counter c_i did not limit the attacker in any way. It plays a role only during the signing, where the smartcard must not produce a signature for a signing instance $j < c_i$. This restriction ensures that only the signing instance $j = c_i$ can succeed since nonces for $j > c_i$ are not known yet. The technique already enforces the signing of non-decreasing sequences, and it only remains to avoid nonce reuse, which is achieved by incrementing the internal counter c_i when producing a signature.

Comparison to Plain Nonce Caching

Just as in plain nonce caching, we manage to avoid costly group operations during the signing. In this subsection, we highlight the differences.

Storage-wise, nonce encryption still requires only constant memory on smartcards, but the central party cannot pre-aggregate nonces anymore as only a single nonce is known at a time. Therefore, the space required by the central party grows linearly in n.

Communication-wise, nonce encryption seemingly needs one additional round to transmit the decryption key. However, this transmission can be automatically piggybacked with the previous signing, where the decryption key for $c_i + 1$ can be revealed, as c_i was already invalidated. As a result, the number of communication rounds can remain the same; only the additional decryption key is transmitted.

```
Algorithm 3: Sign
    Input: Aggregate nonce $R_j$, message $m$,
           session index $j$
    Output: Partial signature $s_{i,j}$
    if $j < c_i$ then abort fi
    $c_i \leftarrow j + 1$
    $r_{i,j} \leftarrow \mathsf{PRF}_{p_i}(j)$
    $s_{i,j} \leftarrow r_{i,j} + H(R_j, X, m)x_i$
    return $s_{i,j}$
```

Figure 4: SHINE signing algorithm.

Finally, computation-wise, symmetric encryption can be realized efficiently, minimizing the additional demands on the smartcards. The only new concern is the aggregate nonce computation by the central party – which cannot be precomputed, as only a single decryption key is known at a time. In the case of burst signing requests, the overall solution would result in a performance decrease compared to the vulnerable nonce caching. But assuming the central party is significantly more capable than the smartcards (which in practice is), the aggregation can be performed relatively quickly and is not a limiting factor.

4.4 Signing Protocol

Figure 4 shows the signing algorithm. The central party initiates the signing of a message m by computing the aggregate nonce. If it already has all appropriate decryption keys, it sums the cached nonces as $R_j = \sum_{i=1}^{n} R_{i,j}$. Otherwise, the central party might need to exchange an additional Cache or Reveal message with some smartcard(s) prior to the aggregation. When the aggregate nonce is computed, the central party sends a signature request to every participating smartcard with the given signing index j, the aggregate nonce R_j, and the message m.

When a smartcard receives a signing request, it first checks whether the signing index is greater or equal to its internal counter. If not, it aborts the protocol. Otherwise, the smartcard sets its internal counter $c_i \leftarrow j + 1$ and continues with the signing. It derives its nonce $r_{i,j}$ and computes its signature $s_{i,j} \leftarrow r_{i,j} + H(R_j, X, m)x_i$, which it outputs. Optionally, the decryption key for index $j+1$ can be revealed together with the signature. Finally, the central party sums the signatures to obtain the resulting (R_j, s_j).

4.5 Implementation and Evaluation

We implemented SHINE for the JavaCard platform using a modified version of the JCMathLib library (Mavroudis and Svenda, 2020) that provides the necessary low-level operations without relying on proprietary smartcard API and thus enables our results to be reproduced on a wide variety of supported smartcards. While the solution achieves a decent speed with JCMathLib, using proprietary API calls would significantly increase its performance in practice, especially for nonce computation and caching operations.

We tested and evaluated our implementation on five JavaCards: (1) `NXP J2E145G`, (2) `NXP J3H145`, (3) `GD SmartCafe 6.0`, (4) `GD SmartCafe 7.0`, and (5) `NXP J3R180`. We measured the time required to compute each phase of the protocol and also their counterparts in a variant without nonce caching so that we would be able to assess its impact. We repeated each measurement 100 times and averaged the results. Table 2 presents a summary of the results.

The implementation achieves an average signing speed of around 700 ms, comparable to the performance of the Myst implementation from (Mavroudis et al., 2017). The signing slowdown caused by key derivation for piggybacking did not exceed 45 ms for any of the cards and thus increased signing latency by 6% at most. Encryption operation added to the nonce caching resulted in less than 36 ms slowdown, which was at most 23% (but mostly only 1%) of the time required to perform the nonce computation.

The differences in time required to compute the nonce are caused by different native algorithm support. JavaCards (2) and (5) supported `ALG_EC_SVDP_DH_PLAIN_XY` algorithm, which made the nonce computation quite close to the native performance of the hardware, while on other cards, additional transformations had to be made. This computation is the precomputed part; thus, it influences the speedup over signing without caching the most.

Table 2: Time (ms) required to compute steps of SHINE on different JavaCards.

	(1)	(2)	(3)	(4)	(5)
Compute	2826	194	2764	1962	60
+ Cache	2854	217	2801	1984	74
Overhead	28	24	36	23	14
(%)	1	12	1	1	23
Sign	802	737	768	637	457
+ Reveal	842	756	813	660	472
Overhead	28	19	45	23	15
(%)	5	3	6	4	3
W/O cache	3627	931	3532	2599	518
W/ cache	842	756	813	660	472
Speedup	2785	175	2719	1938	46
(%)	77	19	77	75	9

We estimate that implementing the signing part using native low-level operations instead of slower software emulation via JCMathLib would achieve performance comparable to ECDSA on a given platform, e.g., around 200 ms for common smartcards (Dzurenda et al., 2017). Nonce caching with encryption would provide an overall speedup of around 50% in such implementations.

Our implementation of SHINE uses a central party (written in Rust), which mediates communication among different devices and serves as a storage of the cached nonces. It contains NE, NC, and ND schemes, which can be used by themselves or jointly with the SHINE applet, demonstrating its interoperability. Both open-source implementations are available in GitHub repositories[5].

5 CONCLUSION

We studied the possibility of interoperability of Schnorr-based multi-party signature schemes and classified the existing schemes based on their approach to the nonce agreement – a key component in signature computation. We identified four classes of nonce agreement approaches – nonce exchange (NE), nonce commitment (NC), nonce delinearization (ND), and deterministic nonce derivation (DN).

NE-based schemes are secure only with limited concurrency, but they are the most efficient and the most interoperable. For these reasons, we suggest that schemes of this class should be considered for the use-cases where computation is costly and sequential execution is acceptable. One such case includes cryptographic smartcards, for which we designed a new multi-signature scheme called SHINE.

Schemes based on DN are the most rigid, the least interoperable, and the most computationally demanding. However, in return for these downsides, they gain the benefits of stateless execution and no need for additional randomness. Therefore, these schemes should be considered for the scenarios where computation is not a limiting factor, the sources of randomness are limited, and a setup of homogeneous implementations can be guaranteed.

ND schemes schemes lie in the middle between NE and DN with regard to both their computational

[5]https://github.com/crocs-muni/SHINE/
https://github.com/crocs-muni/SHINE-mediator/

complexity and their interoperability. In the half-ND variant, these schemes are interoperable with NE schemes without compromising their concurrent security. We thus propose the half-ND schemes to be considered in the most common scenario featuring reasonably fast devices like smartphones, which could benefit from interoperability with more constrained devices, while enjoying the benefits of security under concurrent execution.

The class of NC-based schemes is the only one that currently does not display any apparent additional utility. It provides concurrent security achievable even for computationally restricted devices, but such devices could easily rely on the more efficient NE, which would also allow them to be interoperable with half-ND schemes.

We designed the scheme SHINE to benefit from interoperability while being efficiently executable on cryptographic smartcards. The scheme is based on NE complemented by a novel approach to nonce caching – featuring encryption and avoiding the previous attack by Drijvers et al. (Drijvers et al., 2019).

We implemented SHINE as an applet for the JavaCard platform and evaluated its performance on five different smartcard models. The experiments empirically confirmed the performance improvement of nonce caching on computationally restricted devices over a variant without caching. Furthermore, we provided a Rust implementation of the central party that practically demonstrates the interoperability of SHINE with NE, NC, and half-ND schemes.

We view the interoperability of multi-party protocols as a meaningful and practical way to increase resilience and flexibility of multi-party systems. Following this path, future research could investigate the possibility of interoperability among other types of multi-party protocols, e.g., the recent designs of threshold ECDSA signatures.

ACKNOWLEDGEMENTS

Authors were supported by Czech Science Foundation project GA20-03426S. V. Sedlacek was also supported by the Ph.D. Talent Scholarship – funded by the Brno City Municipality.

REFERENCES

Alper, H. K. and Burdges, J. (2021). Two-round trip schnorr multi-signatures via delinearized witnesses. In *Annual International Cryptology Conference*. Springer.

Bagherzandi, A., Cheon, J.-H., and Jarecki, S. (2008). Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 449–458.

Bellare, M., Namprempre, C., Pointcheval, D., and Semanko, M. (2003). The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *Journal of Cryptology*, 16(3).

Benhamouda, F., Lepoint, T., Loss, J., Orru, M., and Raykova, M. (2021). On the (in) security of ros. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 33–53. Springer.

Bernstein, D. J., Duif, N., Lange, T., Schwabe, P., and Yang, B.-Y. (2012). High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89.

Boneh, D., Drijvers, M., and Neven, G. (2018). Compact multi-signatures for smaller blockchains. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 435–464. Springer.

Crites, E., Komlo, C., and Maller, M. (2021). How to prove schnorr assuming schnorr: Security of multi- and threshold signatures. *Cryptology ePrint Archive*.

Drijvers, M., Edalatnejad, K., Ford, B., Kiltz, E., Loss, J., Neven, G., and Stepanovs, I. (2019). On the security of two-round multi-signatures. In *IEEE Symposium on Security and Privacy*, pages 1084–1101.

Dzurenda, P., Ricci, S., Hajny, J., and Malina, L. (2017). Performance analysis and comparison of different elliptic curves on smart cards. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pages 365–374. IEEE.

Fiat, A. and Shamir, A. (1986). How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer.

Garillot, F., Kondi, Y., Mohassel, P., and Nikolaenko, V. (2021). Threshold schnorr with stateless deterministic signing from standard assumptions. In *Annual International Cryptology Conference*, pages 127–156. Springer.

IANIX (2022). Things that use Ed25519. https://ianix.com/pub/ed25519-deployment.html. Accessed: 2022-02-01.

Komlo, C. and Goldberg, I. (2021). Frost: Flexible round-optimized schnorr threshold signatures. In Dunkelman, O., Jacobson, Jr., M. J., and O'Flynn, C., editors, *Selected Areas in Cryptography*, pages 34–65, Cham. Springer International Publishing.

Mavroudis, V., Cerulli, A., Svenda, P., Cvrcek, D., Klinec, D., and Danezis, G. (2017). A touch of evil: High-assurance cryptographic hardware from untrusted components. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1583–1600.

Mavroudis, V. and Svenda, P. (2020). Jcmathlib: Wrapper cryptographic library for transparent and certifiable javacard applets. *IEEE European Symposium on Security and Privacy Workshops*, pages 89–96.

Maxwell, G., Poelstra, A., Seurin, Y., and Wuille, P. (2019). Simple Schnorr multi-signatures with applications to Bitcoin. *Designs, Codes and Cryptography*, 87(9):2139–2164.

Morita, H., Schuldt, J. C., Matsuda, T., Hanaoka, G., and Iwata, T. (2015). On the security of the schnorr signature scheme and dsa against related-key attacks. In *Information Security and Cryptology - ICISC 2015*, pages 20–35. Springer.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf. Accessed: 2022-02-01.

Nick, J., Ruffing, T., and Seurin, Y. (2021). Musig2: Simple two-round schnorr multi-signatures. In *CRYPTO*, pages 189–221. Springer.

Nick, J., Ruffing, T., Seurin, Y., and Wuille, P. (2020). Musig-dn: Schnorr multi-signatures with verifiably deterministic nonces. In *27th ACM CCS*, pages 1717–1731.

Pointcheval, D. and Stern, J. (2000). Security arguments for digital signatures and blind signatures. *Journal of cryptology*, 13(3):361–396.

Schnorr, C. P. (1991a). Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174.

Schnorr, C. P. (1991b). Method for identifying subscribers and for generating and verifying electronic signatures in a data exchange system. US Patent 4,995,082.

Schnorr, C. P. (2001). Security of blind discrete log signatures against interactive attacks. In *International Conference on Information and Communications Security*, pages 1–12. Springer.

Syta, E., Tamas, I., Visher, D., Wolinsky, D. I., Jovanovic, P., Gasser, L., Gailly, N., Khoffi, I., and Ford, B. (2016). Keeping authorities "honest or bust" with decentralized witness cosigning. In *IEEE Symposium on Security and Privacy*, pages 526–545.

Wagner, D. (2002). A generalized birthday problem. In *Annual International Cryptology Conference*, pages 288–304. Springer.

Wuille, P., Nick, J., and Towns, A. (2020a). Schnorr signatures for secp256k1. https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki. Accessed: 2022-02-01.

Wuille, P., Nick, J., and Towns, A. (2020b). Taproot: Segwit version 1 spending rules. https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki. Accessed: 2022-02-01.

APPENDIX

The Security of SHINE

We aim to reduce the security of SHINE (Figure 5) to the OMDL problem; we use the following algorithms:

- GrGen that outputs a group description $(\mathbb{G}, q, G)$ for a given security parameter λ,
- KeyGen that outputs a standard, uniformly sampled key pair $(x_1, x_1 G)$,
- KeyAgg that performs key aggregation according to the selected group establishment method,
- Verify that performs standard Schnorr signature verification.

The security reduction relies on the *one-more discrete logarithm* (OMDL) problem (Bellare et al., 2003): the adversary has access to an oracle O^{DLog} providing them with up to q_d discrete logarithms, while the goal is presenting discrete logarithms of $q_d + 1$ challenges received from the O^{Chall} oracle.

Theorem 1. If a polynomial adversary $\mathcal{A}$ against the SHINE unforgeability game (Fig. 5) wins with probability ε while making q_S reveal oracle queries and $q_H - q_S$ random oracle queries, then the OMDL problem can be solved in polynomial time with probability ε' in the ROM with the KOSK assumption, such that:

$$\varepsilon' \geq \frac{\varepsilon}{8 q_H q_S} - \frac{q_S^2 k_{\max}^2}{q},$$

where $k_{\max} = 8q_H/\varepsilon \cdot \ln(8n/\varepsilon)$.

Proof. The proof adapts the main idea of the reduction given by Drijvers et al. (Drijvers et al., 2019) for the restricted version of CoSi. Given the adversary $\mathcal{A}$ from the theorem statement, we construct an algorithm $\mathcal{B}$ that simulates the SHINE unforgeability game for $\mathcal{A}$, and succeeds when it does not abort and $\mathcal{A}$ succeeds. The algorithm $\mathcal{B}$ is constructed in a way compatible with the forking lemma, which we then apply to $\mathcal{B}$ to solve the OMDL problem.

The algorithm $\mathcal{B}$ is started with elements from $\mathbb{Z}_q$ for the random oracle outputs $(h_1, \ldots, h_{q_H})$ and is given access to oracles O^{Chall} and O^{DLog}. We sequentially order the O^{Reveal} and O^{H} oracle queries made by $\mathcal{A}$. If the k-th query is a random oracle query, $\mathcal{B}$ replies with h_k. If k-th query is a reveal query, we set $k_j = k$, where j is the input to the query. We also modify O^{Chall} to accept an input element from $\mathbb{Z}_q$. When the oracle is queried on a previously unseen element, it invokes its inner O^{Chall} oracle; otherwise, it outputs the previously output value. Without loss of generality, we assume that $\mathcal{A}$ makes all random oracle queries needed to verify the produced signatures.

First, $\mathcal{B}$ makes a minor modification of nonce caching in the simulation, allowing us to defer nonce sampling to the time when the corresponding decryption key should be revealed – as per SHINE's sequential nature. The change is sampling a random string

```
$\mathsf{Game}^{\mathsf{EUF-CMA}}_{\mathcal{A}}(\lambda)$
$(\mathbb{G}, q, G) \leftarrow \mathsf{GrGen}(1^{\lambda})$
$p_1 \leftarrow \mathbb{Z}_q$; $c \leftarrow 0$; $Q \leftarrow \{\}$
$(x_1, X_1) \leftarrow \mathsf{KeyGen}()$
$X_2, \ldots, X_n \leftarrow \mathcal{A}(X_1)$
$X \leftarrow \mathsf{KeyAgg}(X_1, \ldots, X_n)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{O^{\mathsf{Cache,Reveal,Sign,H}}}(X_1)$
**return** $m^* \notin Q \wedge \mathsf{Verify}(X, m^*, \sigma^*)$
```

```
$O^{\mathsf{Cache}}(j)$
$r_{1,j} \leftarrow \mathsf{PRF}_{p_1}(j)$
$R_{1,j} \leftarrow r_{1,j}G$
$k_{1,j} \leftarrow \mathsf{KDF}_{p_1}(j)$
$E_{1,j} \leftarrow \mathsf{Enc}_{k_{1,j}}(R_{1,j})$
**return** $E_{1,j}$
```

```
$O^{\mathsf{Reveal}}(j)$
**if** $j \geq c$ **then**
    $c \leftarrow j$
**fi**
$k_{1,j} \leftarrow \mathsf{KDF}_{p_1}(c)$
**return** $k_{1,j}$
```

```
$O^{\mathsf{Sign}}(R, m, j)$
**if** $j < c$ **then**
    **abort**
**fi**
$c \leftarrow j + 1$
$r_{1,j} \leftarrow \mathsf{PRF}_{p_1}(j)$
$s_{1,j} \leftarrow r_{1,j} + H(R, X, m)x_1$
$Q \leftarrow Q \cup \{m\}$
**return** $s_{1,j}$
```

Figure 5: The SHINE unforgeability game. O^{H} is the random oracle.

$E_{1,j}$ in the O^{Cache} oracle instead of computing and encrypting $R_{1,j}$. The value $R_{1,j}$ is computed only when the corresponding decryption key should be revealed, i.e., in the corresponding O^{Reveal} call, and the decryption key is chosen so that the decrypted value corresponds to the sampled $R_{1,j}$. For that, we need the encryption to be non-binding. This change is indistinguishable by $\mathcal{A}$, does not introduce additional aborts, and $\mathcal{B}$ succeeds whenever $\mathcal{A}$ succeeds.

Another change that $\mathcal{B}$ makes is setting an output of O^{Chall} as X_1, instead of using the KeyGen algorithm. Because of this change, x_1 is not known anymore, but we still need to be able to correctly simulate the signing queries, which we do as follows.

Before running $\mathcal{A}$, $\mathcal{B}$ tosses a biased coin coin_j that turns out 1 with probability $1/q_S$ for every signing instance $j \in \{1, \ldots, q_S\}$. Whenever $\mathsf{coin}_j = 1$, $\mathcal{B}$ samples $g_j \in \{1, \ldots, q_H\}$. The simulation of O^{Reveal} and O^{Sign} oracles for session j then depends on coin_j.

If $\mathsf{coin}_j = 0$, $\mathcal{B}$ invokes $O^{\mathsf{Chall}}(h_{k_j})$ to get the element $R_{1,j}$ when revealing the decryption key. To successfully answer a signing query, the simulator queries its O^{DLog} oracle with $R_{1,j} + H(R, X, m)X_1$.

If $\mathsf{coin}_j = 1$, $\mathcal{B}$ samples a random $s_{1,j}$ from $\mathbb{Z}_q$ and computes $R_{1,j} = s_{1,j}G - h_{g_j}X$ when revealing the decryption key. The signing query might not always succeed in this case. If $H(R, X, m) \neq h_{g_j}$ for the input provided in the signing query, $\mathcal{B}$ aborts. Otherwise, it outputs $s_{1,j}$ in the corresponding O^{Sign} call.

Due to SHINE's design, at most one corresponding O^{Sign} query has not been made, nor has the session been invalidated by querying O^{Sign} with a higher index. If a session was open when the query for the forgery was made and $\mathsf{coin}_j = 0$, $\mathcal{B}$ aborts.

When $\mathcal{B}$ does not abort, the changes are indistinguishable by $\mathcal{A}$, and $\mathcal{B}$ succeeds whenever $\mathcal{A}$ succeeds. $\mathcal{B}$ does not abort when the session index is guessed correctly, which happens with probability $1/q_S$, and the corresponding hash oracle query is also guessed correctly, which happens with probability $1/q_H$. Furthermore, $\mathcal{B}$ does not abort when no session was open when the fork occurred. From that, we get that the success probability of $\mathcal{B}$ is at least $\varepsilon/q_S q_H$.

Now, we apply the Generalized Forking Lemma (Bagherzandi et al., 2008) to $\mathcal{B}$. By the construction of $\mathcal{B}$, no challenge that is still being usable for signing is known at the time of the fork; thus, the application of the forking lemma does not result in multiple O^{DLog} queries for the same challenge. If the forking of $\mathcal{B}$ succeeds and it outputs two forgeries s and s' for hashes h and h', respectively, we use them to solve the OMDL problem in the following way. First, we compute the discrete logarithm of the group key X as $x = (s - s')/(h - h')$. In the KOSK model, we can compute the discrete logarithm of X_1 directly from x, as $\mathcal{A}$ has to output its private keys $(x_2, \ldots, x_n)$, and x is a linear combination of $(x_1, \ldots, x_n)$. To compute answers to the other challenges, we use x_1 to solve linear equations in the form $r_{1,j} = s_{1,j} - H(R_j, X, m)x_1$; if some outputs of O^{Chall} were not used during signing, we apply O^{DLog} on them.

The probability of solving the OMDL is the probability that the forking succeeded and no collisions occurred among any of the q_S values that $\mathcal{B}$ submits to its O^{Chall} in any of its runs. Thus, we have

$$\varepsilon' \geq \frac{\varepsilon}{8 q_H q_S} - \frac{q_S^2 k_{\max}^2}{q}.$$

□

We needed the KOSK model only at the end of the proof – to extract the private key x_1. The KOSK model can be avoided in settings with a separate group establishment where the group key is fixed prior to issuing signatures; and the proof can be extended to MuSig-like and proof of possession group key aggregation by extracting the key correspondingly.