Non-interactive half-aggregation of EdDSA and variants of Schnorr signatures

Konstantinos Chalkias[1], François Garillot[1], Yashvanth Kondi *[2], and Valeria Nikolaenko[1]

[1]Novi/Facebook
[2]Northeastern University

Abstract

Schnorr's signature scheme provides an elegant method to derive signatures with security rooted in the hardness of the discrete logarithm problem, which is a well-studied assumption and conducive to efficient cryptography. However, unlike pairing-based schemes which allow arbitrarily many signatures to be aggregated to a single *constant* sized signature, achieving significant non-interactive compression for Schnorr signatures and their variants has remained elusive. This work shows how to compress a set of independent EdDSA/Schnorr signatures to roughly half their naive size. Our technique does not employ generic succinct proofs; it is agnostic to both the hash function as well as the specific representation of the group used to instantiate the signature scheme. We demonstrate via an implementation that our aggregation scheme is indeed practical. Additionally, we give strong evidence that achieving better compression would imply proving statements specific to the hash function in Schnorr's scheme, which would entail significant effort for standardized schemes such as SHA2 in EdDSA. Among the others, our solution has direct applications to compressing Ed25519-based blockchain blocks because transactions are independent and normally users do not interact with each other.

1 Introduction

Schnorr's signature scheme [59] is an elegant digital signature scheme whose security is rooted in the hardness of computing discrete logarithms in a given group. Elliptic curve groups in particular have found favour in practical instantiations of Schnorr as they are secured by conservative well-studied assumptions, while simultaneously allowing for fast arithmetic. One

*Yashvanth Kondi did part of this work during an internship at Novi/Facebook.

such instantiation is the EdDSA signature scheme [11], which is deployed widely across the internet (in such protocols as TLS 1.3, SSH, Tor, GnuPGP, Signal and more).

However, the downside of cryptography based on older assumptions is that it lacks the functionality of modern tools. In this work, we are concerned with the ability to *aggregate* signatures without any prior interaction between the signers. Informally speaking, an aggregate signature scheme allows a set of signatures to be compressed into a smaller representative unit, which verifies only if all of the signatures used in its generation were valid. Importantly, this aggregation operation must not require any secret key material, so that any observer of a set of signatures may aggregate them. Quite famously, pairing-based signatures [15, 13] support compression of an arbitrary number of signatures into a constant sized aggregate. Thus far, it has remained unclear how to achieve any sort of non-trivial non-interactive compression for Schnorr signatures without relying on generic tools such as SNARKs.

In order to make headway in studying how to compress Schnorr signatures, we loosely cast this problem as an issue of *information optimality*. We first recap the structure of such a signature in order to frame the problem.

Structure of Schnorr signatures. Assume that we instantiate Schnorr's signature scheme in a group $(\mathbb{G}, +)$ with generator $B \in \mathbb{G}$ of prime order q.

Algorithm 1 Schnorr signature scheme in (R, S)-format

$\mathsf{KeyGen}()$**:** sample $s \xleftarrow{\$} \mathbb{Z}_q$, output $\mathsf{sk} = s$ and $\mathsf{pk} = s \cdot B$.

$\mathsf{Sign}(\mathsf{sk}, m)$**:** sample $r \xleftarrow{\$} \mathbb{Z}_q$, compute $R = r \cdot B$ and $S = r + H_0(R, A, m) \cdot s$, output $\sigma = (R, S)$.

$\mathsf{Verify}(m, \mathsf{pk}, \sigma)$**:** for $\sigma = (R, S)$ and $\mathsf{pk} = A$ accept if $S \cdot B = R + H_0(R, A, m) \cdot A$.

In practice, the groups that are used to instantiate Schnorr signatures are elliptic curves which are believed to be 'optimally hard', i.e. no attacks better than generic ones are known for computing discrete logarithms in these curves groups. Consequently, elliptic curve based Schnorr signatures are quite compact: at a λ-bit security level, instantiation with a 2λ-bit curve yields signatures that comprise only 4λ bits (ignoring a few bits of security loss due to the specific representation of the curve).

This format of Schnorr signature is employed by EdDSA [11]. The EdDSA signature is originally defined over Curve25519 group in its twisted Edwards form. The variant known as Ed25519 provides ~ 128 bits of security and is the most widely deployed variant of Schnorr today. We use this instantiation for benchmarks. The name EdDSA generally refers to instantiation of the scheme over any compatible elliptic curve (another notable instantiation being Ed448 [36, 41] offering ~ 224 bits of security). A concrete instantiation of the scheme would depend on the elliptic curve and the security level, we show a generic instantiation of EdDSA in Algorithm 2. The main difference between Algorithm 1 and Algorithm 2 is in derivation of the secret scalar r, in the latter r is derived deterministically, while in the

former it is sampled randomly. The verification equation is unchanged making techniques in this paper applicable to both algorithms.

Algorithm 2 EdDSA signature scheme

$\mathsf{KeyGen}()$**:** sample uniformly random secret key $\mathsf{sk} \xleftarrow{\$} \{0,1\}^{2\lambda}$, expand the secret with a hash function that gives 4λ-bits outputs: $(s,k) \leftarrow H_1(\mathsf{sk})$, interpret s as a scalar compute $A = s \cdot B$ and output $(\mathsf{sk}, \mathsf{pk} = A)$.

$\mathsf{Sign}(\mathsf{sk}, m)$**:** for $\mathsf{sk} = (s,k)$ and $\mathsf{pk} = A$ derived from sk, generate a pseudorandom secret scalar $r := H_2(k,m)$, compute a curve point: $R := r \cdot B$, compute a scalar $S := (r + H_0(R,A,M) \cdot s)$ and output $\sigma = (R,S)$.

$\mathsf{Verify}(m, \mathsf{pk}, \sigma)$**:** for $\sigma = (R,S)$ and $\mathsf{pk} = A$, accept if $S \cdot B = R + H_0(R,A,M) \cdot A$.

The original form of Schnorr signatures are slightly different: $\sigma = (H(R,\mathsf{pk},m), S)$, the verification rederives R and verifies the hash. Schnorr signatures of this format can be shortened by a quarter via halving the output of the hash function [50, 59], but this format does not allow for half-aggregation (nor is it implemented in widely deployed standards such as EdDSA), *thus we focus on Schnorr-type signatures in the* (R,S) *format.* We explain most of the popular Schnorr variants in Section 2.3 discussing compatibility with our aggregation approach.

Schnorr signatures are not information optimal. Given a fixed public key, a fresh Schnorr signature carries only 2λ bits of information. Indeed for a 2λ-bit curve, there are only $2^{2\lambda}$ pairs of accepting (R,S) tuples. It seems unlikely that we can achieve an information-optimal representation for a single signature[1]. However we can not rule out this possibility when transmitting a larger number of signatures. Transmitting n Schnorr signatures at a λ-bit security level naively requires $4n\lambda$ bits, whereas they only convey $2n\lambda$ bits of information. Therefore we ask:

> How much information do we need to transmit in order to aggregate the effect of n Schnorr signatures?

We specify that we are only interested in aggregation methods that are agnostic to the curve and the hash function used for Schnorr - in particular aggregation must only make oracle use of these objects. This is not merely a theoretical concern, as proving statements that depend on the curve or code of the hash function can be quite involved in practice.

1.1 Our contributions

This work advances the study of non-interactive compression of Schnorr signatures.

[1]Even for shortened Schnorr signatures $\sigma = (H(R,\mathsf{pk},m), S)$, where the output of the hash function is halved, signatures are at least 3λ bits, i.e. 50% larger than the amount of information they carry.

Simple half-aggregation. We give an elegant construction to aggregate n Schnorr signatures over 2λ bit curves by transmitting only $2(n+1)\lambda$ bits of information - i.e. only *half* the size of a naive transmission. This effectively cuts down nearly all of the redundancies in naively transmitting Schnorr signatures. Our construction relies on the Forking Lemma for provable security and consequently suffers from a quadratic security loss similar to Schnorr signatures themselves. Fortunately, this gap between provably secure and actually used parameters in practice has thus far not been known to induce any attacks. We also show how this aggregation method leads to a deterministic way of verifying a batch of Schnorr signatures.

Almost half-aggregation with provable guarantees. In light of the lossy proof of our half-aggregation construction, we give a different aggregation scheme that permits a *tight* reduction to the unforgeability of Schnorr signatures. However this comes at higher cost, specifically $2(n+\epsilon)\lambda$ bits to aggregate n signatures where $\epsilon \in O(\lambda/\log\lambda)$ is independent of n. This construction is based on Fischlin's transformation [29], and gives an uncompromising answer to the security question while still retaining reasonable practical efficiency. More concretely the compression rate of this construction passes 40% as soon as we aggregate 128 signatures, and tends towards the optimal 50% as n increases.

Implementations. We implement and comprehensively benchmark both constructions. We demonstrate that the simple half-aggregation construction is already practical for wide adoption, and we study the performance of our almost half-aggregation construction in order to better understand the overhead of provable security in this setting.

A lower bound. Finally, we give strong evidence that it is not possible to achieve non-trivial compression beyond $2n\lambda$ bits without substantially higher computation costs, i.e. our half-aggregation construction is essentially optimal as far as generic methods go. In particular, we show that aggregating Schnorr signatures from different users (for which no special distribution is fixed ahead of time) at a rate non-trivially better than 50% must necessarily be non-blackbox in the hash function used to instantiate the scheme.

In summary, we propose a lightweight half-aggregation scheme for Schnorr signatures, a slightly worse performing scheme which settles the underlying theoretical question uncompromisingly, and finally strong evidence that achieving a better compression rate is likely to be substantially more computationally expensive.

2 Related work

2.1 Security Proofs

Schnorr signatures were proposed by Claus Schnorr [59], and in the original paper a compact version was proposed, which outputted signatures of size 3λ, where λ is the provided security level (i.e. 128). In 1996, Pointcheval and Stern [55] applied their newly introduced Forking Lemma to provide the first formal security for a 2λ-bit ideal hash assuming the underlying discrete logarithm is hard. In [61] the first proof of Schnorr's ID against active attacks is provided in the GGM (Generic Group Model), but without focus on Fiat-Shamir

constructions.

A significant contribution from Neven et al. [50] was to apply the GGM and other results of [7] to prove security using a λ-bit hash function. Briefly, in their proof, hash functions are not handled as random oracles, but they should offer specific properties, such as variants of preimage and second preimage resistance; but not collision resistance. However, as we mention in Section 2.3, most of the real world applications do not assume honest signers, and thus non-repudiation is an important property, which unfortunately requires a collision resistant H_0.

Finally, the works from Backendal et al. [2] clarified the relation between the UF-security of different Schnorr variants, while in [32] a *tight* reduction of the UF-security of Schnorr signatures to *discrete log* in the Algebraic Group Model [31] (AGM)+ROM was presented.

2.2 Multi-signatures

One of the main advantages of Schnorr signatures compared to ECDSA is its linearity which allows to add two (or more) Schnorr signatures together and get a valid compact aggregated output indistinguishable from a single signature. The concept of multi-signature is to allow co-signing on the same message. Even if the messages are different, there are techniques using indexed Merkle tree accumulators to agree on a common tree root and then everyone signs that root. However, just adding Schnorr signatures is not secure as the requirement to protect against rogue key and other similar attacks is essential, especially in blockchain systems.

There is indeed a number of practical proposals that require two or three rounds of interaction until co-signers agree on a common R and public key A value [7, 58, 3, 45, 62, 12, 47, 24, 52, 43, 51]. One of the most recent is the compact two-round Musig2 [51] which also supports pre-processing (before co-signers learn the message to be signed) of all but the first round, effectively enabling a non-interactive signing process. Musig2 security is proven in the AGM+ROM model and it relies on the hardness of the OMDL problem.

Another promising two-round protocol is FROST [43] which has a similar logic with Musig2, but it utilizes verifiable random functions (VRFs) and mostly considers a threshold signature setting.

Note that even with pre-processing, Musig2 requires an initial setup with broadcasting and maintaining state. Compared to half-aggregation which can work with zero interaction between signers, Musig2 and FROST have a huge potential for controlled environments (i.e., validator sets in blockchains), but might not be ideal in settings where the co-signers do not know each other in advance or when public keys and group formation are rotated/updated very often.

2.3 Schnorr signature variants

There exist multiple variants of the original Schnorr scheme and the majority of them are incompatible between each other. Some of the most notable differences include:

- H_0 is not binding to the public key and thus it's computed as $H_0(R, m)$ instead of

$H_0(R, A, m)$ [59, 33]. Note that these signatures are malleable as shown in the EdDSA paper (page 7, Malleability paragraph) [11].

- change the order of inputs in H_0, such as $H_0(m, R)$. Note that protocols in which m is the first input to the hash function require collision resistant hash functions, as a malicious message submitter (who doesn't know R), can try to find two messages m_0 and m_1 where $H_0(m_0) = H_0(m_1)$. This is the main reason for which the Pure EdDSA RFC 8032 [41] suggests $H_0(R, A, m)$ versus any other combination.
- $H_0(R_x, A, m)$ takes as input only the x-coordinate of R, such as the EC-SDSA-opt in [33] and BIP-Schnorr [54].
- send the scalar H_0 instead of the point R. This variation (often referred to as *compact*) was proposed in the original Schnorr paper [59] and avoids the minor complexity of encoding the R point in the signature, while it allows for potentially shorter signatures by 25%. The idea is that only half of the H_0 bytes suffice to provide SUF-CMA security at the target security level of 128 bits. While this allows 48-byte signatures, there are two major caveats:
  - according to Bellare at al.[6] (page 39), the (R, S) version (mentioned as BNN in that paper) achieves semi-strong unforgeability, while the original 48-byte Schnorr only normal unforgeability. In short, because finding collisions in a short hash function is easy, a malicious signer can break message binding (non-repudiation) by finding two messages m_0 and m_1 where truncated $H_0(R, A, m_0)$ equals to truncated $H_0(R, A, m_1)$,
  - as mentioned, collisions in 128-bit truncated H_0 require a 64-bit effort. But because the SUF-CMA model assumes honest signers, in multi-sig scenarios where potentially distrusting signers co-sign, some malicious coalition can try to obtain a valid signature on a message that an honest co-signer did not intend to sign.

Due to the above, and because compact signatures do not seem to support non-interactive aggregation or batch verification, it is clear that *this work is compatible with most of the (R, S) Schnorr signature variants*, EdDSA being one of them. Also note that half-aggregation achieves an asymptotic 50% size reduction and compares favorably against multiple *compact* Schnorr signatures.

2.4 Non-Schnorr schemes

Some of the best applications of non-interactive signature aggregation include shortening certificate chains and blockchain blocks. Putting Schnorr variants aside, there is a plethora of popular signature schemes used in real world applications including ECDSA, RSA, BLS and some newer post-quantum schemes i.e., based on hash functions or lattices. Regarding ECDSA, although there exist interactive threshold schemes, to the best of our knowledge there is no work around non-interactive aggregation, mainly due to the modular inversion involved [46].

Similarly, in RSA two users cannot share the same modulus N, which makes interactivity essential; however there exist sequential aggregate RSA signatures which however imply interaction [14]. Along the same lines, we are not aware of efficient multi-sig constructions for Lamport-based post-quantum schemes.

On the other hand, BLS is considered the most aggregation and blockchain friendly signature scheme, which by design allows for deriving a single signature from multiple outputs without any prior interaction and without proving knowledge or possession of secret keys [12]. The main practicality drawback of BLS schemes is that they are based on pairing-friendly curves and hashing to point functions for which there are on-going standardization efforts and limited HSM support. Also, the verification function of a rogue-key secure BLS scheme is still more expensive than Schnorr (aggregated or not) mainly due to the slower pairing computations.

2.5 Schnorr batching and aggregation

Similar approaches to generating linear combinations of signatures have been used for batch verification in the past as shown in Section 5. The original idea of operating on a group of signatures by means of a random linear combination of their members is due to Bellare et al [4]. Other approaches consider an aggregated signature from public keys owned by the same user, which removes the requirement for rogue key resistance. For instance, in [34] an interactive batching technique is provided resulting to faster verification using higher degree polynomials.

Half-aggregation has already been proposed in the past, but either in its simple form without random linear combinations (which is prone to rogue key attacks) or without proofs [25] or using non-standard Schnorr variants that are not compatible with EdDSA. Γ-signatures [64] are the closest prior work to our approach, also achieving half aggregation, but with a significantly modified and slightly slower Schnorr scheme. Additionally, their security is based on the custom *non-malleable* discrete logarithm (NMDL) assumption, although the authors claim that it could easily be proven secure against the stronger *explicit* knowledge-of-exponent assumption EKEA. On the other hand, we believe that our security guarantees are much more powerful as they are actually a proof of knowledge of signatures, which means that they can be used as a drop-in replacement in any protocol (where having the exact original signature strings is not important), without changing any underlying assumptions; and therefore be compliant with the standards.

3 Proof-of-knowledge for a collection of signatures

We then construct a three-move protocol for the proof of knowledge of a collection signatures, we then discuss two ways to make it non-interactive with different security/efficiency tradeoffs.

3.1 Three-move (Sigma) protocol

The construction takes inspiration from the batching of Sigma protocols for Schnorr's identification scheme [34].

A Sigma protocol is a three-move protocol run by a prover P and a verifier V for some relation $R = \{(x, w)\}$, for $(x, w) \in R$, x is called an *instance* and w is called a witness. $R \subseteq \{0,1\}^* \times \{0,1\}^*$, where there exists a polynomial p such that for any $(x, w) \in R$, the length of the witness is bounded $|w| \leq p(|x|)$. Often-times, x is a computational problem and w is a solution to that problem. In the Sigma protocol the prover convinces the verifier that it knows a witness of an instance x known to both of them. The protocol produces a transcript of the form (a, e, z) which consists of (in the order of exchanged messages): the *commitment* a sent by P, the *challenge* e sent by V and the *response* z sent by P. The verifier accepts or rejects the transcript. A Sigma protocol for the relation R with n-special soundness guarantees the existence of an extractor Ext which when given valid transcripts (accepted by the verifier) with different challenges (a, e_1, z_1), $(a, e_2, z_2), \ldots (a, e_n, z_n)$ for an instance x, produces (with certainty) a witness w for the statement, s.t. $(x, w) \in R$. We will not be concerned with the zero-knowledge property of the protocol for our application.

For a group $\mathbb{G}$ with generator $B \in \mathbb{G}$ of order $q \in \mathbb{Z}$, define the relation $R_{\mathsf{DL}} = \{(\mathsf{pk}, \mathsf{sk}) \in (\mathbb{G}, \mathbb{Z}_q) : \mathsf{pk} = \mathsf{sk} \cdot B\}$. Schnorr's identification protocol [59] is a two-special sound Sigma protocol for the relation R_{DL}: given two transcripts with the same commitment and different challenges, the secret key (discrete logarithm of pk) can be extracted. It is known how to compress n instances of Schnorr's protocol to produce an n-special sound Sigma protocol at essentially the same cost [34], we use similar ideas to derive a Sigma protocol for the aggregation of Schnorr signatures, i.e. for the following relation (with hash function H_0):

$$
\begin{aligned}
R_{\mathsf{aggr}} = \{(x, w) \mid x = (\mathsf{pk}_1, m_1, \ldots, \mathsf{pk}_n, m_n), w = (\sigma_1, \ldots, \sigma_n), \\
\mathsf{Verify}(m_i, \mathsf{pk}_i, \sigma_i) = \mathsf{true} \text{ for } \forall i \in [n]\} = \\
= \{(x, w) \mid x = (A_1, m_1, \ldots, A_n, m_n), w = (R_1, S_1, \ldots, R_n, S_n), \\
S_i \cdot B = R_i + H_0(R_i, A_i, m_i) \cdot A_i \text{ for } i = 1..n\}
\end{aligned}
$$

Theorem 1. *Protocol 3 is an n-special sound Sigma protocol for R_{aggr}.*

Proof. Completeness is easy to verify. Extraction is always successful due to the following: let $F \in \mathbb{G}[X]$ be the degree $n-1$ polynomial where the coefficient of x^{i-1} is given by $R_i + H(R_i, \mathsf{pk}_i, m_i) \cdot \mathsf{pk}_i$ for each $i \in [n]$. Define $f \in \mathbb{Z}_q[X]$ as the isomorphic degree $n-1$ polynomial over $\mathbb{Z}_q$ such that the coefficient of x^{i-1} in f is S_i (the discrete logarithm of the corresponding coefficient in F). Observe that $f(x) \cdot B = F(x)$ for each $x \in \mathbb{Z}_q$. Given a transcript (a, e, z), V_Σ accepts iff $z \cdot B = F(e)$, which is true iff $z = f(e)$. Therefore n valid transcripts $(a, e_1, z_1), \ldots, (a, e_n, z_n)$ define n distinct evaluations of f (which has degree $n-1$) allowing for recovery of coefficients $[S_i]_{i \in [n]}$ efficiently. This is precisely the operation carried out by Ext_Σ, expressed as a product of matrices. Note that $E = [e_i^j]_{i,j \in [n]}$ is always invertible; each e_i is known to be distinct, and so E is always a Vandermonde matrix. □

Protocol 3 Sigma protocol for a collection of signatures R_{aggr}

For instance $x = \{(\mathsf{pk}_i = A_i, m_i)\}_{i=1}^n$ and witness $w = \{\sigma_i = (R_i, S_i)\}_{i=1}^n$

Prover $P_\Sigma(x, w)$:

1. **Commitment**: $a = [R_1, \ldots, R_n]$
2. **Challenge**: $e \xleftarrow{\$} \mathbb{Z}_q^*$
3. **Response**: $z = \sum_{i \in [n]} S_i \cdot e^{i-1}$

Verifier $V_\Sigma(x, (a, e, z))$: Output 1 iff $z \cdot B = \sum_{i \in [n]} e^{i-1}(R_i + H_0(R_i, A_i, m_i) \cdot A_i)$

Extractor $\mathsf{Ext}_\Sigma((a, e_1, z_1), \ldots, (a, e_n, z_n))$: Define the $n \times n$ matrix $E = [e_i^j]_{i,j \in [n]}$ and the column vector $Z = ([z_i]_{i \in [n]})^T$. Output $[S_1, \ldots, S_n] = (E^{-1}Z)^T$.

3.2 Proof-of-knowledge

A proof-of-knowledge for a relation $R = \{(x, w)\}$ is a protocol that realizes the following functionality:

$$\mathcal{F}^R((x, w), x) = (\emptyset, R(x, w))$$

i.e. the prover and verifier have inputs (x, w) and x respectively, and receive outputs $\emptyset$ and $R(x, w)$ respectively. This definition is taken from Hazay and Lindell [38, 37] who show it to be equivalent to the original definition of Bellare and Goldreich [5]. We additionally let a corrupt verifier learn $\mathsf{aux}(w)$ for some auxiliary information function aux. As we do not care about zero-knowledge at all (only compression) this can simply be the identity function, i.e. $\mathsf{aux}(w) = w$.

Proofs-of-knowledge allow for the drop-in replacement mechanism that we desire: instead of an instruction of the form “A sends n signatures to B” in a higher level protocol, one can simply specify that “A sends n signatures to $\mathcal{F}^R$, and B checks that its output from $\mathcal{F}^R$ is 1”.

Among the several landmark transformations of a Sigma protocol into a non-interactive proof [28, 29, 53], the most commonly used is the Fiat-Shamir transform [28]: for a relation R a valid transcript of the form (a, e, z) can be transformed into a proof by hashing the commitment to generate the challenge non-interactively: $\mathsf{proof} = (a, e = H_1(a, x), z)$. Unfortunately, this transformation induces a security loss, applied directly to the n-sound Sigma protocol for the relation R_{aggr} from the previous section (Protocol 3), the prover will have to be rewinded n times to extract the witness. This transformation however gives a more efficient construction for non-interactive aggregation of signatures that we discuss in Section 4.

To achieve tighter security reduction, we look into the literature on proof-of-knowledge with online extractions [53]. There, extractors can output the witness immediately without rewinding, in addition to the instance and the proof the extractors are given all the hash queries the prover made. We achieve a proof-of-knowledge for the relation R_{aggr} which

immediately gives an aggregate signature scheme whose security can be *tightly* reduced to unforgeability of Schnorr's signatures as we discuss in Section 4.3. We present both protocols in this Section.

3.2.1 Fiat-Shamir transformation

Protocol 4 Non-interactive proof-of-knowledge for R_{aggr}

Parameters: A curve group G with generator $B \in G$ of order $q \in \mathbb{Z}$. For instance $x = \{(\mathsf{pk}_i = A_i, m_i)\}_{i=1}^n$ and witness $w = \{\sigma_i = (S_i, R_i)\}_{i=1}^n$ we define three algorithms. Hash function H_1 modeled as a Random-Oracle.

Prover $P(x, w) \rightarrow \mathsf{proof}$:

1. Compute the scalar $e = H_1(R_1, A_1, m_1, \ldots, R_n, A_n, m_n)$
2. Compute the scalar $S_{\mathsf{aggr}} = \sum_{i=1}^n e^{i-1} \cdot S_i$.
3. Output the proof $\sigma_{\mathsf{aggr}} = [R_1, \ldots, R_n, S_{\mathsf{aggr}}]$.

Verifier $V_{\mathsf{RO}}(x, \mathsf{proof} = [R_1, \ldots, R_n, S_{\mathsf{aggr}}]) \leftarrow 0/1$:

1. Compute the scalar $e = H_1(R_1, A_1, m_1, \ldots, R_n, A_n, m_n)$.
2. If $\sum_{i=1}^n e^{i-1} (R_i + H_0(R_i, A_i, m_i) \cdot A_i) = S_{\mathsf{aggr}} \cdot B$, output *true*,
3. otherwise output *false*.

Theorem 2. *For every prover P that produces an accepting proof with probability ϵ and runtime T having made a list of queries $\mathcal{Q}$ to RO, there is an extractor Ext that outputs a valid signature for each $\mathsf{pk}_i \in \mathsf{pk}_{\mathsf{aggr}}$ in time $nT + \mathsf{poly}(\lambda)$, with probability at least $\epsilon - (n \cdot Q)^2/2^{h+1}$, where h is the bit-length of the H_1's output. It follows that the scheme (P, V, Ext) is a non-interactive proof-of-knowledge for the relation R_{aggr} in the random oracle model.*

Proof. The extractor Ext runs the adversary n times programming the random oracle to output fresh random values on each run, giving n proofs that can be used to obtain n accepting transcripts (a, e_i, z_i) for $i \in [n]$ and invokes Ext_Σ once they are found. Ext runs in time nT, and additionally $poly(\kappa)$ to run Ext_Σ. The extractor fails in case not all of the e_i are distinct which happens with probability at most $(n \cdot Q)^2/2^{h+1}$ by the birthday bound when we estimate the probability of at least one hash-collision between the queries of n runs of the adversary. □

Another form of the protocol with the challenges derived with independent hashes allows for extraction of any single signature with a single rewinding. This protocol is a foundation for the half-aggregation construction for Schnorr signatures described in Section 4.3. To construct an extractor we use a variant of the Forking Lemma. Originally the Forking Lemma was

introduced in the work of Pointcheval and Stern [55]. We use a generalized version described in [7].

Protocol 5 Non-interactive proof-of-knowledge for R_{aggr}

Parameters: A curve group G with generator $B \in G$ of order $q \in \mathbb{Z}$. For instance $x = \{(\mathsf{pk}_i = A_i, m_i)\}_{i=1}^n$ and witness $w = \{\sigma_i = (S_i, R_i)\}_{i=1}^n$ we define three algorithms. Hash function H_1 modeled as a Random-Oracle.

Prover $P(x, w) \to \mathsf{proof}$:

1. For $i \in [n]$ compute the scalars $e_i = H_1(R_1, A_1, m_1, \ldots, R_n, A_n, m_n, i)$
2. Compute the scalar $S_{\mathsf{aggr}} = \sum_{i=1}^n e_i \cdot S_i$.
3. Output the proof $\sigma_{\mathsf{aggr}} = [R_1, \ldots, R_n, S_{\mathsf{aggr}}]$.

Verifier $V_{\mathsf{RO}}(x, \mathsf{proof} = [R_1, \ldots, R_n, S_{\mathsf{aggr}}]) \leftarrow 0/1$:

1. For $i \in [n]$ compute the scalars $e_i = H_1(R_1, A_1, m_1, \ldots, R_n, A_n, m_n, i)$.
2. If $\sum_{i=1}^n e_i \left(R_i + H_0(R_i, A_i, m_i) \cdot A_i\right) = S_{\mathsf{aggr}} \cdot B$, output *true*,
3. otherwise output *false*.

---

[7] **Generalized Forking Lemma.** *Fix an integer $q \geq 1$ and a set H of size $h \geq 2$. Let $\mathcal{A}$ be a randomized algorithm. The algorithm $\mathcal{A}$ is given an input $in = (\mathsf{pk}, h_1, \ldots, h_q)$ and randomness y, it returns a pair, the first element of which is an integer I and the second element of which is a side output* proof*:*

$$(I, \mathsf{proof}) \leftarrow \mathcal{A}(in; y).$$

We say that the algorithm $\mathcal{A}$ succeeds if $I \geq 1$ and fails if $I = 0$. Let IG *be a randomized input generator algorithm. We define the success probability of $\mathcal{A}$ as:*

$$acc = \Pr[I \geq 1; \mathsf{input} \stackrel{\$}{\leftarrow} \mathsf{IG}; (h_1, \ldots, h_q) \stackrel{\$}{\leftarrow} H; (I, \mathsf{proof}) \stackrel{\$}{\leftarrow} \mathcal{A}(\mathsf{input}, h_1, \ldots, h_q)].$$

We define a randomized generalized forking algorithm $F_{\mathcal{A}}$ that depends on $\mathcal{A}$:

$\underline{F_{\mathcal{A}}(\mathsf{input})}$ *forking algorithm:*

1. *Pick coins y for $\mathcal{A}$ at random*
2. $h_1, \ldots, h_q \stackrel{\$}{\leftarrow} H$
3. $(I, \mathsf{proof}) := \mathcal{A}(x, i, h_1, \ldots, h_q; y)$
4. *If $I = 0$ then return* $(0, \perp, \perp)$

5. $h'_1, \ldots, h'_q \xleftarrow{\$} H$

6. $(I', \mathsf{proof}') := \mathcal{A}(x, i, h_1, \ldots, h_{I-1}, h'_I, \ldots, h'_q; y)$

7. *If* $(I = I'$ *and* $h_I \neq h'_I)$ *then return* $(1, \mathsf{proof}, \mathsf{proof}')$

8. *Else return* $(0, \bot, \bot)$.

Let $frk = \Pr[b = 1; \mathsf{input} \xleftarrow{\$} \mathsf{IG}; (b, \mathsf{proof}, \mathsf{proof}') \xleftarrow{\$} F_{\mathcal{A}}(i, x)]$.

Then $frk \geq acc \cdot \left(\frac{acc}{q} - \frac{1}{h}\right)$.

Theorem 3. *For every prover P that produces an accepting proof for a collection of n signatures with probability ϵ and runtime T having made a list of queries $\mathcal{Q}$ to* RO *(H_1), there is an extractor* Ext *that given $i^* \in [n]$ outputs an i^*-th signature that is valid under pk_{i^*} for message m_{i^*} in time $2T \cdot n$, with probability at least $\epsilon \cdot (\epsilon/(n \cdot Q) - 1/2^h)$, where h is the bit-length of the H_1's output.*

Proof. The extractor will run the prover P for the same input twice to obtain two proofs that differ on the last component:

$$\mathsf{proof} = [R_1, \ldots, R_n, S_{\mathsf{aggr}}] \text{ and } \mathsf{proof}' = [R_1, \ldots, R_n, S'_{\mathsf{aggr}}]$$

it will then be able to extract a signature on pk_{i^*}.

We first wrap the prover P into an algorithm $\mathcal{A}$ to be used in the Forking Lemma. The algorithm $\mathcal{A}$ takes input $in = (\{(\mathsf{pk}_i, m_i)\}_{i=1}^n, i^*, h_1, \ldots, h_q)$, for $q = (Q+1) \cdot n$, and a random tape y, it runs the prover P and programs its H_1 random oracle outputs as follows: on the input that was already queried before, output the same value (we record all the past H_1 queries). In case the query can not be parsed as $(R_1, A_1, m_1, \ldots, R_n, A_n, m_n, j) \in (G \times G \times \{0,1\}^*)^n \times [n]$ or in case the public key A_{i^*} does not match the one in the input: $A_{i^*} \neq \mathsf{pk}_{i^*}$, program the oracle to the next unused value of y. Otherwise, if $A_{i^*} = \mathsf{pk}_{i^*}$ and the query is of the form $(R_1, A_1, m_1, \ldots, R_n, A_n, m_n, j) \in (G \times G \times \{0,1\}^*)^n \times [n]$, do the following: (1) for each $i \in [n] \backslash i^*$ program the oracle on index i, i.e. on input $(R_1, A_1, m_1, \ldots, R_n, A_n, m_n, i)$, to the next unused value of y, (2) program the oracle on index i^*, i.e. on input $(R_1, A_1, m_1, \ldots, R_n, A_n, m_n, i^*)$, to the next unused value of h: h_t and (3) record the index into the table $T[R_1, A_1, m_1, \ldots, R_n, A_n, m_n, i^*] := t$.

Note that when the oracle is queried on some $(R_1, A_1, m_1, \ldots, R_n, A_n, m_n, j)$, all the related n queries are determined, those are queries of the form $(R_1, A_1, m_1, \ldots, R_n, A_n, m_n, i)$ for $i \in [n]$, so we program all those n queries ahead of time, when a fresh tuple $(R_1, A_1, m_1, \ldots, R_n, A_n, m_n)$ is queried to the H_1 oracle (i.e. on one real query, we program n related queries). The index t recorded in the table T is the potential forking point, so we program the queries $(R_1, A_1, m_1, \ldots, R_n, A_n, m_n, i)$ for $i \in [n] \backslash i^*$ first, to the values of y, making sure that those values of y [2] are read before the forking point (the positions of y that are used here are

[2] An anonymous reviewer suggested a PRF could be used to derive the values of y from a single seed in order to save space for an implementation of the reduction.

therefore the same between rewindings), we finally program $(R_1, A_1, m_1, \ldots, R_n, A_n, m_n, i^*)$ to the next value of h (the potential forking point, therefore an oracle query at this value may differ between rewindings). Note also that in the process of programming we ignore the index j where the real query has been asked, it is only being used to give back the correct programmed value.

When the prover outputs a $\mathsf{proof} = [R_1, \ldots, R_n, S_{\mathsf{aggr}}]$, the algorithm $\mathcal{A}$ performs additional queries $H_1(R_1, A_1, m_1, \ldots, R_n, A_n, m_n, j)$ for all $j \in [n]$, making sure those are defined, and if the proof is valid, it outputs $I = T[R_1, A_1, m_1, \ldots, R_n, A_n, m_n, i^*]$ and proof, otherwise it outputs $(0, \perp)$.

Next we use the forking lemma to construct an algorithm $F_{\mathcal{A}}$ that produces two valid proofs proof and proof' and an index I. Since the same randomness and the same oracle values were used until index I, it must be the case that two proofs satisfy:

$$\mathsf{proof} = [R_1, \ldots, R_n, S_{\mathsf{aggr}}],\ \sum_{i=1}^{n} e_i \left(R_i + H_0(R_i, A_i, m_i) \cdot A_i\right) = S_{\mathsf{aggr}} \cdot B, \tag{1}$$

$$\mathsf{proof}' = [R_1, \ldots, R_n, S'_{\mathsf{aggr}}],\ \sum_{i=1}^{n} e'_i \left(R_i + H_0(R_i, A_i, m_i) \cdot A_i\right) = S'_{\mathsf{aggr}} \cdot B, \tag{2}$$

where $e_{i^*} \neq e'_{i^*}$ and for $\forall i \neq i^* e_i = e'_i$,

since the latter are programmed before the forking point I.

Subtracting the two equations (Eq. 1 and Eq. 2) we extracted a signature $(S = S_{\mathsf{aggr}} - S'_{\mathsf{aggr}}, R_i)$ on message m_i under the public key A_i.

The success probability of $\mathcal{A}$ is ϵ, hence the probability of successful extraction according to the Forking Lemma is $\epsilon \cdot (\epsilon/(n \cdot Q) - 1/2^h)$. The extractor runs the prover twice and on each one random oracle query programs at most $n - 1$ additional random oracle queries. □

Note that Ext extracts a single signature at a specified position. To extract all of the n signatures, the prover needs to be rewinded n times.

Corollary 1. *For every prover P that produces an accepting proof with probability ϵ and runtime T having made a list of queries $\mathcal{Q}$ to RO, there is an extractor Ext that outputs a full witness (i.e. all valid signatures for all $\mathsf{pk}_i \in \mathsf{pk}_{\mathsf{aggr}}$) in time $(n+1)Tn$, with probability at least $\left(\epsilon \cdot (\epsilon/(n \cdot Q) - 1/2^h)\right)^n$, where h is the bit-length of the H_1's output. It follows that the scheme (P, V, Ext) is a non-interactive proof-of-knowledge for the relation R_{aggr} in the random oracle model.*

3.2.2 Fischlin's transformation

Pass [53] was the first to formalize the online extraction problem in the random oracle model and give a generic transformation from any 2-special sound sigma protocol to a non-interactive proof-of-knowledge with online extraction. Intuitively, Pass's transformation is a cut-and-choose protocol where each challenge is limited to a logarithmic number of bits. The prover can therefore compute transcripts for all of the challenges (since there are a

polynomial number of them), put the transcripts as leaves of the Merkle tree and compute the Merkle root. The extractor will see all of the transcripts on the leaves since it can examine random-oracle queries. The prover may construct an actual challenge by hashing the root of the tree and the original commitment, map the result to one of the leaves and reveal the Merkle path as a proof of correctness which induces a logarithmic communication overhead. Fischlin's transformation [29] implements essentially the same idea (albeit for a specific class of Sigma protocols) where the transcripts for opening the cut-and-choose are selected at *constant* communication overhead, however at the expense of at least twice the number of hash queries in expectation. Roughly, the selection process works by repeatedly querying (a, e_i, z_i) to RO until one that satisfies $\mathsf{RO}(a, e_i, z_i) = 0^\ell$ is found.

A proof-of-knowledge that permits an online extractor is very easy to use in a larger protocol; it essentially implements an oracle that outputs 1 to the verifier iff the prover gives it a valid witness. A reduction that makes use of an adversary for a larger protocol simply receives the witness on behalf of this oracle, while incurring only an additive loss of security corresponding to the extraction error. This is the design principle of Universal Composability [18] and permits modular analysis for higher level protocols, which in this case means that invoking the aggregated proof oracle is "almost equivalent" to simply sending the signatures in the clear.

We construct a non-interactive version of our aggregation protocol with ideas inspired by Fischlin's transformation, so that proofs produced by our protocol will permit online extraction. There are various subtle differences from Fischlin's context, such as different soundness levels for the underlying and compiled protocols to permit compression, and the lack of zero-knowledge, and so we specify the non-interactive protocol directly in its entirety below, and prove it secure from scratch.

The parameters ℓ, r are set to achieve λ bits of security, and adjusted as a tradeoff between computation and communication cost. In particular, the scheme achieves $r(\ell - \log_2(n)) = \lambda$ bits of security, proofs are of size n curve points and r field elements, and take $r \cdot 2^\ell$ hash queries to produce (in expectation).

Theorem 4. *The scheme* (P, V, Ext) *is a non-interactive proof-of-knowledge for the relation* R_{aggr} *in the random oracle model. Furthermore for every prover* P^* *that produces an accepting proof with probability* ϵ *and runtime* T *having made a list of queries* $\mathcal{Q}$ *to* RO*, the extractor* Ext *given* $\mathcal{Q}$ *outputs a valid signature for each* $\mathsf{pk}_i \in \mathsf{pk}_{\mathsf{aggr}}$ *in time* $T + \mathsf{poly}(\lambda)$*, with probability at least* $\epsilon - T \cdot 2^{-\lambda}$*.*

Proof. **Completeness.** It is easy to verify that when P terminates by outputting a proof, V accepts this proof string. P terminates once it has found r independent pre-images of 0^ℓ per RO; in expectation, this takes $r \cdot 2^\ell$ queries, which is polynomial in λ as $\ell \in O(\log \lambda)$ and $r \in O(\mathsf{poly}(\lambda))$. The prover therefore runs in expected polynomial time.

Proof of Knowledge. The extractor Ext works by inspecting queries to RO to find n accepting transcripts (a, e_i, z_i) and invoking Ext_Σ once they are found. First note that Ext runs in at most $|\mathcal{Q}| \leq T$ steps to inspect queries to RO, and additionally $\mathsf{poly}(\lambda)$ to run Ext_Σ. We now focus on bounding the extraction error. As Ext_Σ works with certainty when given $(a, e_1, z_1), \ldots, (a, e_n, z_n)$, it only remains to quantify the probability with which Ext will

Protocol 6 Non-interactive proof-of-knowledge for R_{aggr}

Prover $P_{\mathsf{RO}}(x, w) \to \mathsf{proof}$:

1. Initialize an array of curve points, $\mathbf{a} = [R_1, \ldots, R_n]$.
2. Initialize empty arrays of scalars: $\mathbf{e} = [\bot]^r$ and $\mathbf{z} = [\bot]^r$; $\mathbf{e}, \mathbf{z} \in (\mathbb{Z}_q \cup \bot)^r$.
3. Set $\mathsf{ind} = 1$, $e = 1$.
4. While $\mathsf{ind} \leq r$, do:
   (a) Compute $z = \sum_{i \in [n]} S_i \cdot e^{i-1}$.
   (b) If $\mathsf{RO}(\mathbf{a}, \mathsf{ind}, e, z) \stackrel{?}{=} 0^\ell$:
      - Set $\mathbf{e}_{\mathsf{ind}} = e$ and $\mathbf{z}_{\mathsf{ind}} = z$
      - Increment the ind counter and reset $e = 1$
   (c) Else: increment e
5. Output the proof $(\mathbf{a}, \mathbf{e}, \mathbf{z})$

Verifier $V_{\mathsf{RO}}(x, \mathsf{proof} = (\mathbf{a}, \mathbf{e}, \mathbf{z})) \leftarrow 0/1$:

1. Output 1 (accept) if both of the following equalities hold for every $\mathsf{ind} \in [r]$:

$$\mathsf{RO}(\mathbf{a}, \mathsf{ind}, \mathbf{e}_{\mathsf{ind}}, \mathbf{z}_{\mathsf{ind}}) = 0^\ell \bigwedge \mathbf{z}_{\mathsf{ind}} \cdot G = \sum_{i \in [n]} \mathbf{e}_{\mathsf{ind}}^{i-1}(R_i + H(R_i, \mathsf{pk}_i, m_i) \cdot \mathsf{pk}_i)$$

2. Output 0 (reject) if even one test does not pass.

---

succeed in finding at least n accepting transcripts in the list of RO queries. The event that the extractor fails is equivalent to the event that P^* is able to output an accepting proof despite querying fewer than n valid transcripts (prefixed by the same a) to RO; call this event fail. Define the event fail_a as the event that P^* is able to output an accepting proof $(a, \mathbf{e}, \mathbf{z})$ despite querying fewer than n valid transcripts (prefixed specifically by a) to RO. Define $\mathsf{fail}_{a,\mathsf{ind}}$ as the event that P^* queries fewer than n valid transcripts to RO prefixed specifically by a, ind, for each $\mathsf{ind} \in [r]$. Let $Q_{\mathsf{ind},1}, \ldots, Q_{\mathsf{ind},m}$ index the valid transcripts queried to RO with prefix a, ind. The event $\mathsf{fail}_{a,\mathsf{ind}}$ occurs only when $m < n$, and so the probability that $\mathsf{fail}_{a,\mathsf{ind}}$ occurs for a given ind can therefore be computed as follows:

$$\Pr[\mathsf{fail}_{a,\mathsf{ind}}] = \Pr[\mathsf{RO}(Q_{\mathsf{ind},1}) = 0^\ell \vee \cdots \vee \mathsf{RO}(Q_{\mathsf{ind},m}) = 0^\ell] \leq \sum_{j \in [m]} \Pr[\mathsf{RO}(Q_{\mathsf{ind},j}) = 0^\ell]$$
$$\leq \sum_{j \in [n]} \Pr[\mathsf{RO}(Q_{\mathsf{ind},j}) = 0^\ell] = \sum_{j \in [n]} \frac{1}{2^\ell} = \frac{n}{2^\ell} = \frac{1}{2^{\ell - \log_2(n)}}$$

Subsequently to bound fail_a itself, we make the following observations:

- For fail_a to occur, it must be the case that $\mathsf{fail}_{a,\mathsf{ind}}$ occurs for every $\mathsf{ind} \in [r]$. This follows

easily, because every transcript prefixed by a, ind is of course prefixed by a.

- Each event $\mathsf{fail}_{a,\mathsf{ind}}$ is independent as the sets of queries they consider are prefixed by different ind values and so are completely disjoint.

The probability that fail_a occurs can hence be bounded as follows:

$$\Pr[\mathsf{fail}_a] \leq \Pr[\mathsf{fail}_{a,1} \wedge \cdots \wedge \mathsf{fail}_{a,r}] = \prod_{i \in [r]} \Pr[\mathsf{fail}_{a,i}] \leq \prod_{i \in [r]} \frac{1}{2^{\ell - \log(n)}} = 2^{-r(\ell - \log(n))}$$

The parameters r, ℓ are set so that $r(\ell - \log(n)) \geq \lambda$ and so the above probability simplifies to $2^{-\lambda}$. As P^* runs in time T, in order to derive the overall probability of the extractor's failure (i.e. event fail) we take a union bound over potentially T unique a values, finally giving us $\Pr[\mathsf{fail}] \leq T \cdot 2^{-\lambda}$ which proves the theorem. □

4 Non-interactive half-aggregation of signatures

Following the definition of Boneh et al. [13], we say that a signature scheme supports aggregation if given n signatures on n messages from n public keys (that can be different or repeating) it is possible to compress all these signatures into a shorter signature non-interactively. Aggregate signatures are related to non-interactive multisignatures [40, 48] with independent key generations. In multisignatures, a set of signers collectively sign the same message, producing a single signature, while here we focus on compressing the signatures on distinct messages. Our aggregation could be used to compress certificate chains, signatures on transactions or consensus messages of a blockchain, and everywhere where a batch of signatures needs to be stored efficiently or transmitted over a low-bandwidth channel. The aggregation that we present here can in practice be done by any third-party, the party does not have to be trusted, it needs access to the messages, the public keys of the users and the signatures, but it does not need to have access to users' secret keys.

The aggregate signature scheme consists of five algorithms: $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{AggregateSig}$, $\mathsf{AggregateVerify}$. The first three algorithms are the same as in the ordinary signature scheme:

$\mathsf{KeyGen}(1^\lambda)$: given a security parameter output a secret-public key pair $(\mathsf{sk}, \mathsf{pk})$.

$\mathsf{Sign}(\mathsf{sk}, m)$: given a secret key and a message output a signature σ.

$\mathsf{Verify}(m, \mathsf{pk}, \sigma)$: given a message, a public key and a signature output accept or reject.

$\mathsf{AggregateSig}((m_1, \mathsf{pk}_1, \sigma_1), \ldots, (m_n, \mathsf{pk}_n, \sigma_n)) \to \sigma_{\mathsf{aggr}}$: for an input set of n triplets —message, public key, signature, output an aggregate signature σ_{aggr}.

$\mathsf{AggregateVerify}((m_1, \mathsf{pk}_1), \ldots, (m_n, \mathsf{pk}_n), \sigma_{\mathsf{aggr}}) \to \{\textbf{accept}/\textbf{reject}\}$: for an input set of n pairs —message, public key— and an aggregate signature, output accept or reject.

Some schemes may allow an aggregation of the public keys as well, $\mathsf{AggregatePK}$, but we do not focus on such schemes here.

We now recall the EUF-CMA security and Strong Binding Security (SBS) of the single signature scheme. Intuitively, EUF-CMA (existential unforgeability under chosen message attacks) guarantees that any efficient adversary who has the public key pk of the signer and received an arbitrary number of signatures on messages of its choice: $\{m_i, \sigma_i\}_{i=1}^N$, cannot output a valid signature σ^* for a new message $m^* \notin \{m_i\}_{i=1}^N$ (except with negligible probability). An SBS guarantees that the signature is binding both to the message and to the public key, e.g. no efficient adversary may produce two public keys $\mathsf{pk}, \mathsf{pk}'$, two signatures m, m', s.t. $(\mathsf{pk}, m) \neq (\mathsf{pk}', m')$ and a signature σ that verifies successfully under (pk, m) and (pk', m').

An attacker $\mathcal{A}$ plays the following game:

$\underline{G_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}()}$ security game:

1. $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KeyGen}()$
2. $(m, \sigma) \leftarrow \mathcal{A}^{O_{\mathsf{Sign}(\mathsf{sk}^*, \cdot)}}(\mathsf{pk}^*)$
3. accept if
   $m_i \notin \mathcal{L}_{\mathsf{Sign}} \ \wedge \ \mathsf{Verify}(m, \mathsf{pk}^*, \sigma)$

$\underline{O_{\mathsf{Sign}(\mathsf{sk}^*, \cdot)}}$, the signing oracle, constructs the set $\mathcal{L}_{\mathsf{Sign}}$:

1. On input m, compute $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}^*, m)$
2. $\mathcal{L}_{\mathsf{Sign}} \leftarrow \mathcal{L}_{\mathsf{Sign}} \cup m$
3. return σ

Definition 1. *An attacker $\mathcal{A}$, (t, ϵ)-breaks a EUF-CMA security of the signature scheme if $\mathcal{A}$ runs in time at most t and wins the EUF-CMA game with probability ϵ. A signature scheme is (t, ϵ)-EUF-CMA-secure if no forger (t, ϵ)-breaks it.*

Likewise, if the scheme is (t, ϵ)-EUF-CMA-secure, we say that it achieves $\log_2(t/\epsilon)$-bits *security level.*

Note also that there is an additional requirement on single signature security which becomes increasingly important especially in blockchain applications is Strong Binding [20], it prevents a malicious signer from constructing a signature that is valid against different public keys and/or different messages. We define the associated game:

$\underline{G_{\mathcal{A}}^{\mathsf{SBS}}()}$ security game:

1. $(\mathsf{pk}, m, \mathsf{pk}', m', \sigma) \leftarrow \mathcal{A}()$
2. accept if $(\mathsf{pk}, m) \neq (\mathsf{pk}', m') \ \wedge \ \mathsf{Verify}(m, \mathsf{pk}, \sigma) \ \wedge \ \mathsf{Verify}(m', \mathsf{pk}', \sigma)$

Definition 2. *An attacker $\mathcal{A}$, (t, ϵ)-breaks SBS security of the signature scheme if $\mathcal{A}$ runs in time at most t and wins the SBS game with probability ϵ. A signature scheme is (t, ϵ)-SBS-secure if no forger (t, ϵ)-breaks it.*

4.1 Aggregate signature security

Intuitively, the aggregate signature scheme is secure if no adversary can produce new aggregate signatures on a sequence of chosen keys where at least one of the keys is honest. We follow the definition of [13], the attacker's goal is to produce an existential forgery for an aggregate signature given access to the signing oracle on the honest key. An attacker $\mathcal{A}$ plays the following game parameterized by n, that we call chosen-key aggregate existential forgery under chosen-message attacks (CK-AEUF-CMA).

$\underline{G_{\mathcal{A}}^{\mathsf{CK\text{-}AEUF\text{-}CMA}}(n)}$ security game:

1. $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KeyGen}()$
2. $((m_1, \mathsf{pk}_1), \ldots, (m_n, \mathsf{pk}_n), \sigma_{\mathsf{aggr}}) \leftarrow \mathcal{A}^{O_{\mathsf{Sign}(\mathsf{sk}^*,\cdot)}}(\mathsf{pk}^*)$,
3. accept if $\exists i \in [n]$ s.t. $\mathsf{pk}^* = \mathsf{pk}_i$, and $m_i \notin \mathcal{L}_{\mathsf{Sign}}$, and $\mathsf{AggregateVerify}((m_1, \mathsf{pk}_1), \ldots, (m_n, \mathsf{pk}_n), \sigma_{\mathsf{aggr}})$

$\underline{O_{\mathsf{Sign}(\mathsf{sk}^*,m)}}$ constructs the set $\mathcal{L}_{\mathsf{Sign}}$:
$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}^*, m);\ \mathcal{L}_{\mathsf{Sign}} \leftarrow \mathcal{L}_{\mathsf{Sign}} \cup m;$ return σ

In this game an attacker is given an honestly generated challenge public key pk^*, he can choose all of the rest public keys, except the challenge public key, and may ask any number of chosen message queries for signatures on this key, at the end the adversary should output a sequence of n public keys (including the challenge public key), a sequence of n messages and an aggregate signature where the message corresponding to the public key pk^* did not appear in the signing queries done by the adversary. The adversary wins if the forgery successfully verifies.

Definition 3. *An attacker $\mathcal{A}$, (t, ϵ)-breaks a CK-AEUF-CMA security of aggregate signature scheme if $\mathcal{A}$ runs in time at most t and wins the CK-AEUF-CMA game with probability ϵ. An aggregate signature scheme is (t, ϵ)-CK-AEUF-CMA-secure if no forger (t, ϵ)-breaks it.*

More broadly, we say that an aggregate signature scheme is CK-AEUF-CMA-secure if no polynomial-time (in the security parameter) adversary may break the scheme other than with the negligible probability. Nonetheless, to instantiate the scheme with some concrete parameters, we will use a more rigid definition stated above. If the scheme is (t, ϵ)-CK-AEUF-CMA-secure, we say that it provides $\log_2(t/\epsilon)$-bits of security.

Note that the adversary has the ability to derive the rest of the public keys from the honest key pk^* in hope to cancel out the unknown components in the aggregate verification. Our constructions naturally prevent these attacks, otherwise generic methods of proving the knowledge of the secret keys could be used [48]. Note also that the original definition of Boneh et al. [13] places the honest public key as the first key in the forged sequence, since their scheme is agnostic to the ordering of the keys, our case is different and thus we give an adversary the ability to choose the position for the honest public key in the sequence.

In our constructions of aggregate Schnorr signatures we show that a valid single-signature forgery can be extracted from any adversary on the aggregate scheme.

The SBS definition translates to the aggregate signature defined as follows.
$\underline{G_{\mathcal{A}}^{\mathsf{CK\text{-}ASBS}}(n)}$ security game:

1. $((m_1, \mathsf{pk}_1), \ldots, (m_n, \mathsf{pk}_n), (m'_1, \mathsf{pk}'_1), \ldots, (m'_n, \mathsf{pk}'_n), \sigma_{\mathsf{aggr}}) \leftarrow \mathcal{A}(n)$,
2. accept if $[(m_1, \mathsf{pk}_1), \ldots, (m_n, \mathsf{pk}_n)] \neq [(m'_1, \mathsf{pk}'_1), \ldots, (m'_n, \mathsf{pk}'_n)] \wedge$
   $\mathsf{AggregateVerify}((m_1, \mathsf{pk}_1), \ldots, (m_n, \mathsf{pk}_n), \sigma_{\mathsf{aggr}}) \wedge$
   $\mathsf{AggregateVerify}((m'_1, \mathsf{pk}'_1), \ldots, (m'_n, \mathsf{pk}'_n), \sigma_{\mathsf{aggr}})$

Definition 4. *An attacker $\mathcal{A}$, (t, ϵ)-breaks a CK-ASBS security of aggregate signature scheme if $\mathcal{A}$ runs in time at most t and wins the CK-ASBS game with probability ϵ. An aggregate signature scheme is (t, ϵ)-CK-ASBS-secure if no forger (t, ϵ)-breaks it.*

4.2 Half-aggregation

The half-aggregation scheme for Schnorr's/EdDSA signatures runs the proof-of-knowledge protocol (Protocol 5 from Section 3) to obtain a proof that would serve as an aggregate signature. We present the construction for completeness here in Algorithm 7.

Algorithm 7 Half-aggregation of EdDSA signatures

$\mathsf{AggregateSig}((m_1, \mathsf{pk}_1, \sigma_1), \ldots, (m_n, \mathsf{pk}_n, \sigma_n)) \rightarrow \sigma_{\mathsf{aggr}}$:
1: Parse the signature as the group element and the scalar: $\sigma_i = (R_i, S_i)$.
2: Parse the public key as a group element: $\mathsf{pk}_i = A_i$.
3: For $i \in 1..n$ compute the scalars $e_i \leftarrow H_1(R_1, A_1, m_1, \ldots, R_n, A_n, m_n, i)$.
4: Compute an aggregate scalar $S_{\mathsf{aggr}} = \sum_{i=1}^{n} e_i \cdot S_i$.
5: Output an aggregate signature $\sigma_{\mathsf{aggr}} = [R_1, \ldots, R_n, S_{\mathsf{aggr}}]$.

$\mathsf{AggregateVerify}((m_1, \mathsf{pk}_1), \ldots, (m_n, \mathsf{pk}_n), \sigma_{\mathsf{aggr}}) \rightarrow 0/1$:
1: Parse the aggregate signature as $\sigma_{\mathsf{aggr}} = [R_1, \ldots, R_n, S_{\mathsf{aggr}}]$.
2: Parse each public key as a group element $\mathsf{pk}_i = A_i$.
3: Compute $e_i \leftarrow H_1(R_1, A_1, m_1, \ldots, R_n, A_n, m_n, i)$ for $i \in 1..n$
4: If $\sum_{i=1}^{n} e_i \, (R_i + H_0(R_i, A_i, m_i) \cdot A_i) = S_{\mathsf{aggr}} \cdot B$, output *true*,
5: otherwise output *false*.

Note that the scheme of Algorithm 7 compresses n signatures by a factor of $2 + O(1/n)$: it takes n signatures, where each of them is one group element and one scalar, it compresses the scalars into a single scalar, therefore the resulting aggregate signature is comprised of one scalar and n group elements, compared to n scalar and n group elements before aggregation.

Note that the set of R-s can be pre-published as part of the public key or part of previously signed messages, the aggregate signature becomes constant size, but signatures become stateful, as it should be recorded which R-s have already been used. Reuse of R leads to a complete leak of the secret key. Even small biasis in R weakens the security of the scheme [1]. This approach departs from the deterministic nature of deriving nonces in EdDSA, loosing its potential security benefits, though it will go unnoticed for the verifier.

Note also that for large messages the following optimized aggregation could be used to speed-up the verifier: each e_i could be computed as $e_i = H_1(H_0(R_1, A_1, m_1), \ldots, H_0(R_n, A_n, m_n), i)$, since the verifier computes $H_0(R_i, A_i, m_i)$ anyway, it can reuse those values to compute the coefficients for the aggregation, thus making the length of the input to H_1 smaller[3]. Though this optimization will only work for the (R_i, S_i) form of Schnorr signatures where all of the R_i, A_i, mi are inputs to H_0. Another potential micro-optimization mentioned in the literature is to set one of the e coefficients to 1 without loss of security.

Theorem 5. *If there is an adversary* $\mathsf{Adv_1}$ *that can* (t, ϵ)*-break the CK-AEUF-CMA security of the aggregate signature scheme in Algorithm 7, then this adversary can be transformed into an adversary* $\mathsf{Adv_2}$ *that can* $(2tn, \epsilon \cdot (\epsilon/(nt) - 1/2^h))$*-break the EUF-CMA security of the underlying signature scheme, where* h *is the bit-length of the* H_1*'s output.*

The proof of this Theorem is nearly identical to the proof of Theorem 3. The only caveat here is that to apply the extractor from Theorem 3, it is required to know the index of pk^* in a list of public keys, but this index can be obtained from examining the position of pk^* in the random oracle queries to H_1.

Theorem 6. *No adversary running in time tmay break the CK-ASBS security of the aggregate signature scheme described in Algorithm 7, other than with probability at most* $t^2/2^{2\lambda+1}$*.*

Proof. By statistical argument we show that the adversary may only produce an SBS forgery with negligible probability. For a successful forgery $((A_1, m_1), \ldots, (A_n, m_n), \sigma_{\mathsf{aggr}}) \neq ((A'_1, m'_1), \ldots (A'_2, m'_2), \sigma_{\mathsf{aggr}})$, all $2n$ underlying signatures can be extracted: $\sigma_1, \ldots, \sigma_n, \sigma'_1, \ldots, \sigma'_n$. All of those signatures have the same R components (since those are part of σ_{aggr}), but possibly different S components. When a query is made to the random oracle $H_1(R_1, A_1, m_1, \ldots, R_n, A_n, m_n, i)$, denote the output by h_i^j, where j is the incrementing counter for the unique tuples $(R_1, A_1, m_1, \ldots, R_n, A_n, m_n)$ queried to the random oracle. Denote by s_i^j the discrete log of $R_1 + H_0(R_i, A_i, m_i)A_i$ (here we work under the assumption that the discrete log can always be uniquely determined). Without loss of generality we assume that the adversary verifies the forgery, therefore for some two indices j' and j'' (that correspond to the SBS forgery output by the adversary) it must hold that the linear combination of the $\{s_i^{j'}\}_{i=1}^n$'s with coefficients $\{h_i^{j'}\}_{i=1}^n$ is equal to the linear combination of $\{s_i^{j''}\}_{i=1}^n$'s with coefficients $\{h_i^{j''}\}_{i=1}^n$. Having that in the RO-model, we can assume that the values $\{h_i^{j'}\}_{i=1}^n$ and $\{h_i^{j''}\}_{i=1}^n$ are programmed to uniformly random independent values after the s's values are determined. Each h randomizes the non-zero value of s to an exponent indistinguishable from random, therefore creating a random element as a result of a linear combination. Therefore the probability of a successful forgery for the adversary must be bounded by the collision probability $Q^2/(2 \cdot |\mathbb{G}|)$, where $Q \leq t$ is the number of H_1-queries and $|\mathbb{G}|$ is the size of the group (for prime order groups, or an order of a base point).

[3]Note that many hash libraries allow updating (then cloning) the hash function's state, which enables caching of $h_1 = H_1(H_0(R_1, A_1, m_1), \ldots, H_0(R_n, A_n, m_n))$. The latter results to one-pass on $H_0(R_i, A_i, m_i)$ values; then reuse the hash function's state by just adding the index i each time (as shown in [19] - function `set_initial_hash`). If hash-state cloning is not available, a construction where we pre-calculate h_1 and then compute each e_i as $H_1(h_1, i)$ is also valid.

□

Parameter selection and benchmarks. Theorem 5 has a quadratic security loss in its time-to-success ratio: assuming that EUF-CMA provides 128-bits of security (which is the case for example for Ed25519 signature scheme) the theorem guarantees only 64-bits security for CK-AEUF-CMA with 128-bits H_1-hashes; and assuming that EUF-CMA provides 224-bits of security (which is the case for example for Ed448 signature scheme) the theorem guarantees 112-bits security for CK-AEUF-CMA with 256-bits H_1-hashes[4]. A similar loss in the reduction from single Schnorr/EdDSA signature security to a discrete logarithm problem was not deemed to require the increase in the hardness of the underlying problems (i.e. the discrete logarithm problem). The proof that reduces security of Schnorr/EdDSA to the discrete logarithm problem also uses the Forking Lemma, but no attacks were found to exploit the loss suggested by such proof. Research suggests that the loss given by the Forking Lemma is inevitable for the proof of security of Schnorr/EdDSA signatures [60, 30], whether it is likewise inevitable for non-interactive half-aggregation of Schnorr/EdDSA signatures remains an open question. Recent work [35] on tight state restoration soundness in the Algebraic Group Model [31] may represent a promising direction towards proving that a tight parameterization of our half-aggregation is safe to use.

We benchmark [19] the scheme to understand the effect of using 128-bits of H_1 output vs. 256-bits of H_1 output and present the results in Table 1.[5] Note that the performance loss in aggregate signature's verification between the two approaches is only about 15%, which might not justify the a use of smaller hashes. We also benchmark the use of 512-bits hashes of H_1, same-size scalar are used in the EdDSA signature scheme, the advantage of this approach is that the scalars generated this way are distributed uniformly at random (within negligible statistical distance from uniform).

4.3 Half+ε-aggregation

The half+ε-aggregation scheme for EdDSA/Schnorr's signatures runs the proof-of-knowledge protocol (Protocol 6 from Section 3) to obtain a proof that would serve as an aggregate signature. For completeness we present the constructions in Algorithm 8.

Theorem 7. *If there is an adversary* Adv_1 *that can* (t, ϵ)*-break the CK-AEUF-CMA security of the aggregate signature scheme defined in Algorithm 8 making* Q *oracle queries to* H_1*, then this adversary can be transformed into an adversary* Adv_2 *that can* $(t+\mathsf{poly}(\lambda), \epsilon - t \cdot 2^{-\lambda})$*-break the EUF-CMA security of the underlying signature scheme.*

The theorem is a simple corollary of Theorem 4.

[4]Note that additionally $2\log_2(n) + 1$ bits of security will be lost due to n.

[5]The 'curve25519-dalek' and 'ed25519-dalek' libraries were used for the benchmark of this entire section, which ran on a AMD Ryzen 9 3950X 16-Core CPU. We used the scalar u64 backend of the dalek suite of libraries, to offer comparable results across a wide range of architectures, and the implementation does make use of Pippenger's bucketization algorithm for multi-exponentiation.

n	Sequential verification	Batch verification	AggregateVerify			AggregateSig		
			128	256	512	128	256	512
16	0.8 ms	0.39 ms	0.37 ms	0.43 ms	0.44 ms	9.75 µs	10.6 µs	16.98 µs
32	1.6 ms	0.75 ms	0.68 ms	0.79 ms	0.83 ms	19.25 µs	21.5 µs	33.02 µs
64	3.2 ms	1.39 ms	1.35 ms	1.52 ms	1.58 ms	39.35 µs	41.4 µs	67.63 µs
128	6.4 ms	2.73 ms	2.61 ms	2.95 ms	3.04 ms	78.6 µs	84.9 µs	134.44 µs
256	12.8 ms	4.86 ms	4.69 ms	5.41 ms	5.54 ms	151.6 µs	165.6 µs	260.36 µs
512	25.7 ms	8.92 ms	8.00 ms	9.86 ms	9.54 ms	316.1 µs	341.7 µs	526.50 µs
1024	51.5 ms	16.15 ms	15.25 ms	17.46 ms	18.31 ms	613.5 µs	657.9 µs	1088.0 µs
131072	6.59 s	1.98 s	1.71 s	2.11 s	2.09 s	80.21 ms	84.60 ms	133.96 ms

Table 1: For n individual signatures we compare batch-verification, aggregate-verification and aggregation with $\{128, 256, 512\}$-bits output for H_1, for Ed25519 signatures. SHA-256 cropped to 128-bits used for 128-bits H_1, SHA-256 used for 256-bits H_1, SHA-512 used for 512-bits H_1. The benchmarks are run using the ed25519-dalek library.

Theorem 8. *If there is an adversary* $\mathsf{Adv_1}$ *that can* (t, ϵ)*-break the CK-ASBS-CMA security of the aggregate signature scheme defined in Algorithm 8 making* Q *oracle queries to* H_1*, then this adversary can be transformed into an adversary* $\mathsf{Adv_2}$ *that can* $(t + \mathsf{poly}(\lambda), (\epsilon - t \cdot 2^{-\lambda})^2)$*-break the EUF-CMA security of the underlying signature scheme.*

Proof. From the forgery produced by the adversary $\mathsf{Adv_1}$:

$$((m_1, \mathsf{pk}_1), \ldots, (m_n, \mathsf{pk}_n), (m'_1, \mathsf{pk}'_1), \ldots, (m'g_n, \mathsf{pk}'_n), \sigma_{\mathsf{aggr}}),$$

we extract two sets of signatures by running the extractor of Theorem 4: $(\sigma_1, \ldots, \sigma_n)$ and $(\sigma'_1, \ldots, \sigma'_n)$. Those signatures have the same R-components $(R_1, \ldots, R_n)$, but possibly different S-components $(S_1, S'_1, \ldots, S_n, S'_n)$ when aggregated those components produce the same signature σ, therefore for some random $e \neq e'$, it holds that $\sum_{i=1}^{n} S_i \cdot e^{i-1} = \sum_{i=1}^{n} S'_i \cdot e'^{i-1}$ which may happen with probability at most 2^{λ} when $(S_1, \ldots, S_n) \neq (S'_1, \ldots, S'_n)$. Assuming that $(S_1, \ldots, S_n) = (S'_1, \ldots, S'_n)$, but $[(m_1, \mathsf{pk}_1), \ldots, (m_n, \mathsf{pk}_n)] \neq [(m'_1, \mathsf{pk}'_1), \ldots, (m'_n, \mathsf{pk}'_n)]$, as required for the forgery of $\mathsf{Adv_1}$ to be successful, it follows that at some position $i \in [n]$ where the equality breaks, a successful single SBS-forgery can be constructed: $(m_i, \mathsf{pk}_i, m'_i, \mathsf{pk}'_i, \sigma = (R_i, S_i))$. □

The security loss in this construction is much smaller, for example, the security remains at 128-bits for 128-bits output H_1-hash for Ed25519 signature scheme, and at 224-bits for 256-bits output H_1 for Ed448 signature scheme. But the compression rate for this aggregate signature scheme here is worse than for the previous scheme: the aggregated signature has n group elements, r full scalars and r small scalars of length ℓ in expectation, therefore the size of the signature is n group elements plus $r \cdot \lambda + r \cdot \ell$ bits. If we set λ and r to be constants and increase n, set $\ell = \log_2(n) + \lambda/r$, the size of the aggregate signature will be n group elements plus $O(log(n))$ bits, therefore the compression of the aggregation approaches 50% as n grows.

Algorithm 8 Almost-half-aggregation of EdDSA signatures

$\underline{\mathsf{AggregateSig}}((m_1, \mathsf{pk}_1, \sigma_1), \ldots, (m_n, \mathsf{pk}_n, \sigma_n)) \to \sigma_{\mathsf{aggr}}$:

1: Let $\sigma_i = (R_i, S_i)$.
2: Compute the hash $h_a = H_2(R_1, \cdots, R_n)$.
3: Set the empty arrays of scalars $\mathbf{e} := [\perp]^r$ and $\mathbf{z} := [\perp]^r$; $\mathbf{e}, \mathbf{z} \in (\mathbb{Z}_q \cup \perp)^r$.
4: Set the counter $j := 1$.
5: Set the scalar $e := 1$.
6: **while** $j \leq r$ **do**
7:     Compute $z := \sum_{i=1}^{n} S_i \cdot e^{i-1}$.
8:     **if** $H_1(h_a, j, e, z) = 0^\ell$ **then**
9:         Set $\mathbf{e}_j := e$; set $\mathbf{z}_j := z$; increment the counter j; reset the scalar $e = 1$.
10:     **else**
11:         Increment the scalar e.
12: Output the aggregate signature $\sigma_{\mathsf{aggr}} = ([R_1, \cdots, R_n], \mathbf{e}, \mathbf{z})$.

$\underline{\mathsf{AggregateVerify}}((m_1, \mathsf{pk}_1), \ldots, (m_n, \mathsf{pk}_n), \sigma_{\mathsf{aggr}}) \to 0/1$:

1: Let $\sigma_{\mathsf{aggr}} = ([R_1, \cdots, R_n], \mathbf{e}, \mathbf{z})$.
2: Compute $h_a = H_2(R_1, \cdots, R_n)$.
3: Output 1 (accept) if both of the following equalities hold for every $j \in 1..r$:

$$H_1(h_a, j, \mathbf{e}_j, \mathbf{z}_j) = 0^\ell \text{ and } \mathbf{z}_j \cdot G = \sum_{i=1}^{n} \mathbf{e}_j^{i-1}(R_i + H_0(R_i, \mathsf{pk}_i, m_i) \cdot \mathsf{pk}_i)$$

4: Output 0 (reject) if the test does not pass for some j.

Table 2 shows a selection of values across different trade-offs. Note that despite the aggregation time being rather slow, as the aggregator has to do many oracle-queries, it is highly parallelizable which is not reflected in our benchmarks: given $M \leq r2^\ell$ processors it is straightforward to parallelize aggregation into M threads.

It is most suitable to do aggregation in batches, i.e. aggregate some fixed constant number of signatures, n, choosing the number to achieve a desired trade-off between compression rate, aggregation time and verification time. The computational complexity of the aggregator is $O(r \cdot n \cdot 2^\ell)$ and of the verifier is $O(n \cdot r)$. In fact, in this scheme the verifier is about $r/2 > 1$ times less efficient than verifying signatures iteratively one-by-one, therefore this compression scheme will always sacrifice verifier's computational efficiency for compressed storage or network bandwidth for transmission of signatures. To approximate the aggregator's complexity, we first approximate the compression rate:

$$c = (256 \cdot n + r \cdot 256 + r \cdot \ell)/(512 \cdot n) \approx (n + r)/(2n).$$

We can now estimate the aggregator's time through $r = n(2c-1)$ as $O(n^3 \cdot (2c-1) \cdot 2^{\lambda/n/(2c-1)})$. For a fixed compression rate c it achieves minimum at a batch-size n shown on Figure 1 for

Compression	n	r	AggregateVerify	AggregateSig
0.52	512	16	134.11 ms	197.89 s
	1024	32	516.55 ms	76.857 s
0.53	256	16	74.449 ms	62.649 s
	512	32	291.04 ms	25.272 s
0.57	128	16	41.565 ms	12.007 s
	256	32	147.48 ms	6.1843 s
0.63	32	8	5.7735 ms	46.330 s
	64	16	23.007 ms	4.2622 s
	128	32	82.235 ms	1.3073 s
0.77	16	8	2.9823 ms	12.455 s
	32	16	10.377 ms	1.2994 s
	64	32	42.807 ms	403.55 ms

Table 2: The compression rate, the computation cost (for aggregation and aggregate-verification) for aggregating n Ed25519 signatures with SHA-256 hash function used for H_1. The ℓ is set to be $\ell = \log_2(n) + 128/r$. The benchmarks are run using the ed25519-dalek library.

$\lambda = 128$. The verifier's time can be estimated through compression rate as $O(n^2(2c-1))$, it is therefore most convenient to select an upper bound on the batch size according to Figure 1 and lower the batch-size to trade-off between aggregator's and verifier's runtime. We report optimal aggregation times for the given compression rate in Figure 2 for Ed25519 signature scheme. Amortized verification per signature is constant for constant r, amortized optimal aggregation per signature is linear in the batch size n.

5 Deterministic batch verification of Schnorr signatures

As another application of the proof-of-knowledge techniques we present *deterministic* batch verification. Batch verification is a technique that allows to verify a batch of signatures faster than verifying signatures one-by-one. Not all of the Schnorr's signatures' variants support batch verification, only those that transmit R instead of the hash $H(..)$ do.

Bernstein et al. [11] built and benchmarked an optimized variant for batch verification for EdDSA signatures utilizing the state-of-the-art methods for scalar-multiplication methods. To batch-verify a set of signatures (R_i, S_i) for $i = 1..n$ corresponding to the set of messages $\{m_i\}_{i=1..n}$ and the set of public keys A_i, they propose to choose "independent uniform random 128-bit integers z_i" and verify the equation

$$(-\sum_i z_i S_i \bmod \ell)B + \sum_i z_i R_i + \sum_i (z_i H_0(R_i, A_i, m_i) \bmod \ell)A_i = 0. \quad (3)$$

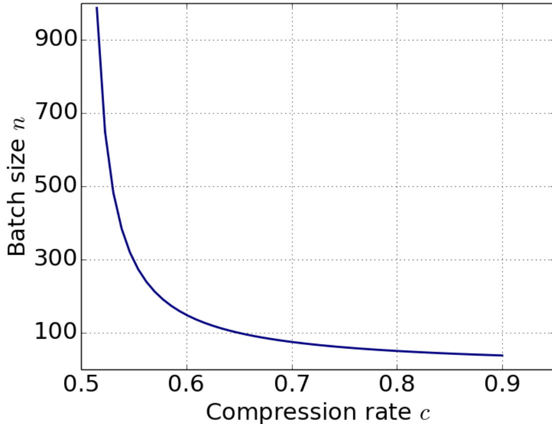

Figure 1: Optimal batch size to achieve the optimal aggregation time.

c	n	AggregateVerify amortized	AggregateSig amortized
0.55	296	500 µs	39.7 ms
0.6	148	562 µs	16.9 ms
0.65	98	609 µs	9.4 ms
0.7	74	582 µs	12.7 ms
0.75	59	562 µs	4.2 ms

Figure 2: Aggregation and verification time amortized per signature. Parameters n, r are set to achieve the smallest aggregation time: n is chosen from Figure 1, $r = 30$.

As we explain in the next paragraph with many real-world examples, it is often dangerous to rely on randomness in cryptographic implementations, particularly so for deployments on a cloud. It would thus be desirable to make protocols not utilize randomness in secure-critical components, such as signature-verifications. We note that batch verification (Eq. 3) is a probabilistic version of the Algorithm 7 for verification of half-aggregation of EdDSA signatures.

From the security proof of half-aggregation it therefore follows that batch verification can be made deterministic by deriving scalars with hashes as $z_i = H_1(R_1, A_1, m_1, \ldots, R_n, A_n, m_n, i)$ or better use $z_i = H_1(H_0(R_1, A_1, m_1), \ldots, H_0(R_n, A_n, m_n), i)$ as mentioned in Section 4.2. In fact, deterministic batching has been mentioned in the past [10] in the form of encrypting the hash of the input batch, which is similar to the above derivation, but by including the S values as well.

It is highlighted that apart from unforgeability guarantees, some applications might require another interesting property: batch verification and normal sequential verification should be consistent. The latter might be desirable in blockchains if we allow block verifiers to decide between batch and sequential verification.

Unfortunately, batch verification without including S in z-derivation is malleable which could lead to output inconsistency between batch and normal verification, and thus we recommend appending all S values in H_1 when this property is important. A concrete malleability example is the following:

Let's assume two legitimate signatures $sig_1 = (R_1, S_1)$ and $sig_2 = (R_2, S_2)$ for two users with public keys A_1, A_2 and signed messages m_1, m_2, respectively. Then a malicious third party could replace S_1 with $S_1' = S_1 + z_2$ and S_2 with $S_2' = S_2 - z_1$. It can be shown that the pair of signatures (R_1, S_1') and (R_2, S_2'), which would obviously fail when signatures are verified independently, will pass batch verification. This is because $z_1S_1' + z_2S_2' = z_1(S_1 + z_2) + z_2(S_2 - z_1) = z_1S_1 + z_1z_2 + z_2S_2 - z_2z_1 = z_1S_1 + z_2S_2$, as the z_1z_2 values are

canceled out.

We stress that the above malleability scenario is not conceptually applicable to aggregation, because the claim is a *proof of knowledge* that the aggregating entity "has seen valid signatures under *these* public keys" without saying anything about what the exact signature strings are. Therefore there is no scope for disagreement about whether a given set of signatures is valid or not; the only guarantee that half aggregation provides is that *somebody* saw valid signatures on those messages under those public keys at some point in time.

Along the same lines of normal vs. batch verification inconsistency, particularly for the Ed25519 signature scheme, it is advised [20] to multiply by a cofactor 8 in single- and batch-verification equations (when batch verification is intended to be used).

Determinism's value in blockchains The history of the flaws of widely-deployed, modern pseudo-random number generator (PRNG) has shown enough variety in root causes to warrant caution, exhibiting bugs [63, 49], probable tampering [21], and poor boot seeding [39]. Yet more recent work has observed correlated low entropy events in public block chains [22, 16], and attributed classes of these events to PRNG seeding.

When juxtaposed with the convenience of deployment afforded by public clouds, often used in the deployment of blockchains, this presents a new challenge. Indeed, deploying a cryptographic algorithm on cloud infrastructure often entails that its components will run as guest processes in a virtualized environment of some sort. Existing literature shows that such guests have a lower rate of acquiring entropy [44, 27], that their PRNG behaves deterministically on boot and reset [26, 57], and that they show coupled entropy in multi-tenancy situations [42].

We suspect the cloud's virtualized deployment context worsen the biases observed in PRNG, and hence recommend the consideration of deterministic variants of both batch verification and aggregation.

The kind of aggregated signature verification in this paper may also be available to deterministic runtimes, which by design disable access to random generator apis. One such example is DJVM [23], where a special Java ClassLoader ensures that loaded classes cannot be influenced by factors such as hardware random number generators, system clocks, network packets or the contents of the local filesystem. Those runtimes are relevant for blockchains, which despise non-determinism including RNG invocations to avoid accidental or malicious misuse in smart contracts that would break consensus. Nonetheless, all blockchains support signature verification. A deterministic batch verifier would hence be very useful in these settings, especially as it applies to batching signatures on different messages too (i.e., independent blockchain transactions).

6 Impossibility of non-interactive compression by more than a half

Given that we have shown that it is possible to compress Schnorr signatures by a constant factor, it is natural to ask if we can do better. Indeed, the existence of succinct proof systems where the proofs are smaller than the witnesses themselves indicates that this is possible, even without extra assumptions or trusted setup if one were to use Bulletproofs [17] or IOP based proofs [8, 9] for instance. This rules out proving any non-trivial lower bound on the communication complexity of aggregating Schnorr's signatures. However, one may wonder what overhead is incurred in using such generic SNARKs, given their excellent compression. Here we make progress towards answering this question, in particular we show that non-trivially improving on our aggregation scheme must rely on the hash function used in the instantiation of Schnorr's signature scheme.

We show in Theorem 9 that if the hash function used by Schnorr's signature scheme is modeled as a random oracle, then the verifier must query the nonces associated with each of the signatures to the random oracle. Given that each nonce has 2λ bits of entropy, it is unlikely that an aggregate signature non-trivially smaller than $2n\lambda$ can reliably induce the verifier to query all n nonces.

The implication is that an aggregation scheme that transmits fewer than $2n\lambda$ bits must not be making oracle use of the hash function; in particular it depends on the code of the hash function used to instantiate Schnorr's scheme. To our knowledge, there are no hash functions that are believed to securely instantiate Schnorr's signature scheme while simultaneously allowing for succinct proofs better than applying generic SNARKs to their circuit representations. Note that the hash function must have powerful properties in order for Schnorr's scheme to be proven secure, either believed to be instantiating a random oracle [56] or having strong concrete hardness [50]. Given that the only known techniques for making use of the code of the hash function in this context is by using SNARKs generically, we take this to be an indication that compressing Schnorr signatures with a rate better than 50% will incur the overhead of proving statements about complex hash functions. For instance compressing n Ed25519 signatures at a rate better than 50% may require proving n instances of SHA-512 via SNARKs.

For "self-verifying" objects such as signatures (aggregate or otherwise) one can generically achieve some notion of compression by simply omitting $O(\log \lambda)$ bits of the signature string, and have the verifier try all possible assignments of these omitted bits along with the transmitted string, and accept if any of them verify. Conversely, one may instruct the signer to generate a signature such that the trailing $O(\log \lambda)$ bits are always zero (similarly to blockchain mining) and need not be transmitted (this is achieved by repeatedly signing with different random tapes). There are two avenues to apply these optimizations:

1. **Aggregating optimized Schnorr signatures.** One could apply these optimizations to the underlying Schnorr signature itself, so that aggregating them even with our scheme produces an aggregate signature of size $2n(\lambda - O(\log \lambda))$ which in practice is considerably better than $2n\lambda$ as n scales. In the rest of this section we only consider

the aggregation of Schnorr signatures that are produced by the regular unoptimized signing algorithm, i.e. where nonces have the full $2n\lambda$ bits of entropy. This quantifies the baseline for the most common use case, and has the benefit of a simpler proof. However, it is simple to adapt our proof technique to show that aggregation with compression rate non-trivially greater than 50% is infeasible with this optimized Schnorr as the baseline as well.

2. **Aggregating unoptimized Schnorr signatures**. One could apply this optimization to save $O(\log \lambda)$ bits overall in the aggregated signature. In this case, $O(\log \lambda)$ is an additive term in the aggregated signature size and its effect disappears as n increases, and so we categorize this a trivial improvement.

Proof Intuition. Our argument hinges on the fact that the verifier of a Fiat-Shamir transformed proof must query the random oracle on the 'first message' of the underlying sigma protocol. In Schnorr's signature scheme, this represents that the nonce R must be queried by the verifier to the random oracle. It then follows that omitting this R value for a single signature in the aggregate signature with noticeable probability will directly result in an attack on unforgeability of the aggregate signature.

We first fix the exact distribution of signatures that must be aggregated, and then reason about the output of any given aggregation scheme on this input.
$\mathsf{GenSigs}(n, 1^\lambda)$:

1. For each $i \in [n]$, sample $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $r_i \leftarrow F_s$, and compute $R_i = r_i \cdot B$ and $\sigma_i = \mathsf{sk}_i \cdot \mathsf{RO}(\mathsf{pk}_i, R_i, 0) + r_i$

2. Output $(\mathsf{pk}_i, R_i, \sigma_i)_{i\in[n]}$

The $\mathsf{GenSigs}$ algorithm simply creates n uniformly sampled signatures on the message '0'.

Theorem 9. *Let* $(\mathsf{AggregateSig}, \mathsf{AggregateVerify})$ *characterize an aggregate signature scheme for* KeyGen, Sign, Verify *as per Schnorr with group* $(\mathbb{G}, B, q)$ *such that* $|q| = 2\lambda$. *Let* $\mathcal{Q}_V$ *be the list of queries made to* RO *by*

$$\mathsf{AggregateVerify}^{\mathsf{RO}}(\mathsf{AggregateSig}^{\mathsf{RO}}(\{\mathit{pk}_i, R_i, \sigma_i\}_{i\in[n]}))$$

where $(\mathit{pk}_i, R_i, \sigma_i)_{i\in[n]} \leftarrow \mathsf{GenSigs}(n, 1^\lambda)$. *Then for any* n, $\mathsf{max}((\Pr[(\mathit{pk}_i, R_i, 0) \notin \mathcal{Q}_V])_{i\in[n]})$ *is negligible in* λ.

Proof. Let $\varepsilon = \max((\Pr[(\mathsf{pk}_i, R_i, 0) \notin \mathcal{Q}_V])_{i\in[n]})$, and let $j \in [n]$ be the corresponding index. We now define an alternative signature generation algorithm as follows,
$\mathsf{GenSigs}^*(n, j, \mathsf{pk}_j, 1^\lambda)$:

1. For each $i \in [n]\backslash j$, sample $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $r_i \leftarrow F_s$, and compute $R_i = r_i \cdot B$ and $\sigma_i = \mathsf{sk}_i \cdot \mathsf{RO}(\mathsf{pk}_i, R_i, 0) + r_i$

2. Sample $\sigma_j \leftarrow F_s$ and $e_j \leftarrow F_s$

3. Set $R_j = \sigma_i \cdot B - e_j \cdot \mathsf{pk}_j$
4. Output $(\mathsf{pk}_i, R_i, \sigma_i)_{i\in[n]}$

Observe the following two facts about $\mathsf{GenSigs}^*$: (1) it does not use sk_j, and (2) the distributions of $\mathsf{GenSigs}$ and $\mathsf{GenSigs}^*$ appear identical to any algorithm that does not query $(\mathsf{pk}_i, R_i, 0)$ to RO. The first fact directly makes $\mathsf{GenSigs}^*$ conducive to an adversary in the aggregated signature game: given challenge public key pk, simply invoke $\mathsf{GenSigs}^*$ with $\mathsf{pk}_j = \mathsf{pk}$ to produce $(\mathsf{pk}_i, R_i, \sigma_i)_{i\in[n]}$ and then feed these to $\mathsf{AggregateSig}$[6]. The advantage this simple adversary is given by the probability that the verifier does not notice that that $\mathsf{GenSigs}^*$ did not supply a valid signature under pk^* to $\mathsf{AggregateSig}$, and we can quantify this using the second fact as follows:

$$
\begin{aligned}
&\Pr[\mathsf{AggregateVerify}^{\mathsf{RO}}(\mathsf{AggregateSig}^{\mathsf{RO}}(\mathsf{GenSigs}^*(n, j, \mathsf{pk}_j, 1^\lambda))) = 1] \\
&= \Pr[\mathsf{AggregateVerify}^{\mathsf{RO}}(\mathsf{AggregateSig}^{\mathsf{RO}}(\mathsf{GenSigs}(n, 1^\lambda))) = 1] - \Pr[(\mathsf{pk}_i, R_i, 0) \in \mathcal{Q}_V] \\
&= 1 - \Pr[(\mathsf{pk}_i, R_i, 0) \in \mathcal{Q}_V] \\
&= 1 - (1 - \varepsilon) = \varepsilon
\end{aligned}
$$

Assuming unforgeability of the aggregated signature scheme, ε must be negligible. □

Acknowledgements. The authors would like to thank Payman Mohassel (Novi/Facebook) and Isis Lovecruft for insightful discussions at the early stages of this work; Daniel J. Bernstein, Jonas Nick, Tim Ruffing and Yannick Seurin for their helpful feedback mostly around batch verification; and all anonymous reviewers of this paper for comments and suggestions that greatly improved the quality of this paper.

Summary of changes

- Mar 16, 2021: First version (very close to the initial submission for the CT-RSA 2021 conference on Dec 02, 2020).
- Mar 19, 2021: Addressing malleability scenarios in deterministic batch verification and recommendations to allow for output consistency between normal and batch verification.

References

[1] Diego F. Aranha, Claudio Orlandi, Akira Takahashi, and Greg Zaverucha. Security of hedged fiat-shamir signatures under fault attacks. In *Eurocrypt*, 2020.

[2] Matilda Backendal, Mihir Bellare, Jessica Sorrell, and Jiahao Sun. The fiat-shamir zoo: relating the security of different signature variants. In *Nordic Conference on Secure IT Systems*, pages 154–170. Springer, 2018.

[6] If necessary, intercept $(\mathsf{pk}_j, R_j, 0)$ queried by $\mathsf{AggregateSig}$ to RO, and respond with e_j as set by $\mathsf{GenSigs}^*$

[3] Ali Bagherzandi, Jung-Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In *ACM CCS*, 2008.

[4] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Lecture Notes in Computer Science*, pages 236–250. Springer Berlin Heidelberg, 1998.

[5] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer, Heidelberg, August 1993.

[6] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Security proofs for identity-based identification and signature schemes. *Journal of Cryptology*, 22(1):1–61, 2009.

[7] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *ACM CCS*, 2006.

[8] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. https://eprint.iacr.org/2018/046.

[9] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *Eurocrypt*, 2019.

[10] Daniel J Bernstein, Jeroen Doumen, Tanja Lange, and Jan-Jaap Oosterwijk. Faster batch forgery identification. In *International Conference on Cryptology in India*, pages 454–473. Springer, 2012.

[11] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In *CHES*, 2011.

[12] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *Asiacrypt*, 2018.

[13] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Eurocrypt*, 2003.

[14] Dan Boneh, Craig Gentry, Ben Lynn, Hovav Shacham, et al. A survey of two signature aggregation techniques, 2003.

[15] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *Asiacrypt*, 2001.

[16] Joachim Breitner and Nadia Heninger. Biased nonce sense: Lattice attacks against weak ecdsa signatures in cryptocurrencies. In *International Conference on Financial Cryptography and Data Security*, pages 3–20. Springer, 2019.

[17] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE S&P*, pages 315–334, 2018.

[18] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.

[19] Konstantinos Chalkias, François Garillot, Yashvanth Kondi, and Valeria Nikolaenko. ed25519-dalek-fiat, branch:half-aggregation. `https://github.com/novifinancial/ed25519-dalek-fiat/tree/half-aggregation`, 2021.

[20] Konstantinos Chalkias, François Garillot, and Valeria Nikolaenko. Taming the many eddsas. Technical report, Cryptology ePrint Archive, Report 2020/1244, https://eprint. iacr. org/2020/1244, 2020.

[21] Stephen Checkoway, Hovav Shacham, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Ralf-Philipp Weinmann, and Eric Rescorla. A Systematic Analysis of the Juniper Dual EC Incident. In *ACM CCS*, 2016.

[22] Nicolas T. Courtois, Pinar Emirdag, and Filippo Valsorda. Private key recovery combination attacks: On extreme fragility of popular bitcoin key management, wallet and cold storage solutions in presence of poor RNG events, 2014.

[23] Djvm - the deterministic jvm library, 2020.

[24] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1084–1101. IEEE, 2019.

[25] Tadge Dryja. Per-block non-interactive schnorr signature aggregation, 2017.

[26] Adam Everspaugh, Yan Zhai, Robert Jellinek, Thomas Ristenpart, and Michael Swift. Not-So-Random Numbers in Virtualized Linux and the Whirlwind RNG. In *2014 IEEE Symposium on Security and Privacy*, pages 559–574. IEEE, May 2014.

[27] Diogo A.B. Fernandes, Liliana F.B. Soares, Mario M. Freire, and Pedro R.M. Inacio. Randomness in Virtual Machines. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 282–286. IEEE, Dec 2013.

[28] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Crypto*, 1987.

[29] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *Crypto*, 2005.

[30] Nils Fleischhacker, Tibor Jager, and Dominique Schröder. On tight security proofs for Schnorr signatures. *Journal of Cryptology*, 32(2):566–599, April 2019.

[31] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In *Crypto*, 2018.

[32] Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind schnorr signatures and signed elgamal encryption in the algebraic group model. In *Eurocrypt*, 2020.

[33] Bundesamt für Sicherheit in der Informationstechnik (BSI). Elliptic curve cryptography, technical guideline tr-03111, 2009.

[34] Rosario Gennaro, Darren Leigh, R. Sundaram, and William S. Yerazunis. Batching Schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices. In *Asiacrypt*, 2004.

[35] Ashrujit Ghoshal and Stefano Tessaro. Tight state-restoration soundness in the algebraic group model. *Cryptology ePrint Archive*, 2020, 2020.

[36] Mike Hamburg. Ed448-goldilocks, a new elliptic curve. Cryptology ePrint Archive, Report 2015/625, 2015. `http://eprint.iacr.org/2015/625`.

[37] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols: Techniques and Constructions*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

[38] Carmit Hazay and Yehuda Lindell. A note on zero-knowledge proofs of knowledge and the zkpok ideal functionality. *IACR Cryptol. ePrint Arch.*, 2010:552, 2010.

[39] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *USENIX Security Symposium*, 2012.

[40] Kazuharu Itakura and Katsuhiro Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development*, 1983.

[41] S. Josefsson and I. Liusvaara. Edwards-Curve Digital Signature Algorithm (EdDSA), 2017.

[42] Brendan Kerrigan and Yu Chen. A study of entropy sources in cloud computers: random number generation on cloud hosts. In *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*, pages 286–298. Springer, 2012.

[43] Chelsea Komlo and Ian Goldberg. Frost: Flexible round-optimized schnorr threshold signatures. *IACR Cryptol. ePrint Arch*, 2020.

[44] Rashmi Kumari, Mohsen Alimomeni, and Reihaneh Safavi-Naini. Performance Analysis of Linux RNG in Virtualized Environments. In *ACM Workshop on Cloud Computing Security Workshop*, 2015.

[45] Changshe Ma, Jian Weng, Yingjiu Li, and Robert Deng. Efficient discrete logarithm based multi-signature scheme in the plain public key model. *Designs, Codes and Cryptography*, 54(2):121–133, 2010.

[46] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. Cryptology ePrint Archive, Report 2018/068, 2018. `https://eprint.iacr.org/2018/068`.

[47] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Designs, Codes and Cryptography*, 87(9):2139–2164, 2019.

[48] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures. In *ACM CCS*, 2001.

[49] Kai Michaelis, Christopher Meyer, and Jörg Schwenk. Randomly failed! the state of randomness in current java implementations. In *Cryptographers' Track at the RSA Conference*. Springer, 2013.

[50] Gregory Neven, Nigel P Smart, and Bogdan Warinschi. Hash function requirements for schnorr signatures. *Journal of Mathematical Cryptology*, 2009.

[51] Jonas Nick, Tim Ruffing, and Yannick Seurin. Musig2: Simple two-round schnorr multi-signatures. Technical report, IACR Cryptol. ePrint Arch, 2020.

[52] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. Musig-dn: Schnorr multi-signatures with verifiably deterministic nonces. In *ACM CCS*, 2020.

[53] Rafael Pass. On deniability in the common reference string and random oracle model. In *Crypto*, 2003.

[54] Wuille Pieter, Nick Jonas, and Ruffing Tim. Bip: 340, schnorr signatures for secp256k1, 2020.

[55] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *Eurocrypt*, 1996.

[56] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000.

[57] T Ristenpart and S Yilek. When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography. In *NDSS*, 2010.

[58] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multi-party signatures against rogue-key attacks. In *Eurocrypt*, 2007.

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

[60] Yannick Seurin. On the exact security of Schnorr-type signatures in the random oracle model. In *Eurocrypt*, 2012.

[61] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *Eurocrypt*, 1997.

[62] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities" honest or bust" with decentralized witness cosigning. In *IEEE S&P)*, 2016.

[63] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When private keys are public: Results from the 2008 debian openssl vulnerability. In *ACM SIGCOMM Internet Measurement Conference, IMC*, 2009.

[64] Yunlei Zhao. Aggregation of gamma-signatures and applications to bitcoin. *IACR Cryptol. ePrint Arch.*, 2018:414, 2018.