Unique Aggregate Signatures with Applications to Distributed Verifiable Random Functions*

Veronika Kuchta and Mark Manulis

Department of Computing, University of Surrey, United Kingdom
v.kuchta@surrey.ac.uk, mark@manulis.eu

Abstract. The computation process of a Distributed Verifiable Random Function (DVRF) on some input specified by the user involves multiple, possibly malicious servers, and results in a publicly verifiable pseudorandom output to the user. Previous DVRF constructions assumed trusted generation of secret keys for the servers and imposed a threshold on the number of corrupted servers.

In this paper we propose the first generic approach for building DVRFs, under much weaker setup assumptions, where we only require existence of a shared random string. More precisely, we first aim at constructions of Distributed Verifiable Unpredictable Functions (DVUF) that can then be converted to DVRF using inner products with a random string as specified by Micali, Rabin, and Vadhan (FOCS'99) for the non-distributed VUF/VRF case.

Our main contribution are generic DVUF constructions from aggregate signatures that satisfy the property of *uniqueness*. We define uniqueness for two flavors of aggregate signatures (with public and sequential aggregation) and show that both flavors can be used to obtain DVUF. By proving uniqueness of existing pairing-based aggregate signature schemes we immediately obtain several concrete communication-efficient DVUF/DVRF instantiations.

1 Introduction

Unique Signatures and VRFs The uniqueness property for digital signatures, introduced by Goldwasser and Ostrovsky [19], guarantees that all signatures produced by one signer on the same message remain "similar" in that there exists an efficient publicly computable function that yields the same unpredictable value on input of any such signature. This property has been explored for traditional signature schemes [19,24] and more recently in the context of advanced schemes such as group signatures [14] and ring signatures [15] where it enabled more efficient anonymity revocation resp. linkability procedures. The

* Following the publication of this work the authors would like to thank Kwangsu Lee for pointing out that the sequential aggregate signature scheme from [22] is not unique. This invalidates our Theorem 2. All remaining theorems remain valid.

uniqueness property doesn't require all signatures to be identical as it is the case for deterministic schemes. In fact, it is sufficient for an unique signature to contain some unique component that can be used to link different signatures of the same signer on the same message.

Goldwasser and Ostrovsky [19] established the equivalence between unique signatures and non-interactive zero-knowledge proofs (NIZK) for hard-to-predict languages. The main application of unique signatures, e.g. in [24,12], has been the construction of Verifiable Random Functions (VRF) [25] — these are pseudorandom functions with a corresponding private/public key pair (sk, pk) that on some input x output a pair $(F(sk, x), \pi(sk, x))$ where $F(sk, x)$ is pseudorandom and $\pi(sk, x)$ represents a proof for the correctness of the computation that can be verified in a public fashion using pk. In order to construct VRFs from unique signatures one first needs to construct a so-called Verifiable Unpredictable Function (VUF) and then apply the transformation from [25] to convert VUF into VRF. For the actual construction of VUF out of a unique signature scheme one simply considers the signer's secret key as a secret seed and treats the resulting unique signature (or its unique component) as a VUF output, whose correctness can be checked publicly using the verification procedure of the signature scheme and the signer's public key. As observed in [1], who constructed VRFs in the identity-based setting, VRFs turned out to be very useful for many applications, including resettable zero-knowledge proofs [26], micropayment schemes [28], updatable zero-knowledge databases [21], and verifiable transaction escrow schemes [20].

Distributed VRFs In a distributed VRF (DVRF) setting, considered by Dodis [11], there are multiple parties (servers), each in possession of its own secret and public key such that any subset of n servers can participate in the computation process. The approach taken in [11] to build a DVRF scheme was to first propose a concrete VRF construction and then turn it into DVRF by using the $(t+1, n)$-secret sharing technique [30] to equip servers with individual shares sk_i of the private VRF key sk. In addition, for each party i an individual public key pk_i is derived from sk_i. In order to compute the DVRF output $(F(sk, x), \pi(sk, x))$ the input x is communicated by the user to each of the n parties that reply with their intermediate VRF outputs $(F(sk_i, x), \pi(sk_i, x))$. If at least $t+1$ intermediate VRF proofs $\pi(sk_i, x)$ are valid (which is checked using corresponding public keys pk_i) then the final DVRF output $(F(sk, x), \pi(sk, x))$ can be computed by the user through polynomial interpolation. The validity of the resulting DVRF proof $\pi(sk, x)$ can be checked publicly using the original pk of the underlying VRF scheme.

The DVRF construction from [11] is reasonably efficient, yet has a few limitations, as discussed in the following. One consequence of using $(t + 1, n)$-secret sharing is that in order to guarantee pseudorandomness of $F(sk, x)$ at least $t+1$ parties involved in its computation process must remain honest. The DVRF scheme from [11] requires a trusted setup procedure for the generation and distribution of shares sk_i, which is a strong assumption. The assumption on trusted setup could possibly be removed by adopting a matching Distributed Key Gener-

ation (DKG) protocol, e.g. [16], yet at the cost of reduced efficiency and possibly further restrictions on the ratio between the threshold value $t+1$ and n.

We observe that the approach taken in [11] to apply threshold cryptography on top of a non-distributed VRF scheme is so far the only known way to construct DVRF schemes.

The original motivation for DVRF schemes given in [11] is the practical realization of random oracles, a theoretical construct introduced in [4] that is frequently used in security proofs of cryptographic schemes. In a nutshell, random oracle is a mathematical function that on any new input outputs a random string from the output domain. Goldreich, Goldwasser and Micali [17] were the first who showed how to simulate a random oracle for fixed-length input and output strings by using a PRF. Canetti et al. [10] showed that no fixed public function can generically replace the random oracle. They demonstrated that a PRF should not be expected to offer a general solution for realizing random oracles. Micali, Rabin, and Vadhan [25] suggested that a random oracle can be realized using VRF schemes. Dodis observed that this would require a significant amount of trust put into a single party that computes VRF outputs and argued that it is desirable to distribute this trust across multiple, ideally independent parties.

Our DVRF Approach: Unique (Sequential) Aggregate Signatures In this work we propose another approach for building DVRF schemes without imposing trust assumptions on the generation of secrets keys sk_i for the involved servers and without requiring any particular threshold on the number of honest servers. Our main contribution is to build DVRF schemes generically from different flavors of aggregate signatures [7,23,22] where each signer i has its own private/public key pair (sk_i, pk_i) and a set of n signers contributes to the computation of an aggregate signature $\bar{\sigma}$ on some set of (possibly different) messages $\mathbf{m} = \{m_1, \ldots, m_n\}$ where the size of resulting $\bar{\sigma}$ is independent of n. The signature can be verified using the set of public keys $\mathbf{pk} = \{pk_1, \ldots, pk_n\}$.

Just as in case of a VRF that can be obtained from a VUF we show that different flavors of aggregate signatures can be used to build a distributed VUF (DVUF), which can then be converted to a DVRF using the techniques from [25]. In order to construct DVUFs from aggregate signatures the latter require some sort of uniqueness. Since the property of uniqueness in the context of aggregate signatures has not been considered so far, we first need to define it. We define uniqueness for aggregate signatures with public aggregation (cf. [7]) and denote such schemes by UAS, and for sequential aggregate signatures (cf. [23,22]), denoted by USAS. Our definition of uniqueness in both cases roughly means that for any aggregate signature $\bar{\sigma}$ produced on the same set of messages $\mathbf{m}$ using the same set of private keys $\mathbf{sk} = \{sk_1, \ldots, sk_n\}$ there exists no other aggregate signature $\bar{\bar{\sigma}}$ such that $\mathtt{Verify}(\mathbf{pk}, \mathbf{m}, \bar{\sigma}) = \mathtt{Verify}(\mathbf{pk}, \mathbf{m}, \bar{\bar{\sigma}}) = 1$.

At a high level, our DVUF construction from any UAS/USAS scheme proceeds as follows: the DVUF public key $\mathbf{pk}$ consists of all UAS/USAS public keys pk_i while each UAS/USAS secret key sk_i is generated individually by the respective DVUF server i. The DVUF output $(F(\mathbf{sk}, x), \pi(\mathbf{sk}, x))$ is essentially given

by $(\boldsymbol{unq}(\bar{\sigma}), \bar{\sigma})$ where $\boldsymbol{unq}(\bar{\sigma})$ determines the *unique* component of aggregate signature $\bar{\sigma}$, which in turn plays the role of the proof. Note that each server signs the same message x that is specified by the user as input to DVUF. The actual computation process and interaction differs for UAS and USAS schemes. Our most efficient UAS-based DVUF construction requires only one communication round in which the user sends x to each of the n servers, obtains their individual signatures and then aggregates them locally to obtain the DVUF output. In the USAS-based DVUF construction the user needs to contact n servers sequentially and obtains the resulting DVUF output and the proof upon contacting the last server in the sequence.

Our UAS/USAS-based approach for constructing DVUF and consequently DVRF has two advantages over [11]: (1) the uniqueness and unforgeability properties of UAS/USAS schemes will guarantee that the DVRF output $F(\mathbf{sk}, x)$ is pseudorandom even if the adversary corrupts up to $n-1$ servers; (2) since each server i can generate her own UAS/USAS key pair (sk_i, pk_i) independently, our DVUF construction doesn't require any trusted setup procedure for the distribution of sk_i. When using the inner product-based technique from [25] to convert out DVUF outputs into DVRF outputs we need to impose existence of a shared random string [13] as an additional, albeit much weaker setup assumption than the trustworthy generation of secret keys adopted in [11].

DVUF/DVRF Instantiations We obtain several concrete DVUF/DVRF instantiations from existing (sequential) aggregate signatures schemes by proving the uniqueness property for the (pairing-based) aggregate signature schemes by Boneh et al. [7], Lu et al. [22], and Schröder [31]. The scheme from [7] is a very efficient random oracle-based construction that supports public aggregation of signatures. The schemes from [22,31] offer sequential aggregation in the standard model and are based on two popular signature schemes; in particular, [22] offers aggregation of Waters signatures [32], while [31] shows how to aggregate Camenisch-Lysyanskaya [9] signatures.

A Note on Multisignatures In our generic DVUF constructions parties compute aggregate signatures on the same input message, specified by the user. This step can also be realized using multisignatures [5] that represent a special case of aggregate signatures in that all signers are required to use the same message in the execution of the signing protocol. Our generic DVUF constructions can therefore be analysed from the perspective of unique multisignatures, yet their instantiations may not necessarily be more communication-efficient than those presented in our work. This is because all existing aggregate signatures are non-interactive in that at most one message needs to be exchanged between the signers, which is not the general case for multisignatures. For instance, the signing process of multisignature schemes from [27,3,2] requires several rounds of interaction amongst the participating signers. Those schemes, if unique, can be possibly used to realize a DVUF but at the cost if the increased communication overhead, in comparison to non-interactive aggregate signature schemes used in our constructions. On the other hand, there exist several multisigna-

ture schemes where the signing process is non-interactive, e.g. [5,22,6,33]. These schemes seem to satisfy the uniqueness property and could possibly be used to obtain communication-efficient DVUF constructions. For instance, Boldyreva's scheme [5] that uses Gap Diffie-Hellman groups and is based on BLS signatures [8], when realized using pairings, would offer a similar performance for a DVUF, in the random oracle model, as the aggregate signature scheme from [7] that is used in our work. Similarly, the multisignature scheme from [22], which is based on the signature scheme by Waters [32], could be used to build DVUF in the standard model. The resulting scheme would offer similar communication performance to the DVUF construction that we obtain by using their aggregate signature scheme. One advantage of using these non-interactive multisignatures in comparison to corresponding aggregate signatures is that by adding further "proofs of secret key possession" from [29] one could obtain a higher level of security against rogue key attacks that is notoriously difficult to achieve for the more general case of aggregate signatures.

2 Preliminaries

All concrete constructions used in this paper are in the setting of bilinear groups, defined in the following.

Definition 1 (Bilinear Groups). *Let $\mathcal{G}(1^\lambda)$, $\lambda \in \mathbb{N}$ be an algorithm that on input a security parameter 1^λ outputs the description of two cyclic groups $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$ of prime order q with $|q| = 1^\lambda$, where possibly $\mathbb{G}_1 = \mathbb{G}_2$, and an efficiently computable $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with $\mathbb{G}_T$ being another cyclic group of order q. The group pair $(\mathbb{G}_1, \mathbb{G}_2)$ is called* bilinear *if $e(g_1, g_2) \neq 1$ and $\forall u \in \mathbb{G}_1,\ v \in \mathbb{G}_2,\ \forall a, b \in \mathbb{Z} :\ e(u^a, v^b) = e(u, v)^{ab}$.*

3 Unique Aggregate Signatures

In this section we recall definitions of aggregate signatures with public aggregation and define their uniqueness property. We adopt the syntax and the security model from [7].

Definition 2 (AS scheme). *An* aggregate signature *scheme AS consists of the following algorithms:*

- $\texttt{ParGen}(1^\lambda)$ *is a PPT algorithm that takes as input the security parameter 1^λ and outputs public system parameters I.*
- $\texttt{KeyGen}(I)$ *is a PPT algorithm that takes as input I and generates a private/public key pair (sk_i, pk_i) for an user i.*
- $\texttt{Sign}(sk_i, m_i)$ *is a possibly deterministic algorithm that takes as input a secret key sk_i and a message m_i and outputs a signature σ_i.*
- $\texttt{Verify}(pk_i, m_i, \sigma_i)$ *is a deterministic algorithm that takes as input a candidate signature σ_i, a public key pk_i, and a message m_i, and outputs 1 if the signature is valid and 0 otherwise.*

$\mathtt{Aggregate}(\mathbf{pk}, \mathbf{m}, \boldsymbol{\sigma})$ *is an algorithm that takes as input a set of signatures* $\boldsymbol{\sigma} = (\sigma_1, ..., \sigma_n)$*, public keys* $\mathbf{pk} = (pk_1, \ldots, pk_n)$*, and messages* $\mathbf{m} = (m_1, \ldots, m_n)$*, and outputs an aggregate signature* $\bar{\sigma}$*.*

$\mathtt{AggVerify}(\mathbf{pk}, \mathbf{m}, \bar{\sigma})$ *is a deterministic algorithm that takes as input a candidate aggregate signature* $\bar{\sigma}$*, a set of messages* $\mathbf{m}$ *and public keys* $\mathbf{pk}$*, and outputs 1 if the signature is valid, or 0 otherwise.*

Definition 3 (Unforgeability of AS). *An aggregate signature scheme is* unforgeable *if for any PPT adversary* $\mathcal{A}$*, running in time at most* t *and invoking the signing oracle at most* q_S *times, the probability that the following experiment outputs* 1 *remains negligible in the security parameter* λ*.*

Experiment $\mathtt{Forge}_A^{AS}(\lambda)$
$I \leftarrow \mathtt{ParGen}(1^\lambda)$
$(sk_c, pk_c) \leftarrow \mathtt{KeyGen}(I)$
$(\mathbf{m}^*, \mathbf{pk}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathtt{OSign}(sk_c, \cdot)}(I, pk_c)$
Let m_c *be the message those index in* $\mathbf{m}^*$ *corresponds to the index of* pk_c *in* $\mathbf{pk}^*$*.*
Output 1 if all of the following holds:
- $\mathtt{AggVrfy}(\sigma^*, \mathbf{m}^*, \mathbf{pk}^*) = 1$,
- $m_c \in \mathbf{m}^*$ *was never submitted to* $\mathtt{OSign}(sk_c, \cdot)$

where $\mathcal{A}$ *is given access to the following aggregate signing oracle:*

$\mathtt{OSign}(sk_c \cdot)$*: The adversarial input to the oracle contains a message* m_i *under the public key* pk_c*, the oracle computes the signature* σ_i *on* m_i *using* sk_c *and gives the signature to* $\mathcal{A}$*.*

Our definition of uniqueness for aggregate signatures with public aggregation is given in Definition 4. This definition fits likewise probabilistic and deterministic schemes due to the use of function $\mathtt{unq}$, even though we are not aware of any (non-interactive) probabilistic scheme that supports public aggregation.

Definition 4 (Unique AS). *An unforgeable AS scheme is said to be unique, denoted by UAS, if there exists an efficient deterministic function* $\mathtt{unq}$ *which on input an aggregate signature* $\bar{\sigma}$ *outputs a string of polynomial-size in the security parameter of the scheme such that for any ordered sequence of messages* $\mathbf{m} = (m_1, \ldots, m_n)$ *and public keys* $\mathbf{pk} = (pk_1, \ldots, pk_n)$ *there exist no two aggregate signatures* $\bar{\sigma}$ *and* $\bar{\bar{\sigma}}$ *for which it holds that* $\mathtt{Verify}(\mathbf{pk}, \mathbf{m}, \bar{\sigma}) = \mathtt{Verify}(\mathbf{pk}, \mathbf{m}, \bar{\bar{\sigma}}) = 1$*, and* $\mathtt{unq}(\bar{\sigma}) \neq \mathtt{unq}(\bar{\bar{\sigma}})$*.*

3.1 Uniqueness of Boneh-Gentry-Lynn-Shacham AS Scheme

We recall the aggregate signature scheme with public aggregation from [7] where the hash function $H : \{0,1\}^* \rightarrow \mathbb{G}_1$ is modeled as a random oracle.

$\mathtt{ParGen}(1^\lambda)$**.** On input of the security parameter 1^λ this algorithm outputs public parameters $I = (\mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \psi, e, \mathbb{G}_T, q)$, with $\psi(g_2) = g_1$, where ψ is a computable isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$.

KeyGen(I)**.** For an user i, choose randomly $x_i \xleftarrow{r} \mathbb{Z}_q$ and compute $v_i \leftarrow g_2^{x_i}$. It outputs $(sk_i, pk_i) = (x_i, v_i)$.

Sign(m_i, sk_i)**.** For all i, it takes as input sk_i and a message $m_i \in \{0,1\}^*$. The algorithm computes $h_i \leftarrow H(m_i)$, where $h_i \leftarrow \mathbb{G}_1$ and $\sigma_i \leftarrow h_i^{x_i}$. Output is $\sigma_i \in \mathbb{G}_1$.

Verify(m_i, pk_i, σ_i)**.** For all $i \in [n]$ the algorithm takes as input m_i and σ_i. It outputs 1 if $e(\sigma_i, g_2) = e(h_i, v_i)$ and 0 otherwise.

Aggregate$(\mathbf{pk}, \mathbf{m}, \boldsymbol{\sigma})$**.** On the input $\mathbf{pk}, \mathbf{m}, \boldsymbol{\sigma}$ the algorithm computes $\bar{\sigma} \leftarrow \prod_{i=1}^{n} \sigma_i$. The aggregate signature is $\bar{\sigma} \in \mathbb{G}_1$.

AggVerify$(\mathbf{pk}, \mathbf{m}, \bar{\sigma})$**.** This algorithm takes as input an aggregate signature $\bar{\sigma}$,a sequence of messages $\mathbf{m} = (m_1, ..., m_n)$ and a sequence of pubic keys $\mathbf{pk} = (v_1, \ldots, v_n) \in \mathbb{G}_2$, for all users u_i. The algorithm outputs 1 if the messages m_i are all distinct and $e(\bar{\sigma}, g_2) = \prod_{i=1}^{n} e(h_i, v_i)$. Otherwise the algorithm outputs 0.

The above scheme offers unforgeability in the random oracle model, as already proven in [7]. Interestingly, our Theorem 1 shows that this scheme is unique without imposing the random oracle assumption on H.

Theorem 1. *The Boneh-Gentry-Lynn-Shacham AS scheme is unique according to Definition 4.*

Proof. Assume that there exist two valid aggregate signatures $\bar{\sigma}$ and $\bar{\bar{\sigma}}$ on an ordered sequence of messages $\mathbf{m} = (m_1, ..., m_i)$ such that the equation ***Verify***$(\mathbf{pk}, \mathbf{m}, \bar{\sigma}) =$ ***Verify***$(\mathbf{pk}, \mathbf{m}, \bar{\bar{\sigma}}) = 1$. We define ***unq***$(\bar{\sigma})$ as an identity function. That is, ***unq***$(\bar{\sigma}) = \bar{\sigma}$ and ***unq***$(\bar{\bar{\sigma}}) = \bar{\bar{\sigma}}$. We know that $\bar{\sigma} = \prod_{j=1}^{i} h_j^{x_j}$. In the following we prove by induction on i that $\bar{\bar{\sigma}} = \bar{\sigma}$:

Base step: $i = 1$**.** The signature $\bar{\bar{\sigma}}$ must satisfy the verification process $e(\sigma, g_2) = e(h, v)$, i.e. $e(\bar{\bar{\sigma}}, g_2) = e(h, g_2^x) = e(h^x, g_2)$. It holds only if $\bar{\bar{\sigma}} = h^x = \bar{\sigma}$.

Induction step: $i - 1 \mapsto i$. Let the theorem hold for $i - 1$. The verification algorithm will accept $\bar{\bar{\sigma}}_i$ if it satisfies the verification equation $e(\bar{\bar{\sigma}}_i, g_2) = \prod_{j=1}^{i} e(h_j, v_j)$. By the induction hypothesis we have the validity for $i - 1$ aggregated signatures, i.e. $\bar{\bar{\sigma}}_{i-1} = \prod_{j=1}^{i-1} h_j^{x_j}$ and $\bar{\bar{\sigma}}_i = \bar{\bar{\sigma}}_{i-1} \cdot \tilde{\sigma}_i$. We put this value

into the verification equation such that:

$$e\left(\left(\prod_{j=1}^{i-1} h_j^{x_j}\right)\cdot\tilde{\sigma}_i, g_2\right) = \prod_{j=1}^{i} e(h_j, v_j) = \prod_{j=1}^{i}(h_j, g_2^{x_j})$$
$$\Leftrightarrow e\left(\prod_{j=1}^{i-1} h_j^{x_j}, g_2\right) e(\tilde{\sigma}_i, g_2) = \prod_{j=1}^{i-1} e(h_j, g_2^{x_j}) e(h_i, g_2^{x_i})$$
$$\Leftrightarrow e(\tilde{\sigma}_i, g_2) = e(h_i, g_2^{x_i}) = e(h_i^{x_i}, g_2) \Leftrightarrow \tilde{\sigma}_i = h_i^{x_i}$$

Therefore we have $\bar{\tilde{\sigma}}_i = \bar{\tilde{\sigma}}_{i-1}\cdot\tilde{\sigma}_i = \prod_{j=1}^{i-1} h_j^{x_j}\cdot h_i^{x_i} = \bar{\sigma}_i$.

□

4 Unique Sequential Aggregate Signatures

In the following we recall definitions of sequential aggregate signatures using the syntax and security model from [23,22] and define their uniqueness.

Definition 5 (SAS scheme). *A* sequential aggregate signature *scheme SAS consists of the following algorithms:*

- $\texttt{ParGen}(1^\lambda)$ *is a PPT algorithm that takes as input the security parameter* 1^λ *and outputs public system parameters* I*.*
- $\texttt{KeyGen}(I)$ *is a PPT algorithm that takes as input* I *and outputs a private/public key pair* (sk_i, pk_i) *for an user* i*.*
- $\texttt{AggSign}(sk_i, m_i, \bar{\sigma}_{i-1}, \mathbf{m}_{i-1}, \mathbf{pk}_{i-1})$ *is a PPT algorithm that on input a private key* sk_i*, a message* $m_i \in \{0,1\}^*$*, an aggregate-so-far signature* $\bar{\sigma}_{i-1}$*, a sequence of messages* $\mathbf{m}_{i-1} = (m_1, \ldots, m_{i-1})$ *and public keys* $\mathbf{pk}_{i-1} = (pk_1, \ldots, pk_{i-1})$*, outputs the aggregate-so-far signature* $\bar{\sigma}_i$ *for the updated sequences* $\mathbf{m}_i = (m_1, \ldots, m_i)$ *and* $\mathbf{pk}_i = (pk_1, \ldots, pk_i)$*.*
- $\texttt{AggVerify}(\bar{\sigma}_i, \mathbf{m}_i, \mathbf{pk}_i)$ *takes as input an aggregate-so-far signature* σ_i*, a sequence of messages* $\mathbf{m}_i$ *and public keys* $\mathbf{pk}_i$ *and outputs 1 if the signature is valid, or 0 otherwise.*

An SAS scheme is said to be complete*, if for any sequence* $(sk_1, pk_1), \ldots, (sk_n, pk_n)$ *with each* $(sk_i, pk_i) \leftarrow \texttt{KeyGen}(I)$*,* $(m_1, ..., m_n)$ *with each* $m_i \in \{0,1\}^*$*, and some non-empty* $\bar{\sigma}_{i-1}$ *for which* $\texttt{AggVerify}(\bar{\sigma}_{i-1}, \mathbf{m}_{i-1}, \mathbf{pk}_{i-1}) = 1$*, for any* $\bar{\sigma}_i \leftarrow \texttt{AggSign}(sk_i, m_i, \bar{\sigma}_{i-1}, \mathbf{m}_{i-1}, \mathbf{pk}_{i-1})$*:* $\texttt{AggVerify}(\bar{\sigma}_i, \mathbf{m}_i, \mathbf{pk}_i) = 1$*.*

Definition 6 (Unforgeability of SAS). *An SAS scheme is unforgeable if for any PPT adversary* $\mathcal{A}$*, running in time at most* t *and invoking the signing oracle at most* q_S *times, the probability that the following experiment outputs 1 remains negligible in the security parameter* λ*.*

Experiment $\texttt{Forge}_A^{SAS}(\lambda)$
$I \leftarrow \texttt{ParGen}(1^\lambda)$
$(sk_c, pk_c) \leftarrow \texttt{KeyGen}(I)$
$(\mathbf{m}^*, \mathbf{pk}^*, \sigma^*) \leftarrow \mathcal{A}^{\texttt{OSeqAgg}(sk_c,\cdot)}(I, pk_c)$
Let C *denote the list of all registered key pairs* (sk_i, pk_i) *and* m_c *be the message those index in* $\mathbf{m}^*$ *corresponds to the index of* pk_c *in* $\mathbf{pk}^*$.
Output 1 if all of the following holds:
- *for any pair* $pk_i, pk_j \in \mathbf{pk}^*$ *with* $i \neq j$: $pk_i \neq pk_j$
- $\texttt{AggVerify}(\sigma^*, \mathbf{m}^*, \mathbf{pk}^*) = 1$,
- $m_c \in \mathbf{m}^*$ *was never amongst the inputs to* $\texttt{OSeqAgg}(sk_c, \cdot)$

where $\mathcal{A}$ *is given access to the following sequential aggregate signing oracle and the key registration oracle:*

$\texttt{OSeqAgg}(sk_c, \cdot)$*: The adversarial input to the signing oracle consists of a message* m*, an aggregate-so-far signature* $\bar{\sigma}_{i-1}$*, a sequence of messages* $\mathbf{m}_{i-1}$ *and public keys* $\mathbf{pk}_{i-1}$*. The oracle computes* $\bar{\sigma}_i \leftarrow \texttt{AggSign}(sk_c, m, \bar{\sigma}_{i-1}, \mathbf{m}_{i-1}||m, \mathbf{pk}_{i-1}||pk_c)$ *and returns* $\bar{\sigma}_i$ *to* $\mathcal{A}$.

Our definition of uniqueness for unforgeable SAS schemes is given in Definition 7. Note that by requiring the existence of an appropriate deterministic function $\texttt{unq}$ we can cover uniqueness in deterministic and probabilistic SAS schemes. For example, USAS instantiations that we focus on later are all probabilistic SAS schemes that output signatures consisting of multiple components from which one component remains unique. We will prove the uniqueness property of those schemes by using appropriate $\texttt{unq}$ functions for each scheme.

Definition 7 (Unique SAS). *An unforgeable SAS scheme is said to be* unique*, denoted by USAS, if there exists an efficient deterministic function* $\texttt{unq}$ *which on input the aggregate-so-far signature* σ_i *outputs a string of polynomial-size in the security parameter of the scheme such that for any ordered sequence of messages* $\mathbf{m}_i$ *and public keys* $\mathbf{pk}_i$ *there exist no two aggregate-so-far signatures* $\bar{\sigma}_i$ *and* $\bar{\bar{\sigma}}_i$ *for which it holds that* $\texttt{AggVerify}(\bar{\sigma}_i, \mathbf{m}_i, \mathbf{pk}_i) = 1, \texttt{AggVerify}(\bar{\bar{\sigma}}_i, \mathbf{m}_i, \mathbf{pk}_i) = 1$*, and* $\texttt{unq}(\bar{\sigma}_i) \neq \texttt{unq}(\bar{\bar{\sigma}}_i)$.

Note that the uniqueness property of an SAS scheme as defined above respects the order of messages in $\mathbf{m} = (m_1, ..., m_n)$. That is, the resulting aggregate signatures output on permuted sequences of messages in $\mathbf{m}$ for the same set of public keys $\mathbf{pk}$ will differ from each other.

4.1 Uniqueness of Lu-Ostrovsky-Sahai-Shacham-Waters SAS Scheme

The SAS scheme proposed by Lu et al. [22] offers sequential aggregation of Waters signatures [32]. We breifly recall their scheme and explore its uniqueness property.

$\texttt{ParGen}(1^\lambda)$**.** On input the security parameter 1^λ output $I = (q, \mathbb{G}, \mathbb{G}_T, g, e)$ for the bilinear group setting according to Definition 1.

KeyGen(I)**.** Pick random $\alpha, y \xleftarrow{r} \mathbb{Z}_q$ and a random vector $\mathbf{y} = (y_1, ..., y_k) \xleftarrow{r} \mathbb{Z}_q^k$. Compute:

$$u' \leftarrow g^{y'}, \quad \mathbf{u} = (u_1, ..., u_k) \leftarrow (g^{y_1}, ..., g^{y_k}), \quad A \leftarrow e(g,g)^{\alpha}.$$

The private key is set to $sk = (\alpha, y', \mathbf{y}) \in \mathbb{Z}_q^{k+2}$, while the the public key is set to $pk = (A, u', \mathbf{u}) \in \mathbb{G}_T \times \mathbb{G}^{k+1}$. The algorithm outputs (sk, pk).

AggSign$(sk_i, m_i, \bar{\sigma}_{i-1}, \mathbf{m}_{i-1}, \mathbf{pk}_{i-1})$**.** If ***AggVerify***$(\bar{\sigma}_{i-1}, \mathbf{m}_{i-1}, \mathbf{pk}_{i-1}) = 1$ proceed; else output 0. Parse $\bar{\sigma}_{i-1}$ as $(S_1^{'}, S_2^{'}) \in \mathbb{G}^2$. For each $1 \leq i \leq n$ and $1 \leq l \leq k$ set $m_i = (m_{i,1}, ..., m_{i,k}) \in \{0,1\}^k$ as k-bit message of user i and $pk_i = (A_i, u_i^{'}, u_{i,1}, ..., u_i^k) \in \mathbb{G}_T \times \mathbb{G}^{k+1}$ as public key of user i. Compute:

$$w_1 \leftarrow S_1^{'} g^{\alpha} \left(S_2^{'}\right)^{y' + \sum_{l=1}^{k} y_l m_l}, \quad w_2 \leftarrow S_2^{'}.$$

Proceed with the re-randomization step, i.e. pick random $\tilde{r} \in \mathbb{Z}_q$ and output $\bar{\sigma}_i = (S_1, S_2)$ where

$$S_1 \leftarrow w_1 \cdot \left(u' \prod_{l=1}^{k} u_l^{m_l}\right)^{\tilde{r}} \cdot \prod_{j=1}^{i} \left(u_j^{'} \prod_{l=1}^{k} u_{j,l}^{m_{j,l}}\right)^{\tilde{r}} \quad \text{and} \quad S_2 \leftarrow w_2 g^{\tilde{r}}.$$

(Note that $\bar{\sigma}_i = (S_1, S_2)$ is an aggregate-so-far signature on an updated list of messages $\mathbf{m}_{i-1}||m_i$ and corresponding public keys $\mathbf{pk}_{i-1}||pk_i$. The re-randomization step results in randomness update to $r + \tilde{r}$.)

AggVerify$(\bar{\sigma}_i, \mathbf{m}_i, \mathbf{pk}_i)$**.** The input is a candidate aggregate signature $\bar{\sigma}_i$ on messages $\mathbf{m}_i$ under public keys $\mathbf{pk}_i$. Set $\bar{\sigma}_i = (S_1, S_2) \in \mathbb{G}$. Check the following equation:

$$e(S_1, g) \cdot e\left(S_2, \prod_{j=1}^{i} \left(u_j^{'} \prod_{l=1}^{k} u_{j,l}^{m_{j,l}}\right)\right)^{-1} = \prod_{j=1}^{i} A_j$$

If the above equation holds output 1, else output 0.

Theorem 2. *The Lu-Ostrovsky-Sahai-Shacham-Waters SAS Scheme is unique according to Definition 7.*

Proof. Let ***unq*** be a function that outputs the first component of the aggregate signature $\bar{\sigma} = (S_1, S_2)$, i.e. ***unq***$(\bar{\sigma}) = S_1$. Assume that there exists another signature $\bar{\bar{\sigma}}$ that passes the verification process for the same set of messages and public keys as $\bar{\sigma}$ and for which ***unq***$(\bar{\bar{\sigma}}) = \hat{S}_1$. In the following we prove that $\hat{S}_1 = S_1$ by induction on i:

Base step: $i = 1$**.** The verification algorithm will accept the signature $\bar{\bar{\sigma}} = (\hat{S}_1, S_2)$ if it satisfies the following verification equation

$$e(\hat{S}_1, g) \cdot e\left(S_2, u^{'} \prod_{l=1}^{k} u_l^{m_l}\right)^{-1} = e(g,g)^{\alpha}$$

$$\Leftrightarrow \quad e(\hat{S}_1, g) \cdot e\left(g, \left(u^{'} \prod_{l=1}^{k} u_l^{m_l}\right)^{r}\right)^{-1} = e(g,g)^{\alpha}.$$

It holds only if $\hat{S}_1 = g^{\alpha}\left(u^{'}\prod_{l=1}^{k} u_l^{m_l}\right)^r$, because we have then:

$$
\begin{aligned}
& e(\hat{S}_1, g)\cdot e\left(g, \left(u^{'}\prod_{l=1}^{k} u_l^{m_l}\right)^r\right)^{-1} = e(g,g)^{\alpha} \\
\Leftrightarrow \quad & e\left(g^{\alpha}\left(u^{'}\prod_{l=1}^{k} u_l^{m_l}\right)^r, g\right)\cdot e\left(g, \left(u^{'}\prod_{l=1}^{k} u_l^{m_l}\right)^r\right)^{-1} = e(g,g)^{\alpha} \\
\Leftrightarrow \quad & e(g^{\alpha}, g) e\left(\left(u^{'}\prod_{l=1}^{k} u_l^{m_l}\right)^r, g\right) e\left(g, \left(u^{'}\prod_{l=1}^{k} u_l^{m_l}\right)^r\right)^{-1} = e(g,g)^{\alpha}.
\end{aligned}
$$

Induction step: $i-1 \mapsto i$**.** Let the theorem hold for $i-1$. The verification algorithm will accept $\mathtt{unq}(\bar{\sigma}_i) = (\hat{S}_1)_i$ if it satisfies the verification equation:

$$
\begin{aligned}
& e((\hat{S}_1)_i, g)\cdot e\left(S_2, \prod_{j=1}^{i}(u_j^{'}\prod_{l=1}^{k} u_{j,l}^{m_{j,l}})\right)^{-1} = \prod_{j=1}^{i} A_j \\
\Leftrightarrow\ & e((\hat{S}_1)_i, g)\cdot e\left(g, \prod_{j=1}^{i}(u_j^{'}\prod_{l=1}^{k} u_{j,l}^{m_{j,l}})^r\right)^{-1} = \prod_{j=1}^{i} e(g,g)^{\alpha_j}
\end{aligned}
$$

By induction hypothesis we have $(\hat{S}_1)_{i-1} = \prod_{j=1}^{i-1} g^{\alpha_j}\prod_{j=1}^{i-1}(u_j^{'}\prod_{l=1}^{k} u_{j,l}^{m_{j,l}})^r$, such that $(\hat{S}_1)_i = (\hat{S}_1)_{i-1}\cdot\delta$. We obtain the following equation:

$$
\begin{aligned}
& e\left((\hat{S}_1)_{i-1}\cdot\delta, g\right)\cdot e\left(S_2, \prod_{j=1}^{i}(u_j^{'}\prod_{l=1}^{k} u_{j,l}^{m_{j,l}})\right)^{-1} = \prod_{j=1}^{i} A_j \\
& \Leftrightarrow e\left((\hat{S}_1)_{i-1}\cdot\delta, g\right)\cdot e\left(g, \prod_{j=1}^{i-1}(u_j^{'}\prod_{l=1}^{k} u_{j,l}^{m_{j,l}})^r\cdot\left(u_i^{'}\prod_{l=1}^{k} u_{i,l}^{m_{i,l}}\right)^r\right)^{-1} \\
& = \prod_{j=1}^{i} e(g,g)^{\alpha_j} \Leftrightarrow \prod_{j=1}^{i-1} e(g,g)^{\alpha_j} e(g,\delta) e\left(g, \left(u_i^{'}\prod_{l=1}^{k} u_{i,l}^{m_{i,l}}\right)^r\right)^{-1} = \prod_{j=1}^{i} e(g,g)^{\alpha_j}
\end{aligned}
$$

The last equation holds if $\delta = g^{\alpha_i}\left(u_i^{'}\prod_{l=1}^{k} u_{i,l}^{m_{i,l}}\right)^r$. This implies the desired equality

$$
(\hat{S}_1)_i = (\hat{S}_1)_{i-1}\cdot\delta = \prod_{j=1}^{i} g^{\alpha_j}\left(u_j^{'}\prod_{l=1}^{k} u_{j,l}^{m_{j,l}}\right)^r = S_1 = \mathtt{unq}(\bar{\sigma}).
$$

□

4.2 Uniqueness of Schröder SAS Scheme

The SAS scheme proposed by Schröder [31] offers sequential aggregation for Camenisch-Lysyanskaya (CL) signatures [9]. The SAS scheme slightly modifies the original CL signatures by introducing an additional signature component, denoted in the following by D. We will essentially rely on this new component when proving the uniqueness property of the scheme.

$\mathit{ParGen}(1^\lambda)$. Output the public parameters $I = (\mathbb{G}, \mathbb{G}_T, g, e)$ for the bilinear group setting according to Definition 1.

$\mathit{KeyGen}(I)$. For each signer i choose $x_i \leftarrow \mathbb{Z}_q$ and $y_i \leftarrow \mathbb{Z}_q$ and sets $X_i = g^{x_i}$, $Y_i = g^{x_i}$ for $i \in [n]$. The algorithm returns $sk_i = (x_i, y_i)$ and $pk_i = (X_i, Y_i)$.

$\mathit{AggSign}(sk_i, m_i, \bar{\sigma}_{i-1}, \mathbf{m}_{i-1}, \mathbf{pk}_{i-1})$. The algorithm takes as input a secret signing key sk_i, a message $m_i \in \mathbb{Z}_q$, an aggregate-so-far $\bar{\sigma}_{i-1}$ a sequence of messages $\mathbf{m}_{i-1} = (m_1, ..., m_{i-1})$ and a sequence of public keys $\mathbf{pk}_{i-1} = (pk_1, ..., pk_{i-1})$. The algorithm first checks that $|\mathbf{m}| = |\mathbf{pk}|$ and that the sequential verification $\mathit{AggVerify}(\bar{\sigma}_{i-1}, \mathbf{m}_{i-1}, \mathbf{pk}_{i-1}) = 1$. If the verification holds, than it parses $\bar{\sigma}_{i-1} = (A', B', C', D')$, where $\mathit{unq}(\bar{\sigma}_{i-1}) = D'$ is the unique component.

$$A' = g^r, \quad B' = \prod_{j=1}^{i} g^{r y_j}, \quad C' = \prod_{j=1}^{i} g^{r(x_j + m_j x_j y_j)}, \quad D' = \prod_{j \neq k}^{i} g^{m_j x_j y_k},$$

and it computes the signature $\bar{\sigma}_i = (A, B, C, D)$:

$$A = g^r, \quad B = B' \cdot A'^{y_i} = \prod_{j=1}^{i} g^{r y_j}, \quad C = C'(A')^{x_i + m_i x_i y_i} = \prod_{j=1}^{i} g^{x_j + m_j x_j y_j},$$

$$D = D' \cdot \left(\prod_{j=1}^{i-1} g^{x_j m_j y_i} g^{y_j x_i m_i} \right) = \prod_{j \neq k}^{i} g^{x_j m_j y_k}$$

$\mathit{AggVerify}(\bar{\sigma}_i, \mathbf{pk}_i, \mathbf{m}_i)$: On input of a sequence of public keys $\mathbf{pk}_i$, sequence of messages $\mathbf{m}_i$ and $\bar{\sigma}_i = (A, B, C, D)$. The verification algorithm first checks if $|\mathbf{m}| = |\mathbf{pk}|$. It then validates the structure of the elements A, B, D:

$$e(A, \prod_{j=1}^{i} Y_j) = e\left(g, \prod_{j=1}^{i} g^{r y_j}\right) \text{ and } \prod_{j \neq k}^{i} e\left(X_k, Y_j\right)^{m_k} = e(g, D)$$

and checks that C is also formed correctly:

$$\prod_{j=1}^{i} \left(e\left(X_j, A\right) \cdot e\left(X_j, B\right)^{m_j}\right) e(A, D)^{-1} = e(g, C).$$

If all equations are valid, then the algorithm outputs 1; otherwise it returns 0.

Theorem 3. *Schröder SAS Scheme is unique according to Definition 7.*

Proof. Let unq be a function that outputs the fourth component of the aggregate signature $\bar{\sigma} = (A, B, C, D)$, i.e. $\mathit{unq}(\bar{\sigma}) = D$. Assume that there exists another aggregate signature $\bar{\bar{\sigma}}$ that passes the verification procedure on the same set of messages and public keys as $\bar{\sigma}$ such that $\mathit{unq}(\bar{\bar{\sigma}}) = \tilde{D}$. We prove by induction on i that in this case $\tilde{D} = D$ must hold. We use $\bar{\bar{\sigma}}$ to check the verification equations.

Base step: $i = 2$**.** The verification algorithm will accept $\bar{\bar{\sigma}}$, if $\tilde{D}$ satisfies the verification equations.

We check first the second equation $\prod_{j \neq k}^{2} e\left(X_j, Y_k\right)^{m_j} = e(g, \tilde{D})$ and compute:

$$
\begin{aligned}
& e\left(X_1, Y_2\right)^{m_1} e\left(X_2, Y_1\right)^{m_2} = e(g, \tilde{D}) \\
& \Leftrightarrow e\left(g^{x_1}, g^{y_2}\right)^{m_1} \cdot e\left(g^{x_2}, g^{y_1}\right)^{m_2} = e(g, \tilde{D}) \\
& \Leftrightarrow e(g, g)^{m_1 x_1 y_2} \cdot e(g, g)^{m_2 x_2 y_1} = e(g, \tilde{D}) \\
& \Leftrightarrow e(g, g)^{m_1 x_1 y_2 + m_2 x_2 y_1} = e(g, \tilde{D}) \\
& \Leftrightarrow e(g, g^{m_1 x_1 y_2 + m_2 x_2 y_1}) = e(g, \tilde{D})
\end{aligned}
$$

The last equation holds only if $\tilde{D} = g^{m_1 x_1 y_2 + m_2 x_2 y_1} = D$.

Induction step: $i - 1 \mapsto i$**.** Let the theorem hold for $i - 1$. The verification algorithm will accept $\mathit{unq}(\bar{\bar{\sigma}}_i) = \tilde{D}_i$ if it satisfies the verification equation $\prod_{j \neq k}^{i} \left(X_j, Y_k\right)^{m_j} = e(g, \tilde{D}_i)$. By the induction hypothesis we have $\tilde{D}_{i-1} = \prod_{j \neq k}^{i-1} g^{m_j x_j y_k}$ such that $\tilde{D}_i = \tilde{D}_{i-1} \cdot \delta$. Considering the following verification equation we get:

$$
\begin{aligned}
& \prod_{j \neq k}^{i} e\left(X_j, Y_k\right)^{m_j} = e(g, \tilde{D}_i) \quad \Leftrightarrow \quad \prod_{j \neq k}^{i} e\left(g^{x_j}, g^{y_k}\right)^{m_j} = e(g, \tilde{D}_i) \\
& \Leftrightarrow \prod_{j \neq k}^{i-1} (g, g)^{m_j x_j y_k} \prod_{j=1}^{i-1} e(g, g)^{m_i x_i y_j + m_j x_j y_i} = e(g, \prod_{j \neq k}^{i-1} g^{m_j x_j y_k} \cdot \delta) \\
& = \prod_{j \neq k}^{i-1} e(g, g)^{m_j x_j y_k} e(g, \delta) \Leftrightarrow \prod_{j=1}^{i-1} e\left(g, g^{m_i x_i y_j + m_j x_j y_i}\right) = e(g, \delta)
\end{aligned}
$$

The last equation holds if $\delta = \prod_{j=1}^{i-1} g^{m_i x_i y_j + m_j x_j y_i}$. We therefore obtain the desired equality $\tilde{D}_i = D_i = \mathit{unq}(\bar{\sigma}_i)$.

□

5 Distributed Verifiable Random Functions

Distributed Verifiable Random Functions (DVRF) were introduced by Dodis [11]. The so-far only DVRF construction in [11] was obtained by first constructing a non-distributed VRF scheme (based on a variant of the well-known Decisional Diffie-Hellman assumption) and then by making it distributed using threshold secret sharing techniques; more precisely by issuing secret shares of the VRF secret key sk to the n servers and then by combining their individual VRF outputs into the DVRF output, whose validity could be checked publicly using the original VRF public key pk. This approach, however, imposed undesirable trust assumptions on the trustworthy generation of secret keys (shares) for the n servers and resulted in a threshold on the number of corrupted servers.

In contrast, our approach for building DVRF is generic, proceeds under much weaker setup assumptions, and requires only one server to remain uncorrupted. As a guideline we adopt the approach by Micali, Rabin, and Vadhan [25] that has been used in a non-distributed VRF case, namely to first focus on a weaker family of functions those outputs are unpredictable but not necessarily pseudorandom, the so-called Verifiable Unpredictable Functions (VUF). We observe that the generic transformation from [25] for converting VUF outputs into VRF outputs — by adding a random string r to the VUF public key pk and then computing VRF outputs as inner products of VUF outputs and r (which takes its roots in [18]) — works just fine for the case where the VUF output has been previously obtained in a distributed way. In a distributed VUF setting the required random string r can be made part of a *shared random string (SRS)* [13], which we consider as the only setup assumption in our DVRF schemes. Note that the SRS model is much weaker than the assumed trustworthy generation of secret keys in [11] and belongs to standard cryptographic assumptions.

Following the above approach we thus need to define the notion of Distributed VUF (DVUF). Our Definition 8 essentially tweaks the original definition of VUF from [25] to the distributed setting.

Definition 8 (Distributed Verifiable Unpredictable Function (DVUF)). *Let $F_{(\cdot)}(\cdot) : \{0,1\}^{a(\lambda)} \rightarrow \{0,1\}^{b(\lambda)}$ denote a family of functions with associated algorithms:*

- $\texttt{Gen}(1^\lambda)$ *is a PPT algorithm that takes as input the security parameter 1^λ and outputs a private/public key pair (sk_i, pk_i) for a server $i \in \{1, \ldots, n\}$. Let $\mathbf{sk} = \{sk_1, \ldots, sk_n\}$ and $\mathbf{pk} = \{pk_1, \ldots, pk_n\}$.*
- $\texttt{Prove}(\mathbf{sk}, \mathbf{pk}, x)$ *is an interactive protocol executed between an user and n servers with common input x chosen by the user and $\mathbf{pk} = (pk_1, \ldots, pk_n)$ such that at the end of the execution the user obtains a VUF value $F(\mathbf{sk}, x) = y$ and the corresponding proof π.*
- $\texttt{Verify}(\mathbf{pk}, x, y, \pi)$ *is a deterministic algorithm that takes as input $\mathbf{pk}, x, y$ and a candidate proof π, and outputs 1 if π is a valid proof for $y = F(\mathbf{sk}, x)$ and 0 otherwise.*

F is a family of Distributed Verifiable Unpredictable Functions (DVUF) *if it satisfies:*

– Uniqueness: *The DVUF value* $y = F(\mathbf{sk}, m)$ *with proof of correctness* π *is unique if there exists no tuple* $(\mathbf{pk}, x, y_1, y_2, \pi)$ *with* $y_1 \neq y_2$ *but* $\mathtt{Verify}(\mathbf{pk}, x, y_1, \pi) = \mathtt{Verify}(\mathbf{pk}, x, y_2, \pi) = 1$.
– Provability: *For all* $(y, \pi) \leftarrow \mathtt{Prove}(\mathbf{sk}, x)$*:* $\mathtt{Verify}(\mathbf{pk}, x, y, \pi) = 1$.
– Residual Unpredictability: *For any PPT algorithm* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *the probability that* $\mathcal{A}$ *succeeds in the following experiment is negligible in the security parameter* 1^λ :
  1. $(sk_i, pk_i) \leftarrow \mathtt{Gen}(1^\lambda)$ *for all* $i \in [n]$.
  2. $[n] \ni c \leftarrow \mathcal{A}_1(\mathbf{pk})$
  3. $(x^*, y^*, \pi^*) \leftarrow \mathcal{A}_2^{\mathtt{OProve}(sk_c, \cdot)}(\mathbf{sk} \setminus \{sk_c\})$.
  4. $\mathcal{A}$ *succeeds if* $x^* \in \{0,1\}^{a(\lambda)}$, $\mathtt{Verify}(\mathbf{pk}, x^*, y^*, \pi^*) = 1$ *and* x^* *was not queried to the* $\mathtt{OProve}(sk_c, \cdot)$ *oracle by* $\mathcal{A}$,

  where

  $\mathtt{OProve}(sk_c, \cdot)$*: The adversarial input to the oracle is a DVUF input* $x \in \{0,1\}^{a(\lambda)}$*. The oracle responds on behalf of server* c *according to the specification of the* $\mathtt{Prove}$ *protocol.*

The following lemma from [25] when applied to the distributed setting shows how to convert DVUF outputs into DVRF outputs. The resulting transformation holds in the shared random string model that provides involved parties with the random string r. Lemma 1 essentially allows us to focus on DVUF constructions in the remaining part of this work.

Lemma 1 (From DVUF to DVRF [25]). *For any DVUF* $(\mathtt{Gen}, \mathtt{Prove}, \mathtt{Verify})$ *with input length* $a(\lambda)$*, output length* $b(\lambda)$*, and security* $s(\lambda)$*, there exists a DVRF in the shared random string model with the following three algorithms:* $(\overline{\mathtt{Gen}}, \overline{\mathtt{Prove}}, \overline{\mathtt{Verify}})$ *with input length* $a'(\lambda) \leq a(\lambda)$*, output length* $b'(\lambda) = 1$*, and security* $s'(\lambda) = s(\lambda)^{1/3}/(poly(\lambda) \cdot 2^{a'(\lambda)})$*:*

– $\overline{\mathtt{Gen}}(1^\lambda, r)$ *where* $r \leftarrow \{0,1\}^{b(\lambda)}$ *is shared random string computes public/private keys* $(sk_i, pk_i) \leftarrow \mathtt{Gen}(1^\lambda)$ *and outputs* $(\overline{\mathbf{sk}}, \overline{\mathbf{pk}}) = (\mathbf{sk}, (\mathbf{pk}, r))$.
– $\overline{\mathtt{Prove}}(\mathbf{sk}, x, r)$ *computes* $(y, \pi) \leftarrow \mathtt{Prove}(\mathbf{sk}, x)$*,* $\overline{y} = \langle y, r \rangle$ *as inner product of* y *and* r*,* $\overline{\pi} := (y, \pi)$ *and outputs* $(\overline{y}, \overline{\pi})$.
– $\overline{\mathtt{Verify}}(\mathbf{pk}, x, \overline{y}, \overline{\pi})$ *outputs* 1 *if* $\mathtt{Verify}(\mathbf{pk}, x, y, \pi) = 1$ *and* $\overline{y} = \langle y, r \rangle$*. Otherwise it outputs* 0.

Proof. Since any DVUF/DVRF family F is also a VUF/VRF family the proof of this lemma is implied by the result from [25, Section 5].

5.1 Generic Construction of DVUF from UAS Schemes

We obtain our first generic DVUF construction from UAS schemes where the aggregation process is public. The major benefit of this construction is that it requires only one communication round between the user and the n servers and is thus as efficient in terms of communication as the approach in [11]. The algorithms of our UAS-based DVUF construction are detailed in the following using the UAS syntax from Definition 2:

$\mathit{Gen}(1^\lambda)$ computes public parameters $I \leftarrow \mathit{ParGen}(1^\lambda)$ of the UAS scheme. Each server S_i, $i \in [n]$ computes its private/public UAS key pair $(sk_i, pk_i) \leftarrow \mathit{KeyGen}(I)$. Let $\mathbf{sk} = (sk_1, ..., sk_n)$ and $\mathbf{pk} = (pk_1, \ldots, pk_n)$.

$\mathit{Prove}(\mathbf{sk}, x)$ *Protocol:* This is a protocol between user U and servers S_i, $i = 1, \ldots, n$ with each server in possession of $sk_i \in \mathbf{sk}$. The common input is x and $\mathbf{pk}$. Each server S_i computes $\sigma_i \leftarrow \mathit{Sign}(sk_i, x)$ and sends it to U. For all $i \in [n]$, U checks whether $\mathit{Verify}(pk_i, x, \sigma_i) = 1$ using the verification algorithm of the UAS scheme. If so U computes $\bar{\sigma} \leftarrow \mathit{Aggregate}(\mathbf{pk}, x, \boldsymbol{\sigma})$ and outputs $(y, \pi) = (\mathit{unq}(\bar{\sigma}), \bar{\sigma})$.

$\mathit{Verify}(\mathbf{pk}, x, y, \pi)$**:** Parse π as $\bar{\sigma}$. If $\mathit{AggVerify}(\mathbf{pk}, x, \bar{\sigma}) = 1$ and $y = \mathtt{unq}(\bar{\sigma})$ then output 1, else output 0.

Theorem 4. *Let UAS be a unique aggregate signature scheme according to Definitions 3 and 4. Then our DVUF construction from UAS fulfills the properties of Definition 8.*

Proof. The uniqueness of UAS scheme implies the uniqueness property of DVUF. Because individual UAS signatures σ_i, which pass the UAS verification procedure Verify from Definition 2 can be aggregated into a signature $\bar{\sigma}$, which satisfies the UAS $\mathit{AggVerify}$ algorithm, we can conclude that for all $(y, \pi) \leftarrow \mathit{Prove}(\mathbf{sk}, x)$ we have $\mathit{Verify}(\mathbf{pk}, m, y, \pi) = 1$, where $y = \mathtt{unq}(\bar{\sigma})$, $\pi = \bar{\sigma}$ and x is a value to be signed. This implies the provability of our DVUF scheme.

In the following we thus focus on the residual unpredictability of our DVUF construction. Assuming an adversary $\mathcal{A}$ which breaks the unpredictability of the DVUF scheme, i.e. outputs a valid tuple (x^*, y^*, π^*) according to the experiment in Definition 8, we construct an adversary $\mathcal{B}$ that simulates the environment of $\mathcal{A}$ and breaks the unforgeability of the underlying UAS scheme by outputting a valid tuple $(\mathbf{m}^*, \mathbf{pk}^*, \sigma^*)$ according to the experiment in Definition 3.

The UAS forger $\mathcal{B}$ is initialized with system parameters I and the challenge public key pk_c. For all $i \in [n]$, $i \neq c$, where c is treated as a random index in $[n]$ it computes $(sk_i, pk_i) \leftarrow \mathit{KeyGen}(I)$ using the key generation algorithm of the UAS scheme and invokes the two-stage DVUF adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. First it invokes $\mathcal{A}_1(\mathbf{pk})$ where $\mathbf{pk}$ is comprised of all generated pk_i and pk_c whereby index c for pk_c in $\mathbf{pk}$ is assigned randomly by $\mathcal{B}$. If the index c output by $\mathcal{A}_1(\mathbf{pk})$ doesn't match that of pk_c the simulation aborts. The probability that the index matches is given by $1/n$. Otherwise, $\mathcal{B}$ invokes $\mathcal{A}_2(\mathbf{sk}')$, where $\mathbf{sk}'$ is comprised of all generated sk_i (i.e. doesn't include sk_c which is unknown to $\mathcal{B}$) and answers the $\mathit{OProve}(sk_c, \cdot)$ oracle queries of $\mathcal{A}_2$ using its own oracle $\mathit{OSign}(sk_c, \cdot)$. That is, $\mathcal{B}$ performs the computation step of the protocol Prove on behalf of server S_c by obtaining individual signatures σ_c on a given DVUF input x from its own signing oracle. At some point, $\mathcal{A}_2$ outputs a tuple (x^*, y^*, π^*) aiming to break the unpredictability property of the DVUF scheme. This tuple is valid if $\mathcal{A}_2$ never queried x^* to its $\mathit{OProve}(sk_c, \cdot)$ oracle and $\mathit{Verify}(\mathbf{pk}, x^*, y^*, \pi^*) = 1$. $\mathcal{B}$ checks the validity of the tuple and if valid outputs $(\mathbf{m}^*, \mathbf{pk}^*, \sigma^*) = (\mathbf{x}^*, \mathbf{pk}, \pi^*)$ where $\mathbf{x}^*$ is a set consisting of n values x^* as its own forgery.

Let $\mathtt{Succ}_{\mathcal{B}}$ denote the probability that $\mathcal{B}$ outputs a valid forgery for the UAS scheme and $\mathtt{Succ}_{\mathcal{A}}$ denote the probability that $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ breaks the DVUF

construction. If the index c assigned by $\mathcal{B}$ matches the one output by $\mathcal{A}_1$ then its simulation for $\mathcal{A}$ is perfect. It is easy to see that in this case the resulting tuple $(\mathbf{x}^*, \mathbf{pk}, \pi^*)$ constitutes a valid forgery for the UAS scheme since $\mathcal{B}$ never queried the message x^* to its $\mathit{OSign}(sk_c, \cdot)$ oracle. Considering that indices match with probability $1/n$ we get $\mathtt{Succ}_{\mathcal{A}} \leq n \cdot \mathtt{Succ}_{\mathcal{B}}$.

□

5.2 Generic Construction of DVUF from USAS Schemes

Our second generic DVUF construction is based on an USAS scheme where the aggregation process is sequential. This implies that the user must approach each server one-by-one until it obtains the resulting DVUF output from the last server in the sequence. The algorithms of our USAS-based DVUF are detailed in the following using the USAS syntax from Definition 5:

$\mathit{Gen}(1^\lambda)$ computes public parameters $I \leftarrow \mathit{ParGen}(1^\lambda)$ of the USAS scheme. Each server S_i, $i \in [n]$ computes its private/public USAS key pair $(sk_i, pk_i) \leftarrow \mathit{KeyGen}(I)$. Let $\mathbf{sk} = (sk_1, ..., sk_n)$ and $\mathbf{pk} = (pk_1, \ldots, pk_n)$.

$\mathit{Prove}(\mathbf{sk}, x)$ *Protocol:* This is a protocol between user U and servers S_i, $i = 1, \ldots, n$ with each server in possession of $sk_i \in \mathbf{sk}$. The common input is x and $\mathbf{pk}$. Each server S_i computes $\bar{\sigma}_i \leftarrow \mathit{AggSign}(sk_i, x, \bar{\sigma}_{i-1}, \mathbf{pk}_{i-1})$ and sends it to U. For all $i \in [n]$, U checks whether $\mathit{AggVerify}(\bar{\sigma}_i, x, \mathbf{pk}_i) = 1$ using the verification algorithm of the USAS scheme. If so U gives as input to server S_{i+1} an aggregate-so-far $\bar{\sigma}_i$ and value x. Finally it outputs $(y, \pi) = (\mathit{unq}(\bar{\sigma}), \bar{\sigma})$.

$\mathit{Verify}(\mathbf{pk}, x, y, \pi)$**:** Parse π as $\bar{\sigma}$. If $\mathit{AggVerify}(\mathbf{pk}, x, \bar{\sigma}) = 1$ and $y = \mathtt{unq}(\bar{\sigma})$ then output 1, else output 0.

Theorem 5. *Let USAS be a unique sequential aggregate signature scheme according to Definitions 6 and 7. Then our DVUF construction from USAS fulfills the properties of Definition 8.*

Proof. The uniqueness of USAS scheme implies the uniqueness property of DVUF. Because each aggregate-so-far signature $\bar{\sigma}_{i-1}$ from USAS scheme, which pass the USAS verification procedure $\mathit{AggVerify}$ from Definition 5 can be aggregated into an aggregate signature $\bar{\sigma}_i$ by adding the signature σ_i on message m signed by signer i, we can conclude that for all $(y, \pi) \leftarrow \mathit{Prove}(\mathbf{sk}, x)$ we have $\mathit{Verify}(\mathbf{pk}, m, y, \pi) = 1$, where $y = \mathtt{unq}(\bar{\sigma}_i)$, $\pi = \bar{\sigma}_i$ and x is a value to be signed. This implies the provability of our DVUF scheme.

Similar to the last construction we thus focus here on the residual unpredictability of our DVUF construction. Assuming an adversary $\mathcal{A}$ which breaks the unpredictability of the DVUF scheme, i.e. outputs a valid tuple (x^*, y^*, π^*) according to the experiment in Definition 8, we construct an adversary $\mathcal{B}$ that simulates the environment of $\mathcal{A}$ and breaks the unforgeability of the underlying USAS scheme by outputting a valid tuple $(\mathbf{m}^*, \mathbf{pk}^*, \sigma^*)$ according to the experiment in Definition 6.

The USAS forger $\mathcal{B}$ is initialized with system parameters I and the challenge public key pk_c. For all $i \in [n]$, $i \neq c$, where c is treated as a random index in $[n]$ it computes $(sk_i, pk_i) \leftarrow \mathit{KeyGen}(I)$ using the key generation algorithm of the USAS scheme and invokes the two-stage DVUF adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. Because the first stage adversary $\mathcal{A}_1(\mathbf{pk})$ with $\mathbf{pk}$ being comprised of all generated pk_i and given pk_c with a randomly assigned index $c \in [n]$ runs in analogue way to the proof of Theorem 4, we skip here its description and proceed with the invocation of $\mathcal{A}_2(\mathbf{sk}')$, where $\mathbf{sk}'$ is comprised of all generated sk_i, i.e. $\mathbf{sk}'$ doesn't include sk_c which is unknown to $\mathcal{B}$. $\mathcal{B}$ answers the $\mathit{OProve}(sk_c, \cdot)$ oracle queries of $\mathcal{A}_2$ on input $(x, \bar{\sigma}_{c-1})$ where x is the provided DVUF input and $\bar{\sigma}_{c-1}$ is the aggregate-so-far signature that is expected by the server S_c during the execution of the Prove protocol as follows. Upon receiving such query from $\mathcal{A}_2$ it queries its own oracle $\mathit{OAggSign}(sk_c, \cdot)$ on input $(x, \bar{\sigma}_{c-1}, \mathbf{x}_{c-1}, \mathbf{pk}_{c-1})$ where $\mathbf{x}_{c-1}$ is a set of $c-1$ messages all of which are equal to x and $\mathbf{pk}_{c-1}$ is comprised of all pk_i, $i = 1, \ldots, c-1$. Recall that the entire set of DVUF public keys $\mathbf{pk}$ is considered as common input to the Prove protocol. In response to its query, $\mathcal{B}$ obtains the aggregate-so-far signature $\bar{\sigma}_c$ that it forwards on to $\mathcal{A}_2$ which is inline with the specification of the Prove protocol. At some point, $\mathcal{A}_2$ outputs a tuple (x^*, y^*, π^*) aiming to break the unpredictability property of the DVUF scheme. This tuple is valid if $\mathcal{A}_2$ never queried x^* to its $\mathit{OProve}(sk_c, \cdot)$ oracle and $\mathit{Verify}(\mathbf{pk}, x^*, y^*, \pi^*) = 1$. $\mathcal{B}$ checks the validity of the tuple and if valid outputs $(\mathbf{m}^*, \mathbf{pk}^*, \bar{\sigma}^*) = (\mathbf{x}^*, \mathbf{pk}, \pi^*)$ where $\mathbf{x}^*$ is a set consisting of n values x^* as its own forgery.

Let $\mathtt{Succ}_{\mathcal{B}}$ denote the probability that $\mathcal{B}$ outputs a valid forgery for the USAS scheme and $\mathtt{Succ}_{\mathcal{A}}$ denote the probability that $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ breaks the DVUF construction. If the index c assigned by $\mathcal{B}$ for pk_c matches the one output by $\mathcal{A}_1$ then its simulation for $\mathcal{A}$ is perfect. It is easy to see that in this case the resulting tuple $(\mathbf{x}^*, \mathbf{pk}, \pi^*)$ constitutes a valid forgery for the USAS scheme since $\mathcal{B}$ never queried the message x^* to its $\mathit{OAggSign}(sk_c, \cdot)$ oracle. Considering that indices match with probability $1/n$ we get $\mathtt{Succ}_{\mathcal{A}} \leq n \cdot \mathtt{Succ}_{\mathcal{B}}$. □

6 Conclusion

We explored the uniqueness property of aggregate signatures and showed that it gives rise to generic DVUF constructions, whose outputs can be made pseudorandom in the shared random string model using the techniques from [25]. This gives us first generic DVRF constructions that do not impose assumptions on trusted generation of secret keys and those outputs remain pseudorandom even in presence of up to $n-1$ corrupted servers. A number of concrete DVRF constructions follows immediately from our proofs of uniqueness for the aggregate signature schemes from [7,22,31].

Acknowledgements

This research was supported by the German Science Foundation (DFG) through the project PRIMAKE (MA 4957).

References

1. M. Abdalla, D. Catalano, and D. Fiore: Verifiable Random Functions from Identity-Based Key Encapsulation. In EUROCRYPT '09, LNCS 5479, pp. 554–571, Springer, 2009.
2. A. Bagherzandi, J. H. Cheon, and S. Jarecki: Multisignatures Secure under the Discrete Logarithm Assumption and a Generalized Forking Lemma. In ACM CCS '08, pp. 449-458, ACM, 2008.
3. M. Bellare and G. Neven: Multi-Signatures in the Plain Public-Key Model and a General Forking Lemma. In ACM CCS '06, pp. 390–399, ACM, 2006.
4. M. Bellare and P. Rogaway: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In ACM CCS '93, pp. 62-73, ACM, 1993.
5. A. Boldyreva: Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap Diffie-Hellman-Group Signature Scheme. In PKC 03, LNCS 2567, pp. 31-46, Springer, 2003.
6. A. Boldyreva, C. Gentry, A. O'Neill, and D. H. Yum: Ordered Multisignatures and Identity-Based Sequential Aggregate Signatures, with Applications to Secure Routing. In ACM CCS '07, pp. 276–285, ACM, 2007.
7. D. Boneh, C. Gentry, B. Lynn, and H. Shacham: Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In EUROCRYPT '03, LNCS 2656, pp. 416-432, Springer, 2003.
8. D. Boneh, B. Lynn, and H. Shacham: Short Signatures from the Weil Pairing. In *Journal of Cryptology* 17:297-319, Springer, 2004.
9. J. Camenisch and A. Lysyanskaya: Signature Schemes and Anonymous Credentials from Bilinear Maps. In CRYPTO '04, LNCS 3152, pp. 56–72, Springer, 2004.
10. R. Canetti, O. Goldreich, and S. Halevi: The Random Oracle Methodology, Revisited. In ACM STOC '98, pp. 209-218, 1998.
11. Y. Dodis: Efficient Construction of (Distributed) Verifiable Random Functions. In PKC '03, LNCS 2567, pp. 1–17, Springer, 2003.
12. Y. Dodis and A. Yampolskiy: A Verifiable Random Function with Short Proofs and Keys. In PKC '05, LNCS 3386, pp. 416–431, Springer, 2005.
13. U. Feige, J. Killian, and M. Naor: A Minimal Model for Secure Computation. In ACM STOC '94, pp. 554–563, ACM, 1994.
14. M. Franklin and H. Zhang: Unique Group Signatures. In ESORICS '12, LNCS 7459, pp. 643–660, Springer, 2012.
15. M. Franklin and H. Zhang: A Framework for Unique Ring Signatures. In IACR Cryptology ePrint Archive, Report 2012/577, 2012.
16. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin: Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In EUROCRYPT '99, LNCS 1592, pp. 295–310, Springer, 1999.
17. O. Goldreich, S. Goldwasser and S. Micali.: How to construct random functions. *Journal of ACM,* 33(4): pp. 792-807, 1986.
18. O. Goldreich and L. A. Levin: A Hard-Core Predicate for All One-Way Functions. In ACM STOC '89, pp. 25–32, ACM, 1989.

19. S. Goldwasser and R. Ostrovsky: Invariant Signatures and Non-Interactive Zero-Knowledge Proofs are Equivalent. In CRYPTO '92, LNCS 740, pp. 228–244, Springer, 1992.
20. S. Jarecki and V. Shmatikov: Handcuffing Big Brother: An Abuse-Resilient Transaction Escrow Scheme. In EUROCRYPT '04, LNCS 3027, pp. 590–608, Springer, 2004.
21. M. Liskov: Updatable Zero-Knowledge Databases. In ASIACRYPT '05, LNCS 3788, pp. 174–198, Springer, 2005.
22. S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters: Sequential Aggregate Signatures and Multisignatures Without Random Oracles. In EUROCRYPT '06, LNCS 4004, pp. 465–485, Springer, 2006.
23. A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham: Sequential Aggregate Signatures from Trapdoor Permutations. In EUROCRYPT'04, LNCS 3027, pp. 74–90, Springer, 2004.
24. A. Lysyanskaya: Unique Signatures and Verifiable Random Functions from the DH-DDH Separation. In CRYPTO '02, LNCS 2442, pp. 597–612, Springer, 2002.
25. S. Micali, M. Rabin, and S. Vadhan: Verifiable Random Functions. In IEEE FOCS '99, pp. 120–130, IEEE Computer Society, 1999.
26. S. Micali and L. Reyzin: Soundness in the Public-Key Model. In CRYPTO '01, LNCS 2139, pp. 542–565, Springer, 2001.
27. S. Micali, K. Ohta, and L. Reyzin: Accountable-subgroup multisignatures: extended abstract. In ACM CCS '01, pp. 245–254, ACM, 2001.
28. S. Micali and R. L. Rivest: Micropayments Revisited. In CT-RSA '02, LNCS 2271, pp. 149–163, Springer, 2002.
29. T. Ristenpart and S. Yilek: The Power of Proofs-of-Possession: Securing Multiparty Signatures against Rogue-Key Attacks. In EUROCRYPT '07, LNCS 4515, pp. 228–245, Springer, 2007.
30. A. Shamir: How to Share a Secret. In *Communications of the ACM*, 22(11):612–613, ACM, 1979.
31. D. Schröder: How to aggregate CL signatures. In ESORICS '11, LNCS 6879, pp. 298-314, Springer, 2011.
32. B. Waters: Efficient Identity-Based Encryption without Random Oracles. In EUROCRYPT '05, LNCS 3494, pp. 114–127, Springer, 2005.
33. Y. Zhou, H. Qian, and X. Li: Non-Interactive CDH-Based Multisignature Scheme in the Plain Public Key Model with Tighter Security. In ISC 2011, LNCS 7001, pp. 341-354, Springer, 2011.