Aggregatable Distributed Key Generation

Kobi Gurkan [⋆], Philipp Jovanovic [⋆⋆], Mary Maller [⋆⋆⋆], Sarah Meiklejohn [†], Gilad Stern [‡], Alin Tomescu [§]

Abstract. In this paper, we introduce a distributed key generation (DKG) protocol with aggregatable and publicly-verifiable transcripts. Compared with prior *publicly-verifiable* approaches, our DKG reduces the size of the final transcript and the time to verify it from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$, where n denotes the number of parties. As compared with prior non-publicly-verifiable approaches, our DKG leverages *gossip* rather than *all-to-all communication* to reduce verification and communication complexity. We also revisit existing DKG security definitions, which are quite strong, and propose new and natural relaxations. As a result, we can prove the security of our aggregatable DKG as well as that of several existing DKGs, including the popular Pedersen variant. We show that, under these new definitions, these existing DKGs can be used to yield secure threshold variants of popular cryptosystems such as El-Gamal encryption and BLS signatures. We also prove that our DKG can be securely combined with a new efficient verifiable unpredictable function (VUF), whose security we prove in the random oracle model. Finally, we experimentally evaluate our DKG and show that the per-party overheads scale linearly and are practical. For 64 parties, it takes 71 ms to share and 359 ms to verify the overall transcript, while for 8192 parties, it takes 8 s and 42.2 s respectively.

1 Introduction

System designers who strive to remove single points of failure often rely on tools provided by threshold cryptography [21, 58] and secure multi-party computation [20, 34]. In this paper, we study *distributed key generation (DKG)* [31, 53], a method from threshold cryptography that often plays an essential role during the setup of distributed systems, including Byzantine consensus [6, 64], time-stamping services [14, 61], public randomness beacons [29, 59], and data archive systems [45, 63]. A DKG enables a set of parties to generate a keypair such that any sufficiently large subset can perform an action that requires the secret key while any smaller subset cannot. To achieve this, a DKG essentially turns each party into a dealer for a verifiable secret sharing (VSS) scheme [19, 25, 54]. This

[⋆] cLabs, Ethereum Foundation. Email: `me@kobi.one`
[⋆⋆] University College London. Email: `p.jovanovic@ucl.ac.uk`
[⋆⋆⋆] Ethereum Foundation. Email: `mary.maller@ethereum.org`
[†] University College London, Google. Email: `s.meiklejohn@ucl.ac.uk`
[‡] Hebrew University. Email: `gilad.stern@mail.huji.ac.il`
[§] VMware Research. Email: `alint@vmware.com`

process yields a single collective public key, generated in a distributed manner, with each party keeping a share of the secret key for themselves.

Current DKGs [27, 31, 32, 53] commonly require that all n parties broadcast $\mathcal{O}(n)$-sized messages that are then used by each party to verify the shares they received from their peers. This results in each party communicating $\mathcal{O}(n^2)$ sized messages via broadcast. While some DKGs have $\mathcal{O}(n)$ communication and verification per party, they rely on constant-sized polynomial commitment schemes that require trusted setup [43, 42]. In this work, we show how to reduce the size of the final DKG transcript to $\mathcal{O}(n)$ by making the parties' contributions *aggregatable*. This enables us to relay (partial) transcripts in an efficient and resilient manner, *e.g.*, over gossip networks, ensuring that transcripts do not grow in size since aggregation can be done in a continuous manner. Aggregatability also enables us to refresh the transcript if and when shares get compromised.

Our DKG transcripts contain the information needed for parties to decrypt their secret shares. During aggregation it is therefore essential to ensure that only valid (partial) transcripts are aggregated. We achieve this by making our transcripts *publicly-verifiable* so that anybody receiving and aggregating transcripts can verify their correctness. Making the transcripts publicly-verifiable has several other advantages: It ensures that all parties can obtain their secret shares, even if they go offline momentarily, and also enables us to remove the "complaint rounds" that are used in previous DKGs to expose misbehaving parties. This improves overall latency, since fewer communication rounds are required, and reduces the protocol's complexity from an implementation perspective.

A consequence of our approach is that the DKG secret key and its shares are group elements rather than field elements. While this prevents us from using it for many well-known cryptosystems, we demonstrate its applicability by introducing a *verifiable unpredictable function (VUF)* [23, 48] whose secret key is a group element, and prove its security in the random oracle model. Threshold VUFs are useful in the construction of verifiable random beacons, which themselves are invaluable in building proof-of-stake-based cryptocurrencies [33, 44]. To the best of our knowledge, our VUF is the first that takes a group element as the secret key, and its performance is also reasonable: our VUF output consists of 6 source group elements and can be verified using 10 pairings. We also provide further optimizations enabling us to reduce the VUF contributions in the threshold protocol to just 2 source group elements that can be verified with 3 pairings.

We also revisit the definitions for DKGs in the hope of reducing complexities and inefficiencies. In particular, previous definitions [31] required *secrecy*, in the sense that the output of the DKG must be indistinguishable from random. While this notion has the benefit of making the DKG modular (one can replace key generation with a DKG in any context), it also is difficult to realise. Indeed, Gennaro et al. [31] demonstrated that the popular Pedersen DKG does not have secrecy, and introduced an alternative and considerably less efficient protocol to achieve secrecy. Additionally, no one-round DKG can achieve secrecy because a rushing adversary (an adversary that plays last) can always influence the final distribution. Instead, we look to prove that a DKG is *robust* (see Definition 6) and *security-preserving* (see Definition 8) in the sense that any adversary that breaks the security of a threshold version of the scheme (i.e., one using a DKG) also breaks the original security property.

Gennaro et al. [32] previously observed that the Pedersen DKG suffices to construct threshold Schnorr signatures [57]. Recently, Benhamouda et al. [10] found an attack on this approach when the adversary is *concurrent*. (Gennaro et al. had not considered concurrent adversaries.) Komlo and Goldberg [47] show that it is possible to avoid the attack but, in doing so, they lose robustness (*e.g.*, if a single party goes offline a signature will not be produced). This raises questions as to whether it is still okay to use the Pedersen DKG with respect to other signature schemes such as BLS. In this paper, we provide a positive answer in the form of a security proof that holds concurrently and does not rely on rewinding the adversary. Specifically, we show that the Pedersen DKG is security-preserving with respect to any *rekeyable* encryption scheme, signature scheme, or VUF scheme where the sharing algorithm is the same as encryption or signing (see Definition 5).

Our contributions. In Section 5, we construct an *aggregatable* and *publicly-verifiable* DKG. The aggregation can be completed by any party (there are no additional secrets) and can also be done incrementally. The cost of verifying our transcripts is $\mathcal{O}(n \log n)$ whereas prior approaches were $\mathcal{O}(n^2)$ [27]. If any user temporarily goes offline, they can still recover their secret shares. Dealing DKG shares takes $\mathcal{O}(n \log n)$ time and aggregation costs are $\mathcal{O}(n)$.

We prove security of our DKG using a natural definition (see Section 3.6), which roughly states that, if it is possible to break a cryptosystem's security game with a DKG swapped in, then it is possible to break that cryptosystem's original security game that did not use the DKG for key generation. We further demonstrate that, counter-intuitively, it is possible to prove that a DKG realises this definition without needing a separate proof for each cryptosystem. Indeed, we show that any encryption scheme, signature scheme, or VUF that are *rekeyable*, and where the sharing algorithm is the same as encryption or signing, can be securely instantiated using a *key-expressable* DKG (see Definition 7). This includes El-Gamal encryption, BLS signatures, a new VUF we introduce in Section 7, and, we suspect, many others.

We further demonstrate the applicability of our techniques by showing that all three of the Pedersen DKG [53], the Fouque-Stern DKG [27], and our aggregatable DKG are key-expressable and thus can be used securely with rekeyable encryption schemes, signature schemes, and VUFs whose decryption/ signing algorithms are the same as the algorithms to generate decryption/ signature shares. Our proof allows for rushing adversaries and holds concurrently (*i.e.*, with respect to an adversary that can open many sessions at the same time). We cannot cover Schnorr signatures, however, because their threshold variants do not appear to be rekeyable.

Our final contribution, in Section 8, is a Rust implementation of our aggregatable DKG to demonstrate its practicality by showing that its overheads are indeed linear. For example, the evaluation of our implementation shows that for

64 / 128 / 8192 nodes it takes 71 ms / 137 ms / 8,000 ms to share one secret and 359 ms / 747 ms / 42,600 ms to verify the corresponding transcript.

2 Related Work

DKG	**Broadcasts**		**P2P**	**PV**	**Complaints**	**Rounds**		**Prover**	**Verifier**	
	$\mathcal{O}(n)$	$\mathcal{O}(1)$				*Broadcast*	*Gossip*		*Local*	*Global*
Pedersen	n	–	n	no	yes	3	–	$n \lg n$	n^2	–
Kate	–	n	n	no	yes	3	–	n^2	n	–
AMT	–	n	$n \lg n$	no	yes	3	–	$n \lg n$	$n \lg n$	–
Fouque-Stern	n	–	n^2	yes	no	1	–	$n \lg n$	n^2	n^2
Our work	$\lg n$	n	$n \lg^2 n$	yes	no	2	$\lg n$	$n \lg n$	$n \lg^2 n$	$n \lg n$

Table 1. Complexities of prior DKG protocols with n parties, per party. In the "Broadcast" column, we count the number of broadcasts by size (either $O(n)$ or $O(1)$-sized). "P2P" means the total size of the messages sent over public and private communication channels (excludes broadcast messages). "PV" means publicly-verifiable. "Verifier local" indicates the per-party time spent verifying their shares from other parties, while "global" indicates the time to verify the final DKG transcript.

We provide an asymptotic overview of the state-of-the-art for DKGs in Table 2. Here we assume that the threshold t is linear in n. Our comparisons consider the optimistic case where there is no more than a constant number of complaints for protocols where these are relavant (recall in our protocol there are no complaints).

Pedersen introduced the first efficient DKG protocol for discrete log-based cryptosystems [54], building on top of Feldman's VSS [25]. Gennaro et al. [32] showed Pedersen's DKG does not generate uniformly distributed secrets, and proposed a protocol that does but at the cost of lower efficiency. They also fix problems with the complaint phase in Pedersen's DKG. Neji et al. [50] gave a more efficient protocol that ensures uniformity in Pedersen's DKG. Kate [42] reduced the broadcast overhead per DKG party from $\mathcal{O}(n)$ to $\mathcal{O}(1)$ using their constant-sized polynomial commitment scheme [43]. However, there scheme depends on a trusted setup algorithm, the costs of which are not considered in Table 2. Trusted setup algorithms have a round complexity of t, and each of these rounds require users to broadcast $\mathcal{O}(n)$ sized messages [15, 36]. Unlike our protocol, all of these protocols rely on complaints rounds, are not publicly verifiable and have $\mathcal{O}(n^2)$ communication complexity.

Fouque and Stern present a one-round, publicly-verifiable DKG that uses only public channels. However, their final transcript size is $\mathcal{O}(n^2)$ whereas ours is $\mathcal{O}(n)$ because we can aggregate. Furthermore, their security proof does not allow for rushing adversaries. While they do not measure performance, their use of Paillier encryption [52] is likely to make their DKG slow and have high communication costs. Nonetheless, unlike our DKG, theirs has the advantage of outputting secrets that are field, rather than group, elements.

Other works tackle the DKG problem from different angles. Canetti et al's DKG [16] has adaptive security, while ours is secure only against static adversaries that fix the set of corrupted parties before the protocol starts. Canny and Sorkin [17] study DKG protocols with poly-logarithmic communication and computation cost per-party, but their protocol relies on a trusted dealer that permutes the parties before the protocol starts. Kate et al. [41, 42] and Kokoris-Kogias et al. [46] study DKG protocols in the *asynchronous* setting, unlike our work and most previous work. Schindler et al. [56] use the Ethereum blockchain to instantiate the synchronous broadcast channel all DKG protocols mentioned so far assume, including ours. Tomescu et al. [60] lower the computational cost of dealing in Kate et al.'s DKG [43], at a logarithmic increase in communication. Lastly, several works implement and benchmark synchronous, statically-secure DKG protocols for discrete log-based cryptosystems [56, 51, 40, 22, 39].

Abe et al. [1] observed that any fully structure preserving signature scheme that depends solely on algebraic operations cannot be used as a VUF or VRF. Unlike our VUF (which is not algebraic), this rules out a number of structure preserving signatures from being candidates for building VUFs [4, 2, 62, 3].

3 Definitions

3.1 Preliminaries

If x is a binary string then $|x|$ denotes its bit length. If S is a finite set then $|S|$ denotes its size and $x \overset{\$}{\leftarrow} S$ denotes sampling a member uniformly from S and assigning it to x. We use $\lambda \in \mathbb{N}$ to denote the security parameter and 1^λ to denote its unary representation. Algorithms are randomized unless explicitly noted otherwise. "PPT" stands for "probabilistic polynomial time." We use $\boldsymbol{y} \leftarrow A(\boldsymbol{x}; r)$ to denote running algorithm A on inputs $\boldsymbol{x}$ and randomness r and assigning its output to $\boldsymbol{y}$. We use $\boldsymbol{y} \overset{\$}{\leftarrow} A(\boldsymbol{x})$ to denote $y \leftarrow A(x; r)$ for uniformly random r. We use $[A(\boldsymbol{x})]$ to denote the set of values that have non-zero probability of being output by A on input $\boldsymbol{x}$. For two functions $f, g : \mathbb{N} \to [0, 1]$, we use $f(\lambda) \approx g(\lambda)$ to denote $|f(\lambda) - g(\lambda)| = \lambda^{-\omega(1)}$. We use code-based games in security definitions and proofs [9]. A game $\mathsf{Sec}_{\mathcal{A}}(\lambda)$, played with respect to a security notion Sec and adversary $\mathcal{A}$, has a MAIN procedure whose output is the output of the game. The notation $\mathsf{Pr}[\mathsf{Sec}_{\mathcal{A}}(\lambda)]$ denotes the probability that this output is 1.

We formalize bilinear groups via a *bilinear group sampler*, which is an efficient *deterministic* algorithm $\mathsf{GroupGen}$ that given a security parameter 1^λ (represented in unary), outputs a tuple $\mathsf{bp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, \hat{h}_1)$ where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are groups with order divisible by the prime $p \in \mathbb{N}$, g_1 generates $\mathbb{G}_1$, $\hat{h}_1$ generates $\mathbb{G}_2$, and $e \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a (non-degenerate) bilinear map.

Galbraith et al. distinguish between three types of bilinear group samplers [28]. Type I groups have $\mathbb{G}_1 = \mathbb{G}_2$ and are known as *symmetric* bilinear groups. Types II and III are *asymmetric* bilinear groups, where $\mathbb{G}_1 \neq \mathbb{G}_2$. Type II groups have an efficiently computable homomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$, while Type III groups do not have an efficiently computable homomorphism in either direction. Certain assumptions are provably false with respect to certain group types (*e.g.*, SXDH only holds for Type III groups), and we work only with Type III groups.

3.2 Communication and threat models

In this section we discuss our communication and threat models.

Synchrony: We assume perfect synchrony. There is a strict time bound between rounds. All messages (honest and adversarial) within a round will be seen by all parties by the end of the round.

Communication channel: We assume the existence of a broadcast channel for sending messages. If a non-faulty party broadcasts a message then it will be seen by everyone by the end of the round. It is not possible to forge messages from non-faulty parties.

Adversarial threshold: We denote by t the adversarial threshold; i.e., the number of parties that the adversary can corrupt. The total number of parties is denoted by n. We set no specific bounds on the adversarial threshold because a rational adversary might prefer to attack the secrecy of the DKG over blocking the communication channels [5, 30].

Assumptions on the adversary: Our security proofs are given with respect to static adversaries, meaning the adversary must state at the start of the security game all of the parties that it has corrupted. We allow the adversary to control the ordering of messages within a round, and in particular the adversary can wait to receive all messages within a round before it broadcasts its message (this is called a *rushing* adversary). The adversary can also choose not to participate at all.

Byzantine adversary A byzantine adversary is a malicious entity that may differ arbitrarily from the protocol.

Crashed party A crashed party is a party that has gone offline e.g. due to a faulty internet connection. After a party has crashed they will not send any more messages.

3.3 Assumptions

Our security proofs are provided in the random oracle model; i.e., there exists a simulator that can program the output of a hash function provided that their chosen outputs are indistinguishable from random.

We rely on the SXDH assumption [7, 8], which is an extension of the DDH assumption to Type III bilinear groups. Informally, it states that given g_1^{α} and g_1^{β} it is hard to distinguish $g_1^{\alpha\beta}$ from random.

The BDH assumption is an extension of the CDH assumption to Type III bilinear groups [12]. Informally, it states that given $g_1^{\alpha}, g_1^{\beta}, \hat{h}_1^{\gamma}, \hat{h}_1^{\alpha\gamma}$ it is hard to compute $e(g_1, \hat{h}_1)^{\alpha\beta}$.

3.4 Verifiable unpredictable functions (VUFs)

A VUF allows a party with a secret key to compute a deterministic (keyed) function and prove to an external verifier that the result is correct. The notion is related to signatures, with the extra requirement that the output of the signer must be unique, even to a party that can choose the secret key. We have made the following changes to prior definitions [23] in order to better suit our setting: (1) we include a global setup algorithm to generate a common reference string; (2) we include a derive algorithm to map the prover's output onto the unique function output.

Definition 1 (Verifiable Unpredictable Function). *Let $\Pi = (\mathsf{VUF.Setup}, \mathsf{VUF.Gen}, \mathsf{VUF.Eval}, \mathsf{VUF.Sign}, \mathsf{VUF.Derive}, \mathsf{VUF.Ver})$ be the following set of efficient algorithms:*

$\mathsf{crs}_{\mathsf{vuf}} \leftarrow \mathsf{VUF.Setup}(1^{\lambda})$: *a DPT algorithm that takes as input the security parameter and outputs a common reference string.*

$(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{VUF.Gen}(\mathsf{crs}_{\mathsf{vuf}})$*: a PPT algorithm that takes as input a common reference string and returns a public key and a secret key.*

$\mathsf{out} \leftarrow \mathsf{VUF.Eval}(\mathsf{crs}_{\mathsf{vuf}}, \mathsf{sk}, m)$*: a DPT algorithm that takes as input a common reference string, secret key, and message $m \in \{0,1\}^{\lambda}$ and returns* $\mathsf{out} \in \{0,1\}^{\lambda}$.

$\sigma \xleftarrow{\$} \mathsf{VUF.Sign}(\mathsf{crs}_{\mathsf{vuf}}, \mathsf{sk}, m)$*: a PPT algorithm that takes as input a common reference string, secret key, and message, and returns a signature σ.*

$\mathsf{out} \leftarrow \mathsf{VUF.Derive}(\mathsf{crs}_{\mathsf{vuf}}, \mathsf{pk}, m, \sigma)$*: a DPT algorithm that takes as input a common reference string, public key, message and signature and returns* $\mathsf{out} \in \{0,1\}^{\lambda}$.

$0/1 \leftarrow \mathsf{VUF.Ver}(\mathsf{crs}_{\mathsf{vuf}}, \mathsf{pk}, m, \sigma)$*: a DPT algorithm that takes as input a common reference string, public key, message and signature and returns* 1 *to indicate acceptance and* 0 *to indicate rejection.*

We say that Π is a verifiable unpredictable function (VUF) *if it satisfies correctness, uniqueness, and unpredictability (defined below).*

A VUF is correct if an honest signer always convinces an honest verifier and always outputs a seed such that the derive function outputs the correct value.

Definition 2 (Correctness). *A VUF is correct if for all $\lambda \in \mathbb{N}$ and $m \in \{0,1\}^\lambda$ we have that*

$$\Pr\left[\begin{array}{l|l} \mathsf{crs_{vuf}} \leftarrow \mathsf{Setup}(1^\lambda), & \mathsf{VUF.Derive}(\mathsf{crs_{vuf}}, \mathsf{pk}, m, \sigma) = \\ (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{VUF.Gen}(\mathsf{crs_{vuf}}), & \mathsf{VUF.Eval}(\mathsf{crs_{vuf}}, \mathsf{sk}, m) \\ \sigma \xleftarrow{\$} \mathsf{VUF.Sign}(\mathsf{crs_{vuf}}, \mathsf{sk}, m) & \wedge\ \mathsf{VUF.Ver}(\mathsf{crs_{vuf}}, \mathsf{pk}, m, \sigma) = 1 \end{array}\right] = 1.$$

A VUF is unique if an adversary (even one that chooses the secret key) cannot output a verifying signature such that the derive function outputs the wrong value.

Definition 3 (Uniqueness). *For a VUF Π and an adversary $\mathcal{A}$, let* $\mathsf{Adv}^{\mathsf{unique}}_{\mathcal{A}}(\lambda) = \Pr[\mathsf{Game}^{\mathsf{unique}}_{\mathcal{A}}(\lambda)]$*, where* $\mathsf{Game}^{\mathsf{unique}}_{\mathcal{A}}(\lambda)$ *is defined as follows:*

```
MAIN $\mathsf{Game}^{\mathsf{unique}}_{\mathcal{A}}(\lambda)$
$\mathsf{crs_{vuf}} \leftarrow \mathsf{VUF.Setup}(1^\lambda)$
$(\mathsf{pk}, m, \sigma_1, \sigma_2) \xleftarrow{\$} \mathcal{A}(\mathsf{crs_{vuf}})$
$y_1 \leftarrow \mathsf{VUF.Derive}(\mathsf{crs_{vuf}}, \mathsf{pk}, m, \sigma_1)$
$y_2 \leftarrow \mathsf{VUF.Derive}(\mathsf{crs_{vuf}}, \mathsf{pk}, m, \sigma_2)$
return $(y_1 \neq y_2)\ \wedge\ \mathsf{VUF.Ver}(\mathsf{crs_{vuf}}, \mathsf{pk}, m, \sigma_1)\ \wedge\ \mathsf{VUF.Ver}(\mathsf{crs_{vuf}}, \mathsf{pk}, m, \sigma_2)$
```

We say that Π is unique if for all PPT adversaries $\mathcal{A}$ we have that $\mathsf{Adv}^{\mathsf{unique}}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$.

Finally, a VUF is unpredictable if an adversary cannot predict the output of the function VUF.Eval on a message for which it has not seen any valid signatures.

Definition 4 (Unpredictability). *For a VUF Π and an adversary $\mathcal{A}$, let* $\mathsf{Adv}^{\mathsf{predict}}_{\mathcal{A}}(\lambda) = \Pr[\mathsf{Game}^{\mathsf{predict}}_{\mathcal{A}}(\lambda)]$ *where* $\mathsf{Game}^{\mathsf{predict}}_{\mathcal{A}}(\lambda)$ *is defined as follows:*

```
MAIN $\mathsf{Game}^{\mathsf{predict}}_{\mathcal{A}}(\lambda)$
$H \leftarrow \emptyset$
$\mathsf{crs_{vuf}} \leftarrow \mathsf{VUF.Setup}(1^\lambda)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{VUF.Gen}(\mathsf{crs_{vuf}})$
$(m, y) \xleftarrow{\$} \mathcal{A}^{\mathsf{VUF.Sign}(\mathsf{crs_{vuf}}, \mathsf{sk}, \cdot)}(\mathsf{crs_{vuf}}, \mathsf{pk})$
return $(\mathsf{VUF.Eval}(\mathsf{crs_{vuf}}, \mathsf{sk}, m) = y)\ \wedge\ (m \notin H)$

ORACLE $\mathcal{O}^{\mathsf{VUF.Sign}(\mathsf{crs_{vuf}}, \mathsf{sk}, m)}$
add $m$ to query set $H$
return $\mathsf{VUF.Sign}(\mathsf{crs_{vuf}}, \mathsf{sk}, m)$
```

We say that Π is unpredictable if for all PPT adversaries $\mathcal{A}$ we have that $\mathsf{Adv}^{\mathsf{predict}}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$.

3.5 Rekeyability

To show that existing cryptographic primitives can be instantiated with our DKG, and other DKGs in the literature, we rely on a property called *rekeyability*. Intuitively, rekeyability says that it is possible to transform an object (e.g., a ciphertext or signature) that was formed using one cryptographic key into an object formed with a related key. As one concrete example, in the BLS signature scheme, in which a signature on a message m is of the form $\sigma = H(m)^{\mathsf{sk}_1}$, it is possible to transform this into a signature under the key $\alpha\mathsf{sk}_1 + \mathsf{sk}_2$ by computing $\sigma^\alpha \cdot H(m)^{\mathsf{sk}_2}$. This means that BLS can be efficiently rekeyed with respect to the secret key. While this notion is related to the idea of re-randomizability [35, 55, 26], we are not aware of any formalizations in the literature and it may be of independent interest.

Definition 5 (Rekeyability). *For a public-key primitive $\Pi = (\mathsf{KeyGen}, \Pi_1, \ldots, \Pi_n)$ and functions $f_{\mathsf{k}}(\alpha, \mathsf{k}_1, \mathsf{k}_2)$ that outputs $\alpha\mathsf{k}_1 \oplus \mathsf{k}_2$ for some binary operator $\oplus$ (typically $+$ or $\times$), we define rekeyability as follows for all $\alpha \in \mathbb{N}$ and $(\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_2, \mathsf{sk}_2) \in [\mathsf{KeyGen}(1^\lambda)]$:*

- *We say that an algorithm Π_i is* rekeyable *with respect to the secret key if there exists an efficient function rekey_i such that*

$$\mathsf{rekey}_i(\alpha, \mathsf{pk}_1, \mathsf{sk}_2, x, \Pi_i(\mathsf{sk}_1, x; r)) = \Pi_i(f_{\mathsf{sk}}(\alpha, \mathsf{sk}_1, \mathsf{sk}_2), x; r)$$

  for all $x \in \mathsf{Domain}(\Pi_i)$ and randomness r. Likewise, we say that an algorithm Π_j is rekeyable *with respect to the public key if there exists an efficient function rekey_j such that*

$$\mathsf{rekey}_j(\alpha, \mathsf{pk}_1, \mathsf{sk}_2, \Pi_j(\mathsf{pk}_1, x; r)) = \Pi_j(f_{\mathsf{pk}}(\alpha, \mathsf{pk}_1, \mathsf{pk}_2), x; r)$$

  for all $x \in \mathsf{Domain}(\Pi_i)$ and randomness r.
- *We say that (Π_i, Π_j) is* rekeyable *with respect to the secret key if (1) Π_i is rekeyable with respect to the secret key and (2)*

$$\Pi_j(\mathsf{pk}_1, y) = \Pi_j(f_{\mathsf{pk}}(\alpha, \mathsf{pk}_1, \mathsf{pk}_2), \mathsf{rekey}_i(\alpha, \mathsf{pk}_1, \mathsf{sk}_2, y)).$$

  Likewise we say that (Π_i, Π_j) is rekeyable *with respect to the public key if (1) Π_i is rekeyable with respect to the public key and (2)*

$$\Pi_j(\mathsf{sk}_1, y) = \Pi_j(f_{\mathsf{sk}}(\alpha, \mathsf{sk}_1, \mathsf{sk}_2), \mathsf{rekey}_i(\alpha, \mathsf{pk}_1, \mathsf{sk}_2, y)).$$

For encryption, we would want that $(\mathsf{Encrypt}, \mathsf{Decrypt})$ is rekeyable with respect to the public key, meaning new key material can be folded into ciphertexts without affecting the ability to decrypt. For signing, we would want that $(\mathsf{Sign}, \mathsf{Verify})$ is rekeyable with respect to the secret key, meaning that if signatures verify then so do their rekeyed counterparts.

3.6 Distributed key generation (DKG)

We define a *distributed key generation* (DKG) as an interactive protocol that is used to generate a keypair $(\mathsf{pk}, \mathsf{sk})$. We define this as $(\mathsf{transcript}, \mathsf{pk}) \xleftarrow{\$} \mathtt{DKG}(I, n)$, where n is the number of participants in the DKG, I is the indices of the adversarial participants (so $|I| \leq t$), pk is the resulting public key, and $\mathsf{transcript}$ is some representation of the messages that have been exchanged.

We additionally consider an algorithm $\mathsf{Reconstruct}$ that, given $\mathsf{transcript}$ and the shares submitted by $t+1$ honest parties, outputs the secret key sk corresponding to pk. With this in place, we can define an omniscient interactive protocol $(\mathsf{transcript}, (\mathsf{pk}, \mathsf{sk}), \mathsf{state}_{\mathcal{A}}) \xleftarrow{\$} \mathtt{OmniDKG}(I, n)$ that is aware of the internal state of each participant and thus can output sk (by running the $\mathsf{Reconstruct}$ algorithm) and $\mathsf{state}_{\mathcal{A}}$; i.e., the internal state of the adversary.

The $\mathsf{Reconstruct}$ algorithm is useful not only in defining this extra interactive protocol, but also in defining a notion of *robustness* for DKGs (initially called correctness by Gennaro et al. [31]). We define this as follows:

Definition 6 (Robustness). *A DKG protocol is* robust *if the following properties hold:*

- *A DKG transcript* dkg *determines a public key* pk *that all honest parties agree on.*
- *There is an efficient algorithm* $\mathsf{Reconstruct}$

$$\mathsf{sk} \leftarrow \mathsf{Reconstruct}(\mathsf{dkg}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\ell) \text{ for } t+1 \leq \ell \leq n$$

that takes as input a set of secret key shares where at least $t+1$ *are from honest parties and verifies them against the public transcript produced by the DKG protocol. It outputs the unique value* sk *such that* $\mathsf{pk} \leftarrow \mathsf{KeyGen}(1^\lambda; \mathsf{sk})$.

Beyond robustness, we also want a DKG to *preserve security* of the underlying primitive for which it is run. Previous related definitions of *secrecy* for DKGs required there to exist a simulator that could fix the output of the DKG; i.e., given an input y, could output $(\mathsf{transcript}, y)$ that the adversary could not distinguish from a real $(\mathsf{transcript}, \mathsf{pk})$ output by the DKG run with t adversarial participants. While general, this definition is strong and required previous constructions to have more rounds or constraints than would otherwise be necessary; *e.g.*, there seem to be significant barriers to satisfying this definition in any DKG where the adversary is allowed to go last, as they the know the entire transcript and can bias the final result.

In defining what it means for a DKG to preserve security, we first weaken this previous definition. Rather than require a simulator given pk_1 to have the DKG output exactly pk_1, we consider that it can instead fix the output public key to have a known relation with its input public key. In particular, a simulator given pk_1 can fix the output of the DKG to be pk, where the simulator knows $(\alpha, \mathsf{pk}_2, \mathsf{sk}_2)$ such that $\mathsf{pk} = f(\alpha, \mathsf{pk}_1, \mathsf{pk}_2)$ for $\alpha \neq 0$ and f as defined in the rekeyability definition (see Definition 5). We call this property *key expressability.*

Definition 7 (Key expressability). *For a simulator* Sim*, define as* $(\mathsf{transcript}, \mathsf{pk}, \alpha, \mathsf{pk}_2, \mathsf{sk}_2) \xleftarrow{\$} \mathtt{SimDKG}(\mathsf{Sim}, I, n)$ *a run of the DKG protocol in which all honest participants are controlled by* Sim*, which takes as input a public key* pk_1 *and has private outputs* α*,* pk_2*, and* sk_2*. We say that a DKG is* key-expressable *if there exists such a simulator* Sim *such that (1)* $(\mathsf{transcript}, \mathsf{pk})$ *is distributed identically to the output of* $\mathtt{DKG}(I, n)$*, (2)* $(\mathsf{pk}_2, \mathsf{sk}_2)$ *is a valid keypair, and (3)* $\mathsf{pk} = f(\alpha, \mathsf{pk}_1, \mathsf{pk}_2) = \alpha\mathsf{pk}_1 \oplus \mathsf{pk}_2$.

To now define a security-preserving DKG, we intuitively consider a DKG being run in the context of a security game. To keep our definition as general as possible, our only requirements are that (1) the security game contains a line of the form $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$ (it also works if KeyGen takes a common reference string as additional input), and (2) pk is then later given as input to the adversary. We then say that the DKG preserves security if it is not possible for an adversary participating in the DKG to do better than it would have done in the original security game, in which it was given pk directly. Formally, we have the following definition.

Definition 8 (Security-preserving). *Define* Game *as any security game containing the line* $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$*, denoted* $\mathsf{line}_{\mathsf{pk}}$*, and where* pk *is later input to an adversary* $\mathcal{A}$ *(in addition to other possible inputs). Define* $\mathsf{Game}'(\mathsf{line}, x)$*, parameterized by a starting line* line *and some value* x*, as* Game *but with* $\mathsf{line}_{\mathsf{pk}}$ *replaced by* line *and* $\mathcal{A}$ *given* x *as input rather than* pk*. It is clear that* $\mathsf{Game} = \mathsf{Game}'(\mathsf{line}_{\mathsf{pk}}, \mathsf{pk})$.

Define $\mathsf{line}_{\mathsf{dkg}}$ *as the line* $(\mathsf{transcript}, (\mathsf{pk}, \mathsf{sk}), \mathsf{state}_{\mathcal{A}}) \xleftarrow{\$} \mathtt{OmniDKG}(I, n)$*, and define* $\mathsf{DKG\text{-}Game} \leftarrow \mathsf{Game}'(\mathsf{line}_{\mathsf{dkg}}, \mathsf{state}_{\mathcal{A}})$*. We say the DKG* preserves security *for* Game *if*

$$\mathsf{Adv}^{\mathsf{DKG\text{-}Game}}_{\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{Game}}_{\mathcal{A}}(\lambda) + \mathsf{negl}(\lambda)$$

for all PPT adversaries $\mathcal{A}$*.*

We do not view our requirements for the original security game as restrictive, given the number of security games that satisfy them. For signature unforgeability, for example, our definition says that an adversary that participates in the DKG, and can carry its state from that into the rest of the game (including all of the messages it saw), cannot achieve better advantage than when it is just given the public key (as in the standard EUF-CMA game).

While the relationship between key expressability and security-preserving DKGs is not obvious, we show in the full version of our paper [38] that it is typically the case that when key-expressable DKGs are used for rekeyable primitives, they preserve the security of that primitive's underlying security game.

4 Our Enhanced Scrape PVSS

A *secret sharing scheme* allows a *dealer* to deal out n secret *shares* so that any subset of $t + 1$ shares suffices to reconstruct the secret, but subsets of size $\leq t$ shares do not. A *publicly verifiable secret sharing (PVSS)* scheme is a secret sharing scheme in which any third party can verify that the dealer has behaved honestly. Importantly, PVSS obviates the need for a complaint round in VSS protocols, which simplifies designing PVSS-based DKG protocols [27]. Cascudo and David designed an elegant PVSS scheme called *Scrape* [18] with $O(n)$ verification costs. In this section, we describe a slightly-modified variant of Scrape that supports *aggregation* and uses Type III pairings and an additional element

$\hat{u}_2 \in \mathbb{G}_2$ that will help our DKG security proofs later on. We rely on Type III pairings, not only for efficiency, but also because the SXDH assumption does not hold in symmetric groups. We give a formal description in Fig. 1. In Section 5, we use this slightly-modified variant of Scrape to construct our DKG.

Common reference string (CRS). All parties use the same CRS consisting of (1) a bilinear group description bp (which fixes $g_1 \in \mathbb{G}_1$ and $\hat{h}_1 \in \mathbb{G}_2$), (2) a group element $\hat{u}_1 \in \mathbb{G}_2$ and (3) *encryption keys* $\mathsf{ek}_i \in \mathbb{G}_2$ for every party P_i with corresponding *decryption keys* $\mathsf{dk}_i \in \mathbb{F}$ known only to P_i such that $\mathsf{ek}_i = \hat{h}_1^{\mathsf{dk}_i}$.

Dealing. Scrape resembles other Shamir-based [58] secret sharing schemes. The Scrape dealer will share a secret $\hat{h}_1^{a_0} \in \mathbb{G}_2$, whose corresponding $a_0 \in \mathbb{F}$ the dealer knows. (This is different than other VSS schemes, which typically share a secret in $\mathbb{F}$ rather than in $\mathbb{G}_2$.) The dealer picks a random, degree-t polynomial $f(X) = (a_0, a_1, \ldots, a_t)$, where $f(0) = a_0$, and commits to it via Feldman [25] as $F_i = g_1^{a_i}, \forall i \in [0, t]$. Party P_i's share will be $\hat{h}_1^{f(\omega_i)}$. The dealer then computes Feldman commitments $A_i = g_1^{f(\omega_i)}$ and encryptions $\mathsf{ek}_i^{f(\omega_i)}$ of each share. (The term "encryption" here is slightly abused since these are not IND-CPA-secure ciphertexts.) The *PVSS transcript* will consist of the Feldman commitments to $f(X)$ and to the shares, plus the encryptions of the shares. Additionally, we augment the transcript with $\hat{u}_2 = \hat{u}_1^{a_0}$, which helps our DKG security proofs.

Verifying. Each party P_i can verify that the PVSS transcript is a correct sharing of $\hat{h}_1^{a_0}$. For this, P_i checks the Feldman commitments A_i to the shares $f(\omega_i)$ are consistent with the Feldman commitment to $f(X)$ via Lagrange interpolation in the exponent (see Fig. 1). Then, each P_i checks their encryption of $f(\omega_i)$ against A_i. Altogether, this guarantees that the encrypted shares are indeed the evaluations of the committed polynomial f.

Aggregating transcripts. One of our key contributions is an algorithm for aggregating two Scrape PVSS transcripts pvss_1 and pvss_2 for polynomials f_1 and f_2 into a single transcript for their sum $f_1 + f_2$. This is a key ingredient of our DKG from Section 5. Our aggregation leverages the homomorphism of Feldman commitments and of the encryption scheme. Indeed, suppose we have Feldman commitments to f_b consisting of $F_{b,i} = g_1^{a_{b,i}}, \forall i \in [0, t]$, where $a_{b,i}$'s are the coefficients of f_b, for $b \in \{1, 2\}$. Then, $F_i = F_{1,i} F_{2,i} = g^{a_{1,i} + a_{2,i}}, \forall i \in [0, t]$ will be a Feldman commitment to $f_1 + f_2$. Similarly, we can aggregate the share commitments $A_{b,i} = g^{f_b(\omega_i)}$ as $A_i = A_{1,i} A_{2,i} = g^{(f_1+f_2)(\omega_i)}, \forall i \in [n]$. Lastly, the encryptions $\mathsf{ek}_i^{f_b(\omega_i)}$ can be aggregated as $\mathsf{ek}_i^{(f_1+f_2)(\omega_i)} = \mathsf{ek}_i^{f_1(\omega_i)} \mathsf{ek}_i^{f_2(\omega_i)}$. We summarize this aggregation algorithm in Fig. 1.

Reconstructing the secret. At the end of the PVSS protocol, each party P_i can decrypt their share as $\hat{A}_i = \hat{Y}_i^{\mathsf{dk}_i^{-1}} = (\mathsf{ek}_i^{f(\omega_i)})^{\mathsf{dk}_i^{-1}} = \hat{h}_1^{f(\omega_i)}$. Recall that the degree t polynomial $f(X)$ encodes the secret $f(0) = a_0$. Thus, any set S of $\geq t + 1$ honest parties can reconstruct $\mathsf{sk} = \hat{h}_1^{f(0)}$ as follows:

```
$\underline{\mathsf{Scrape.Deal}(\mathsf{bp}, \mathbf{ek}, \hat{u}_1, a_0) \to \mathsf{pvss}}$
$(a_1, \ldots, a_t) \xleftarrow{\$} \mathbb{F}^t,\ f(X) \leftarrow \sum_{i=0}^{t} a_i X^i$
$F_0, \ldots, F_t \leftarrow g_1^{a_0}, \ldots, g_1^{a_t}$
$\hat{u}_2 \leftarrow \hat{u}_1^{a_0}$
$A_1, \ldots, A_n \leftarrow g_1^{f(\omega_1)}, \ldots, g_1^{f(\omega_n)}$
$\hat{Y}_1 \ldots, \hat{Y}_n \leftarrow \mathsf{ek}_1^{f(\omega_1)}, \ldots, \mathsf{ek}_n^{f(\omega_n)}$
return $\boldsymbol{F}, \hat{u}_2, \boldsymbol{A}, \hat{\boldsymbol{Y}}$

$\underline{\mathsf{Scrape.Verify}(\mathsf{bp}, \mathbf{ek}, \hat{u}_1, \hat{u}_2, \mathsf{pvss}) \to 0/1}$
$\boldsymbol{F}, \hat{u}_2, \boldsymbol{A}, \hat{\boldsymbol{Y}} \leftarrow \mathsf{parse}(\mathsf{pvss})$
$\alpha \xleftarrow{\$} \mathbb{F}$
check $\prod_{j=1}^{n} A_j^{\ell_j(\alpha)} = \prod_{j=0}^{t} F_j^{\alpha^j}$
check $e(F_0, \hat{u}_1) = e(g_1, \hat{u}_2)$
check $e(g_1, \hat{Y}_j) = e(A_j, \mathsf{ek}_j)$ for $1 \leq j \leq n$
return 1 if all checks pass, else return 0

$\underline{\mathsf{Scrape.Aggregate}(\mathsf{bp}, \mathsf{pvss}_1, \mathsf{pvss}_2) \to \mathsf{pvss}}$
$((F_{1,0}, \ldots, F_{1,t}), \hat{u}_{1,2}, (A_{1,1}, \ldots, A_{1,n}), (\hat{Y}_{1,1}, \ldots, \hat{Y}_{1,n})) \leftarrow \mathsf{parse}(\mathsf{pvss}_1)$
$((F_{2,0}, \ldots, F_{2,t}), \hat{u}_{2,2}, (A_{2,1}, \ldots, A_{2,n}), (\hat{Y}_{2,1}, \ldots, \hat{Y}_{2,n})) \leftarrow \mathsf{parse}(\mathsf{pvss}_2)$
for $0 \leq i \leq t$:
    $F_i \leftarrow F_{1,i} F_{2,i}$
for $1 \leq i \leq n$:
    $A_i \leftarrow A_{1,i} A_{2,i},\ \hat{Y}_i \leftarrow \hat{Y}_{1,i} \hat{Y}_{2,i}$
$\hat{u}_2 \leftarrow \hat{u}_{1,2} \hat{u}_{2,2}$
return $\boldsymbol{F}, \hat{u}_2, \boldsymbol{A}, \hat{\boldsymbol{Y}}$
```

Fig. 1. Dealing, verification and aggregation algorithms for the Scrape PVSS. Here, $\mathbf{ek}, \boldsymbol{F}, \boldsymbol{A}, \hat{\boldsymbol{Y}}$ denote vectors of ek_i's, F_i's, A_i's and $\hat{Y}_i$'s. The polynomial $\ell_j(X)$ denotes the Lagrange polynomial equal to 1 at ω_j and 0 at $\omega_i \neq \omega_j$. The ω_i's are public predetermined values which, for efficiency purposes, should be chosen as roots of unity of degree n. For more details, see the full version of our paper [38].

1. For each share $\hat{A}_i$ provided, check that $e(A_i, \hat{h}_1) = e(g_1, \hat{A}_i)$, where $A_i = g_1^{f(\omega_i)}$ is part of the PVSS transcript. If this check fails, or if P_i does not provide a share, then remove P_i from S.
2. Return, $\mathsf{sk} = \prod_{i \in S} \hat{A}_i^{\ell_{S,i}(0)}$ where $\ell_{S,i}(X)$ is a Lagrange polynomial equal to 0 at $\omega_j \in S$ for $i \neq j$, and 1 at ω_i.

5 Distributed Key Generation

In this section, we describe our distributed key generation (DKG) protocol for generating a key-pair $(\mathsf{pk}, \mathsf{sk})$ of the form

$$\mathsf{pk} = (g_1^a, \hat{u}_1^a) \in \mathbb{G}_1 \times \mathbb{G}_2 \quad \text{and} \quad \mathsf{sk} = \hat{h}_1^a \in \mathbb{G}_2, \text{ where } a \in \mathbb{F}$$

We often refer to $a \in \mathbb{F}$ as the *DKG secret*. All parties P_i use the same Scrape CRS (see Section 4) but augmented with *verification keys* vk_i (defined later).

At a high level, our DKG protocol resembles previous protocols based on verifiable secret sharing: each party P_i deals a secret $\hat{h}_1^{c_i}$ to all other parties using the Scrape PVSS from Section 4. Additionally, each party P_i includes a proof-of-knowledge of their secret c_i. At this point, each party P_j would have to verify the PVSS transcript of every other party P_i, resulting in $O(n^2)$ work. Then, the final secret would be $\mathsf{sk} = \hat{h}_1^a$ with $a = \sum_{i \in Q} c_i$, where Q is the set of all parties who dealt honestly (i.e., whose PVSS transcript verified). Note that since PVSS transcripts are publicly-verifiable, all parties P_i agree on Q and there is no need for a complaint round. We often refer to an honest party P_i as having *contributed* to the final secret key and to c_i as its *contribution*.

Gossip and aggregate. To avoid the $O(n^2)$ verification work per party, we leverage aggregation of Scrape PVSS transcripts. We observe that a party who verified several transcripts can aggregate them into a single one and forward it to another party, who can now verify this aggregated transcript faster. By carefully aggregating and *gossiping* transcripts in this manner, we decrease verification time per party from $O(n^2)$ to $O(n \log^2 n)$. One caveat is that, due to the randomized nature of gossiping, a party's contribution c_i might be incorporated multiple times, say w_i times, into the final secret $\mathsf{sk} = \hat{h}_1^a$. As a result, the final $a = \sum_{i \in Q} w_i c_i$, where w_i is called the *weight* of each c_i.

Signatures-of-knowledge of contributions. Similar to previous DKGs [32], our DKG requires each party P_i to prove knowledge of its contribution c_i to the final DKG secret. However, since our DKG transcripts must be publicly-verifiable, we also require each party to sign their contributions. We achieve both of these goals using a *signature-of-knowledge (SoK)*. Specifically, P_i signs $C_i = g_1^{c_i}$ using its secret key sk_i, with corresponding *verification key* $\mathsf{vk}_i = g_1^{\mathsf{sk}_i}$:

$$\sigma_i = (\sigma_{i,1}, \sigma_{i,2}) = (\mathsf{Hash}_{\mathbb{G}_2}(C_i)^{c_i}, \mathsf{Hash}_{\mathbb{G}_2}(\mathsf{vk}_i, C_i)^{\mathsf{sk}_i})$$

where $\mathsf{Hash}_{\mathbb{G}_2}$ is a hash function that maps to $\mathbb{G}_2$. Any verifier with vk_i can verify the signature-of-knowledge σ_i of c_i as:

$$e(C_i, \mathsf{Hash}_{\mathbb{G}_2}(C_i)) = e(g_1, \sigma_{i,1}) \ \wedge \ e(\mathsf{vk}_i, \mathsf{Hash}_{\mathbb{G}_2}(\mathsf{vk}_i, C_i)) = e(g_1, \sigma_{i,2})$$

Our signatures of knowledge are simulation-sound and thus cannot be compressed or combined. However, since they are constant-sized, this is not problematic. We refer to the signing algorithm as $\mathsf{SoK.Sign}(C_i, \mathsf{sk}_i, c_i) \to \sigma_i$ and the verification algorithm as $\mathsf{SoK.Verify}(\mathsf{vk}_i, C_i, \sigma_i) \to 0/1$.

DKG transcripts. To maintain their public-verifiability, aggregated PVSS transcripts must keep track of the weights w_i of each party's contribution c_i and of the σ_i's. This gives rise to a new notion of a *DKG transcript* defined as:

$$\mathsf{transcript} = ((C_1, \ldots, C_n), (w_1, \ldots, w_n), (\sigma_1, \ldots, \sigma_n), \mathsf{pvss}), \tag{1}$$

where $C_i = g_1^{c_i}$ is a commitment to the contribution c_i of party P_i, w_i is its weight, σ_i is the SoK of c_i and pvss is an (aggregated) PVSS transcript for secret $a = \sum_{i \in [n]} w_i c_i$.

Recall from Fig. 1 that pvss stores a Feldman commitment $\boldsymbol{F}$ to a polynomial $f(X)$ with $f(0) = a$ and that $F_0 = g_1^a$. In our protocol, each party

P_i initializes their DKG transcript by picking $c_i \xleftarrow{\$} \mathbb{F}$ and setting $\mathsf{pvss} \leftarrow \mathsf{Scrape.Deal}(\mathsf{bp}, \mathbf{ek}, \hat{u}_1, c_i)$, $C_i \leftarrow g_1^{c_i}$, $w_i \leftarrow 1$ and $\sigma_i \leftarrow \mathsf{SoK.Sign}(C_i, \mathsf{sk}_i, c_i)$. For $j \neq i$, P_i sets $C_j \leftarrow \perp$, $w_j \leftarrow 0$ and $\sigma_j \leftarrow \perp$. Importantly, in our protocol, each party will broadcast the C_i commitment to their contribution and gossip the rest of their DKG transcript to a subset of the other parties (we discuss this in more detail later on).

Verifying DKG transcripts. To verify the DKG $\mathsf{transcript}$ from Eq. (1), one first checks that its inner pvss transcript verifies. Second, for all *non-trivial contributions* with $w_i \neq 0$, one first checks if their signature of knowledge σ_i verifies. Finally, one checks that the contributions correctly combine to the commitment F_0 to the zero coefficient of $f(X)$ shared in pvss; i.e., that $C_1^{w_1} \cdots C_n^{w_n} = F_0$. If $\mathsf{transcript}$ passes these checks, then one can be sure that the players P_i which have $w_i \neq 0$ in $\mathsf{transcript}$ have contributed to its corresponding DKG secret. See Fig. 2 for a full description.

Aggregating DKG transcripts. Given two input DKG transcripts

$$(C_{b,1}, \ldots, C_{b,n}), (w_{b,1}, \ldots, w_{b,n}), (\sigma_{b,1}, \ldots, \sigma_{b,n}), \mathsf{pvss}_b, \text{ for } b \in \{1, 2\}$$

we can easily aggregate them into a single DKG transcript

$$(C_1, \ldots, C_n), (w_1, \ldots, w_n), (\sigma_1, \ldots, \sigma_n), \mathsf{pvss}$$

We first aggregate the pvss_b transcripts into pvss via $\mathsf{Scrape.Aggregate}$ (see Fig. 1). Second, we aggregate the weights, which are field elements, as $w_i = w_{1,i} + w_{2,i}, \forall i \in [n]$. Third, if P_i contributed in one of the input transcripts, then P_i's contribution should also be reflected in the aggregated transcript. In other words, for any $C_{b,i} \neq \perp$ and valid $\sigma_{b,i}$, we simply set $C_i = C_{b,i}$ and $\sigma_i = \sigma_{b,i}$. The choice of $C_{b,i}$ does not matter when they are both $\neq \perp$ since they were both obtained from the broadcast channel, so they must be equal. As a result, their corresponding $\sigma_{b,i}$'s will also be equal since our signatures of knowledge are unique.

Reconstructing the secret. As explained in the beginning of this section, the final key-pair will be $\mathsf{pk} = (g_1^{f(0)}, \hat{u}_2) = (g_1^{f(0)}, \hat{u}_1^{f(0)})$ and $\mathsf{sk} = \hat{h}_1^{f(0)}$. Since the final DKG transcript is just an augmented Scrape PVSS transcript, reconstruction of sk works as explained in Section 4.

5.1 A gossip protocol

In Step 4 of our DKG, we rely on a gossip protocol to communicate the $\mathcal{O}(n)$-sized DKG transcripts. By using gossip, we avoid both the need to broadcast these larger messages, which is expensive, and the need for a central aggregator. We detail our protocol in the full version of our paper [38], but provide some insight here into how it works.

We take an optimistic approach and provide robustness for up to $t_r < n/2 - \log n$ crashed parties but only up to $\log n$ Byzantine adversaries. We believe this

```
$\underline{\mathsf{DKG.Aggregate}(\mathsf{bp}, \mathsf{transcript}_1, \mathsf{transcript}_2) \to \mathsf{transcript}}$
$((C_{1,1},\ldots,C_{1,n}),(w_{1,1},\ldots,w_{1,n}),(\sigma_{1,1},\ldots,\sigma_{1,n}),\mathsf{pvss}_1) \leftarrow \mathsf{parse}(\mathsf{transcript}_1)$
$((C_{2,1},\ldots,C_{2,n}),(w_{2,1},\ldots,w_{2,n}),(\sigma_{2,1},\ldots,\sigma_{2,n}),\mathsf{pvss}_2) \leftarrow \mathsf{parse}(\mathsf{transcript}_2)$

for $1 \leq i \leq n$:
    $w_i \leftarrow w_{1,i} + w_{2,i}$
    if $\sigma_{1,i} \neq \bot$: $\sigma_i \leftarrow \sigma_{1,i}$, else: $\sigma_i \leftarrow \sigma_{2,i}$
    if $C_{1,i} \neq \bot$: $C_i \leftarrow C_{1,i}$, else: $C_i \leftarrow C_{2,i}$

$\mathsf{pvss} \leftarrow \mathsf{Scrape.Aggregate}(\mathsf{bp}, \mathsf{pvss}_1, \mathsf{pvss}_2)$
return $(C_1,\ldots,C_n),(w_1,\ldots,w_n),(\sigma_1,\ldots,\sigma_n),\mathsf{pvss}$

$\underline{\mathsf{DKG.Verify}(\mathsf{bp}, (\mathsf{ek}_i, \mathsf{vk}_i)_{i\in[n]}, \hat{u}_1, \mathsf{transcript}) \to 0/1}$
$((C_1,\ldots,C_n),(w_1,\ldots,w_n),(\sigma_1,\ldots,\sigma_n),\mathsf{pvss}) \leftarrow \mathsf{parse}(\mathsf{transcript})$
$((F_0,\ldots,F_t),\hat{u}_2,(A_1,\ldots,A_n),(\hat{Y}_1,\ldots,\hat{Y}_n)) \leftarrow \mathsf{parse}(\mathsf{pvss})$
check $\mathsf{Scrape.Verify}(\mathsf{bp},(\mathsf{ek}_1,\ldots,\mathsf{ek}_n),\hat{u}_1,\hat{u}_2,\mathsf{pvss}) = 1$

for $1 \leq i \leq n$:
    if $w_i \neq 0$: check $\mathsf{SoK.Verify}(\mathsf{vk}_i, C_i, \sigma_i) = 1$

check $C_1^{w_1} \cdots C_n^{w_n} = F_0$
return 1 if all checks pass, else return 0
```

Fig. 2. Aggregation algorithm for the distributed key generation protocol.

approach is often reasonable in practice because if a Byzantine adversary attacks the robustness of a DKG, the only outcome is that the computation required to output the DKG is higher. Furthermore, Byzantine attacks on robustness are detectable, so any faulty party can be manually removed from the system. This is in contrast to an attack on the security preservation of the DKG, which could have far more serious consequences. If we want a security threshold of t_s, then we have to assume that t_s parties respond. A direct implication is that $n - t_r$ must be at least t_s, showing an inherent tradeoff between the security and robustness thresholds. In our scheme we can set t_s to be exactly equal to $n - t_r$.

The gossip protocol has each party send its currently aggregated DKG transcript to $\mathcal{O}(c \log n)$ parties in expectation in each round, and terminate when it has agreed on a *"full" transcript*; i.e., a valid transcript with at least $t_s + 1$ contributions. Here c is a small success parameter such that $c \geq 4$. However, deciding when to terminate is non-trivial, because the aggregated "full" transcripts may all be different. We thus still rely on broadcast to agree on which transcript to use, but our goal is to minimize the number of total broadcasts. We do this by having each party with a full transcript broadcast it with probability $2/n$ in a given round. We argue that this makes the protocol likely to terminate within $\mathcal{O}(c \log n)$ rounds. Parties agree to use the transcript whose public key has a binary representation with the smallest bit-count (but any other publicly-verifiable convention works too). In terms of complexity, our gossip protocol

Our aggregatable DKG protocol

Common reference string: Scrape CRS consiting of $\mathsf{bp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, \hat{h}_1)$, encryption and verficiation keys $(\mathsf{ek}_i, \mathsf{vk}_i)_{i \in [n]}$, nth roots of unity $(\omega_i)_{i \in [n]}$ in $\mathbb{F}$, random $\hat{u}_1 \in \mathbb{G}_2$ such that nobody knows $\log_{\hat{h}_1}(\hat{u}_1)$.

Party P_i's private input: Decryption key dk_i for ek_i and secret key sk_i for vk_i.

1. Each P_i picks random $c_i \in \mathbb{F}$, computes $C_i = g_1^{c_i}$ and broadcasts C_i.
2. Each P_i picks random polynomial $f_i(X) \in \mathbb{F}[X]$ of degree at most t

$$f_i(X) = a_{i,0} + a_{i,1}X + \cdots + a_{i,t}X^t$$

such that $a_{i,0} = c_i$. They compute $f_i(\omega_j)$ for $j \in [n]$. Each party *gossips* (see Section 5.1) their DKG transcript consisting of (1) $F_{i,k} = g_1^{a_{i,k}}$ for $k \in [0, t]$; (2) $\hat{u}_{i,2} = \hat{u}_1^{c_i}$; (3) a vector $\boldsymbol{w}_i$ such that $w_{i,j} = 1$ if $i = j$ and 0 otherwise; (4) $A_{i,j} = g_1^{f_i(\omega_j)}$ for $j \in [n]$; (5) $Y_{i,j} = \mathsf{ek}_j^{f_i(\omega_j)}$ for $j \in [n]$; and (6) a vector $\boldsymbol{\sigma}_i$ such that $\sigma_{i,j} = (\mathsf{Hash}_{\mathbb{G}_2}(C_i)^{c_i}, \mathsf{Hash}_{\mathbb{G}_2}(\mathsf{vk}_i, C_i)^{\mathsf{sk}_i})$ if $i = j$ and $\perp$ otherwise.
3. During the gossip phase, each P_i verifies the transcripts it receives using DKG.Verify (see Fig. 2). If two transcripts verify, it aggregates them using DKG.Aggregate (also in Fig. 2), and gossips the aggregated transcript. The aggregated transcripts contain a list of weights $(w_1, \ldots, w_n) \in \mathbb{F}^n$ indicating how many times each party has contributed to the current transcript. When a party receives a *"full" transcript* with $\geq t+1$ non-zero weights, it broadcasts this as a candidate final transcript.
4. Parties terminate in the round where they first broadcast a "full" transcript. If several candidate "full" transcripts were broadcast, the one whose pk has the lowest bit count is chosen as the final one. The final $\mathsf{pk} = (\prod_{i=1}^n C_i^{w_i}, \prod_{i=1}^n \hat{u}_{i,2}^{w_i})$. Each party computes their secret key share as $Y_i^{\mathsf{dk}_i^{-1}}$ such that they can reconstruct.

Fig. 3. Our DKG with reconstruction threshold $t+1$ run by parties $P_1, \ldots, P_n$.

requires $\mathcal{O}(cn^2 \log n)$ total words to be communicated in private messages and $\mathcal{O}(c \log^2 n)$ broadcasts.

5.2 Security analysis

Robustness Our DKG is robust in the sense that all honest parties agree on the final public key, and in the sense that any set S containing at least $t+1$ honest parties can reconstruct the secret key.

Theorem 1 (DKG is robust). *The scheme in Figure 3 is robust for any primitive with keys of the form* $\mathsf{pk} = (g_1^a, \hat{u}_1^a) \in \mathbb{G}_1 \times \mathbb{G}_2$.

Proof. First we show that all honest parties have the same value pk. By perfect synchrony we have that in each round all honest parties agree on a completing set of broadcasts. From the broadcast messages that complete and verify, one must have the most sparse binary decomposition. This message defines a public key pk that all parties agree on.

We show that reconstruction always succeeds on input of n shares where at least $t+1$ are input by non-faulty parties. First observe that if the DKG transcript verifies, then for some random value α we have that

$$A_1^{\ell_1(\alpha)} \cdots A_n^{\ell_n(\alpha)} = F_0 F_1^{\alpha} \cdots F_t^{\alpha^t}$$

By the Schwartz-Zippel Lemma this implies that with overwhelming probability

$$f(X) = f_0 + f_1 X + \cdots + f_t X^t = a_1 \ell_1(X) + \cdots + a_n \ell_n(X)$$

and $a_i = f(\omega_i)$. Second observe that $e(A_i, \hat{h}_1) = e(g_1, \hat{A}_i)$ if and only if $\hat{A}_i = \hat{h}_1^{f(\omega_i)}$. Where at least $t+1$ parties are honest the reconstruction algorithm receives at least $t+1$ verifying shares. With $t+1$ verifying shares the reconstruction algorithm always succeeds because f has degree t.

Security preserving We now prove that our DKG satisfies key expressability; i.e., we construct a simulator that is able to fix the output to be a value $\alpha \mathsf{pk}_1 + \mathsf{pk}_2$, where pk_1 is given as input and $\alpha \neq 0$. This does not directly prove that the DKG preserves security, but in the full version of our paper we detail how combining a key-expressable DKG with rekeyable encryption schemes, signature schemes, and VUFs implies that the DKG also preserves security of these primitives. We cover these three due to their popularity (and our VUF construction in Section 7), but envisage that there are many other primitives that are rekeyable and thus similarly preserve their security when combined with key-expressable DKGs.

Theorem 2 (DKG). *The scheme in Figure 3 is key-expressable as per Definition 7 in the random oracle model for any primitive with keys of the form* $\mathsf{pk} = (g_1^a, \hat{u}_1^a) \in \mathbb{G}_1 \times \mathbb{G}_2$ *and* $\mathsf{sk} = \hat{h}_1^a \in \mathbb{G}_2$.

Proof. We design an adversary $\mathcal{B}$ that takes as input pk_1 such that whenever the DKG outputs pk, $\mathcal{B}$ outputs $\alpha, \mathsf{pk}_2, \mathsf{sk}_2$ such that $\mathsf{pk} = \alpha\mathsf{pk}_1 + \mathsf{pk}_2$. Suppose $\mathcal{B}$ receives input $\mathsf{pk}_1 = (g_2, \hat{v}_2)$.

First $\mathcal{B}$ runs the DKG with $\mathcal{A}$. Let $\mathbb{I}_B \subset [1, n]$ be the set of corrupted (i.e. "bad") parties and $\mathbb{I}_G \subset [1, n]$ be the set of uncorrupted ("good") parties. For good parties P_k, $\mathcal{B}$ *simulates* the adversarial view of this party's output, so that public view $C_k, \hat{u}_{k,2}$ sent by P_k is equal to $(g_2^{a_k}, \hat{v}_2^{a_k})$.

In the course of this simulation, $\mathcal{B}$ answers $\mathcal{A}$'s queries to the oracle $\mathsf{Hash}_{\mathbb{G}_2}$ by selecting $r \xleftarrow{\$} \mathbb{F}$ at random, and returning $\hat{h}_1^r$.

In the registration round, when $\mathcal{A}$ queries $\mathcal{B}$ on the k-th honest value, $\mathcal{B}$ chooses $\mu_k, \kappa_k \xleftarrow{\$} \mathbb{F}$ randomly from the field and returns the public key $(\mathsf{ek}_k, \mathsf{vk}_k) = (\hat{u}_1^{\mu_k}, g_2^{\kappa_k})$.

In the broadcast round, $\mathcal{B}$ chooses $a_k \xleftarrow{\$} \mathbb{F}$ randomly for each honest party and computes $C_k = g_2^{a_k}$. It then samples $\chi_k, \psi_k \xleftarrow{\$} \mathbb{F}$ and programs $\mathsf{Hash}_{\mathbb{G}_2}$ to return $\hat{u}_1^{\chi_k}$ and $\hat{u}_1^{\psi_k}$ on input C_k and (vk_k, C_k) respectively. Finally it broadcasts C_k. With overwhelming probability, $\mathcal{A}$ is yet to query the randomised value C_k.

In the share creation round, when queried on P_k, $\mathcal{B}$ is required to output

$$(\boldsymbol{F}_k, \hat{u}_{k,2}, \hat{\sigma}_k, \boldsymbol{A}_k, \hat{\boldsymbol{Y}}_k)$$

that are indistinguishable from a valid output. Assume without loss of generality that $|\mathbb{I}_B| = t$. It then behaves as follows

1. Choose random $\bar{x}_{k,j} \xleftarrow{\$} \mathbb{F}$ for each $j \in \mathbb{I}_B$ and interpolate in the exponent to find $(F_{k,0}, \dots, F_{k,t})$ such that $F_{k,i} = g_1^{c_i}$, where $\sum_{i=0}^{t} c_i X^i$ evaluates to $\bar{x}_{k,j}$ at ω_j for $j \in \mathbb{I}_B$ and $a_k \log_{g_1}(g_2)$ at 0. These c_i values are unknown to $\mathcal{B}$.
2. Set $\hat{u}_{k,2} = \hat{v}_2^{a_k}$.
3. Set $\sigma_k = (\hat{v}_2^{a_k\chi_k}, \hat{v}_2^{\kappa_k\psi_k})$.
4. To compute $A_{k,1}, \dots, A_{k,n}$, set $A_{k,j} = \prod_{i=0}^{t} F_{k,i}^{\omega_j^i}$.
5. To compute $\hat{Y}_{k,j}$ for $j \in \mathbb{I}_B$, return $\mathsf{ek}_j^{\bar{x}_{k,j}}$. To compute $\hat{Y}_{k,j}$ for $j \in \mathbb{I}_G$, interpolate in the exponent to find $\hat{u}_1^{c_0}, \dots, u_1^{c_t}$ for $c_0, \dots, c_{t-1}$ as in Step 1 (recall that $\mathcal{B}$ knows $\hat{u}_1^{\log_{g_1}(g_2)}$). Return $\hat{Y}_{k,j} = \prod_{i=0}^{t} \hat{u}_1^{c_i \mu_j \omega_j^i}$.

This simulation is perfect. Indeed $c_0 = \log_{g_1}(C_k)$ and $c_1, \dots, c_t$ are randomly distributed. We have that $\hat{u}_{k,2} = \hat{v}_2^{a_k(\nu+1)} = \hat{u}_1^{\log_{g_1}(C_k)}$. Also, $\sigma_{k,1} = \mathsf{Hash}_{\mathbb{G}_2}(C_k)^{\log_{g_1}(C_k)}$ and $\sigma_{k,2} = \mathsf{Hash}_{\mathbb{G}_2}(\mathsf{vk}_k, C_k)^{\log_{g_1}(\mathsf{vk}_k)}$. The values $A_{k,1}, \dots, A_{k,n}$ are computed honestly and are the unique encryptions that satisfy the verifier.

Suppose that the DKG terminates with transcript $((C_1', \dots, C_n'), (w_1, \dots, w_n), (\sigma_1', \dots, \sigma_n'), \mathsf{pvss})$. The public key is given by $C = C_1' \cdots C_n', \hat{u}_2 = \hat{u}_{1,2} \cdots \hat{u}_{n,2}$. For each adversarial contribution C_i', $\mathcal{B}$ looks up r such that $\mathsf{Hash}_{\mathbb{G}_2}(C_j') = \hat{h}_1^r$. Here, i can be any index, as $\mathcal{A}$ might have forged one of $\mathcal{B}$'s contributions. If the adversary has not queried $\mathsf{Hash}_{\mathbb{G}_2}$ on C' then the probability of them returning a verifying signature σ is negligible. To get the secret key share, $\mathcal{B}$ extracts $\hat{C}_i = \hat{\sigma}^{\frac{1}{r}}$ such that $\hat{C}_i = \hat{h}_1^{\log_{g_1}(C_i)}$.

If $\mathcal{A}$ has included at least one of $\mathcal{B}$'s contributions, then $\mathcal{B}$ computes $z = \sum_{k \in S} w_k$ for S the set of honest participants whose contribution is included in the transcript. Additionally, $\mathcal{B}$ computes $\mathsf{pk}_2 = (\prod_{i \notin S} C'_i, \prod_{i \notin S} \hat{u}_{i,2})$ and $\mathsf{sk}_2 = \prod_{i \notin S} \hat{C}_i$. Then, we have that $\mathsf{pk} = \alpha \mathsf{pk}_1 + \mathsf{pk}_2$ for $\alpha \neq 0$ and sk_2 is a key for pk_2. Thus $\mathcal{B}$ returns (α, sk_2).

If $\mathcal{A}$ has not included any contributions from $\mathcal{B}$, then that $\mathcal{A}$ has forged a signature σ'_k with respect to some $\mathsf{vk}_k = g_2^{\kappa_k}$ and contribution C'_k. Using the oracle queries, $\mathcal{B}$ looks up r such that $\mathsf{Hash}_{\mathbb{G}_2}(\mathsf{vk}_k, C'_k) = \hat{h}_1^r$. Since $\sigma'_k = (\sigma'_{k,1}, \sigma'_{k,2})$ verifies, we have that $\sigma'_{k,2} = \hat{h}_1^{r\kappa_k \log_{g_1}(g_2)}$. Thus, $\mathcal{B}$ computes $\mathsf{sk}_1 = (\sigma'_{k,2})^{\frac{1}{r\kappa_k}}$. Additionally, $\mathcal{B}$ computes $\mathsf{pk}_2 = (g_2^{-1} \prod_i C'_i, \hat{v}_2^{-1} \prod_i \hat{u}_{i,2})$ and $\mathsf{sk}_2 = \mathsf{sk}_1^{-1} \prod_i \hat{C}_i$. Then, we have that $\mathsf{pk} = \mathsf{pk}_1 + \mathsf{pk}_2$ and sk_2 is a key for pk_2 and $\mathcal{B}$ returns $(1, \mathsf{sk}_2)$.

6 Alternative DKGs Have Provable Security

In this section we demonstrate that two popular DKGs, the Pedersen DKG and the Fouque-Stern DKG, are also key-expressable. As a direct consequence, they can be used to securely instantiate a DKG for both El-Gamal encryption and BLS signatures, as we prove in the full version of our paper [38]. Our results generalise to other rekeyable constructions that have public keys in $\mathbb{G}$ and secret keys in $\mathbb{F}$. In addition to justifying the applicability of our security definitions and proof techniques, we hope this also fills a gap in the literature as we are unaware of other works that provide correct proofs for these DKGs.

6.1 Pedersen DKG from Feldman's VSS

We prove that key expressability holds for Pedersen's DKG provided the threshold of adversarial participants is less than $n/2$. It is our belief that this bound on the number of adversarial participants can be removed provided that one gives signatures of knowledge of the individual contributions. Pedersen's DKG can be seen as n parallel instantiations of the Feldman VSS [25]. We remind the reader that key expressability does not imply secrecy (invalidating the attack of Gennaro et al. [31]) but does allow us to prove the security preservation of certain rekeyable schemes. A proof of the following theorem is provided in the full version of our paper.

Theorem 3. *The Pedersen DKG is a key-expressable DKG against static adversaries with adversarial threshold $t < n/2$ for any scheme whose key generation outputs values $\mathsf{pk} = g_1^a \in \mathbb{G}_1$, $\mathsf{sk} = a \in \mathbb{F}$.*

6.2 The Fouque-Stern publicly verifiable DKG

We now show the key expressability of the publicly verifiable Fouque Stern DKG [27]. This DKG has the benefit of outputting field elements as secret keys, but the total communication and verification costs are of order $\mathcal{O}(n^2)$. Unlike Fouque and Stern's original argument, we allow for the existence of rushing adversaries. Indeed Fouque and Stern rely in their reduction on an honest party playing last. Instantiating such an assumption would require the use of a trusted third party and therefore negate the benefits of distributing the key generation. A proof of the following theorem is provided in the full version of our paper.

Theorem 4. *The Fouque-Stern DKG is a key-expressable DKG in the random oracle model against static adversaries under the decisional composite residuosity assumption for any scheme whose key generation outputs values* $\mathsf{pk} = g_1^a \in \mathbb{G}_1$, $\mathsf{sk} = a \in \mathbb{F}$.

6.3 El-Gamal and BLS

In the full version of our paper, we observe that El-Gamal encryption and BLS signatures are both rekeyable (and both have field elements as secret keys). We thus obtain the following two corollaries:

Corollary 1. *The El-Gamal encryption scheme is IND-CPA-secure when instantiated with the Pedersen DKG or the Fouque-Stern DKG.*

Corollary 2. *The BLS signature scheme is EUF-CMA-secure when instantiated with the Pedersen DKG or the Fouque-Stern DKG.*

7 A Structure-Preserving VUF

In this section, we introduce a verifiable unpredictable function (VUF), secure in the random oracle model, that has group elements as the secret key. We can thus securely instantiate our VUF using our DKG.

As one application, VUFs can be used to create randomness beacons, where unlike in, *e.g.*, BLS multi-signatures [11], if a threshold of signers is reached, then the same signature is always produced. By hashing the outcome of this VUF with a random oracle we can obtain a verifiable random function (VRF). Abe et al. [1] proved that it is impossible to construct an algebraic VUF with a secret key as a group element. Since we are using a hash function, however, we are not fully algebraic and therefore sidestep this impossibility result.

7.1 Our construction

Our VUF scheme is given in Figure 4. The techniques were inspired by a combination of BLS signatures [13] and Escala-Groth NIZKs [24] (which are an improvement of Groth-Sahai proofs [37]). Unlike BLS signatures our secret keys are group elements and unlike Escala-Groth NIZKs our VUFs are non-malleable.

Given an input $m \in \mathbb{F}$ under public key $g_1^a, \hat{u}_1^a$ and secret key $\hat{h}_1^a$, the unique output given by $\mathsf{VUF.Eval}(\mathsf{sk}, m)$ is $e(\mathsf{Hash}_{\mathbb{G}_1}(m), \hat{h}_1^a)$. Given $g_1^a \in \mathbb{G}_1$ and $\mathsf{Hash}_{\mathbb{G}_1}(m) \in \mathbb{G}_1$, it is hard for an adversary to compute $e(\mathsf{Hash}_{\mathbb{G}_1}(m), \hat{h}_1)^a \in \mathbb{G}_T$.

We formally prove in Theorem 5 and 6 that our VUF satisfies uniqueness (see Definition 3) and unpredictability (see Definition 4) under the SXDH and BDH assumptions.

```
$\underline{\mathsf{VUF.Setup}(\mathsf{bp}, \mathsf{Hash}_{\mathbb{G}_1})}$
$\hat{u}_1, \hat{h}_2, \hat{h}_3, \hat{h}_4 \xleftarrow{\$} \mathbb{G}_2$
$\mathsf{crs}_{\mathsf{vuf}} \leftarrow (\mathsf{bp}, \mathsf{Hash}_{\mathbb{G}_1}, \hat{h}_2, \hat{h}_3, \hat{h}_4)$
return $\mathsf{crs}_{\mathsf{vuf}}$
```

```
$\underline{\mathsf{VUF.Gen}(\mathsf{crs}_{\mathsf{vuf}})}$
$a \xleftarrow{\$} \mathbb{F}$, $\mathsf{pk} \leftarrow g_1^a, \hat{u}_1^a \in (\mathbb{G}_1 \times \mathbb{G}_2)$
$\mathsf{sk} \leftarrow \hat{h}_1^a \in \mathbb{G}_2$
return $(\mathsf{pk}, \mathsf{sk})$
```

```
$\underline{\mathsf{VUF.Eval}(\mathsf{crs}_{\mathsf{vuf}}, \mathsf{sk}, m)}$
$Z \leftarrow \mathsf{Hash}_{\mathbb{G}_1}(m)$
return $e(Z, \mathsf{sk})$
```

```
$\underline{\mathsf{VUF.Derive}(\mathsf{crs}_{\mathsf{vuf}}, \mathsf{pk}, m, \sigma)}$
$(\pi_1, \pi_2, \pi_3, \pi_4 \in \mathbb{G}_1^4, \hat{\pi}_1, \hat{\pi}_2 \in \mathbb{G}_2^2) \leftarrow \mathsf{parse}(\sigma)$
$Z \leftarrow \mathsf{Hash}_{\mathbb{G}_1}(m)$
return $e(Z, \hat{\pi}_2)e(\pi_2, \hat{h}_3)e(\pi_4, \hat{h}_4)$
```

```
$\underline{\mathsf{VUF.Sign}(\mathsf{crs}_{\mathsf{vuf}}, \mathsf{sk}, m)}$
$Z \leftarrow \mathsf{Hash}_{\mathbb{G}_1}(m)$
$\alpha, \beta \xleftarrow{\$} \mathbb{F}$
$\pi_1, \pi_2, \pi_3, \pi_4 \leftarrow g_1^\alpha, Z^\alpha, g_1^\beta, Z^\beta$
$\hat{\pi}_1, \hat{\pi}_2 \leftarrow \hat{h}_1^{-\alpha}\hat{h}_2^{-\beta},\ \hat{h}_3^{-\alpha}\hat{h}_4^{-\beta} \cdot \mathsf{sk}$
return $(\pi_1, \pi_2, \pi_3, \pi_4, \hat{\pi}_1, \hat{\pi}_2)$
```

```
$\underline{\mathsf{VUF.Ver}(\mathsf{crs}_{\mathsf{vuf}}, \mathsf{pk}, m, \sigma)}$
$(A, \hat{u}_2) \leftarrow \mathsf{parse}(\mathsf{pk})$
$(\pi_1, \pi_2, \pi_3, \pi_4 \in \mathbb{G}_1^4, \hat{\pi}_1, \hat{\pi}_2 \in \mathbb{G}_2^2) \leftarrow \mathsf{parse}(\sigma)$
$Z \leftarrow \mathsf{Hash}_{\mathbb{G}_1}(m)$
check $1 = e(g_1, \hat{\pi}_1)e(\pi_1, \hat{h}_1)e(\pi_3, \hat{h}_2)$
check $1 = e(Z, \hat{\pi}_1)e(\pi_2, \hat{h}_1)e(\pi_4, \hat{h}_2)$
check $e(A, \hat{h}_1) = e(g_1, \hat{\pi}_2)e(\pi_1, \hat{h}_3)e(\pi_3, \hat{h}_4)$
return 1 if all checks pass, else return 0
```

Fig. 4. Verifiable unpredictable function with group elements as the secret key.

Setup: The setup algorithm is a transparent algorithm that takes as input the bilinear group $\mathsf{bp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, \hat{h}_1)$ and returns four group elements in the second source group: $\hat{u}_1, \hat{h}_2, \hat{h}_3, \hat{h}_4 \in \mathbb{G}_2^4$.

KeyGen: The $\mathsf{VUF.Gen}$ algorithm takes as input the common reference string. It samples a random field element $a \xleftarrow{\$} \mathbb{F}$. The public key $\mathsf{pk} \in \mathbb{G}_1 \times \mathbb{G}_2$ and the secret key $\mathsf{sk} \in \mathbb{G}_2$ are given as $\mathsf{pk} = (g_1^a, \hat{u}_1^a)$ and $\mathsf{sk} = \hat{h}_1^a$.

Sign: The $\mathsf{VUF.Sign}$ algorithm first hashes the message m to obtain $Z \in \mathbb{G}_1$ as $Z = \mathsf{Hash}_{\mathbb{G}_1}(m)$. The signer generates a commitment to sk by sampling random elements $\alpha, \beta \in \mathbb{F}$ and computing

$$(\hat{\pi}_1, \hat{\pi}_2) = (\hat{h}_1^{-\alpha}\hat{h}_2^{-\beta}, \mathsf{sk} \cdot \hat{h}_3^{-\alpha}\hat{h}_4^{-\beta}).$$

If $\hat{h}_1, \hat{h}_2, \hat{h}_3, \hat{h}_4$ are randomly distributed, this commitment is perfectly hiding. However, if $\hat{h}_1, \hat{h}_2, \hat{h}_3, \hat{h}_4$ form an SXDH challenge, then there exists some ξ such that $\hat{h}_3 = \hat{h}_1^\xi$ and $\hat{h}_4 = \hat{h}_2^\xi$, meaning that the commitment forms an El-Gamal encryption of sk. In this case, we say that the commitment is perfectly binding.

Having generated $(\hat{\pi}_1, \hat{\pi}_2)$, the signer now generates $(\pi_1, \pi_2, \pi_3, \pi_4) \in \mathbb{G}_1^4$ such that

$$(\pi_1, \pi_2, \pi_3, \pi_4) = (g_1^{\alpha},\, Z^{\alpha},\, g_1^{\beta},\, Z^{\beta})$$

These signature elements have been designed such that the random blinders α, β are canceled out in the verifier's equations.

The signer returns the output $\sigma = (\pi_1, \pi_2, \pi_3, \pi_4, \hat{\pi}_1, \hat{\pi}_2)$.

Derive: The $\mathsf{VUF.Derive}$ computes $Z = \mathsf{Hash}_{\mathbb{G}_1}(m)$ and then returns

$$T = e(Z, \hat{\pi}_2)e(\pi_2, \hat{h}_3)e(\pi_4, \hat{h}_4)$$

as the unique and unpredictable component. If the signer is honest then $T = e(Z, \mathsf{sk}) = \mathsf{VUF.Eval}(\mathsf{crs_{vuf}}, \mathsf{sk}, m)$.

Verify: The $\mathsf{VUF.Ver}$ algorithm parses the signature to check that $(\pi_1, \pi_2, \pi_3, \pi_4)$ is in $\mathbb{G}_1^4$, and $(\hat{\pi}_1, \hat{\pi}_2)$ is in $\mathbb{G}_2^2$. The verifier computes Z identically to the signer, *i.e.*, $Z = \mathsf{Hash}_{\mathbb{G}_1}(m)$. The verifier then checks that three pairing equations are satisfied in order to be convinced that there exist α, β such that

$$(\pi_2, \pi_4, \hat{\pi}_2) = (Z^{\alpha}, Z^{\beta}, \hat{h}_3^{-\alpha}\hat{h}_4^{-\beta} \cdot \mathsf{sk})$$

Specifically, they check that:

$$1 = e(g_1, \hat{\pi}_1)e(\pi_1, \hat{h}_1)e(\pi_3, \hat{h}_2) \tag{2}$$

$$1 = e(Z, \hat{\pi}_1)e(\pi_2, \hat{h}_1)e(\pi_4, \hat{h}_2) \tag{3}$$

$$e(\mathsf{pk}, \hat{h}_1) = e(g_1, \hat{\pi}_2)e(\pi_1, \hat{h}_3)e(\pi_3, \hat{h}_4) \tag{4}$$

They return 1 if all these checks pass and 0 otherwise.

Given a signature that satisfies these equations, an extractor that knows a trapdoor SXDH relation between the CRS elements can output a valid witness sk. However, there also exists a simulated CRS indistinguishable from random such that we can simulate signatures without knowing sk.

Threshold VUF Scheme We discuss how to transform our VUF into a threshold VUF. The individual VUF shares can be made shorter using an optimisation in the full version of our paper [38]. Suppose that there are n parties $P_1, \ldots, P_n$ and we want that any $t+1$ of them can jointly sign a message, but that t of them cannot. We use Shamir's secret sharing scheme and choose a degree t polynomial $f(X)$. Let $\omega_1, \ldots, \omega_n$ denote unique evaluation points and $\ell_{S,1}(X), \ldots, \ell_{S,t+1}(X)$ denote the Lagrange polynomials such that for all $\omega_j \in S$ we have that $\ell_{S,i}(\omega_j)$ is equal to 1 if $i = j$ and 0 otherwise.

The threshold setup algorithm runs identically to the non-threshold version to return $\mathsf{crs_{vuf}}$. The key generation outputs a public key and n secret key shares of the form

$$\mathsf{pk} = (g_1^{f(0)}, \hat{u}_1^{f(0)}), \mathsf{sk}_1 = \hat{h}_1^{f(\omega_1)}, \ldots, \mathsf{sk}_n = \hat{h}_1^{f(\omega_n)}.$$

To compute their share of the threshold signature on m party P_i outputs

$$\sigma_i = (\pi_{i,1}, \pi_{i,2}, \pi_{i,3}, \pi_{i,4}, \hat{\pi}_{i,1}, \hat{\pi}_{i,2}) \xleftarrow{\$} \mathsf{VUF.Sign}(\mathsf{crs}_{\mathsf{vuf}}, \mathsf{sk}_i, m)$$

To aggregate t signature shares on m from parties $\{P_i\}_{i \in S}$ compute

$$\sigma = \left(\prod_{i \in S} \pi_{i,1}^{\ell_{S,i}(0)}, \prod_{i \in S} \pi_{i,2}^{\ell_{S,i}(0)}, \prod_{i \in S} \pi_{i,3}^{\ell_{S,i}(0)}, \prod_{i \in S} \pi_{i,4}^{\ell_{S,i}(0)}, \prod_{i \in S} \hat{\pi}_{i,1}^{\ell_{S,i}(0)}, \prod_{i \in S} \hat{\pi}_{i,2}^{\ell_{S,i}(0)} \right)$$

The verification and derive algorithms run identically to their non-threshold counterparts on the input $(\mathsf{crs}_{\mathsf{vuf}}, \mathsf{pk}, m, \sigma)$

We briefly show that σ is correct. Set $Z = \mathsf{Hash}_{\mathbb{G}_1}(m)$ and see that $\sigma = (\pi_1, \pi_2, \pi_3, \pi_4, \hat{\pi}_1, \hat{\pi}_2)$ is given by

$$\begin{aligned}
\pi_1 &= \textstyle\prod_{i \in S} \pi_{i,1}^{\ell_{S,i}(0)} &&= g_1^{\sum_{i \in S} \alpha_i \ell_{S,i}(0)} \\
\pi_2 &= \textstyle\prod_{i \in S} \pi_{i,2}^{\ell_{S,i}(0)} &&= Z^{\sum_{i \in S} \alpha_i \ell_{S,i}(0)} \\
\pi_3 &= \textstyle\prod_{i \in S} \pi_{i,3}^{\ell_{S,i}(0)} &&= g_1^{\sum_{i \in S} \beta_i \ell_{S,i}(0)} \\
\pi_4 &= \textstyle\prod_{i \in S} \pi_{i,4}^{\ell_{S,i}(0)} &&= Z^{\sum_{i \in S} \beta_i \ell_{S,i}(0)} \\
\hat{\pi}_1 &= \textstyle\prod_{i \in S} \hat{\pi}_{i,1}^{\ell_{S,i}(0)} &&= \hat{h}_1^{-\sum_{i \in S} \alpha_i \ell_{S,i}(0)} \hat{h}_2^{-\sum_{i \in S} \beta_i \ell_{S,i}(0)} \\
\hat{\pi}_2 &= \textstyle\prod_{i \in S} \hat{\pi}_{i,2}^{\ell_{S,i}(0)} &&= \hat{h}_3^{-\sum_{i \in S} \alpha_i \ell_{S,i}(0)} \hat{h}_4^{-\sum_{i \in S} \beta_i \ell_{S,i}(0)} \textstyle\prod_{i \in S} \mathsf{sk}_i^{\ell_{S,i}(0)} \\
& &&= \hat{h}_3^{-\sum_{i \in S} \alpha_i \ell_{S,i}(0)} \hat{h}_4^{-\sum_{i \in S} \beta_i \ell_{S,i}(0)} \hat{h}_1^{f(\omega_i) \ell_{S,i}(0)}
\end{aligned}$$

Since f has degree t we have that $f(\omega_i)\ell_{S,i}(0) = f(0)$. Denote $\alpha = \sum_{i \in S} \alpha_i \ell_{S,i}(0)$ and $\beta = \sum_{i \in S} \beta_i \ell_{S,i}(0)$ in the above equation to get that

$$(\pi_1, \pi_2, \pi_3, \pi_4, \hat{\pi}_1, \hat{\pi}_2) = (g_1^{\alpha}, Z^{\alpha}, g_1^{\beta}, Z^{\beta}, h_1^{-\alpha} h_2^{-\beta}, h_3^{-\alpha} h_4^{-\beta} \hat{h}_1^{f(0)}) \ .$$

Thus the threshold signature is distributed identically to the non-threshold counterpart and the verifier and deriver output 1 and $e(Z, \hat{h}_1)^{f(0)}$, respectively.

Aggregatable Signature Scheme It is also possible to use our VUF to instantiate an aggregatable signature scheme with secret keys as group elements. For aggregating, one simply takes the product of the public key elements output by $\mathsf{VUF.Gen}$ and the signature elements output by $\mathsf{VUF.Sign}$. Similar to the BLS scheme, this aggregatable signature scheme would be susceptible to *rogue key attacks* [49]. It is thus important to provide simulation-extractable proofs of knowledge of secret keys as part of a public key infrastructure.

7.2 Security analysis

To prove that our VUF is secure, we need to prove that it satisfies uniqueness and unpredictability.

Theorem 5. *The VUF in Figure 4 satisfies uniqueness (Definition 3) under the* SXDH *assumption in the random oracle model.*

Theorem 6. *The VUF in Figure 4 satisfies unpredictability (Definition 4) under the* SXDH *and the* BDH *assumption in the random oracle model.*

We provide formal proofs of these theorems in the full version of our paper. Intuitively, uniqueness relies on the fact that it would be statistically impossible to satisfy the verifiers equations for a wrong evaluation if the CRS was made up of an SXDH instance. Since VUF.Eval is deterministic there can only be one correct evaluation. Thus if an adversary could break uniqueness in the general case, then we could use them as a subroutine to determine SXDH instances from random.

Our unpredictability proof uses an adversary who predicts the VUF to compute a BDH output. To do this we embed one component of the BDH challenge into the public key being targeted, and the other into the adversaries random oracle queries. However, we also need to simulate responses to the adversaries signature requests, and to do this (after jumping to a hybrid game with a structured CRS) we need to program the oracle such that we know a discrete log. This could present a collision as the adversary may have already queried that point. To counteract, we take a random guess as to which oracle query the adversary will output their prediction for, and if we guess wrong we abort. Thus our reduction is not tight, but does provide us with a polynomial chance of success whenever the adversary succeeds.

After observing that our VUF is rekeyable, we prove the following corollary in the full version of our paper.

Corollary 3. *The VUF in Figure 4 is unique and unpredictable when instantiated with the DKG in Figure 3.*

8 Implementation

We implement our DKG and VUF and summarise the performance of our schemes in Tables 2 and 3. Our implementation is written in Rust on top of the `libzexe` library, which performs efficient finite field arithmetic, elliptic curve arithmetic, and finite field FFTs. We evaluate our DKG and VUF on a desktop machine with an $i7$-8700k CPU at 3.7GHz and 32GB of DDR4 RAM. We use the BLS12-381 curve. For hashing to groups, we use the try-and-reject method by instantiating a ChaCha20 RNG with a Blake2s hash of the input message, sampling field elements and checking if they are valid x-coordinates, deriving the corresponding point if so. Our implementation is not constant-time. Upon publication, we plan to release our implementation as open-source software.

We utilise a few optimization techniques throughout the implementation. First, when verifying multiple pairing equations, we instead compute a randomised check of a single pairing equation so as to amortise the cost of the final exponentiations. We then compute the pairing product efficiently using the underlying `libzexe` implementation. In the same vein, when verifying pairing equations where two pairings are computed with respect to the same source

Parties	DKG.Deal (ms)	Scrape.Verify (ms)	DKG.Verify (ms)	Transcript size (kB)
64	72	96	376	25
128	124	178	704	50
256	271	346	1305	99
8192	8000	9900	42 600	3146

Table 2. The performance of our DKG, averaged across 10 samples of each operation. For n parties, we use a threshold of $t = 2n/3$.

	Our VUF	Our optimised VUF	BLS [13]
Key prove (ms)	-	2.89	-
Public key (bytes)	48	336	96
Key verify (ms)	-	4.00	-
Sign (ms)	3.47	0.58	0.44
Signature size (bytes)	384	96	48
Verify (ms)	4.73	2.39	2.15
Derive (ms)	2.37	2.37	-

Table 3. The performance of our VUF (Section 7), our optimised VUF, and the BLS signature scheme. These numbers were averaged across four distinct runs, with 100 samples of each operation per run.

group element, we combine the two into a randomised check. For large multi-exponentiations we use the `libzexe` implementation of Pippenger's algorithm. For large polynomial evaluations we use FFTs. We additionally utilise batch normalization of projective points.

We evaluate our DKG with respect to 64, 128, 256, and 8192 parties. We see that the time taken to compute, verify, and aggregate a transcript all increase linearly in the number of parties. Verifying a transcript with 256 parties takes a little more than a second.

In addition to our VUF presented in Section 7, we also evaluate an optimised VUF that we present in the full version of our paper [38]. We compare the performance of our VUF and our optimised VUF with BLS [13], which is the state of the art in the random oracle model. We do not give the derivation time for BLS because this is the identity function. It can be seen that signing and verifying our optimised VUF is only fractionally more expensive than BLS, but that verifying our full VUF is approximately twice as expensive.

Acknowledgements

Thank you to Ittai Abraham for helpful discussions and feedback. Sarah Meiklejohn was supported in part by EPSRC Grant EP/N028104/1. Gilad Stern was supported by the HUJI Federmann Cyber Security Research Center in conjunction with the Israel National Cyber Directorate (INCD) in the Prime Minister's Office.

References

[1] M. Abe, J. Camenisch, R. Dowsley, and M. Dubovitskaya. "On the Impossibility of Structure-Preserving Deterministic Primitives". In: *J. Cryptology* 32.1 (2019), pp. 239–264.

[2] M. Abe, M. Chase, B. David, M. Kohlweiss, R. Nishimaki, and M. Ohkubo. "Constant-Size Structure-Preserving Signatures: Generic Constructions and Simple Assumptions". In: *J. Cryptol.* 29.4 (2016), pp. 833–878.

[3] M. Abe, J. Groth, M. Kohlweiss, M. Ohkubo, and M. Tibouchi. "Efficient Fully Structure-Preserving Signatures and Shrinking Commitments". In: *J. Cryptol.* 32.3 (2019), pp. 973–1025.

[4] M. Abe, J. Groth, M. Ohkubo, and M. Tibouchi. "Unified, Minimal and Selectively Randomizable Structure-Preserving Signatures". In: *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings.* 2014, pp. 688–712.

[5] I. Abraham, D. Dolev, R. Gonen, and J. Y. Halpern. "Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation". In: *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, July 23-26, 2006.* 2006, pp. 53–62.

[6] I. Abraham, D. Malkhi, and A. Spiegelman. "Asymptotically Optimal Validated Asynchronous Byzantine Agreement". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing.* PODC '19. 2019.

[7] G. Ateniese, J. Camenisch, and B. de Medeiros. "Untraceable RFID tags via insubvertible encryption". In: *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA, November 7-11, 2005.* 2005, pp. 92–101.

[8] L. Ballard, M. Green, B. de Medeiros, and F. Monrose. *Correlation-Resistant Storage via Keyword-Searchable Encryption.* IACR Cryptol. ePrint Arch. 2005/417. 2005.

[9] M. Bellare and P. Rogaway. "The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs". In: *Advances in Cryptology - EUROCRYPT 2006.* Springer Berlin Heidelberg, 2006, pp. 409–426.

[10] F. Benhamouda, T. Lepoint, M. Orrù, and M. Raykova. *On the (in)security of ROS.* Cryptology ePrint Archive, Report 2020/945. 2020.

[11] A. Boldyreva. "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme". In: *PKC 2003.* Ed. by Y. G. Desmedt. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 31–46.

[12] D. Boneh and X. Boyen. "Efficient Selective Identity-Based Encryption Without Random Oracles". In: *J. Cryptology* 24.4 (2011), pp. 659–693.

[13] D. Boneh, B. Lynn, and H. Shacham. "Short Signatures from the Weil Pairing". In: *Advances in Cryptology - ASIACRYPT 2001.* 2001, pp. 514–532.

[14] A. Bonnecaze and P. Trebuchet. "Threshold Signature for Distributed Time Stamping Scheme". In: *Ann. Telecommun.* 62 (2007), pp. 1353–1364.

[15] S. Bowe, A. Gabizon, and I. Miers. "Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model". In: *IACR Cryptol. ePrint Arch.* 2017 (2017), p. 1050. URL: http://eprint.iacr.org/2017/1050.

[16] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. "Adaptive Security for Threshold Cryptosystems". In: *Advances in Cryptology — CRYPTO' 99.* Ed. by M. Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 98–116. ISBN: 978-3-540-48405-9.

[17] J. Canny and S. Sorkin. "Practical Large-Scale Distributed Key Generation". In: *Advances in Cryptology - EUROCRYPT 2004.* Ed. by C. Cachin and J. L. Camenisch. Springer Berlin Heidelberg, 2004, pp. 138–152.

[18] I. Cascudo and B. David. "SCRAPE: Scalable Randomness Attested by Public Entities". In: *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings.* 2017, pp. 537–556.

[19] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. "Verifiable secret sharing and achieving simultaneity in the presence of faults". In: *26th Annual Symposium on Foundations of Computer Science (sfcs 1985).* 1985, pp. 383–395.

[20] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. "How to Share a Function Securely". In: *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing.* STOC '94. 1994, pp. 522–533.

[21] Y. Desmedt and Y. Frankel. "Threshold cryptosystems". In: *Advances in Cryptology — CRYPTO'89.* 1990, pp. 307–315.

[22] DFINITY. *Distributed Key Generation in JS.*

[23] Y. Dodis and A. Yampolskiy. "A Verifiable Random Function with Short Proofs and Keys". In: *Public Key Cryptography - PKC 2005.* Ed. by S. Vaudenay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 416–431. ISBN: 978-3-540-30580-4.

[24] A. Escala and J. Groth. "Fine-Tuning Groth-Sahai Proofs". In: *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings.* 2014, pp. 630–649.

[25] P. Feldman. "A Practical Scheme for Non-interactive Verifiable Secret Sharing". In: *Proceedings of the 28th Annual Symposium on Foundations of Computer Science.* SFCS '87. IEEE Computer Society, 1987, pp. 427–438.

[26] N. Fleischhacker, J. Krupp, G. Malavolta, J. Schneider, D. Schröder, and M. Simkin. "Efficient Unlinkable Sanitizable Signatures from Signatures with Re-Randomizable Keys". In: *Proceedings of PKC 2016.* 2016.

[27] P. Fouque and J. Stern. "One Round Threshold Discrete-Log Key Generation without Private Channels". In: *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001, Proceedings.* 2001, pp. 300–316.

[28] S. D. Galbraith, K. G. Paterson, and N. P. Smart. "Pairings for cryptographers". In: *Discrete Applied Mathematics* 156.16 (2008). Applications of Algebra to Cryptography, pp. 3113 –3121.

[29] D. Galindo, J. Liu, M. Ordean, and J.-M. Wong. *Fully Distributed Verifiable Random Functions and their Application to Decentralised Random Beacons.* Cryptology ePrint Archive, Report 2020/096. 2020.

[30] J. A. Garay, J. Katz, U. Maurer, B. Tackmann, and V. Zikas. "Rational Protocol Design: Cryptography against Incentive-Driven Adversaries". In: *54th Annual*

IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA. 2013, pp. 648–657.

[31] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". In: *Journal of Cryptology* 20 (2007), pp. 51–83.

[32] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. "Secure Applications of Pedersen's Distributed Key Generation Protocol". In: *Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003, San Francisco, CA, USA, April 13-17, 2003, Proceedings*. 2003, pp. 373–390.

[33] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. "Algorand: Scaling Byzantine Agreements for Cryptocurrencies". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. SOSP '17. 2017.

[34] O. Goldreich, S. Micali, and A. Wigderson. "How to Play ANY Mental Game". In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC '87. Association for Computing Machinery, 1987, pp. 218–229.

[35] J. Groth. "Rerandomizable and replayable adaptive chosen ciphertext attack secure cryptosystems". In: *TCC 2004*. Vol. 2951. LNCS. 2004, pp. 152–170.

[36] J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers. "Updatable and Universal Common Reference Strings with Applications to zk-SNARKs". In: *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*. 2018, pp. 698–728.

[37] J. Groth and A. Sahai. "Efficient Noninteractive Proof Systems for Bilinear Groups". In: *SIAM J. Comput.* 41.5 (2012), pp. 1193–1232.

[38] K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. *Aggregatable Distributed Key Generation*. https://eprint.iacr.org/2021/005. 2021.

[39] Heiko Stamer. *Distributed Privacy Guard*.

[40] GNOSIS. *Distributed Key Generation*.

[41] A. Kate and I. Goldberg. "Distributed Key Generation for the Internet". In: *29th IEEE International Conference on Distributed Computing Systems*. 2009, pp. 119–128.

[42] A. Kate. "Distributed Key Generation and Its Applications". PhD thesis. Waterloo, Ontario, Canada, 2010.

[43] A. Kate, G. M. Zaverucha, and I. Goldberg. "Constant-Size Commitments to Polynomials and Their Applications". In: *ASIACRYPT'10*. 2010. ISBN: 978-3-642-17373-8.

[44] A. Kiayias, A. Russell, B. David, and R. Oliynykov. "Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol". In: *Advances in Cryptology – CRYPTO 2017*. 2017.

[45] E. Kokoris-Kogias, E. C. Alp, L. Gasser, P. Jovanovic, E. Syta, and B. Ford. *Verifiable Management of Private Data under Byzantine Failures*. Cryptology ePrint Archive, Report 2018/209. 2018.

[46] E. Kokoris-Kogias, D. Malkhi, and A. Spiegelman. *Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures*. Cryptology ePrint Archive, Report 2019/1015. 2019.

[47] C. Komlo and I. Goldberg. "FROST: Flexible Round-Optimized Schnorr Threshold Signatures". In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 852.

[48] S. Micali, M. Rabin, and S. Vadhan. "Verifiable Random Functions". In: *40th Annual Symposium on Foundations of Computer Science*. 1999, pp. 120–130.

[49] S. Micali, K. Ohta, and L. Reyzin. "Accountable-Subgroup Multisignatures: Extended Abstract". In: *Proceedings of the 8th ACM Conference on Computer and Communications Security*. CCS '01. Philadelphia, PA, USA: Association for Computing Machinery, 2001, 245–254.

[50] W. Neji, K. Blibech, and N. Ben Rajeb. "Distributed key generation protocol with a new complaint management strategy". In: *Security and Communication Networks* 9.17 (2016), pp. 4585–4595.

[51] Orbs Network. *Orbs Network: DKG for BLS threshold signature scheme on the EVM using solidity*. 2018.

[52] P. Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Advances in Cryptology — EUROCRYPT '99*. Ed. by J. Stern. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238.

[53] T. P. Pedersen. "A Threshold Cryptosystem without a Trusted Party". In: *Advances in Cryptology – EUROCRYPT '91*. Ed. by D. W. Davies. Springer Berlin Heidelberg, 1991, pp. 522–526.

[54] T. P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Proceedings on Advances in Cryptology*. Ed. by J. Feigenbaum. CRYPTO '91. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 129–140.

[55] M. Prabhakaran and M. Rosulek. "Rerandomizable RCCA encryption". In: *Advances in Cryptology - CRYPTO 2007*. Vol. 4622. LNCS. 2007, pp. 517–534.

[56] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl. *ETHDKG: Distributed Key Generation with Ethereum Smart Contracts*. Cryptology ePrint Archive, Report 2019/985. 2019.

[57] C. P. Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. Ed. by G. Brassard. New York, NY: Springer New York, 1990, pp. 239–252. ISBN: 978-0-387-34805-6.

[58] A. Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979).

[59] E. Syta et al. "Scalable Bias-Resistant Distributed Randomness". In: *38th IEEE Symposium on Security and Privacy*. San Jose, CA, May 2017.

[60] A. Tomescu et al. "Towards Scalable Threshold Cryptosystems". In: *IEEE S&P'20*. 2020.

[61] D. Tulone. "A Scalable and Intrusion-tolerant Digital Time-stamping System". In: *2006 IEEE International Conference on Communications*. Vol. 5. 2006, pp. 2357–2363.

[62] Y. Wang, Z. Zhang, T. Matsuda, G. Hanaoka, and K. Tanaka. "How to Obtain Fully Structure-Preserving (Automorphic) Signatures from Structure-Preserving Ones". In:

[63] T. M. Wong, C. Wang, and J. M. Wing. "Verifiable Secret Redistribution for Archive Systems". In: *First International IEEE Security in Storage Workshop*. 2002, pp. 94–105.

[64] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. "HotStuff: BFT Consensus with Linearity and Responsiveness". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. PODC '19. 2019.