Distributed Verifiable Random Function With Compact Proof

Ahmet Ramazan Ağırtaş[1][0000−0002−4574−0067],
Arda Buğra Özer[2,⋆][0000−0002−6505−7038],
Zülfükar Saygı[3][0000−0002−7575−3272], and
Oğuz Yayla[2][0000−0001−8945−2780]

[1] Nethermind, London, UK
a.r.agirtas@gmail.com
[2] Institute of Applied Mathematics, Middle East Technical University, Ankara, Türkiye
abozer@gmail.com, oguz@metu.edu.tr
[3] Department of Mathematics, TOBB University of Economics and Technology, Ankara, Türkiye
zsaygi@etu.edu.tr

Abstract. Verifiable Random Functions (VRFs) are cryptographic primitives that generate unpredictable randomness along with proofs that are verifiable, a critical requirement for blockchain applications in decentralized finance, online gaming, and more. Existing VRF constructions often rely on centralized entities, creating security vulnerabilities. Distributed VRFs (DVRFs) offer a decentralized alternative but face challenges like large proof sizes or dependence on computationally expensive bilinear pairings. In this research, a unique distributed VRF (DVRF) system called DVRFwCP with considerable improvements is proposed. DVRFwCP has constant-size proofs, which means that the size of the proof does not change based on the number of participants. This overcomes a significant drawback of earlier DVRF systems, which saw proof size increase with participant count. Furthermore, DVRFwCP produces more efficient verification than previous systems by eliminating the requirement for bilinear pairings throughout the verification process. These innovations contribute to a more secure and scalable solution for generating verifiable randomness in decentralized environments. We compare our construction to well-established DVRF instantiations such as DDH-DVRF and GLOW-DVRF while also pointing out the major improvement in the estimated gas cost of these algorithms.

Keywords: Cryptography, Verifiable Random Function, Distributed Verifiable Random Function, Blockchain

1 Introduction

True randomness is a key element of computing, and its applications range from generating cryptographic keys to digital signing, online gaming, and gambling.

⋆ Corresponding author

Recently, with the introduction and rapid growth of blockchain technologies and Web3-based applications, especially in decentralized finance and the online gaming industry, the demand for trustworthy sources of randomization has increased substantially. In most of these applications involving multiple participants, ensuring that the randomness used is neither predictable nor biased toward any one party is not only important but crucial. Many platforms that rely on blockchain technology have consensus protocols that involve authorizing the creation of blocks to a block producer, whose selection mechanism frequently involves a means for collective sampling of random values. A typical solution to avoid dependence on a trusted party is to utilize a process that permits the distributed generation of verifiable randomness.

Micali, Rabin, and Vadhan [16] introduced verifiable random functions (VRF) that offer such capabilities. A VRF may be viewed as the public-key equivalent of a keyed cryptographic hash. For a secret key chosen uniformly at random, a VRF, on the input of a plaintext α, outputs γ and a proof π, which can publicly be used to verify the correct evaluation of γ. They realized that VRFs and unique signatures share similarities and constructed an RSA signature based VRF. Dodis and Yampolskiy [6] constructed a more optimal VRF using bilinear pairings and collision-resistant hash functions. A more efficient construction using elliptic curves was introduced by Papadopoulos et al. [17]. This was later brought into IETF standardization by Goldberg et al. [12]. In [2], Buser et al. considered post-quantum VRFs based on symmetric primitives. Later, Esgin et al. [7] studied few-time verifiable Lattice-Based post-quantum secure VRFs, and a few other post-quantum secure VRFs have appeared since.

Many blockchains utilize VRF through smart contracts, including Algorand [11] and Polkadot [20]. Chainlink [3] offers one of the most popular VRF services, but their construction is not decentralized; hence, when a specific node is compromised, the secret key is known to the attacker, and in turn, the output of the VRF is completely predictable. Therefore, it is crucial and mandatory to construct decentralized VRF services.

A distributed VRF (DVRF) poses many challenges, such as communication and computational complexity. To our knowledge, Dodis [5] proposed the first DVRFs under the requirement of a trusted dealer. Using unique aggregate signatures, Kuchta and Manulis [15] proposed a generic construction of DVRFs. Hanke et al. [13] introduced a BLS-pairing DVRF. Galindo et al. [9] provided the first systematic analysis and definitions for (Non-Interactive) Fully Distributed Verifiable Random Functions (DVRFs), where they introduced DDH-DVRF (a Decisional Diffie Hellman DVRF) and GLOW-DVRF (a BLS-pairing DVRF) and provided comparisons to [13], which they named Dfinity-DVRF. The major drawback of these constructions is that the proof size of DDH-DVRF is $O(t)$, where t is the threshold number of participants. Therefore, the number of group actions needed for verification is in the same order. On the other hand, the construction of GLOW-DVRF is similar; although it has compact proofs, verification is done using costly procedures such as bilinear pairings. Recently, Kate et al. [14] introduced the notion of Output-Private DVRF, which is based on the GLOW-DVRF mechanism.

In this work, we propose a distributed verifiable random function with compact proofs (DVRFwCP) that does not require bilinear pairings and has constant proof size, independent of the threshold number of participants. Our approach alters and utilizes the NIZK proof system (see Section 2.3) by adding a t-out-of-n threshold structure and needs at least t participants to generate verifiable randomness, whereas requiring these parties to generate the NIZK proof jointly.

We first present preliminary information about verifiable secret sharing and certain VRF primitives in Section 2. In Section 3, we introduce our distributed VRF with compact proofs. In Section 4, we compare our construction to DDH-DVRF and GLOW-DVRF in terms of the required number of mathematical operations and estimated gas cost.

2 Preliminaries

We recall the underlying hard problem, the formal definition of Distributed Verifiable Random Function (DVRF), and other cryptographic primitives we utilize in our construction. We use the notation $s \overset{R}{\leftarrow} S$ to mean *the element s is chosen with uniform probability from the set S*. Let $\mathbb{Z}_q$ be the residue ring of prime order q. From hereon, let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the parties involved in the mentioned secret sharing scheme.

2.1 Lagrangian Interpolation

For a reconstruction set $\mathcal{S}$, the Lagrange basis polynomials are $\lambda_{j,\mathcal{S}}(x) = \prod_{k \in \mathcal{S}\setminus\{j\}} \frac{x-k}{j-k} \in \mathbb{Z}_q[X]$ with the Lagrange coefficients $\lambda_{i,j,\mathcal{S}} = \lambda_{j,\mathcal{S}}(0) \in \mathbb{Z}_q^*$. For any polynomial $f \in \mathbb{Z}_q[X]$ of degree at most $|\mathcal{S}|-1$ this produces a way to reconstruct $f(X) = \sum_{i\in\mathcal{S}} \lambda_{0,i,\mathcal{S}}\ f(i)$. The process of reconstructing $f(0)$ is called *Lagrangian Interpolation*. We shall drop the subscript $\mathcal{S}$ and simplify the notation to write $\lambda_{i,j} = \lambda_{i,j,\mathcal{S}}$ for these coefficients, whenever $\mathcal{S}$ is clear from the context.

2.2 Verifiable Secret Sharing

Let p and q be two primes, such that $q \mid p-1$, where the lengths of p and q are polynomial in a fixed security parameter λ. For any such pair of primes, let $\mathbb{G} \subset \mathbb{Z}_p^*$ be a subgroup of elements of order q with a generator $g \in \mathbb{G}$. An element $y \in \mathbb{G}$ can be written as $y = g^x \bmod p$ for $x \in [1, \ldots, q]$; the integer x is called *the discrete logarithm of y with respect to g*, denoted $\log_g y$. Let $\mathcal{A}$ be a probabilistic polynomial-time adversary.

Definition 1. *(DDH assumption) Let $\mathbb{G} = \langle g \rangle$ be a (cyclic) group of prime order q. Let $X \leftarrow \left(\mathbb{G}, q, g, g^{\alpha}, g^{\beta}\right)$ where $\alpha, \beta \overset{R}{\leftarrow} \mathbb{Z}_q^*$. The Decisional Diffie-Hellman* *assumption holds if for* $\gamma \xleftarrow{R} \mathbb{Z}_q^*$ *the value*

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{DDH}}(\lambda) = \left|\Pr\left[\mathcal{A}\left(X, g^{\alpha\beta}\right) = 1\right] - \Pr\left[\mathcal{A}\left(X, g^{\gamma}\right) = 1\right]\right|$$

is negligible in λ.

In [21], Shamir defined a way to share a secret $\sigma \in \mathbb{Z}_q$ among n parties using a polynomial of degree $t-1$ so that any t parties or more could recover the secret, using the following protocol.

Protocol Shamir-SS.

1. The dealer chooses a random polynomial $f(X)$ over $\mathbb{Z}_q$ of degree $t-1$, such that $f(0) = \sigma$, the secret being shared.
2. The dealer secretly transmits a share $s_i = f(i) \bmod q$ to the party P_i.

It is evident that $t-1$ or fewer parties cannot gather any information about the secret σ, whereas t or more parties can reconstruct it using Lagrange polynomial interpolation as described in Section 2.1.

However, Shamir's protocol could fail as a dealer can share values that do not lie on a polynomial of degree $t-1$. Also, dishonest parties can contribute incorrect shares to disrupt reconstruction. Various challenges, including those mentioned above, demonstrated the necessity for *Verifiable Secret Sharing* (VSS) protocols.

In [8], Feldman extended Shamir's secret-sharing scheme, enabling share recipients to verify that the shares they receive from the dealer are consistent with the secret being shared (i.e., any subset of t shares yields the same unique secret) and to rule out the erroneous shares that dishonest parties submit during reconstruction. In Feldman's approach [8], which we call Feldman-VSS, the secret is merely *computationally secure*, i.e., the verification values are leaked; nevertheless, it can be shown that an adversary that holds $t-1$ or fewer shares cannot obtain any information on the secret except what can be learned from the public values.

Protocol Feldman-VSS.

1. The dealer chooses a random polynomial $f(X) = \sum_{k=0}^{t-1} a_k X^k$ over $\mathbb{Z}_q$ of degree $t-1$, such that $f(0) = \sigma$, the secret being shared.
2. The dealer broadcasts verification values $A_k = g^{a_k} \bmod p$ for $k = 0, \ldots, t-1$.
3. The dealer secretly transmits a share $s_i = f(i) \bmod q$ to the party P_i.
4. The parties can verify that the shares s_i define a secret by checking
$$g^{s_i} \stackrel{?}{=} \prod_{k=0}^{t-1} (A_k)^{i^k} \bmod p\ . \tag{1}$$
5. (a) Each party P_i verifies (1) for their share s_i

(b) If verification fails, P_i broadcasts a *complaint* against the dealer.
(c) For each complaining party P_i, the dealer reveals the share s_i matching the equation (1).
(d) If any of the revealed shares fails (1), the dealer is disqualified.
(e) At the time of reconstruction of the secret σ, the same equation is used to verify the shares submitted by the parties so that dishonest parties may be identified. If at least t honest parties exist, the secret may be reconstructed using lagrangian interpolation.

Pedersen's Verifiable Secret Sharing (Pedersen-VSS) [19], in comparison to Feldman-VSS, offers perfect (information-theoretic) security for the shared secret, which means that the adversary can merely view values unrelated to the secret being shared under the assumption that any adversary, especially the dealer, cannot solve the discrete logarithm problem.

In addition to the notation of Feldman's VSS, this scheme requires an additional element $\bar{g} \in \mathbb{G}$ where it is assumed that $\log_g(\bar{g})$ is hard to compute. To share a secret σ to a group of participants $\mathcal{P} = \{P_1, \ldots, P_n\}$, Pedersen-VSS works as follows.

Protocol Pedersen-VSS.

1. The dealer generates two random polynomials $f(X) = \sum_{k=0}^{t-1} a_k X^k$ and $\bar{f}(X) = \sum_{k=0}^{t-1} b_k X^k$ over $\mathbb{Z}_q$, both of degree $t-1$, such that $f(0) = \sigma$.
2. The dealer publishes $C_k = g^{a_k} \bar{g}^{b_k} \bmod p$ for $k = 0, \ldots, t-1$
3. The dealer transmits $(s_i, \bar{s}_i) = \big(f(i), \bar{f}(i)\big)$ to P_i, for $i = 1, \ldots, n$.
4. (a) Each party $P_i \in \mathcal{P}$ checks

$$g^{s_i} \bar{g}^{\bar{s}_i} \stackrel{?}{=} \prod_{k=0}^{t-1} (C_k)^{i^k} \bmod p \,. \tag{2}$$

(b) If check fails, P_i complains against the dealer.
(c) For each complaining party P_i, the dealer reveals $(s_i, \bar{s}_i)$ matching (2).
(d) If any of the revealed shares fails (2), the dealer is disqualified.

In [19], it was shown that if the dealer is not disqualified, all honest parties can interpolate a unique polynomial of degree $t-1$, and any t of these honest parties can obtain the secret efficiently and where each share can be verified publicly.

Based on the Feldman-VSS protocol, in [18], Pedersen proposed the first distributed key generation protocol, which executes n parallel Feldman-VSS protocols for the parties $P_1, \ldots, P_n$. In homage to [10], we use the name Joint Feldman Distributed Key Generation (JF-DKG) for a simplified version of Pedersen's DKG protocol.

Protocol JF-DKG.

1. Acting as a dealer, each party $P_i \in \mathcal{P}$
   (a) chooses a random polynomial $f_i(X) = a_{i,0} + a_{i,1}X + \cdots + a_{i,t-1}X^{t-1}$ over $\mathbb{Z}_q$ of degree $t-1$, where $a_{i,0} = z_i$ is the secret to be shared,
   (b) broadcasts $A_{i,k} = g^{a_{i,k}} \bmod p$ for $k = 0, \ldots, t-1$,
   (c) computes the shares $s_{i,j} = f_i(j) \bmod q$ for $j = 1, \ldots, n$ and sends $s_{i,j}$ secretly to party P_j.
2. Acting as a receiver, each $P_j \in \mathcal{P}$ verifies the shares they received from the other parties by checking for $i = 1, \ldots, n$:

$$g^{s_{i,j}} \overset{?}{=} \prod_{k=0}^{t-1} (A_{i,k})^{j^k} \bmod p \tag{3}$$

   If the check fails for an index i, P_j broadcasts a complaint against P_i.
3. For every complaint against P_i, acting as a dealer, for each complaining party P_j, P_i reveals the share $s_{i,j}$ matching (3). If any revealed shares fail this equation, P_i is disqualified. We define $\mathcal{Q}$ as the set of non-disqualified parties.
4. The public verification values are computed as $A_k = \prod_{i\in\mathcal{Q}} A_{i,k} \bmod p$ for $k = 0, \ldots, t-1$. Each party P_j sets their share of the secret as $x_j = \sum_{i\in\mathcal{Q}} s_{i,j} \bmod q$. The secret shared value, which is (usually) not computed by any party, is $x = \sum_{i\in\mathcal{Q}} z_i \bmod q = \sum_{i\in\mathcal{Q}} a_{i,0} \bmod q$. Finally, the public value y is computed as $y = \prod_{i\in\mathcal{Q}} y_i \bmod p = \prod_{i\in\mathcal{Q}} A_{i,0} \bmod p$.

It was shown that during a run of the protocol JF-DKG, an attacker could affect the distribution of the pair (x, y), resulting in a non-uniform distribution, eventually proving that JF-DKG cannot be used as a generic secure DKG protocol. Consequently, Gennaro et al. [10] proposed the following protocol, which is utilized in most distributed systems, including the one we propose in this paper.

Protocol Secure-DKG.
Generating x :

1. Each party $P_i \in \mathcal{P}$ performs a Pedersen-VSS of a random value z_i as a dealer:
   (a) P_i chooses two random secret polynomials $f_i(X), \bar{f}_i(X)$ over $\mathbb{Z}_q$ of degree $t-1$:

$$\begin{aligned} f_i(X) &= a_{i,0} + a_{i,1}X + \cdots + a_{i,t-1}X^{t-1}, \\ \bar{f}_i(X) &= b_{i,0} + b_{i,1}X + \cdots + b_{i,t-1}X^{t-1}, \end{aligned}$$

   where $a_{i,0} = z_i = f_i(0)$.
   (b) P_i broadcasts $C_{i,k} = g^{a_{i,k}} \bar{g}^{b_{i,k}} \bmod p$ for $k = 0, \ldots, t-1$.
   (c) P_i computes the shares $s_{i,j} = f_i(j)$ and $\bar{s}_{i,j} = \bar{f}_i(j) \bmod q$, and sends $(s_{i,j}, \bar{s}_{i,j})$ to each party $P_j \in \mathcal{P}$.

(d) Each party $P_j \in \mathcal{P}$ verifies the shares they received from the other parties, i.e., for each $P_i \in \mathcal{P}$, P_j checks whether

$$g^{s_{i,j}} \; \bar{g}^{\bar{s}_{i,j}} \stackrel{?}{=} \prod_{k=0}^{t-1} (C_{i,k})^{j^k} \bmod p \tag{4}$$

If the check fails for any index i, P_j broadcasts a *complaint* against P_i.

(e) Each party P_i, acting as a dealer, who received a complaint from a party P_j, broadcasts the values $(s_{ij}, \bar{s}_{ij})$ that satisfy 4.

(f) Each party P_j marks P_i as *disqualified* if either
   - P_i received more than $t-1$ complaints in Step 1(d), or
   - P_i answered a complaint in Step 1(e) with values that do not satisfy 4.

2. Each party builds the set of non-disqualified parties $\mathcal{Q}$, where it was shown in [10] that all honest parties build the same $\mathcal{Q}$.
3. The distributed secret value x is (usually) not explicitly computed by any party, but it is

$$x = \sum_{i \in \mathcal{Q}} z_i \bmod q \ . \tag{5}$$

Each party P_i sets their share of the secret as $x_i = \sum_{j \in \mathcal{Q}} s_{j,i} \bmod q$ and $\bar{x}_i = \sum_{j \in \mathcal{Q}} \bar{s}_{j,i} \bmod q$.

Extracting $y = g^x \bmod p$:

1. Each party $P_i \in \mathcal{Q}$ publishes $y_i = g^{z_i} \bmod p$ via Feldman-VSS as follows.

(a) P_i broadcasts $A_{i,k} = g^{a_{i,k}} \bmod p$ for $k = 0, \ldots, t-1$.

(b) Each party $P_j \in \mathcal{Q}$ verifies the values publicised by the other parties in $\mathcal{Q}$, i.e, for each $P_i \in \mathcal{Q}$, P_j checks if

$$g^{s_{i,j}} \stackrel{?}{=} \prod_{k=0}^{t-1} (A_{i,k})^{j^k} \bmod p \ . \tag{6}$$

If the check fails for a party member P_i, P_j broadcasts the values $(s_{ij}, \bar{s}_{ij})$ that satisfy 4 but do not satisfy 6, thereby broadcasts a *complaint* against P_i.

(c) For parties P_i who receive at least one <u>valid</u> complaint, i.e., values which satisfy 4 and not 6, the remaining parties run the reconstruction phase of Pedersen-VSS to compute $z_i, f_i(X)$, and $A_{i,k}$ for $k = 0, \ldots, t-1$. For all parties in $\mathcal{Q}$, let $y_i = A_{i,0} = g^{z_i} \bmod p$ and compute y as

$$y = \prod_{i \in \mathcal{Q}} y_i \bmod p \ .$$

We now present a new augmented version of the Secure-DKG protocol, which will be used in our construction.

Protocol Augmented Secure-DKG. This protocol extends the Secure-DKG protocol above to share commitments to the shared secret x relative to an auxiliary element $h \in \mathbb{G}$, where it is assumed that $log_g(h)$ is hard to compute. This is done to reduce the number of interactions in our proposed algorithm. The generation phase of the shared secret x is the same, whereas the modifications to the extraction phase are emphasized in blue.

Extracting $y = g^x \bmod p$ **:**

1. Each party $P_i \in \mathcal{Q}$ publishes $y_i = g^{z_i} \bmod p$ via Feldman-VSS as follows.
   (a) P_i broadcasts $A_{i,k} = g^{a_{i,k}} \bmod p$ and $B_{i,k} = h^{a_{i,k}} \bmod p$ for $k = 0, \ldots, t-1$.
   (b) Each party $P_j \in \mathcal{Q}$ verifies the values publicised by the other parties in $\mathcal{Q}$, i.e, for each $P_i \in \mathcal{Q}$, P_j checks if

$$g^{s_{i,j}} \overset{?}{=} \prod_{k=0}^{t-1} (A_{i,k})^{j^k} \bmod p \; . \tag{7}$$

$$h^{s_{i,j}} \overset{?}{=} \prod_{k=0}^{t-1} (B_{i,k})^{j^k} \bmod p \; . \tag{8}$$

   If the checks fail for a party member P_i, P_j broadcasts the values $(s_{ij}, \bar{s}_{ij})$ that satisfy 4 but do not satisfy 6, thereby broadcasts a *complaint* against P_i.
   (c) For parties P_i who receive at least one valid complaint, i.e., values which satisfy 4 and not 6, the remaining parties run the reconstruction phase of Pedersen-VSS to compute $z_i, f_i(X)$, and $A_{i,k}$ for $k = 0, \ldots, t-1$. For all parties in $\mathcal{Q}$, compute

$$g^x = \prod_{i \in \mathcal{Q}} A_{i,0} \bmod p$$

   and

$$h^x = \prod_{i \in \mathcal{Q}} B_{i,0} \bmod p \; .$$

2.3 NIZK Proof for Equality of Discrete Logarithms

The DDH-DVRF construction uses Non-Interactive Zero Knowledge (NIZK) proof of exponents by Chaum and Pedersen [4]. The Equality of Discrete Logarithms proof system $(\text{ProveEq}_H, \text{VerifyEq}_H)$ to show $k = \log_g x = \log_h y$ is as follows. Consider a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$.

– $\text{ProveEq}_H(g, h, x, y; k)$ chooses $r \overset{R}{\leftarrow} \mathbb{Z}_q$, computes $u \leftarrow g^r$, $v \leftarrow h^r$ and sets $c \leftarrow H(g, h, x, y, u, v)$. The output is a non-interactive proof $\pi = (c, s)$, where $s = r - k \cdot c$.
– $\text{VerifyEq}_H(g, h, x, y, \pi)$ parses $\pi = (c, s)$, computes $\bar{u} \leftarrow g^s x^c$ and $\bar{v} \leftarrow h^s y^c$ and outputs $c \overset{?}{=} H(g, h, x, y, \bar{u}, \bar{v})$.

2.4 Distributed Verifiable Random Functions

In 2020, Galindo et al. [9] introduced DDH-DVRF, a DDH-based DVRF with non-compact proofs, and GLOW-DVRF, which is built on top of DDH-DVRF with aggregate proofs using pairings.

DDH-DVRF. Let $(\mathbb{G}, g, q)$ be a multiplicative group where the DDH assumption holds. Let $H_1 : \{0,1\}^* \to \mathbb{G}$ and $H_2 : \{0,1\}^* \to \mathbb{Z}_q$ be two hash functions. Let $\mathcal{V}^{\mathrm{DDH-DVRF}} = (\text{DistKeyGen, PartialEval, Combine, Verify})$, where:
DistKeyGen $\left(1^\lambda, t, n\right)$ is run by n participating nodes $\mathcal{P} = \{P_1, \ldots, P_n\}$. Each node P_i chooses a random polynomial $f_i(X) = a_{i,0} + a_{i,1}X + \cdots + a_{i,t}X^{t-1}$. The protocol outputs a set of qualified nodes $\mathrm{QUAL} \subseteq \mathcal{P}$, a secret key $\mathrm{sk}_i = \sum_{j \in \mathrm{QUAL}} f_j(i) \in \mathbb{Z}_q$ and a verification key $\mathrm{vk}_i = g^{\mathrm{sk}_i} \in \mathbb{G}$ for each $i \in \mathrm{QUAL}$, an implicit distributed secret value $\mathrm{sk} = \sum_{i \in \mathrm{QUAL}} a_{i,0} \in \mathbb{Z}_q$, and a global public key $\mathrm{pk} = \prod_{i \in \mathrm{QUAL}} g^{a_{i,0}}$.
PartialEval $(\alpha, \mathrm{sk}_i, \mathrm{vk}_i)$ outputs $s_\alpha^i = (i, \gamma_i, \pi_i)$ for a plaintext α, where $\gamma_i \leftarrow H_1(\alpha)^{\mathrm{sk}_i}$ and $\pi_i \leftarrow \mathrm{ProveEq}_{H_2}\left(g, \mathrm{vk}_i, H_1(\alpha),\ \gamma_i; \mathrm{sk}_i\right)$.
Combine$(\mathrm{pk}, \mathcal{VK}, \alpha, \mathcal{S})$ parses list $\mathcal{S} = \left\{s_\alpha^{j_1}, \ldots, s_\alpha^{j_{|\mathcal{S}|}}\right\}$ of $|\mathcal{S}| \geq t$ partial evaluation candidates originating from $|\mathcal{S}|$ different nodes, and obtains verification keys $\mathrm{vk}_{j_1}, \ldots, \mathrm{vk}_{j_{|\mathcal{S}|}}$. Next,

1. Identifies an index subset $I = \{i_1, \ldots, i_t\}$ such that $\mathrm{VerifyEq}_{\mathrm{H}_2}\left(g, \mathrm{vk}_i, H_1(\alpha), \gamma_i, \pi_i\right) = 1$ holds for every $i \in I$, where $(i, \gamma_i, \pi_i) \leftarrow s_\alpha^i$. If no such subset exists, aborts.
2. Sets $\gamma \leftarrow \prod_{i \in I} \gamma_i^{\lambda_{0,i,I}}$ and $\pi \leftarrow \left\{s_\alpha^i\right\}_{i \in I}$
3. Outputs (γ, π).

Verify$(\mathrm{pk}, \mathcal{VK}, \alpha, \gamma, \pi)$ parses $\pi = \left\{s_\alpha^i\right\}_{i \in I}$ such that $|I| = t$ and $I \subseteq \mathrm{QUAL}$

1. Parses $s_\alpha^i = (i, \gamma_i, \pi_i)$ for $i \in I$.
2. Checks if $\mathrm{VerifyEq}_{\mathrm{H}_2}\left(g, \mathrm{vk}_i, H_1(\alpha), \gamma_i, \pi_i\right) \overset{?}{=} 1$ for every $i \in I$; if some of the checks fail, outputs 0.
3. Checks if $\gamma \overset{?}{=} \prod_{i \in I} \gamma_i^{\lambda_{0,i,I}}$; if so outputs 1 ; otherwise outputs 0 .

GLOW-DVRF. Let $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, h_1, h_2)$ be a bilinear paring where g_1, h_1 and g_2, h_2 are generators of $\mathbb{G}_1, \mathbb{G}_2$, respectively. GLOW-DVRF is a pairing-based DVRF similar to DDH-DVRF that achieves compact proofs as elements of $\mathbb{G}_1$ that are validated using pairing equations in the Verify algorithm. We refer the reader to [9] for details.

3 DVRFwCP: Distributed Verifiable Random Function with Compact Proof

Since we are working in a distributed environment, we assume that the participants communicate with each other in a secure and authenticated channel.

Definition 2. *(Distributed Verifiable Random Function with Compact Proof) A t-out-of-n Distributed Verifiable Random Function with Compact Proof (DVRFwCP)* $\boldsymbol{V}$ *consists of the six algorithms, i.e.,* $\boldsymbol{V} = ($***DistKeyGen***, ***PartialEval***, ***EvalCombine***, ***PartialProofGen***, ***ProofCombine***, ***Verify***$)$*, described below.*

- ***DistKeyGen*** $\left(1^\lambda, t, n\right)$*: On input of a security parameter* λ*, the number of participating nodes* n*, and the threshold parameter* t *as inputs, and outputs a public key* pk*, a list of partial secret keys* $\mathcal{SK} = (x_1, \ldots, x_n)$*, and a list of partial verification keys* $\mathcal{VK} = (y_1, \ldots, y_n)$*.*
- ***PartialEval***(α, x_i, y_i)*: On input of a plaintext* α*, a partial secret and verification keys* (x_i, y_i) *as inputs, and outputs a partial evaluation* γ_i *and a partial evaluation NIZK proof* π_i*.*
- ***EvalCombine***$(\mathcal{VK}, \alpha, \mathcal{S})$*: On input of the list of verification keys* $\mathcal{VK}$*, the plaintext* α *and a list* $\mathcal{S} = \left\{s_{i_1}, \ldots, s_{i_{|\mathcal{S}|}}\right\}$ *of partial evaluation proofs from at least* t *nodes, outputs either a combined evaluation* γ *or aborts.*
- ***PartialProofGen***(α, x_i, y_i)*: On input of a plaintext* α*, a partial secret and verification key* (x_i, y_i) *as inputs and outputs a partial NIZK proof* $\Pi_i = (c, s_i)$*.*
- ***ProofCombine***$(\mathcal{VK}, \alpha, \mathcal{S})$*: On input of the list of verification keys* $\mathcal{VK}$*, the plaintext* α *and a list* $\mathcal{S} = \left\{\Pi_{i_1}, \ldots, \Pi_{i_{|\mathcal{S}|}}\right\}$ *of partial proofs from at least* t *nodes, outputs either a combined NIZK proof* $\Pi = (\gamma, c, \mathbf{s})$ *or aborts.*
- ***Verify***(pk, α, Π)*: On input of the public key* pk*, the plaintext* α*, a NIZK proof* Π *as inputs, it outputs* $0/1$*.*

A DVRFwCP needs to satisfy the following requirements.

- **Correctness:** This requires that for a correctly generated VRF value γ, Verify always outputs 1.
- **Uniqueness:** This requires that for every plaintext α a unique value γ passes the verification test. It is infeasible for an adversary to calculate two different output values γ_1 and γ_2 where both values pass the verification test with respect to α, even if the secret keys of the honest parties are compromised.
- **Consistency:** This means that, for a given plaintext α, successful execution of the protocol should produce the same VRF output γ, regardless of the collection of correctly formed shares used.
- **Robustness:** This guarantees the availability of computing the random function value on any plaintext in the presence of an active adversary, i.e., if Combine does not abort, its output passes the verification, even if the adversary inputs malformed values to the combination.
- **Strong pseudorandomness:** This guarantees that even if at most $t-1$ participants are malicious, on input of the plaintext α, they are not able to distinguish between an honest output y of the DVRF protocol from a uniformly random value in the output range; this holds even when the malicious parties are involved in multiple executions on inputs $\alpha_1, \alpha_2 \ldots$ that are not α. It should be noted that, through these executions, the malicious parties learn not only the DVRF outputs y_i on inputs of α_i but also the partial evaluations.

Our construction aims to aggregate the individual NIZK proofs used in DDH-DVRF (see Section 2.4) [9] in a way that it can be verified at once. To that end, we modified the NIZK proof system (see Section 2.3) by adding a *t-out-of-n* threshold structure over it. In other words, our approach needs at least t participants to generate a verifiable random value, and it also needs the same parties to generate the NIZK proof jointly.

We build a DVRF with compact proofs using the notation and setup of DDH-DVRF 2.4, also assume that we have a set $\mathcal{P}$ of l participants, i.e., $\mathcal{P} = \{P_1, \dots, P_l\}$. Additionally, assume that $H_3 : \{0,1\}^* \to \{0,1\}^{b(\lambda)}$ is another hash function. Let $\mathcal{V}^{\text{DVRFwCP}}$ be defined via the following six algorithms.

DistKeyGen: On input of the public parameters *pp*, the participants $P_1, \dots, P_l$ jointly run the protocol Secure-DKG as in 2.2 to obtain their partial secret keys $x_1, \dots, x_n \in \mathbb{Z}_q$ of the shared secret $\mathbf{x} \in \mathbb{Z}_q$, and the group public key

$$y = \prod_{i \in \mathcal{S}} (g^{x_i})^{\lambda_i} = g^{\sum\limits_{i \in \mathcal{S}} x_i \lambda_i} = g^{\mathbf{x}}$$

where $\mathcal{Q}$ is the set of n qualified participants, $\mathcal{S} \subset \mathcal{Q}$ is a subset of at least t qualified participants, and λ_i's are the appropriate Lagrange coefficients.

PartialEval: Each $P_i \in \mathcal{Q}$ performs the following actions.

1. Compute $h \leftarrow H_1(\alpha) \in \mathbb{G} - \{1\}$.
2. Compute partial evaluation $\gamma_i = h^{x_i} \in \mathbb{G}$.
3. Compute partial evaluation proof $\pi_i \leftarrow \text{ProveEq}_{H_2}(g, h, \mathrm{g}^{x_i}, \gamma_i; \mathrm{x}_i)$.
4. Broadcast (γ_i, π_i).

EvalCombine: Each $P_i \in \mathcal{Q}$ performs the following actions.

1. Upon receiving (γ_j, π_j) from P_j, check $\text{VerifyEq}_{\mathrm{H}_2}(g, h, \mathrm{g}^{x_i}, \gamma_i, \pi_i) \overset{?}{=} 1$.
2. If the checks do not pass for at least t partial proofs, abort. Otherwise, define $\mathcal{I}$ as the set of indices of the participants whose partial evaluation proofs are valid.
3. Perform Lagrange interpolation to reconstruct $\gamma = h^{\mathbf{x}} = \prod\limits_{j \in \mathcal{I}} \gamma_j^{\lambda_j}$, where λ_j is the appropriate Lagrange coefficient with respect to P_j and $\mathcal{I}$ is the set of indices of at least t received valid γ_j's.

PartialProofGen: Each $P_i \in \mathcal{Q}$ performs the following actions.

1. Participate in the Augmented Secure-DKG protocol from 2.2 amongst the members of $\mathcal{Q}$ using the elements $g, h \in \mathbb{G}$ to generate a shared secret $\mathbf{k}$. At the end of this step, each P_i obtains a partial ephemeral secret k_i, public values $g^{\mathbf{k}}$ and $h^{\mathbf{k}}$.
2. Compute the joint challenge, i.e., $c \leftarrow H_2(g, h, y, \gamma, g^{\mathbf{k}}, h^{\mathbf{k}})$.
3. After obtaining the joint challenge c, compute $s_i \leftarrow k_i - cx_i \mod q$,
4. Broadcast the partial proof $\Pi_i = (c, s_i)$.

ProofCombine:

1. Upon receiving partial proof $\Pi_j = (c, s_j)$ from P_j, P_i verifies them by checking whether $g^{k_j} \overset{?}{=} y_j^c \cdot g^{s_j}$ and $h^{k_j} \overset{?}{=} \gamma_j^c \cdot h^{s_j}$.
2. If the checks do not pass for at least t partial proofs, abort. Otherwise, define $\mathcal{S}$ as the set of indices of the participants whose partial proofs are valid.
3. Perform Lagrange interpolation to obtain the aggregated proof, i.e., $\mathbf{s} = \sum\limits_{i \in \mathcal{S}} s_i \lambda_i \pmod q$, where λ_i's are the appropriate Lagrange coefficients.
4. Output the VRF value $\beta \leftarrow H_3(\gamma)$ and the proof as $\Pi = (\gamma, c, \mathbf{s})$.

Verify: Given the group public key y, the proof $\Pi = (\gamma, c, \mathbf{s})$, the VRF value β and the plaintext α, perform verification as follows:

1. Compute $u = y^c \cdot g^{\mathbf{s}}$.
2. Given α, obtain $h = H_1(\alpha)$, compute $v = \gamma^c \cdot h^{\mathbf{s}}$.
3. Check if $c \overset{?}{=} H_2(g, h, y, \gamma, u, v)$ and $\beta \overset{?}{=} H_3(\gamma)$. If both are satisfied, output 1; otherwise, output 0.

Remark 1. In the first step of the Combine algorithm, each P_i verifies the partial proof s_j to ensure that partial proofs to be aggregated are valid. Notice that the values g^{k_j}, h^{k_j} and γ_j's are known to P_i from the third step of the PartialEval algorithm.

Security

DVRFwCP satisfies the correctness property because of the following.

$$\begin{aligned} u &= y^c \cdot g^{\mathbf{s}} \\ &= (g^{\mathbf{x}})^c \cdot g^{\sum_{i \in \mathcal{S}} s_i \lambda_i} \\ &= g^{c\mathbf{x}} \cdot g^{\sum_{i \in \mathcal{S}} (k_i - c x_i)\lambda_i} \\ &= g^{c\mathbf{x}} \cdot g^{\sum_{i \in \mathcal{S}} (k_i \lambda_i - c x_i \lambda_i)} \\ &= g^{c\mathbf{x}} \cdot g^{\sum_{i \in \mathcal{S}} k_i \lambda_i - c \sum_{i \in \mathcal{S}} x_i \lambda_i} \\ &= g^{c\mathbf{x}} \cdot g^{\mathbf{k} - c\mathbf{x}} \\ &= g^{\mathbf{k}} \end{aligned} \tag{9}$$

and

$$\begin{aligned} v &= \gamma^c \cdot h^{\mathbf{s}} \\ &= (h^{\mathbf{x}})^c \cdot h^{\sum\limits_{i \in \mathcal{S}} s_i \lambda_i} \\ &= h^{c\mathbf{x}} \cdot h^{\sum\limits_{i \in \mathcal{S}} (k_i - c x_i) \lambda_i} \\ &= h^{c\mathbf{x}} \cdot h^{\sum\limits_{i \in \mathcal{S}} (k_i \lambda_i - c x_i \lambda_i)} \\ &= h^{c\mathbf{x}} \cdot h^{\sum\limits_{i \in \mathcal{S}} k_i \lambda_i - c \sum\limits_{i \in \mathcal{S}} x_i \lambda_i} \\ &= h^{c\mathbf{x}} \cdot h^{\mathbf{k} - c\mathbf{x}} \\ &= h^{\mathbf{k}} \end{aligned} \tag{10}$$

and from (9) and (10), we have

$$c = H_2 (g, h, y, \gamma, u, v) = H_2 \left(g, h, y, \gamma, g^{\mathbf{k}}, h^{\mathbf{k}}\right) \ .$$

The properties of Secure-DKG [10] immediately imply consistency. Uniqueness follows from the fact that the output is consistent and the hash functions are based on random oracles. We argue that the DVRFwCP satisfies robustness and strong pseudorandomness, as the proposed algorithm is a modified version of the DDH-DVRF proposed in [9]; therefore, the security is reduced to the security of DDH-DVRF if the utilized distributed key generation algorithm is secure.

4 Comparison

In Table 1, we compare our construction with the DVRF schemes proposed in [9]. This comparison includes the number of operations required in each phase of the schemes, the number of rounds of interaction, and the proof size. In Table 2, we give a rough gas cost estimation for the verification and the proof size of the schemes compared in Table 1, assuming the EIP-2537: Precompile for BLS12-381 curve operations [22].

In the DistKeyGen phase, our scheme requires the same number of operations as in DDH-DVRF, which requires less than GLOW-DVRF.

In the PartialEval and EvalCombine phases, our scheme requires the same number of operations as in DDH-DVRF and GLOW-DVRF.

Our scheme has two additional algorithms, i.e., ParitalProofGen and ProofCombine. All the operations in these phases can be counted extra operations which do not exist in the other schemes in comparison. However, for example, in blockchains, these parts are handled off-chain; hence, no gas cost is accrued.

In the Verify phase, DVRFwCP requires fewer operations. We want to emphasize that a noteworthy achievement of our construction is that in the Verify phase, our algorithm does not use bilinear pairings, which are known to be slow; also, DVRFwCP has constant time verification as opposed to the linear time verification in the quantity of the threshold of DDH-DVRF. This is an important improvement, especially in the context of blockchains. For example, when a smart contract acts as a public verifier, each node should run the same code for verification. Since our construction requires a constant number of group operations, the gas cost required by the verifier smart contract would be much lower than the verification gas cost required by DDH-DVRF and GLOW-DVRF, as they require either much more expensive operations like bilinear pairings or $O(t)$ group operations, where t is the threshold number of participants, as seen in Table 2.

Another crucial achievement of our proposed scheme is that the proof size is constant (as in GLOW-DVRF), i.e., it consists of one group element and two $\mathbb{Z}_q$ elements. This is also a very important improvement, especially for the blockchain use cases. Note that, other than the computational complexity of the verification algorithm, the size of the data (call data) that needs to be sent to the verifier smart contract also increases the gas cost. For example, in Ethereum, one should pay 16 gas per Byte of call data in addition to the verification cost of a VRF proof [1].

On the other hand, our construction requires one more round of interactions. For more details, one can see Table 1 and Table 2 below.

5 Conclusion

Verifiable randomness is a fundamental feature of blockchains and Web3 applications. Because of the increasing demand for this resource, fresh challenges are emerging. In this paper, we propose a distributed verifiable random function with compact proof, DVRFwCP, which is more efficient in terms of verification and proof size but less efficient in terms of proof generation.

We conclude that, due to its constant proof size and efficient verification algorithm, our construction may be a good candidate for distributed systems, especially solutions that use smart contracts as verifiers.

References

1. Akhunov, A., Ben Sasson, E., Brand, T., Guthmann, L., Levy, A.: "eip-2028: Transaction data gas cost reduction," ethereum improvement proposals, no. 2028 (May 2019), https://eips.ethereum.org/EIPS/eip-2028, last accessed 01/07/2024
2. Buser, M., Dowsley, R., Esgin, M.F., Kasra Kermanshahi, S., Kuchta, V., Liu, J.K., Phan, R.C.W., Zhang, Z.: Post-quantum verifiable random function from symmetric primitives in pos blockchain. In: European Symposium on Research in Computer Security. pp. 25–45. Springer (2022). https://doi.org/10.1007/978-3-031-17140-6_2
3. Chainlink: "chainlink vrf" (June 2024), https://docs.chain.link/vrf, last accessed 01/07/2024
4. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Annual international cryptology conference. pp. 89–105. Springer (1992). https://doi.org/10.1007/3-540-48071-4_7

Table 1. Comparison with DVRF schemes from [9]

	DDH-DVRF	GLOW-DVRF	This work
DistKeyGen	2 poly. in $\mathbb{Z}_q$ $4t+3$ exp. in $\mathbb{G}$ $n+3t-2$ mul. in $\mathbb{G}$ $n-1$ add. in $\mathbb{Z}_q$	2 poly. in $\mathbb{Z}_q$ $4t+3$ exp. in $\mathbb{G}$ $n+3t-2$ mul. in $\mathbb{G}$ $n-1$ add. in $\mathbb{Z}_q$ $2n$ pairings	2 poly. in $\mathbb{Z}_q$ $4t+3$ exp. in $\mathbb{G}$ $n+3t-2$ mul. in $\mathbb{G}$ $n-1$ add. in $\mathbb{Z}_q$
PartialEval	1 hash to $\mathbb{G}$ 3 exp. in $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$ 1 add. in $\mathbb{Z}_q$ 1 mult. in $\mathbb{Z}_q$	1 hash to $\mathbb{G}_1$ 3 exp. in $\mathbb{G}_1$ 1 hash to $\mathbb{Z}_q$ 1 add. in $\mathbb{Z}_q$ 1 mul. in $\mathbb{Z}_q$	1 hash to $\mathbb{G}$ 3 exp. in $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$ 1 mul. in $\mathbb{Z}_q$ 1 add. in $\mathbb{Z}_q$
EvalCombine	$5t$ exp. in $\mathbb{G}$ $3t-1$ mul. in $\mathbb{G}$ t hashes to $\mathbb{Z}_q$	$5t$ exp. in $\mathbb{G}_1$ $3t-1$ mul. in $\mathbb{G}_1$ t hashes to $\{0,1\}^{b(\lambda)}$	$5t$ exp. in $\mathbb{G}$ $3t-1$ mul. in $\mathbb{G}$ 1 Hash to $\mathbb{Z}_q$
PartialProofGen$^{(b)}$	-	-	2 poly. in $\mathbb{Z}_q$ $6t+4$ exp. in $\mathbb{G}$ $2n+4t-2$ mul. in $\mathbb{G}$ n add. in $\mathbb{Z}_q$ 1 mul. in $\mathbb{Z}_q$ 1 hash to $\mathbb{Z}_q$
ProofCombine$^{(b)}$	-	-	$6t$ exp. in $\mathbb{G}$ $2t$ mul. in $\mathbb{G}$ $t-1$ add. in $\mathbb{Z}_q$ t mul. in $\mathbb{Z}_q$ 1 Hash to $\{0,1\}^{b(\lambda)}$
Verify	$5t$ exp. in $\mathbb{G}$ $3t-1$ mul. in $\mathbb{G}$ t hashes to $\mathbb{Z}_q$	2 pairings 1 hash to $\mathbb{G}_1$ 1 hash to $\{0,1\}^{b(\lambda)}$	4 exp. in $\mathbb{G}$ 2 mul. in $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$ 1 hash to $\{0,1\}^{b(\lambda)}$
Interactions$^{(a)}$	1 round	1 round	2 rounds
Proof size	t $\mathbb{G}$ elements	1 $\mathbb{G}_1$ element 1 $b(\lambda)$-bit element	1 $\mathbb{G}$ element 2 $\mathbb{Z}_q$ elements

(a) We use the following abbreviations: "poly." for "polynomial evaluations", "exp" for "exponentiations", "mul." for "multiplications", and "add." for "additions". (b) These algorithms do not exist in the corresponding DVRFs. (c) Note that we do not take DistKeyGen into account as it is a one-time interaction.

Table 2. Gas cost estimation comparison for the BLS12-381 curve

Gas cost	DDH-DVRF	GLOW-DVRF	DVRFwCP
verification cost	$\sim 70,000t$	$\sim 200,000$	$\sim 50,000$
calldata cost	$768t$	$1,280$	$1,792$
Total cost	$\sim 70,768t$	$\sim 201,280$	$\sim 51,792$

(a) We assume that the subgroup $\mathbb{G}_1$ of BLS12-381 curve is used as the cyclic group for both DDH-DVRF and DVRFwCP. (b) We estimate the gas cost only for pairings, exponentiation, and multiplications in $\mathbb{G} = \mathbb{G}_1$. (c) We take $|\mathbb{G}_1| = 48$ bytes and $|\mathbb{Z}_q| = 32$ bytes.

5. Dodis, Y.: Efficient construction of (distributed) verifiable random functions. In: Public Key Cryptography—PKC 2003: 6th International Workshop on Practice and Theory in Public Key Cryptography Miami, FL, USA, January 6–8, 2003 Proceedings 6. pp. 1–17. Springer (2002). https://doi.org/10.1007/3-540-36288-6_1
6. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: International Workshop on Public Key Cryptography. pp. 416–431. Springer (2005). https://doi.org/10.1007/978-3-540-30580-4_28
7. Esgin, M.F., Kuchta, V., Sakzad, A., Steinfeld, R., Zhang, Z., Sun, S., Chu, S.: Practical post-quantum few-time verifiable random function with applications to algorand. In: Borisov, N., Diaz, C. (eds.) Financial Cryptography and Data Security. pp. 560–578. Springer Berlin Heidelberg, Berlin, Heidelberg (2021). https://doi.org/10.1007/978-3-662-64331-0_29
8. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: 28th Annual Symposium on Foundations of Computer Science (sfcs 1987). pp. 427–438 (Oct 1987). https://doi.org/10.1109/SFCS.1987.4
9. Galindo, D., Liu, J., Ordean, M., Wong, J.M.: Fully distributed verifiable random functions and their application to decentralised random beacons. In: 2021 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 88–102 (2021). https://doi.org/10.1109/EuroSP51992.2021.00017
10. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. Journal of Cryptology **20**, 51–83 (05 2007). https://doi.org/10.1007/s00145-006-0347-3
11. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles. p. 51–68. SOSP '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3132747.3132757, https://doi.org/10.1145/3132747.3132757
12. Goldberg, S., Reyzin, L., Papadopoulos, D., Včelák, J.: Verifiable Random Functions (VRFs). RFC 9381 (Aug 2023). https://doi.org/10.17487/RFC9381, https://www.rfc-editor.org/info/rfc9381
13. Hanke, T., Movahedi, M., Williams, D.: Dfinity technology overview series, consensus system (2018). https://doi.org/10.48550/arXiv.1805.04548, https://arxiv.org/abs/1805.04548
14. Kate, A., Mangipudi, E.V., Maradana, S., Mukherjee, P.: Flexirand: Output private (distributed) vrfs and application to blockchains. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 1776–1790 (2023). https://doi.org/10.1145/3576915.3616601
15. Kuchta, V., Manulis, M.: Unique aggregate signatures with applications to distributed verifiable random functions. In: Abdalla, M., Nita-Rotaru, C., Dahab, R. (eds.) Cryptology and Network Security. pp. 251–270. Springer International Publishing, Cham (2013). https://doi.org/10.1007/978-3-319-02937-5_14
16. Micali, S., Rabin, M., Vadhan, S.: Verifiable random functions. In: 40th annual symposium on foundations of computer science (cat. No. 99CB37039). pp. 120–130. IEEE (1999). https://doi.org/10.1109/SFFCS.1999.814584
17. Papadopoulos, D., Wessels, D., Huque, S., Naor, M., Včelák, J., Reyzin, L., Goldberg, S.: Making NSEC5 practical for DNSSEC. Cryptology ePrint Archive, Paper 2017/099 (2017), https://eprint.iacr.org/2017/099, https://eprint.iacr.org/2017/099

18. Pedersen, T.P.: A threshold cryptosystem without a trusted party. In: Davies, D.W. (ed.) Advances in Cryptology — EUROCRYPT '91. pp. 522–526. Springer Berlin Heidelberg, Berlin, Heidelberg (1991). https://doi.org/10.1007/3-540-46416-6_47
19. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) Advances in Cryptology — CRYPTO '91. pp. 129–140. Springer Berlin Heidelberg, Berlin, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_9
20. Polkadot: "cryptography on polkadot" (June 2024), https://wiki.polkadot.network/docs/learn-cryptography, last accessed 01/07/2024
21. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (Nov 1979). https://doi.org/10.1145/359168.359176, http://doi.acm.org/10.1145/359168.359176
22. Vlasov, A., Olson, K., Stokes, A., Sanso, A.: "eip-2537: Precompile for bls12-381 curve operations [draft]," ethereum improvement proposals, no. 2537 (February 2020), https://eips.ethereum.org/EIPS/eip-2537, last accessed 01/07/2024