# A Survey on Distributed Randomness

Kevin Choi

Spring 2021

## Contents

**1 Building Blocks** **3**
- 1.1 Perfectly Synchronous Distributed Randomness . . . 3
- 1.2 Commit-Reveal . . . 3
- 1.3 Verifiable Secret Sharing . . . 3
  - 1.3.1 Feldman-VSS . . . 4
  - 1.3.2 Pedersen-VSS . . . 4
- 1.4 Distributed Key Generation . . . 5
  - 1.4.1 Joint-Feldman . . . 6
  - 1.4.2 Joint-Pedersen . . . 7
- 1.5 Publicly Verifiable Secret Sharing . . . 7
  - 1.5.1 Schoenmakers . . . 7
  - 1.5.2 Scrape's Variations . . . 9
- 1.6 Verifiable Delay Function . . . 9
  - 1.6.1 Wesolowski's . . . 9
  - 1.6.2 Pietrzak's . . . 9
  - 1.6.3 Group Instantiation . . . 9

**2 Protocols** **10**
- 2.1 PVSS-based . . . 10
  - 2.1.1 RandHound . . . 10
  - 2.1.2 Scrape . . . 10
  - 2.1.3 HydRand . . . 10
- 2.2 DVRF-based . . . 10
  - 2.2.1 RandHerd . . . 11
  - 2.2.2 Dfinity . . . 11
  - 2.2.3 DDH-DVRF . . . 11
- 2.3 VDF-based . . . 11
  - 2.3.1 Extending Commit-Reveal . . . 11

2.3.2 Extending Public Randomness . . . . . 11
2.3.3 RandRunner . . . . . 11

# 1 Building Blocks

## 1.1 Perfectly Synchronous Distributed Randomness

Suppose $n$ participants attempt to source a common random number in a distributed yet agreeable manner. In an ideal world where the communication model is perfectly synchronous such that all participants are able to successfully send and receive their random shares $s_i$ (which will make up the common random number) to and from all other participants at the same time, each participant can simply add all shares involved and call $s = \sum s_i$ to be the randomness produced in that round. Such protocol would be not only simple, but also agreeable, as its nature in fact parallels that of rock-paper-scissors, which many play in real life as part of a quick decision-making process. While distributed randomness is hardly a hard problem in this setting, situations can change dramatically in practical settings where participants can go offline (perhaps temporarily) due to network failure, messages can be delayed either non-significantly or significantly, and Byzantine attackers that maliciously try to predict or bias the randomness to their benefit can exist.

## 1.2 Commit-Reveal

In the case of a strawman solution, the classical commit-reveal provides an intuitive way to source a random number from a group of nodes by first allowing each node to commit (in the cryptographic sense) to a secret random number and then adding all shares (revealed later) from all nodes to compute the final random number of a round. The problem with this is that the last person to reveal his share can in fact check the round's output faster than others and hence can decide not to reveal his number if he doesn't like the round's output, biasing the distributed randomness. This is called the *last revealer attack*.

## 1.3 Verifiable Secret Sharing

The issue with the classical Shamir's secret sharing scheme is that either the dealer or other participants could in fact be acting maliciously, e.g. Alice (a participant) needs to trust that she has received her share correctly from the dealer while she also needs to trust that other participants' revealed shares are correct.

This issue can be fixed by the notion of verifiable secret sharing (VSS). The idea is that we add an additional verification process to the usual Shamir's secret sharing scheme. Here, we take note of two commonly used VSS schemes: Feldman-VSS and Pedersen-VSS.

The mathematical setup is as follows. We choose primes $p$ and $q$ such that $q \mid p-1$ and let $g$ be a generator of $G_q$, a cyclic subgroup of $\mathbb{Z}_p^*$. This setup has the effect of achieving the following: $a \equiv b \pmod{q} \iff g^a \equiv g^b \pmod{p}$. As a result, it should be understood that modular arithmetics are done in modulo $q$

whenever the numbers involved concern the exponents while in modulo $p$ otherwise. For convenience, we may omit mod $p$ such that one can aptly assume arithmetics are done in modulo $p$ given an equation unless stated otherwise.

### 1.3.1 Feldman-VSS

The following summarizes a simple VSS scheme (where $t$ among $n$ participants can reconstruct the group secret) proposed by Paul Feldman.

- $f(x) = \sum_{i=0}^{t-1} a_i x^i$ is randomly selected by the dealer, where $a_i \in \mathbb{Z}_q$ and $f(0) = a_0$ is the secret
- The shares are $f(1), f(2), ..., f(n)$ in mod $q$ and are distributed to $n$ participants, respectively
- Also distributed from the dealer are commitments to coefficients of $f$, i.e. $c_j = g^{a_j}$ for $j = 0, ..., t-1$
- Given her share $f(k)$ and the polynomial coefficient commitments, Alice (a participant) can verify her share by checking:

$$g^{f(k)} = g^{\sum_{i=0}^{t-1} a_i k^i} = \prod_{j=0}^{t-1} c_j^{k^j} = c_0 c_1^k c_2^{k^2} \cdots c_{t-1}^{k^{t-1}}$$

- Any $t$ number of participants (say) $i = 1, 2, ..., t$ can recover the secret $a_0$ by performing Lagrange interpolation involving Lagrange coefficients $\lambda_i = \prod_{j \neq i} \frac{j}{j-i}$ in mod $q$:

$$a_0 = f(0) = \sum_{i=1}^{t} f(i)\lambda_i$$

What is new here (compared to Shamir's secret sharing scheme) is the inclusion of commitments to polynomial coefficients in the scheme. These commitments enable participants to verify the validity of their corresponding shares.

### 1.3.2 Pedersen-VSS

Reminiscent of Pedersen commitment, Pedersen-VSS is a variation that involves two random polynomials generated by the dealer as opposed to one. Namely, the scheme runs as follows.

- $f(x) = \sum_{i=0}^{t-1} a_i x^i$ and $f'(x) = \sum_{i=0}^{t-1} b_i x^i$ are randomly selected by the dealer, where $a_i, b_i \in \mathbb{Z}_q$ and $f(0) = a_0$ is the secret (as before)
- The shares are $(f(1), f'(1)), ..., (f(n), f'(n))$ in mod $q$ and are distributed to $n$ participants, respectively

- Also distributed from the dealer are commitments to coefficients of $f$ and $f'$, i.e. $c_j = g^{a_j} h^{b_j}$ for $j = 0, ..., t-1$
- Given her share $(f(k), f'(k))$ and the polynomial coefficient commitments, Alice (a participant) can verify her share by checking:

$$g^{f(k)} h^{f'(k)} = \prod_{j=0}^{t-1} c_j^{k^j}$$

- Any $t$ number of participants (say) $i = 1, 2, ..., t$ can recover the secret $a_0$ by performing Lagrange interpolation involving Lagrange coefficients $\lambda_i = \prod_{j \neq i} \frac{j}{j-i}$ in mod $q$:

$$a_0 = f(0) = \sum_{i=1}^{t} f(i)\lambda_i$$

Effectively, what Pedersen-VSS is able to achieve is the decoupling of $g^{a_0}$ (as the public key corresponding to the secret key $a_0$) and $g^{a_0} h^{b_0}$ (as the published commitment for verification purposes). In other words, the verification process in which the participants verify their shares does not (even information-theoretically) leak any information regarding the initial secret $a_0$, a fact that is not true with Feldman-VSS.

### 1.4 Distributed Key Generation

The motivation for distributed key generation (DKG), which basically comprises $n$ parallel instances of a VSS (run by each participant), is to achieve and utilize a group secret in a leaderless manner such that any threshold $t$ number of participants should be able to recover the same group secret. The basic idea is that each participant becomes a dealer (i.e. leader) for a VSS, in which case the protocol deals with $n$ random polynomials in mod $q$ $\{f_i\}_{i=1,...,n}$ as opposed to one. Though not computed explicitly, the implicit group polynomial $f = \sum f_i$ then embeds the group secret in the form of $f(0)$ (denoted by $x$) as well as the corresponding public key $g^{f(0)}$ (denoted by $y$) via commitments to coefficients of $f_i$ as per each VSS. Notably, this facet is the one that allows a leaderless configuration where there does not exist a leader for $f$ even if each participant $P_i$ remains a leader for $f_i$ with the knowledge of $f_i(0)$. As a result, it is only collectively (i.e. with the collaboration of at least $t$ number of nodes in a group) that the group can make use of $x$ and $y$ in the typical sense of asymmetric cryptography (e.g. signing a message) after performing a DKG.

Before delineating the process of a DKG, we include what it means for a DKG to be uniformly secure. Namely, a uniformly secure DKG should satisfy the following three correctness properties and one secrecy property.

**Definition 1.1.** In the setting where at most $t-1$ nodes can be controlled by an attacker without compromising the protocol, a DKG is *uniformly secure* if the following properties are satisfied.

**Correctness**

1. Any subset of shares of size $t$ can be used to recover the same group secret key $x$.
2. All honest parties have access to the same public key $y = g^x$.
3. $x$ is uniformly distributed in $\mathbb{Z}_q$, and thus $y$ is uniformly distributed in $G_q$ (subgroup of $\mathbb{Z}_p^*$ generated by $g$).

**Secrecy**

1. Other than from the fact that $y = g^x$, no information on $x$ is leaked. Equivalently, this can be stated using a simulator $SIM$: for every probabilistic polynomial-time adversary $\mathcal{A}$, there exists a probabilistic polynomial-time simulator $SIM$ that, given $y \in G_q$ as input, produces an output distribution which is polynomially indistinguishable from $\mathcal{A}$'s view of a DKG protocol that ends with $y$ as its public key output while $\mathcal{A}$ is allowed to corrupt up to $t-1$ participants.

The third correctness property is the one that motivates the .....
As the goal in practical terms is to use

### 1.4.1 Joint-Feldman

1. Each participant $P_i$ runs a Feldman-VSS by choosing a random polynomial $f_i(z) = \sum_{j=0}^{t-1} a_{ij} z^j$ and sending a "subshare" $f_i(j)$ to player $P_j$ for all $j$
2. To satisfy the verifiability portion of the VSS, $P_i$ broadcasts $A_{ik} = g^{a_{ik}}$
3. Upon receiving the subshares and the corresponding commitments (e.g. in the form of a verification vector), $P_j$ can use the verification mechanism per VSS to verify the subshares. If a verification fails, $P_j$ can broadcast a complaint against $P_i$.
4. If $P_i$ receives at least $t$ complaints, then $P_i$ is disqualified. Otherwise, $P_i$ needs to reveal the subshare $f_i(j)$ per $P_j$ that has broadcasted a complaint. We call $QUAL$ the set of non-disqualified players.
5. Once $QUAL$ is set, we define $f(z) = \sum_{i \in QUAL} f_i(z) = \sum_{i=0}^{t-1} a_i z^i$ such that each participant $P_j$ in $QUAL$ can compute the group public key $y = g^{f(0)} = \prod_{i \in QUAL} A_{i0}$, commitments to $f$'s coefficients $A_k = g^{a_k} = \prod_{i \in QUAL} A_{ik}$,

and $P_j$'s share (of the group secret) from subshares $f(j) = \sum_{i \in QUAL} f_i(j)$. Though not computed explicitly, the group secret key $x$ is then equal to both $\sum_{i \in QUAL} a_{i0}$ and the Lagrange interpolation involving the shares $\{f(j)\}_{j \in QUAL}$.

### 1.4.2 Joint-Pedersen

1. Each participant $P_i$ runs a Pedersen-VSS by choosing two random polynomials $f_i(z) = \sum_{j=0}^{t-1} a_{ij} z^j$ and $f'_i(z) = \sum_{j=0}^{t-1} b_{ij} z^j$ and sending a "subshare" $(f_i(j), f'_i(j))$ to player $P_j$ for all $j$

2. To satisfy the verifiability portion of the VSS, $P_i$ broadcasts $C_{ik} = g^{a_{ik}} h^{b_{ik}}$

3. Upon receiving the subshares and the corresponding commitments (e.g. in the form of a verification vector), $P_j$ can use the verification mechanism per VSS to verify the subshares. If a verification fails, $P_j$ can broadcast a complaint against $P_i$.

4. If $P_i$ receives at least $t$ complaints, then $P_i$ is disqualified. Otherwise, $P_i$ needs to reveal the subshare $(f_i(j), f'_i(j))$ per $P_j$ that has broadcasted a complaint. We call $QUAL$ the set of non-disqualified players.

5. Once $QUAL$ is set, we define $f(z) = \sum_{i \in QUAL} f_i(z) = \sum_{i=0}^{t-1} a_i z^i$ such that each participant $P_j$ in $QUAL$ can compute the group public key $y = g^{f(0)} = \prod_{i \in QUAL} A_{i0}$, commitments to $f$'s coefficients $A_k = g^{a_k} = \prod_{i \in QUAL} A_{ik}$, and $P_j$'s share (of the group secret) from subshares $f(j) = \sum_{i \in QUAL} f_i(j)$. Though not computed explicitly, the group secret key $x$ is then equal to both $\sum_{i \in QUAL} a_{i0}$ and the Lagrange interpolation involving the shares $\{f(j)\}_{j \in QUAL}$.

## 1.5 Publicly Verifiable Secret Sharing

While VSS allows verification of the involved shares, the fact of the matter is that such verification can only be done by the involved participants in the secret sharing process. In other words, the verification process of a VSS is not public. Publicly verifiable secret sharing (PVSS), on the other hand, assumes a public setup where the verification process can be achieved publicly (e.g. by bystanders) such that the messages exchanged during the protocol are modified accordingly in order to reflect the fact that some information kept private in a typical VSS can be public in a PVSS. Naturally, this added facet of public verifiability is useful and desirable in models such as the public bulletin board model (e.g. for blockchains). We delineate Schoenmakers' PVSS scheme and Scrape's variations of it here.

### 1.5.1 Schoenmakers

**Initialization.** The group $G_q$ and the generators $g, G$ are selected using an appropriate public procedure. Participant $P_i$ generates a private key $x_i \leftarrow \mathbb{Z}_q^*$ and

registers $y_i = G^{x_i}$ as its public key.

**Distribution.** The protocol consists of two steps:

1. *Distribution of the shares.* Suppose without loss of generality that the dealer wishes to distribute a secret among participants $P_1, ..., P_n$. The dealer picks a random polynomial $p$ of degree at most $t-1$ with coefficients in $\mathbb{Z}_q$

$$p(x) = \sum_{i=0}^{t-1} a_i x^i$$

and sets $s = a_0$. The dealer keeps this polynomial secret but publishes the related commitments $C_j = g^{a_j}$ for $0 \leq j < t$. The dealer also publishes the encrypted shares $Y_i = y_i^{p(i)}$ for $1 \leq i \leq n$ using the public keys of the participants. Finally, let $X_i = \prod_{j=0}^{t-1} C_j^{i^j}$. The dealer shows that the encrypted shares are consistent by producing a proof of knowledge of the unique $p(i)$ for $1 \leq i \leq n$ satisfying:

$$X_i = g^{p(i)}, \qquad Y_i = y_i^{p(i)}.$$

The non-interactive proof is the $n$-fold parallel composition of the protocols for $DLEQ(g, X_i, y_i, Y_i)$. Applying Fiat-Shamir's technique, the challenge $c$ for the protocol is computed as a cryptographic hash of $X_i, Y_i, a_{1i}, a_{2i}, 1 \leq i \leq n$. The proof consists of the common challenge $c$ and the $n$ responses $r_i$.

2. *Verification of the shares.* The verifier computes $X_i = \prod_{j=0}^{t-1} C_j^{i^j}$ from the $C_j$ values. Using $y_i, X_i, Y_i, r_i, 1 \leq i \leq n$ and $c$ as input, the verifier computes $a_{1i}, a_{2i}$ as

$$a_{1i} = g^{r_i} X_i^c, \qquad a_{2i} = y_i^{r_i} Y_i^c$$

and checks that the hash of $X_i, Y_i, a_{1i}, a_{2i}, 1 \leq i \leq n$ matches $c$.

**Reconstruction.** The protocol consists of two steps:

1. *Decryption of the shares.* Using its private key $x_i$, each participant finds the share $S_i = G^{p(i)}$ from $Y_i$ by computing $S_i = Y_i^{1/x_i}$. They publish $S_i$ plus a proof that the value $S_i$ is a correct decryption of $Y_i$. To this end, it suffices to prove knowledge of an $\alpha$ such that $y_i = G^\alpha$ and $Y_i = S_i^\alpha$, which is accomplished by the non-interactive version of the protocol $DLEQ(G, y_i, S_i, Y_i)$.

2. *Pooling the shares.* Suppose without loss of generality that participants $P_i$ produce correct values for $S_i$, for $i = 1, ..., t$. The secret $G^s$ is obtained by Lagrange interpolation:

$$\prod_{i=1}^{t} S_i^{\lambda_i} = \prod_{i=1}^{t} \left(G^{p(i)}\right)^{\lambda_i} = G^{\sum_{i=1}^{t} p(i)\lambda_i} = G^{p(0)} = G^s$$

where $\lambda_i = \prod_{j \neq i} \frac{j}{j-i}$ is a Lagrange coefficient.

Note that the participants do not need nor learn the values of the exponents $p(i)$. Only the related values $S_i = G^{p(i)}$ are required to complete the reconstruction of the secret value $S = G^s$. Also, note that participant $P_i$ does not expose its private key $x_i$; consequently participant $P_i$ can use its key pair in several runs of the PVSS scheme. The type of encryption used for the shares has been optimized for performance; however, if desired, it is also possible to use standard ElGamal encryption instead.

Clearly, the scheme is homomorphic. For example, given the dealer's output for secrets $G^{s_1}$ and $G^{s_2}$, the combined secret $G^{s_1+s_2}$ can be obtained by applying the reconstruction protocol to the combined encrypted shares $Y_{i1}Y_{i2}$.

### 1.5.2 Scrape's Variations

Scrape offers two variations of Schoenmakers' PVSS: one that relies on the DDH (decisional Diffie-Hellman) assumption in the random oracle model and one that relies on the DBS (decisional bilinear square) assumption in the plain model. Accordingly, the former utilizes the same NIZK (i.e. $DLEQ$ by Chaum and Pedersen) from Schoenmakers' whereas the latter utilizes a bilinear pairing, which serves as a valid size-efficient substitute to the NIZK at the cost of pairing-related computations. Both variations make novel use of Reed-Solomon codes from coding theory in a way that the total number of exponentiations needed to verify the shares distributed by the dealer in a PVSS is decreased from $O(nt)$ (which becomes quadratic if $t = \frac{n}{2}$ for instance) to $O(n)$. In other words, the approaches optimize the verifier's computational complexity before the Reconstruction phase.

The idea is that the verifier does not have to compute $t$ exponentiations via $X_i = \prod_{j=0}^{t-1} C_j^{i^j}$ per share, before which the dealer publishes the commitments $C_j$ to the polynomial coefficients. Instead, the dealer publishes $v_i = g^{p(i)}$ directly alongside the dual code $\mathcal{C}^\perp$ of the $[n, k, n-k+1]$ Reed-Solomon code $\mathcal{C}$ corresponding to the shares calculated by the dealer in the form $\mathcal{C} = \{(p(1), ..., p(n)) \mid p(x) \in \mathbb{Z}_q[x], \deg(p(x)) \leq t-1\}$, in which case the verifier can randomly sample a codeword $c^\perp = (c_1^\perp, ..., c_n^\perp)$ from $\mathcal{C}^\perp$, indirectly compute the inner product of $c^\perp$ and the share vector $(s_1, ..., s_n)$, and verify that $\prod v_i^{c_i^\perp} = g^{\sum s_i c_i^\perp} = g^0 = 1$ with $O(n)$ exponentiations.

## 1.6 Verifiable Delay Function

### 1.6.1 Wesolowski's

### 1.6.2 Pietrzak's

### 1.6.3 Group Instantiation

RSA group vs class group of $\mathbb{Q}(\sqrt{p})$ (non-trusted setup)

# 2 Protocols

## 2.1 PVSS-based

### 2.1.1 RandHound

One-off protocol

### 2.1.2 Scrape

Protocol $\pi_{DDH}$ is run between $n$ parties $P_1, ..., P_n$ who have access to a public ledger where they can post information for later verification. A PVSS protocol is used as a subprotocol and it is assumed that the Setup phase is already done and the public keys $pk_i$ of each party $P_i$ are already registered in the ledger. The protocol proceeds as follows:

1. **Commit:** For $1 \leq j \leq n$, party $P_j$ executes the Distribution phase of the PVSS subprotocol as the Dealer with threshold $t = \frac{n}{2}$, publishing the encrypted shares and the verification information $PROOF_D^j$ on the public ledger, and also learning the random secret $h^{s^j}$ and $s^j$. $P_j$ also publishes a commitment to the secret exponent $\mathsf{Com}(s^j, r_j)$ (with fresh randomness $r_j \leftarrow \mathbb{Z}_q$), for $1 \leq j \leq n$.

2. **Reveal:** For every set of encrypted shares and the verification information $PROOF_D^j$ published in the public ledger, all parties run the Verification phase of the PVSS subprotocol. Let $\mathcal{C}$ be the set of all parties who published commitments and valid shares. Once $\frac{n}{2}$ parties have posted their commitments and valid shares on the ledger, party $P_j$ opens its commitment, posting $\mathsf{Open}(s^j, r_j)$ on the ledger, for $j \in \mathcal{C}$.

3. **Recovery:** For every party $P_a \in \mathcal{C}$ that does not publish $\mathsf{Open}(s^a, r_a)$ in the Reveal phase, party $P_j$ runs the Reconstruction phase of the PVSS protocol posting $\tilde{s}_j^a$ and $PROOF_j^a$ to the public ledger, for $1 \leq j \leq n$. Once $\frac{n}{2}$ valid decrypted shares are published, every party reconstructs $h^{s^a}$.

The final randomness is computed as $\rho = \prod_{j \in \mathcal{C}} h^{s^j}$.

### 2.1.3 HydRand

Delayed commit-reveal with a leader per round, with reconstruction via PVSS

## 2.2 DVRF-based

Distributed verifiable random function (DVRF) is quite naturally a distributed version of VRF, where the VRF's secret key is distributed among a group of participants.

Hash of VUF $\rightarrow$ VRF
Threshold signature can be interpreted under this bucket

#### 2.2.1 RandHerd

Involves threshold Schnorr + CoSi (collective signing = multisig aggregation + communication tree) in a fairly complicated way

#### 2.2.2 Dfinity

League of Entropy's drand
Stalling the network? 100% eventual liveness assumption?

#### 2.2.3 DDH-DVRF

Does not use pairings

### 2.3 VDF-based

#### 2.3.1 Extending Commit-Reveal

Unicorn: pre-VDF
RANDAO + VDF

#### 2.3.2 Extending Public Randomness

Taking some closing stock price or a block hash + VDF

#### 2.3.3 RandRunner

A VDF chain involving a group of trapdoor VDFs

## References

[1] Feldman, Paul. "A practical scheme for non-interactive verifiable secret sharing." 28th Annual Symposium on Foundations of Computer Science (sfcs 1987). IEEE, 1987.

[2] Dodis, Yevgeniy, and Aleksandr Yampolskiy. "A verifiable random function with short proofs and keys." International Workshop on Public Key Cryptography. Springer, Berlin, Heidelberg, 2005.