# Non-interactive VSS using Class Groups and Application to DKG

Aniket Kate
Supra Research/Purdue University
aniket@purdue.edu

Easwar Vivek Mangipudi
Supra Research
e.mangipudi@supraoracles.com

Pratyay Mukherjee
Supra Research
p.mukherjee@supraoracles.com

Hamza Saleem
Supra Research/University of Southern California
h.saleem@supraoracles.com

Sri Aravinda Krishnan Thyagarajan
NTT Research
t.srikrishnan@gmail.com

## ABSTRACT

Verifiable secret sharing (VSS) allows a dealer to send shares of a secret value to parties such that each party receiving a share can verify (often interactively) if the received share was correctly generated. Non-interactive VSS (NI-VSS) allows the dealer to perform secret sharing such that every party (including an outsider) can verify their shares along with others' *without* any interaction with the dealer as well as among themselves. Existing NI-VSS schemes employing either exponentiated ElGamal or lattice-based encryption schemes involve zero-knowledge range proofs, resulting in higher computational and communication complexities.

In this work, we present cgVSS, a NI-VSS protocol that uses class groups for encryption. In cgVSS, the dealer encrypts the secret shares in the exponent through a class group encryption such that the parties can directly decrypt their shares. The existence of a subgroup where a discrete logarithm is tractable in a class group allows the receiver to efficiently decrypt the share though it is available in the exponent. This yields a novel-yet-simple VSS protocol where the dealer publishes the encryptions of the shares and the zero-knowledge proof of the correctness of the dealing. The linear homomorphic nature of the employed encryption scheme allows for an efficient zero-knowledge proof of correct sharing. Given the rise in demand for VSS protocols in the blockchain space, especially for publicly verifiable distributed key generation (DKG), our NI-VSS construction can be particularly impactful. We implement our cgVSS protocol using the BICYCL library and compare its performance with a simplified version of the state-of-the-art NI-VSS by Groth. Our implementation shows that cgVSS outperforms (a simplified implementation of) Groth's protocol in overall communication complexity by 5.6x and about $2.4 - 2.7$x in computation time per node (for a 150 node system).

## 1 INTRODUCTION

In a (threshold) secret sharing scheme [8, 51], a dealer distributes a secret among a set of $n$ parties so that the secret can be reconstructed only if a threshold number of $t+1$ or more parties provide their shares. In a verifiable secret sharing (VSS) scheme [24], each party receives a share of the secret and proof that their share is a valid part of the secret. This ability, to confirm the validity of the shares without reconstructing the secret itself, is useful in several secret-sharing applications such as secure multi-party computation, threshold cryptography, and distributed key generation.

The recent adoption of threshold cryptosystems [28] in the blockchain space has invariably increased the demand for VSS mechanisms. In the blockchain space, two additional complementary properties are emerging as crucial: public verifiability and non-interactivity. Public verifiability allows any party to verify the correctness of a protocol execution, and non-interactivity forgoes interaction among parties improving the round complexity of the protocols.

In a traditional (computational) VSS [7, 33], the validity of shares is assured only to the protocol participants through an interactive process; however, in publicly verifiable secret sharing (PVSS) [52], anybody can verify the correctness of the sharing. A typical VSS protocol [45] consists of the dealer generating and sending verifiable secret shares to all the others. Every party receiving the share verifies their share and broadcasts a complaint when the verification fails. When more than $t$ parties raise the complaint, the dealing is marked as invalid; else, the dealer gets an opportunity to publish the correct public response for every complaint. A PVSS protocol [52] instead involves publishing encrypted shares and proving the correctness of the encryptions and sharing [34, 36, 50]. Multiple encryption schemes including ElGamal [36] and Lattice-based encryption schemes [35] have been employed to realize publicly verifiable VSS. If the proof mechanism is non-interactive, the protocol will be a non-interactive VSS protocol that removes the need for the complaint phase and the corresponding interactive verification. Moreover, a publicly verifiable protocol ensures the correctness of the protocol even in the case of a security compromise, as anybody can verify the correctness of the secret sharing. However, since we employ public-key encryption in PVSS, unlike in VSS, we have to surrender the possibility of unconditional secrecy/hiding of secret sharing in PVSS.

In a PVSS, a dealer generates shares $s_i$ for each party $P_i$ in the system and encrypts them. 'Exponentiated' or 'lifted' ElGamal encryption (with the scalar in the exponent) of a message $m$ as $(\bar{g}^r, \bar{g}^m \bar{h}^r)$ is a natural candidate for the encryption, as it allows linear homomorphic operations on the message $m$. Here, $\bar{g}$ is a random generator of the underlying group $\mathbb{G}$ of prime order $q$, $r$ is a random scalar from $\mathbb{Z}_q$ and $\bar{h}$ is the public key of the party.

If a share value $s_i$ is encrypted as $(\bar{g}^r, \bar{g}^{s_i} \bar{h}_i^r)$ for the public key $\bar{h}_i$ of the party indexed $i$, the receiver is expected first to decrypt the value $\bar{g}^{s_i}$ and then solve the discrete logarithm to compute the share value $s_i$. However, if solving the discrete logarithm is considered difficult in the underlying group $\mathbb{G}$, the decryption is inefficient and computationally hard. To avoid such a predicament, one can divide each share value $s_i$ into smaller 'chunks' $s_{ij}$ where

$s_i = s_{i1}||s_{i2}||\cdots||s_{im}$, encrypt each chunk individually using exponentiated ElGamal encryption as $(\bar{g}^{r_j}, \bar{g}^{s_{ij}}\bar{h}^{r_i})$ (an approach taken by Groth et al. [36]). It would now be easier for the receiver to compute the values $s_{ij}$ by solving the discrete logarithm through, for example, a brute-force search. The receiver computes the smaller chunks $s_{ij}$ and uses them to retrieve the share value $s_i$. Apart from the correctness of sharing, this approach would require the dealer to prove in zero-knowledge that the size of each chunk is "sufficiently small" and within a small range. The whole process of dividing shares into smaller chunks, individually encrypting them, and proving that the chunks are correctly formed makes the message complexity of the whole system $O(m \cdot n)$ for $n$ shares and $m$ chunks per share. Chunking could be avoided if there is an efficient way to obtain the share value $s_i$ from the exponent in $\bar{g}^{s_i}$ following the decryption. We propose to use a class group-based encryption mechanism to achieve this.

## 1.1 Our Work

This work considers an encryption scheme in the class-group setting [21]. Unlike traditional (elliptic curve and finite field) discrete logarithm settings, in a class group, there exists a subgroup where solving discrete logarithms is efficient i.e., given a value $f^{s_i}$ in the subgroup and the corresponding generator $f$, computing $s_i$ is efficient. For VSS using class groups, the dealer generates the shares $s_i$ as evaluations of a random polynomial $f(x)$, encrypts each $s_i$ as $(g^r, f^{s_i}h_i^r)$. Here, $g$[1] is the generator of the underlying class group $G$, $f$ is the generator of the subgroup $F$ where discrete logarithm is tractable, $h_i$ is the public key of the receiver, and $r$ is an appropriate scalar. As before, the secret share value is present in the exponent as $f^{s_i}$ allowing for homomorphic operations (in the exponent). However, since discrete logarithm is tractable in the subgroup $F$, the receiver can recover the value $s_i$ after obtaining $f^{s_i}$ following the decryption. This allows the dealer to publish just the encryptions of the shares and the corresponding zero-knowledge proof, thereby achieving a message complexity of $O(n)$ for $n$ shares. The dealer chooses the parameters such that the order of the subgroup supports the bit length of the shares that he encrypts.

We propose a *non-interactive publicly verifiable* VSS scheme based on class groups; we call it cgVSS. In cgVSS, the dealer publishes class group encrypted shares and an efficient zero-knowledge proof (ZKP) of correct sharing. We employ the ZKP of exponent in class groups and adapt it to achieve efficient proof of sharing – a non-typical solution direction. The receivers decrypt their shares after successful verification. We employ the cgVSS and realize a distributed key generation (DKG) mechanism [32, 36, 39]. A DKG protocol provides threshold shares of a secret key to each party in the system such that no set of parties less than the threshold in number have any information about the secret key. However, any set of more than the threshold number can compute the secret key together. The public key corresponding to the shared secret key is known to all the parties by the end of the protocol. The proposed cgVSS is used to realize a DKG mechanism in a straightforward manner where each party acts as the dealer and performs the verifiable secret sharing. After all the dealers publish the encrypted shares and the corresponding proofs of correct sharing, every party verifies the dealing from public information and agrees on the set of dealers whose dealing has been verified. Every party computes their share locally as a summation of verified shares received. Since the verification is from public information, the parties need not interact further to agree on the qualified set of dealers whose dealings have been verified.

The DKG from cgVSS, which we call cgDKG, is non-interactive, publicly verifiable, and has a message complexity of – $O(n^2)$ for $n$ parties. The improvement in message complexity of DKG is a direct consequence of the proposed VSS protocol. Non-interactive DKG (NI-DKG) protocols find increasing use of blockchain as a broadcast channel; this leads to all the communicated messages being stored on the blockchain. Using class groups for DKG and reducing the message complexity also improves the storage complexity of the blockchain used as the broadcast channel. The proposed DKG protocol achieves sharing of an (unpredictable) secret value and is safe for applications involving discrete logarithm keys. However, it suffers from the biasing public key attack [32] similar to DKG protocols proposed earlier [36], [46]. To overcome this, we use an extension [44] by adding a round of interaction to the protocol where every party publishes the exponentiated version of their share $s_i$ with a different generator $g'$ as $g'^{s_i}$. This achieves an efficient mechanism for overcoming the attack (see Appendix B for details).

**Benchmarking.** We implement our NI-VSS protocol cgVSS and compare that with the closest existing scheme by Groth's [36] in terms of dealer/receiver times, and the total bit-length of the message broadcast by the dealer (in Section 6). For comparison, we implement a simplified version of the VSS mechanism (referred to as GrothVSS henceforth) proposed by Groth [36] without forward secrecy. Our implementation shows that the bit-length of the total broadcast message for a single execution for 150 users is 296.51 Kb for the cgVSS compared to 1.66 Mb in GrothVSS which is a 5.6x improvement. Also, in the same setting the gain in dealer's/receiver's computation time is about 2.4 – 2.7x. In summary, our protocol cgVSS outperforms the state-of-art GrothVSS both in communication and computation. This is despite the class-group operations being in the regime of other similar composite order groups, such as RSA. Essentially the performance gain can be attributed for the design simplification, in that any range proof (or proof-of-chunking ala Groth [36]) is totally dispensed with.[2] Importantly, this means that our scheme cgVSS scales much better with the increasing number of parties compared to GrothVSS. We also benchmark the DKG protocols (cf. Sec. 6) end-to-end and compare GrothDKG with cgDKG. For a 50 node network, GrothDKG takes 43.058sec while cgDKG takes 17.950sec.

## 1.2 Related Work

**PVSS**. In their seminal paper, Chor et al. [24] introduced the notion of verifiable secret sharing (VSS). Stadler [52] first proposed publicly verifiable VSS (PVSS) and two constructions using verifiable ElGamal encryption. A long line of works [9, 31, 50] ,[10, 16, 18, 30, 36, 38, 47, 48, 52, 57] realized publicly verifiable and non-interactive VSS schemes. They typically employ encryption mechanisms, including

[1]Notice that we use symbols with bar eg: $\bar{g}$ for generators of prime order (elliptic curve) groups and without bar eg: $g$ for class group generators.

[2]Moreover, the simplified design itself is a substantial advantage from engineering/deployment perspective as well.

Paillier [38, 47, 48], ElGamal-in-the-exponent [36], pairing[29, 55] and lattice-based encryptions[35]. The schemes, that use Paillier encryption suffer from long exponentiations and proof size, that use ElGamal in the exponent [36] requires small exponents due to hardness of DLog. The schemes involving pairing generate shares are group elements (not scalars) and are not suitable for settings such as threshold signatures. PVSS schemes based on lattice-encryption schemes [35] are indeed asymptotically efficient, albeit require large public keys and ciphertext sizes. In another line of work, asynchronous VSS schemes [1, 14, 37, 58] are proposed but public verifiability is not defined in such systems. In addition, they suffer from a so-called high replication factor ($n > 3t$) for a threshold $t$.

**DKG**. Several DKG protocols to support DLog based threshold systems have been studied [1, 2, 17, 32, 36, 40, 40, 41, 43] in the literature in the synchronous and asynchronous settings. However, to achieve public verifiablity, the nodes need to perform PVSS (instead of VSS or asynchronous VSS). Any aggregatable PVSS scheme [43] which supports homomorphic operations on the secret shares [36, 38, 47] may be employed to realize a publicly verifiable DKG mechanism. To achieve non-interactive DKG, one should employ a PVSS for the secret sharing. Groth [36] proposed a non-interactive distributed key generation mechanism using ElGamal encryptions of shares that can be publicly verified by all the parties from the commitments of the polynomial coefficients. We use a simplified variant of this scheme as our baseline.

*Biasing the public key:* Recently, Katz [40] proposed two round-optimal constructions for a 'robust' DKG mechanism, where they define *robustness* as guaranteed-output-delivery. Their definition requires that the DKG mechanism outputs an *unbiased* public key. However, unbiased public keys are not an absolute requirement for DKG mechanisms, since it has been shown that biased public keys can be securely employed for certain systems as long as the secret key is secure. Gennaro et al. [32] show that biased public keys can be securely employed for any cryptographic system relying on the DLog assumption, like the threshold version of the Schnorr signature scheme. Bacho and Loss [5] show that DKG mechanisms that output biased public keys can be employed for generating key shares of adaptively secure BLS scheme as long as they can be shown to be *oracle-aided-algebraic-simulatable* (see [5, Sections 3,4.3]. Braun, Damgård, and Orlandi [13] propose an encryption scheme based on class groups that is secure even with biased public keys.

# 2 PRELIMINARIES

## 2.1 Notation

We use $\mathbb{Z}$ for all integers, and $\mathbb{N}$ for all natural numbers $\{1, 2 \ldots\}$. Vectors are denotes as $\vec{v}$. For a vector $\vec{v}_i$, its $j$-th element is denoted $\vec{v}_{i,j}$. We use the notation $x \xleftarrow{\$} \mathcal{D}$ to indicate that $x$ has been randomly sampled from the distribution $\mathcal{D}$ and the notation $h \leftarrow y$ to indicate that the $h$ has been assigned the value $y$. Also, for any algorithm $A$ we denote $y \leftarrow A(x)$ to express that $A$ on input $x$ yields the output $y$. Unless mentioned otherwise, all algorithms considered in this paper are probabilistic polynomial time (PPT). Sometimes, we explicitly use the notation $A(x; r)$ to denote the output of the algorithm $A$ when run on input $x$ and fixed randomness $r$. Even if $A$ is probabilistic, the notation $A(x; r)$ indicates that it runs on input $x$ with fixed randomness $r$, outputs a unique $y$ – this is also known as determinization of $A$. If a group has unknown order, then we denote it with a hat $\widehat{G}$. We indicate the set $\{1, 2, \cdots, n\}$ by $[n]$. We use the symbol $\stackrel{?}{=}$ to indicate a check of equality of the left and right-hand side entities of the symbol. $(a \stackrel{?}{=} b)$ returns a boolean value denoting whether the equality holds or not. The computational security parameter is denoted by $\lambda$ (a typical value 128), and the statistical security parameter is denoted by $\lambda_{\mathrm{st}}$ (typical value 40). We say that a function is negligible in $\lambda$, if it grows as $2^{-\Omega(\lambda)}$.

## 2.2 Shamir Secret Sharing

We use Shamir's secret sharing [51]. In a typical Shamir's secret sharing, a field element $s \in \mathbb{Z}_q$ can be shared in a $t$ out of $n$ fashion by choosing a $t$-degree uniform random polynomial $P(x) \xleftarrow{\$} \mathbb{Z}_q[x]^t$ with constraint $P(0) = s$. The $i$-th share is computed as $s_i \leftarrow P(i)$. To reconstruct one may use Lagrange coefficients $L_i$s as $s = \sum_{i=1}^{t+1} L_i s_i$. Due to linearity, this can be performed in the exponent without computing $s$. We denote this by $\mathsf{SSS}_{n,t,q}(s) = (s_1, \ldots, s_n)$.

## 2.3 Class Groups Setting

Castagnos and Laguillaumie [21] propose an ElGamal-like encryption scheme using class groups. The main idea is to use a composite order group of unknown order with an underlying subgroup of known order where the discrete logarithm is easy. Since then, a number of works showed the feasibility of several cryptographic tasks [13, 19, 53, 54] including two-party ECDSA [19] and multi-party computation [13].

In this paper, we follow a presentation similar to [13]. We consider a finite abelian group $\widehat{G}$ of *unknown* order $q \cdot \hat{s}$ with an unknown $\hat{s}$, and known $q$ such that $q$ and $\hat{s}$ are co-prime; $\widehat{G}$ is factored as $\widehat{G} \simeq \widehat{G}^q \times F$, where $F = \langle f \rangle$ is the unique subgroup of order $q$. An upper bound $\bar{s}$ is known for $\hat{s}$. We also consider a cyclic subgroup $G = \langle g \rangle$ of $\widehat{G}$, such that $G$ has order $q \cdot s$. Unlike $\widehat{G}$, the elements of $G$ are not efficiently recognizable. $G^q = \langle g_q \rangle$ denotes the cyclic subgroup of $G$ of the $q$-th power. So, $G$ can be factored as $G \simeq G^q \times F$ and $g = g_q \cdot f$. We also consider two distributions $\mathcal{D}$ and $\mathcal{D}_q$ over $\mathbb{Z}$ $\{g^x \mid x \leftarrow \mathcal{D}\}$ and $\{g_q^x \mid x \leftarrow \mathcal{D}_q\}$, such that they induce distributions over $G$ and $G^q$ respectively, that are statistically close (within distance $2^{-\lambda_{\mathrm{st}}}$) to uniform distributions over respective domains.

The framework specifies algorithms $(\mathsf{CG.ParamGen}, \mathsf{CG.Solve})$ with the following description:

- $(q, \lambda, \lambda_{\mathrm{st}}, \bar{s}, f, g_q, \widehat{G}, F, \mathcal{D}, \mathcal{D}_q; \rho) \leftarrow \mathsf{CG.ParamGen}(1^\lambda, 1^{\lambda_{\mathrm{st}}}, q)$. This algorithm, on input the computational security parameter $\lambda$, the statistical security parameter $\lambda_{\mathrm{st}}$ and a modulus $q$, outputs the group parameters and the randomness $\rho$ used to generate them. For convenience, we include the descriptions of the distributions $\mathcal{D}$ and $\mathcal{D}_q$ as well.
- $x \leftarrow \mathsf{CG.Solve}(f^x, (q, \lambda, \lambda_{\mathrm{st}}, \bar{s}, f, g_q, \widehat{G}, F, \mathcal{D}, \mathcal{D}_q))$. This algorithm deterministically solves the discrete log in group $F$.

**Hardness assumptions on class groups**. We need the *unknown order* and the *hard subgroup membership* assumptions as described below.

*Definition 2.1 (Unknown order assumption[13]).* For the security parameters $\lambda, \lambda_{\mathsf{st}} \in \mathbb{N}$, modulus $q \in \mathbb{Z}$ consider a set of public parameters $pp_{\mathsf{CG}} := (q, \lambda, \lambda_{\mathsf{st}}, \bar{s}, f, g_q, \widehat{G}, F, \mathcal{D}, \mathcal{D}_q; \rho) \leftarrow \mathsf{CG.ParamGen}(1^{\lambda}, 1^{\lambda_{\mathsf{st}}}, q)$ generated using a uniform random $\rho$. We say that the unknown order assumption holds over the classgroup framework, if for any PPT adversary $\mathcal{A}$, the following probability is negligible in $\lambda$.

$$\Pr\left[h^e = 1 \wedge h \in (\hat{G} \setminus F) \wedge e \in \mathbb{N} \mid (h, e) \leftarrow \mathcal{A}(pp_{\mathsf{CG}})^{\mathsf{CG.Solve}(\cdot)}\right]$$

*Definition 2.2 (Hard subgroup membership assumption [13]).* For the security parameters $\lambda, \lambda_{\mathsf{st}} \in \mathbb{N}$ and modulus $q \in \mathbb{Z}$ consider a set of public parameters $\mathsf{pp}_{\mathsf{CG}} := (q, \lambda, \lambda_{\mathsf{st}}, \bar{s}, f, g_q, \widehat{G}, F, \mathcal{D}, \mathcal{D}_q; \rho) \leftarrow \mathsf{CG.ParamGen}(1^{\lambda}, 1^{\lambda_{\mathsf{st}}}, q)$ generated using a uniform random $\rho$. Sample $x \leftarrow \mathcal{D}$ and $x_q \leftarrow \mathcal{D}_q$. Sample a bit $b \xleftarrow{\$} \{0,1\}$ uniformly at random. If $b = 0$, define $h^* \leftarrow g^x$, otherwise if $b = 1$ define $h^* \leftarrow g^{x_q}$. Then we say that the hard subgroup membership assumption holds over the classgroup framework, if for any PPT adversary $\mathcal{A}$, the following probability is negligible in $\lambda$.

$$\left|\Pr\left[b = b^* \mid b^* \leftarrow \mathcal{A}(pp_{\mathsf{CG}}, h^*)^{\mathsf{CG.Solve}(\cdot)}\right] - \frac{1}{2}\right|$$

# 3 BUILDING BLOCKS

Our NI-VSS scheme is based on three building blocks: (i) a Schnorr's NIZK proof for knowledge of exponent (over class-group); (ii) an ElGamal-like multi-receiver encryption scheme; (iii) and a Schnorr-like compact proof of correct secret-sharing. In this section, we present them in order.

## 3.1 Schnorr's NIZK for Knowledge of Exponent over class-groups.[20]

Our construction uses non-interactive zero-knowledge (NIZK) proof for knowledge of exponents over class groups. In particular, consider the class-group parameters $pp_{\mathsf{CG}} = (q, \lambda, \lambda_{\mathsf{st}}, \bar{s}, f, g_q, \widehat{G}, F, \mathcal{D}, \mathcal{D}_q; \rho)$ an instance $\mathsf{inst} = (g_q, h)$ and witness $\mathsf{wit} = k \xleftarrow{\$} \mathcal{D}_q$ such that $h \leftarrow g_q^k \in G^q$. Also consider a hash function $H : \{0,1\}^* \rightarrow \mathcal{B}$ for a bound $\mathcal{B} = 2^{O(\lambda)}$. The set of public parameters for the proof system is defined as $pp_{\mathsf{Kex}} \leftarrow (H, \mathcal{B}) \cup \{pp_{\mathsf{CG}}\}$. Then the proof system consists of the following two algorithms:

- $\mathsf{Kex.Prove}(pp_{\mathsf{Kex}}, \mathsf{inst}, \mathsf{wit}) \rightarrow \pi$. This randomized algorithm takes an instance-witness pair $(\mathsf{inst}, \mathsf{wit}) = ((g, h), k)$ as input. Then it executes the following steps:
  - Samples a value $r \xleftarrow{\$} [\mathcal{B} \cdot |\mathcal{D}_q| \cdot 2^{\lambda_{\mathsf{st}}}]$
  - $\alpha \leftarrow g^r$;
  - $c \leftarrow H(g, h, \alpha) \in \mathcal{B}$;
  - $s \leftarrow r + k \cdot c \in \mathbb{Z}$;
  - Output the NIZK proof $\pi = (c, s)$
- $\mathsf{Kex.Ver}(pp_{\mathsf{Kex}}, \mathsf{inst}, \pi) \rightarrow 1/0$. This deterministic algorithm takes an instance $\mathsf{inst} = (g, h)$ and a candidate proof $\pi = (c, s)$ as input. Then:
  - Compute $\alpha \leftarrow g^s \cdot (h^c)^{-1}$;
  - Output $(c \stackrel{?}{=} H(g, h, \alpha)) \in \{0, 1\}$.

**Security**. The completeness and soundness follow immediately from Schnorr [49]. For zero-knowledge, a crucial difference is the computation of $s$. Note that we compute it over integer because the group order is unknown – this is in contrast with the typical Schnorr setting where the group order is known. We need to ensure that the value $s$ can be simulated without the knowledge of $k$. For that, we rely on a statistical argument. In particular, we choose a "mask" $r$ randomly from a range larger than the range of $kc$ by a factor of $2^{\lambda_{\mathsf{st}}}$. So, to simulate, it is possible to sample $s$ from a range such that the simulated value is within statistical distance $2^{-\lambda_{\mathsf{st}}}$ to the actual value. The rest can be argued, following the footsteps of Schnorr's proof.

## 3.2 Multi-receiver Encryption from Class-group

We present a multi-receiver linearly homomorphic encryption from class-groups in this section. Our construction adapts the ElGamal-like encryption scheme from [21] in a multi-receiver setting. The encryption mechanism based on our class-group framework is IND-CPA and employs the class groups $G$ with a subgroup $F$ where the discrete log is easy. Let $pp_{\mathsf{CG}}$ be the public parameters which are the same as the class-group parameters $pp_{\mathsf{CG}} := (q, \lambda, \lambda_{\mathsf{st}}, \bar{s}, f, g_q, \widehat{G}, F, \mathcal{D}, \mathcal{D}_q; \rho)$. The multi-receiver encryption scheme is comprised of three algorithms CGE.KeyGen, CGE.mrEnc and CGE.Dec for generating the keys, (multi-receiver) encryption and decryption, respectively:

- $\mathsf{CGE.KeyGen}(pp_{\mathsf{CG}}) \rightarrow (sk, h)$:
  - $sk \xleftarrow{\$} \mathcal{D}_q$
  - $h \leftarrow g^{sk}$
- $\mathsf{CGE.mrEnc}(pp_{\mathsf{CG}}, \{h_i, m_i\}_{i \in [k]}) \rightarrow (R, \{E_i\}_{i \in [k]})$
  - $r \xleftarrow{\$} \mathcal{D}_q$
  - $R \leftarrow g^r$
  - For all $i \in [k]$: $E_i \leftarrow f^{m_i} h_i^r$
- $\mathsf{CGE.Dec}(pp_{\mathsf{CG}}, sk, R, E) \rightarrow m$
  - $M \leftarrow \frac{E}{R^{sk}}$
  - $m \leftarrow \mathsf{CG.Solve}(pp_{\mathsf{CG}}, M)$

In the above, the encryption scheme takes a number of public keys and messages as input and produces a multi-receiver ciphertext containing a common randomness value $R$, and a specific message-dependent part $E_i$. Each ciphertext can be individually parsed as $(R, E_i)$.

*Definition 3.1 (Security of class-group based multi-receiver Encryption).* Let $(\mathsf{CGE.KeyGen}, \mathsf{CGE.mrEnc}, \mathsf{CGE.Dec})$ be a multi-receiver encryption scheme based on class groups. Then, we say that the scheme is secure if for a correctly generated class-group parameters $pp_{\mathsf{CG}}$, any $n, t \in \mathbb{N}$ $(n > t)$ for any PPT adversary $\mathcal{A}$ the probability that the following experiment outputs 1 is bounded by at most $\mathsf{negl}(\lambda)$ away from $1/2$:

- Run $\mathsf{CG.ParamGen}(pp_{\mathsf{CG}})$ $n$ times to get $(sk_1, h_1), \ldots, (sk_n, h_n)$.
- Give $\{h_i\}_{i \in [n]}$ to $\mathcal{A}$.
- Receive $C \subset [n]$ of size $t$. Give $\{sk_i\}_{i \in C}$ to $\mathcal{A}$.
- Receive challenge vectors $(\vec{m}_0, \vec{m}_1)$ of length $n$ from $\mathcal{A}$ such that for all $i \in C : m_{0,i} = m_{1,i}$

- Choose a uniform random $b$ and encrypt $(R, \{E_i\}_{i\in[n]}) \leftarrow \mathsf{CGE.mrEnc}(pp_{\mathsf{CG}}, \{h_i, m_i\}_{i\in[n]})$.
- Receive $b'$ from $\mathcal{A}$, output $(b \stackrel{?}{=} b')$.

### 3.3 Proof of Correct Secret-Sharing.

Looking ahead, in our NI-VSS protocol we shall require the dealer to produce a non-interactive zero-knowledge proof of correct sharing, where shares are encrypted with the above multi-receiver encryption. We essentially use the Groth's [36] variant of Schnorr proof, adapted to our class-group setting. The overall idea, as we recall from [36], is to use Schnorr's proof for knowledge of exponent in a compact fashion. Note that, the multi-ciphertext consists of a group element $R = g^r$ and another $n$ group elements (in our case $k = n$) of the form $E_i = f^{s_i} h_i^r$. The dealer is required to prove that encrypted messages form a $t$ out of $n$ Shamir's secret sharing in addition to the knowledge of plaintext and randomness. The main idea is to combine these different knowledge of exponents in a way such that the exponents are consistent with the evaluation of $t$-degree secret polynomial used for secret-sharing – to enable this dlog commitments of the secret polynomial are used. Let us now describe the scheme in detail.

Consider any cyclic group $\bar{G}$ of prime order $q$ and a randomly chosen generator $\bar{g} \stackrel{\$}{\leftarrow} \bar{G}$. Note that the group $\bar{G}$ is isomorphic to group $F$. Also, consider hash functions (modeled as random oracles) $H, H'$ both mapping $\{0,1\}^* \rightarrow \mathbb{Z}_q$. The public parameter of the proof system is defined as $pp_{\mathsf{PoC}} := \{\bar{g}, \bar{G}, H, H'\} \cup pp_{\mathsf{CG}}$. We use the generator $\bar{g}$ for commitments, on group $\bar{G}$, which is typically an elliptic curve.

Now consider a secret $s \in \mathbb{Z}_q$, and let $(s_1, \ldots, s_n)$ be a $t$ out of $n$ Shamir's secret-sharing of $s$, done using a $t$-degree secret polynomial $P(x)$ over $\mathbb{Z}_q$ such that $P(i) = s_i$ for all $i \in [n]$. Also, denote the coefficients of $P$ by $a_0, a_1, \ldots, a_t$ each in $\mathbb{Z}_q$ and corresponding dlog commitments as $A_0, A_1, \ldots, A_t$. The shares $s_1, \ldots, s_n$ are then encrypted by the dealer using the multi-receiver encryption scheme described above as $\mathsf{CGE.mrEnc}(pp_{\mathsf{CG}}, \{h_i, s_i\}_{i\in[n]}; r)$ using randomness $r$ (we determinize the encryption algorithm here) to produce a ciphertext tuple $(R, \{E_i\}_{i\in[n]})$. The proof-system described in this section proves a relation $\mathfrak{R}_{\mathsf{CS}}$ that consists of an instance inst and a witness wit where:

- $\mathsf{inst} = \Big(\{h_i\}_{i\in[n]}, (R, \{E_i\}_{i\in[n]}), (A_0, \ldots, A_t)\Big)$;
- $\mathsf{wit} = ((s_1, \ldots, s_n), r)$

for the statement:

- there exists a $t$-degree polynomial $P(x) = a_0 + a_1 x + \ldots a_t x^t$ over $\mathbb{Z}_q$ such that for all $i \in [n]$: $s_i = P(i)$; and for all $j \in \{0, \ldots, t\}$: $A_j = \bar{g}^{a_j}$;
- encrypting $s_1, \ldots, s_n$ with randomness $r$ using public keys $h_1, \ldots, h_n$ yields a multi-receiver ciphertext of the form $(R, \{E_i\}_{i\in[n]})$

For an instance inst which has the same format as a correct instance (as described above), but does not satisfy the statement (and therefore has no witness), we denote $\mathsf{inst} \notin \mathfrak{R}_{\mathsf{CS}}$. The algorithms SharingProof and SharingVerify are described in Figure 1.

We require the above algorithms to satisfy the following secrity requirements.

**Proof of Correct Sharing**

- $\mathsf{SharingProof}(pp_{\mathsf{PoC}}, \mathsf{inst}, \mathsf{wit}) \rightarrow \pi_{\mathsf{CS}}$ :
  - Parse wit as $\{(s_1, \ldots, s_n), r\}$.
  - Sample $\alpha \stackrel{\$}{\leftarrow} \mathbb{Z}_q, \rho \leftarrow [q \cdot |\mathcal{D}_q| \cdot 2^{\lambda_{\mathsf{st}}}]$.
  - $W \leftarrow g_q^{\rho}$ and $X \leftarrow \bar{g}^{\alpha}$
  - Compute:
    - $\gamma \leftarrow H(\mathsf{inst})$.
    - $Y \leftarrow f^{\alpha} \cdot \left(h_1^{\gamma} \cdot h_2^{\gamma^2} \cdot \ldots \cdot h_n^{\gamma^n}\right)^{\rho} \in G$.
    - $\gamma' \leftarrow H'(\gamma, W, X, Y)$.
    - $z_r \leftarrow r\gamma' + \rho \in \mathbb{Z}$.
    - $z_s \leftarrow \gamma' \sum_{i=1}^{n} s_i \gamma^i + \alpha \in \mathbb{Z}_q$.
  - Finally return $\pi_{\mathsf{CS}} \leftarrow (W, X, Y, z_r, z_s)$
- $\mathsf{SharingVerify}(pp_{\mathsf{PoC}}, \mathsf{inst}, \pi_{\mathsf{CS}}) \rightarrow 1/0$ :
  - Parse $\pi_{\mathsf{CS}}$ as $(W, X, Y, z_r, z_s)$.
  - Compute:
    - $\gamma \leftarrow H(\mathsf{inst})$.
    - $\gamma' \leftarrow H'(\gamma, W, X, Y)$.
  - Verify the following equality:
    - $W \cdot R^{\gamma'} \stackrel{?}{=} g_q^{z_r} \in G^q$;
    - $X \cdot (\prod_{j=0}^{t} A_j^{\sum_{i=1}^{n} i^j \gamma^i})^{\gamma'} \stackrel{?}{=} \bar{g}^{z_s} \in \bar{G}$;
    - $(\prod_{i=1}^{n} E_i^{\gamma^i})^{\gamma'} \cdot Y \stackrel{?}{=} f^{z_s} \cdot \prod_{i=1}^{n} (h_i^{\gamma^i})^{z_r} \in G$.
  - Return 1 if all of the above holds, and 0 otherwise.

**Figure 1: Proof System of Correct Sharing.**

*Definition 3.2 (Security of Proof of Correct Sharing).* Consider the security parameters $\lambda, \lambda_{\mathsf{st}} \in \mathbb{N}$ and a modulus $q \in \mathbb{Z}$. Consider a correctly generated $pp_{\mathsf{PoC}}$ as above. Then the pair of algorithms (SharingProof, SharingVerify) is called a secure NIZK proof system for correct sharing if they satisfy:

- **Completeness.** For each legitimate instance-witness pair $(\mathsf{inst}, \mathsf{wit}) \in \mathfrak{R}_{\mathsf{CS}}$ the following probability is 1.

$$\Pr\Big[1 \leftarrow \mathsf{SharingVerify}(pp_{\mathsf{PoC}}, \mathsf{inst}, \pi_{\mathsf{CS}}) \;\Big|\; \pi_{\mathsf{CS}} \leftarrow \mathsf{SharingProof}(pp_{\mathsf{PoC}}, \mathsf{inst}, \mathsf{wit})\Big]$$

- **Statistical Soundness.** For any unbounded adversary $\mathcal{A}$, the following probability is upper bounded by $\mathsf{negl}(\lambda_{\mathsf{st}})$:

$$\Pr[1 \leftarrow \mathsf{SharingVerify}(pp_{\mathsf{PoC}}, \mathsf{inst}, \pi_{\mathsf{CS}}) \wedge \mathsf{inst} \notin \mathfrak{R}_{\mathsf{CS}} \;|\; (\mathsf{inst}, \pi_{\mathsf{CS}}) \leftarrow \mathcal{A}^{H,H'}(pp_{\mathsf{PoC}})]$$

- **Zero-knowledge.** For any PPT adversary, there exists a PPT simulator $\mathcal{S}_{\mathsf{CS}}$ such that the following probability is upper-bounded by $\mathsf{negl}(\lambda)$.

$$\Big|\Pr[\mathcal{A}^{H,H',\mathsf{SharingProof}(pp_{\mathsf{PoC}},\cdot)}(pp_{\mathsf{PoC}})] - \Pr[\mathcal{A}^{H,H',\mathcal{S}'}(pp_{\mathsf{PoC}})]\Big|$$

  where $\mathcal{S}'$ on input $(\mathsf{inst}, \mathsf{wit})$, returns $\mathcal{S}_{\mathsf{CS}}(\mathsf{inst})$ if $(\mathsf{inst}, \mathsf{wit}) \in \mathfrak{R}_{\mathsf{CS}}$, or returns $\perp$ if $(\mathsf{inst}, \mathsf{wit}) \notin \mathfrak{R}_{\mathsf{CS}}$.

Next we show how the construction (cf. Fig. 1) satisfies the above definition by formally proving the following theorem.

Theorem 3.3. For any security parameters $\lambda, \lambda_{st} \in \mathbb{N}$ and any modulus $q \in \mathbb{N}$, our construction, described in Fig. 1, satisfies the security definition given in Def. 3.2 assuming DLog is tractable in $F$, DDH is hard in $G$, the strong root and $\omega$-low order assumptions [53] hold in $G$ in the random oracle model.

We provide the proof sketch for completeness, soundness and zero-knowledge in Appendix C.

## 4 NI-VSS USING CLASS GROUPS

We realize cgVSS, a non-interactive verifiable secret sharing mechanism from class groups and employ it to achieve a non-interactive distributed key generation protocol cgDKG. Our cgVSS scheme uses the encryption scheme and proofs of correct sharing from the previous sections.

**System Model**. For our non-interactive construction, we assume that all parties have access to a *broadcast channel*. The adversary controls the communication channel and can delay the messages; however, it has to deliver those before the synchrony communication bound $\Delta$. The adversary is also rushing and can delay the messages of the parties and inject its own messages after observing honest nodes' messages during the current round.

The system consists of $n+1$ parties, $n$ receivers $P_1, \ldots, P_n$ and a dealer $P_D$. We consider a $t$-bounded static adversary who can maliciously corrupt at most $t$ parties, which may or may not include the dealer. We consider static corruption, in that the set of corrupt parties is decided in the beginning of an execution and stays the same throughout. While the protocol is non-interactive, we consider a one-time public-key setup phase, which can be reused across different VSS execution. Each VSS execution consists of the dealer broadcasting one message to all receivers.

**Ideal Functionality $\mathcal{F}_{vss}$**. We describe the ideal VSS functionality in Figure 2. To our knowledge this is the first attempt of defining VSS through an ideal functionality. This is helpful in our context, as we can capture the weaker security provided by our non-hiding commitment scheme through the leakage query. Comparing with the VSS definitions in the literature, such as [23], we observe that our ideal functionality captures the fundamental properties intutively:

- **Correctness.** Correctness is captured as if there is no corruption, each party receives a correct share of the value $s$, shared by the dealer, which would verify correctly through the verification query.
- **Privacy (with leakage).** If the dealer is honest, and at most $t$ receivers are corrupt, then the value $s$ is hidden except the leakage (looking ahead, that basically comes from the dlog commitments) – this can be simulated through the leakage query. A correct simulation in this case means, the simulator is able to simulate dealer's message just from the leakage, and nothing else.
- **Strong commitment.** In this case the dealer is corrupt, and the simulator must detect a malformed message, which is not a correct sharing of a value $s$, from dealer. A good simulator should be able to capture this and then invoke the `Error` query to send $\perp$ to parties. If a corrupt dealer is able to cheat so that the simulator can not detect a malformed message, then the environment would be able to distinguish the output of honest parties from the real world.

$\mathcal{F}_{vss}$

The ideal functionality interacts with $n$ parties $P_1, \ldots, P_n$, and a dealer $P_D$. It also interacts with an ideal adversary or simulator $\mathcal{S}$. Furthermore, it is parameterized with a leakage function $L$, a threshold $t < n$ and a modulus $q$. It uses a list $T$ indexed by the sid such that each entry $T[\text{sid}]$ contains an $n+1$ tuple (e.g. $(s_0, \ldots, s_n)$) if sid is marked `Active` and $\perp$ otherwise; $s_i$ can be accessed as $T[\text{sid}][i]$.

- On receiving a query $(\text{sid}, \texttt{Share}, s)$ from dealer $P_D$:
  - If $P_D$ is corrupt, then skip. Else go to the next step.
  - Send $(\text{sid}, \texttt{Live})$ to $\mathcal{S}$.
  - When receives $(\text{sid}, \texttt{Distribute})$ from $\mathcal{S}$:
    - Check if sid is unmarked; if that is not true then do nothing and stop. Otherwise mark sid `Active` and go to the next step.
    - Choose a uniform random $t$ degree polynomial $P(x) \in \mathbb{Z}_q[x]^t$ subject to $P(0) = s_0$.
    - Append $(s_0, s_1, \ldots, s_n)$ into the list $T[\text{sid}]$.
    - Then for all $i \in [n]$ send $(\text{sid}, s_i)$ to each $P_i$.
- On receiving a query $(\text{sid}, \texttt{Share}, s)$ from simulator $\mathcal{S}$:
  - If $P_D$ is honest, then skip. Else go to the next step.
  - Check if sid is unmarked; if that is not true then do nothing and stop. Otherwise mark sid `Active` and go to the next step.
  - Choose a uniform random $t$ degree polynomial $P(x) \in \mathbb{Z}_q[x]^t$ subject to $P(0) = s_0$.
  - Append $(s_0, s_1, \ldots, s_n)$ into the list $T[\text{sid}]$.
  - Then for all $i \in [n]$ send $(\text{sid}, s_i)$ to each $P_i$.
- On receiving $(\text{sid}, \texttt{Error})$ from $\mathcal{S}$:
  - If $P_D$ is honest, then skip. Otherwise go to the next step.
  - If sid is marked, then do nothing, else mark it `Error` and send $\perp$ to all $P_i$.
- On receiving a query $(\text{sid}, \texttt{Verify}, \text{sid}, P_j)$ from $P_i$ or $\mathcal{S}$:
  - If sid is marked `Active`, then return `Success`, otherwise return `Failure`.
- On receiving a query $(\text{sid}, \texttt{Leakage})$ from $\mathcal{S}$, return $L(s)$.

**Figure 2: Ideal Functionality for VSS**

**Real World**. In the real world $n$ receivers $P_1, \ldots, P_n$ and a dealer $P_D$ interacts in a specific way. Let $pp \in \mathcal{PP}$ be a set of public parameters chosen from a specific set $\mathcal{PP}$, such that $pp$ includes a security parameter $\lambda \in \mathbb{N}$ integers $n, t \in \mathbb{N}$ $(n > t)$ and a modulus $q \in \mathbb{Z}$. Then consider the following algorithms:

- $\text{KeySetup}(pp) \rightarrow (sk, pk, \pi)$. The setup algorithm produces a key-pair and a proof that the pair is legitimate. For a party $P_i$, the corresponding values are denoted by $(sk_i, pk_i, \pi_i)$.
- $\text{KeyVer}(pp, (pk, \pi)) \rightarrow 1/0$. This algorithm verifies the legitimacy of a public-key $pk$ (that is whether the public-key owner indeed knows the secret key) with respect to the associated proof $\pi$.

- $\mathsf{Share}(pp, s) \rightarrow (\{s_i\}_{i\in[n]}, \mathsf{cmt})$. The sharing algorithm produces $t$ out of $n$ Shamir's secret shares of a value $s$ such that $(s_1 \ldots, s_n) = \mathsf{SSS}_{n,t,q}(s)$ and the associated commitment cmt.
- $\mathsf{ShareEnc}(pp, \mathsf{cmt}, \{s_i, pk_i\}_{i\in[n]}) \rightarrow (R, \{E_i\}_{i\in[n]}, \pi_{\mathsf{CS}})$. On input $n$ many shares $s_1, s_2, \ldots$, the associated commitment cmt, and corresponding public keys , this algorithm outputs a multi-ciphertext $(R, E_1, E_2, \ldots, E_n)$ with a common first element $R$ plus a proof of correct sharing $\pi_{\mathsf{CS}}$.
- $\mathsf{Verify}(pp, \mathsf{cmt}, R, \{E_i, pk_i\}_{i\in[n]}, \pi_{\mathsf{CS}}) \rightarrow 1/0$. This algorithm verifies the entire ciphertext tuple with respect to the proof $\pi_{\mathsf{CS}}$ and the commitment to output a decision bit.
- $\mathsf{ShareDec}(pp, sk_i, R, E_i) \rightarrow s_i$. The decryption algorithm uses a specific secret-key $sk_i$ to decrypt ciphertext $(R, E_i)$. Note that, only the party who posses $sk_i$ can decrypt $(R, E_i)$.

Now consider the protocol $\Pi_{\text{ni-vss}}$ described in Figure 3. Finally we provide the UC definition of a *leaky* NI-VSS prtocol.

*Definition 4.1 (Leaky Non-interactive Verifiable Secret Sharing (NI-VSS)).* Let $pp \in \mathcal{PP}$ be a set of public parameters chosen from a specific set $\mathcal{PP}$, such that $pp$ includes a security parameter $\lambda \in \mathbb{N}$ integers $n, t \in \mathbb{N}$ $(n > t)$ and a modulus $q \in \mathbb{Z}$. Then, we say that a protocol instantiation of $\Pi_{\text{ni-vss}}$ is called an $L$-leaky NI-VSS if it realizes the ideal functionality $\mathcal{F}_{\text{vss}}$ with leakage function $L$ that is, for every PPT adversary $\mathcal{A}$ in the real world, there is a PPT simulator $\mathcal{S}$ in the ideal world such that:

$$\text{Real}_{\mathcal{A},\Pi_{\text{ni-vss}}} \approx_c \text{Ideal}_{\mathcal{S},\mathcal{F}_{\text{vss}}}$$

A generic NI-VSS protocol

**Input and Output** The dealer has an input $s \in \mathbb{Z}_q$. No receiver has any input. After execution, each receiver $P_i$ has an output $y_i$, the dealer has no output.

- **Key Generation**. Each receiver $P_i$ runs $\mathsf{KeySetup}(pp)$ to generate $(sk_i, pk_i, \pi_i)$ and broadcasts $(pk_i, \pi_i)$.
- **Dealing**. The dealer $P_D$ receives $\{(pk_i, \pi_i)\}_{i\in[n]}$. It then runs for all $i \in [n]$: $\mathsf{KeyVer}(pp, pk_i, \pi_i)$. If KeyVer returns 0 for any $i$, exit. Otherwise, it executes the following steps.
  - $(\{s_i\}_{i\in[n]}, \{a_j\}_{j\in[t]}, \mathsf{cmt}) \leftarrow \mathsf{Share}(pp, s)$
  - Compute $(R, \{E_i\}_{i\in[n]}, \pi_{\mathsf{CS}}) \leftarrow \mathsf{ShareEnc}(pp, \{s_i, pk_i\}_{i\in[n]})$
  - Broadcast $(R, \{E_i\}_{i\in[n]}, \mathsf{cmt}, \pi_{\mathsf{CS}})$ to all receivers $\{P_i\}_{i\in[n]}$.
- **Receiving**. Each receiver $P_i$ for $i \in [n]$ performs the following steps:
  - For all $j$ such that $(j \in [n]) \wedge (j \neq i)$, run $\mathsf{KeyVer}(pp, pk_i, \pi_i)$. If KeyVer returns 0 for any $j$, then exit; otherwise go to the next step.
  - $e \leftarrow \mathsf{Verify}(pp, \mathsf{cmt}, R, \{E_i\}_{i\in[n]}, \pi_{\mathsf{CS}})$
  - If $e = 1$ then $s_i \leftarrow \mathsf{ShareDec}(pp, sk_i, R, E_i)$ and outputs $y_i \leftarrow s_i$ as its share corresponding to the dealing. Otherwise, if $e = 0$ reject dealing, and set $y_i \leftarrow \perp$.

**Figure 3: The general structure of an NI-VSS protocol.**

## 4.1 Our NI-VSS Protocol: cgVSS

In this section we provide a concrete instantiation of our NI-VSS protocol based on the multi-receiver encryption scheme (cf. Section 3.2), a corresponding proof of correct sharing (cf. Section 3.3) and a Schnorr's proof for knowledge of exponent (cf. Section 3.1. The instantiation is provided in Figure 4.

We state the following theorem:

Theorem 4.2. *Let $\Pi^{\text{cgVSS}}_{\text{ni-vss}}$ be a NI-VSS protocol instantiated with the* cgVSS *algorithms. Then the protocol $\Pi^{\text{cgVSS}}_{\text{ni-vss}}$ realizes the ideal functionality $\mathcal{F}_{\text{vss}}$ as long as as the underlying multi-receiver encryption scheme (cf. Section 3.2) is secure and the NIZK proof of correctness is a secure proof system (cf. Section 3.3).*

# 5 NI-DKG USING CLASS GROUPS

In an NI-DKG protocol, a number of parties engage in a one-round (non-interactive) protocol to jointly own a secret-key and corresponding public-key. In particular, in an $t$ out of $n$ threshold system at the end of the protocol, each party privately owns $k_i$ such that $(k_1, \ldots, k_n)$ forms a $t$ out of $n$ Shamir's secret-sharing of the secret-key $k$. The individual public keys $\bar{g}^{k_i}$ and the whole public-key $\bar{g}^k$ should be known to everyone, where $\bar{g}$ is a generator of a cyclic group $\bar{G}$ of prime order $q$. An NI-DKG protocol can be thought of as a symmetric extension of NI-VSS, with the crucial difference that no one knows the secret in NI-DKG. Indeed, following prior works (e.g. [32, 36, 46]), we construct NI-DKG by deploying our NI-VSS scheme in Figure 5. Our description directly uses the algorithms from the NI-VSS scheme, which is then instantiated with cgVSS in our implementation. The basic idea is each party $P_i$ now runs an NI-VSS instance using her own secret $z_i$; after the completion of protocol, $k_i$ is computed by *linearly combining* own share of $z_i$ with shares of $z_j$ received from other $P_j$.

## 5.1 Complexity analysis

We analyze the message and computational complexity of our cgVSS and present the performance evaluation using a reference implementation. Since the non-interactive VSS presented by Groth et al. [36] is the closest to our scheme, we compare our scheme against it. We provide a version of their VSS without forward secrecy for proper comparison (see Appendix A), we call it GrothVSS.

**Class Group NI-VSS**. In the cgVSS, the dealer encrypts the receiver shares $s_i$ as $(g^r, f^{s_i} h_i^r)$. The encryption is a multi-receiver encryption where the randomness $r$ is reused across the encryptions, with the total number of elements in the ciphertext being $n + 1$ elements. With $\beta$ bits for each element, the total ciphertext size is $(n+1) \cdot \beta$ bits. For the $n+1$ encryptions, the dealer takes $O(n)$ time. The dealer also generates a NIZK proof of correct sharing and forwards it to all the receivers. The proof consists of two elements from the class group, one elliptic curve element and two scalars. Let the length of the NIZK proof be $k$.

Each receiver decrypts their share and also verifies the correctness of sharing by the dealer. The receiver $i$ first ElGamal decrypts the exponentiated share $f^{s_i}$ and solves the discrete-log problem to obtain $s_i$. They also verify the NIZK proof forwarded by the dealer. The decryption and the verification of the proof by the receiver take $O(1)$ time.

cgVSS-Algorithms

- **Ingredients.** The NI-VSS algorithms described below uses the following ingredients.
  - A Schnorr proof of knowledge-of-exponent with algorithms (cf. Section 3.1) (Kex.Prove, Kex.Ver) and public parameters $pp_{\mathsf{Kex}}$.
  - A multi-receiver encryption scheme (cf. Section 3.2) with algorithms (CGE.KeyGen, CGE.mrEnc, CGE.Dec) and public parameters $pp_{\mathsf{CG}}$.
  - An associated proof system of correct sharing (cf. Section 3.3) with algorithms (SharingProof, SharingVerify) and public parameters $pp_{\mathsf{PoC}}$.
- **Public parameters.** The public parameter $pp$ is defined as $pp \leftarrow \{pp_{\mathsf{CG}}, pp_{\mathsf{PoC}}, pp_{\mathsf{Kex}}\}$.

**Construction**

- $\mathsf{KeySetup}(pp) \rightarrow (sk, pk, \pi)$:
  - $(sk, h) \leftarrow \mathsf{CGE.KeyGen}(pp_{\mathsf{CG}})$.
  - $\pi \leftarrow \mathsf{Kex.Prove}(pp_{\mathsf{Kex}}, h, sk)$.
  - Set $pk \leftarrow h$.
- $\mathsf{KeyVer}(pp, (pk, \pi)) \rightarrow 1/0$:
  - Output $\mathsf{Kex.Ver}(pp_{\mathsf{Kex}}, pk, \pi)$.
- $\mathsf{Share}(pp, s) \rightarrow (\{s_i\}_{i\in[n]}, \mathsf{cmt})$:
  - Sample $a_j \xleftarrow{\$} \mathbb{Z}_q, j \in [t]$.
  - Set $s_0 \leftarrow s$.
  - Define $P(x) = a_0 + a_1 x + \ldots + a_t x^t$.
  - For each $i \in [n]$: set $s_i \leftarrow P(i)$.
  - Compute for all $j \in \{0, \ldots, t\}$: $A_j \leftarrow \bar{g}^{a_j}$.
  - Set $\mathsf{cmt} \leftarrow \{A_0, \ldots, A_t\}$.
- $\mathsf{ShareEnc}(pp, \mathsf{cmt}, \{s_i, pk_i\}_{i\in[n]}) \rightarrow (R, \{E_i\}_{i\in[n]}, \pi_{\mathsf{CS}})/\perp$.
  - For all $i \in [n]$: $e_i \leftarrow \mathsf{Kex.Ver}(pp_{\mathsf{Kex}}, h_i, \pi_i)$ (as $h_i = pk_i$).
  - If *any* $e_i = 0$, output $\perp$. Otherwise do as follows:
    - Sample $r \xleftarrow{\$} \mathcal{D}_q$
    - Compute $(R, \{E_i\}_{i\in[n]}) \leftarrow \mathsf{CGE.mrEnc}(pp_{\mathsf{CG}}, \{h_i, s_i; r\}_{i\in[n]})$.
    - Define:
      - $\mathsf{inst} = \left(\{h_i\}_{i\in[n]}, (R, \{E_i\}_{i\in[n]}), \mathsf{cmt}\right)$.
      - $\mathsf{wit} = ((s_1, \ldots, s_n), r)$.
    - Compute $\pi_{\mathsf{CS}} \leftarrow \mathsf{SharingProof}(pp_{\mathsf{PoC}}, \mathsf{inst}, \mathsf{wit})$.
- $\mathsf{Verify}(pp, \mathsf{cmt}, R, \{E_i, pk_i\}_{i\in[n]}, \pi_{\mathsf{CS}}) \rightarrow 1/0$:
  - Parse $\mathsf{inst} \leftarrow \left(\{h_i\}_{i\in[n]}, (R, \{E_i\}_{i\in[n]}), \mathsf{cmt}\right)$.
  - Output $\mathsf{SharingVerify}(pp_{\mathsf{PoC}}, \mathsf{inst}, \pi_{\mathsf{CS}})$.
- $\mathsf{ShareDec}(pp, sk_i, R, E_i) \rightarrow s_i$:
  - Compute $s_i \leftarrow \mathsf{CGE.Dec}(pp_{\mathsf{CG}}, sk_i, R, E_i)$.

**Figure 4: Algorithms that constitute** cgVSS

cgDKG

- **Key Generation.** Every party has access to the public parameters $pp$. Each $P_i$ runs $(sk_i, pk_i, \pi_i) \leftarrow \mathsf{KeySetup}(pp)$ and broadcasts $(pk_i, \pi_i)$.
- **Dealing.** Each party $P_i$ receives $\{(pk_j, \pi_j)\}_{j\in[n]\wedge j\neq i}$. It then runs for all $j \in [n] \wedge j \neq i$: $\mathsf{KeyVer}(pp, pk_j, \pi_j)$. If KeyVer returns 0 for any $j$, exit. Otherwise it executes the following steps.
  - $z_i \xleftarrow{\$} \mathbb{Z}_q$.
  - $(\{s_{ij}\}_{j\in[n]}, \mathsf{cmt}_i) \leftarrow \mathsf{Share}(pp, z_i)$.
  - $(R_i, \{E_{ij}\}_{j\in[n]}, \pi_{\mathsf{CS}i}) \leftarrow \mathsf{ShareEnc}(pp, \mathsf{cmt}, \{s_{ij}, pk_j\}_{j\in[n]})$
  - Broadcast $(R_i, \{E_{ij}\}_{j\in[n]}, \mathsf{cmt}_i, \pi_{\mathsf{CS}i})$.
- **Receiving.** Each party $P_i$ receives $n-1$ tuples:

  $$\{(R_j, \{E_{jj'}\}_{j'\in[n]}, \mathsf{cmt}_j, \pi_{\mathsf{CS}j})\}_{j\in[n]\wedge j\neq i}$$

  then execute the following steps.
  - For all $j \in [n] \wedge j \neq i$: compute

    $$e_j \leftarrow \mathsf{Verify}(pp, \mathsf{cmt}_j, R_j, \{E_{jj'}\}_{j'\in[n]}, \pi_{\mathsf{CS}j})$$
  - Let $U$ consist of $j$ if and only if $e_j = 1$.
  - $|U| \leq t$ then exit. Otherwise, go to the next step.
  - Initialize $k_i \leftarrow 0$
  - For all $j \in U$:
    - $s_{ji} \leftarrow \mathsf{ShareDec}(pp, sk_i, R_j, E_{ji})$.
    - $k_i \leftarrow k_i + s_{ji}$
  - Define its share to be $k_i$ and individual public-key as $\bar{g}^{k_i}$.
  - To compute the system public-key initialize $y = 1 \in G$ and for each $j \in U$:
    - Parse $\mathsf{cmt}_j$ as $\{A_{0j}, \ldots, A_{tj}\}$.
    - $y = y \cdot A_{0j}$.
  - Output $y$ as system public key.

**Figure 5:** cgDKG- **Non-interactive distributed key generation using class groups**

For the DKG protocol, cgDKG each party acts as the dealer and performs cgVSS. For $n$ parties, the total ciphertext length broadcast in the system is $O(n \cdot (n+1)\beta) \sim O(\beta n^2)$ bits while the NIZK proof length is $nk$. After the dealing phase, the receivers compute the secret key from the first $t+1$ valid sharings.

**Groth's NI-VSS**. In the GrothVSS, the dealer generates the shares $s_i$ and divides each share $s_i$ into $m$ chunks. Thus there are a total of $m \cdot n$ chunks for each dealer, for $n$ parties. The dealer encrypts each of the chunks using ElGamal encryption. He reuses the randomness $r_j$ across encryptions of the chunks as $(\bar{g}_1^{r_j}, \bar{h}_i^{r} \cdot \bar{g}_1^{m_{ij}}) \forall i \in [0, n]$. Each of the chunks is individually encrypted, the total time taken for the encryption is $O(mn)$. Thus the cipher text generated by the dealer consists of $mn + m$ group elements. The total length of the ciphertext is $(m(n+1) \cdot \alpha)$ bits with $\alpha$ bits for each element.

For the proof of correctness, each dealer generates proof of correct sharing and the proof of correct chunking. The proof of chunking involves showing that each chunk of the share is smaller than a certain value. For the proof of correct sharing, the sender forwards three group elements of groups $G_1, G_2$ and 2 group $\mathbb{Z}_q$ elements. For the proof of chunking the dealer uses approximate range proofs for which the dealer forwards a set of elements, including $2\ell + 2$ group elements and $\ell + n + 1$ masked values of the chunks for a parameter $\ell$.

Each party decrypts the $m$ chunks corresponding to their share to compute their share value. First, the chunks in the exponentiated form are ElGamal decrypted, and the chunk value is solved for, using the Baby Step - Gaint step algorithm. This leads to a decryption time of $O(m)$ (ElGamal and Baby Step - Gaint Step) per receiver. For

the DKG protocol, each dealer performs the VSS, and the receivers compute the secret key from the first $t+1$ valid sharings. For $n$ dealers, the total broadcast message length is $O(n \cdot m(n+1) \cdot \alpha) \sim O(\alpha m n^2)$ bits per dealer in the DKG protocol.

## 6 EXPERIMENTATION AND PERFORMANCE ANALYSIS

**Implementation and Setup**. We implement cgVSS in C++ using the BICYCL library [11] for class groups, Miracl C++ library for cryptographic operations with ~ 1858 lines of code. For comparison, we adapt and realize a version of the implementation of GrothVSS without forward secrecy in Rust in ~ 4178 lines of code [3].

We run the experiments with each node realized on a Google Cloud Platform (GCP) instance with an Intel Xeon 2.8GHz CPU with 16 cores and 16GB RAM. We use HotStuff state machine replication [56] to realize the broadcast. Our SMR instance is realized over four GCP instances separate from the DKG nodes. All the reported timings are averages over 10 runs of the protocols.

**Computation Overhead**. Figure 7 shows the time taken by the dealer and the receiver in the cgVSS protocol. The dealer's computation time includes the time to generate the multi-receiver ciphertext and the NIZK proof of correctness whereas a receiver's computation time includes the decryption time and the time for proof verification. We use multi-exponentiation to compute the product of multiple exponentiated values in the generation and verification procedures of the proof of correct sharing. For a 100 party system, the dealer takes 117 msec for generating the ciphertext and 230 msec to generate the proof, whereas for a 150 party system, it takes 176 msec for encryption and 312msec for the proof generation. The decryption takes 38 msec, while the proof verification takes 661 msec for a 100 user system and 1.18 sec for a 150 party system (the decryption time stays the same irrespective of the number of parties). Figure 7b shows the total receiver times taken by the party to verify the sharing and decrypt their shares.

**Computation Overhead**. Figure 7 shows the time taken by the dealer and the receiver in the cgVSS protocol. The dealer's computation time includes the time to generate the multi-receiver ciphertext and the NIZK proof of correctness whereas a receiver's computation time includes the decryption time and the time for proof verification. We use multi-exponentiation to compute the product of multiple exponentiated values in the generation and verification procedures of the proof of correct sharing. For a 100 party system, the dealer takes 1.22sec for generating the ciphertext and 734msec to generate the proof, whereas for a 150 party system, it takes 1.80sec for encryption and 377msec for the proof generation. The decryption takes 38msec, while the proof verification takes 734msec for a 100 user system and 1.23sec for a 150 party system (the decryption time stays the same irrespective of the number of parties). Figure 7b shows the total receiver times taken by the party to verify the sharing and decrypt their shares.

In GrothVSS, to encrypt a share value, (assume) each share is divided into 24 chunks and encrypted individually. The ElGamal encryption constitutes two group elements; however, since the randomness is re-used across different users, the total number of

[3] https://github.com/Entropy-Foundation/class-group

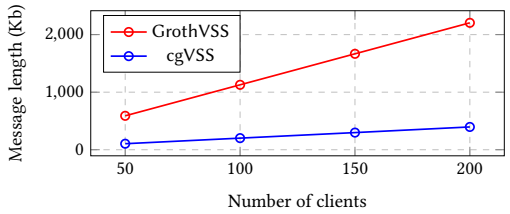
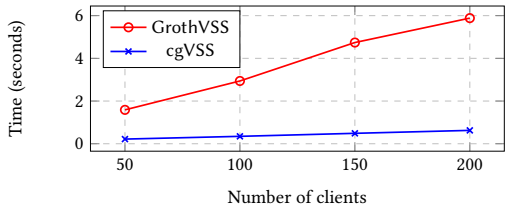
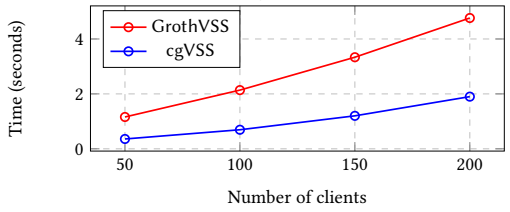



**Figure 6: Comparison of broadcast (dealing) message length where $n = 2t + 1$. cgVSS dealing consists of encryptions and proof of correct sharing, while GrothVSS also consists of proof of correct chunking.**




**(a) Comparison of dealer times. cgVSS dealer time consists of times for encryption and proof of correct sharing, while GrothVSS also involves proof of correct chunking.**




**(b) Comparison of receiver times. cgVSS receiver time consists of decryption time and verification of correct sharing, while GrothVSS also involves verification of correct chunking.**

**Figure 7: Comparison of dealer and receiver times for cgVSS and GrothVSS.**

elements for randomness is 24, amounting to $24 * 381 = 9144$ bits. For $n$ users, the total bit-length of ciphertexts is $9144 \cdot (n+1)$, including the random values. The dealer also commits to the $t$ coefficients of the polynomial, which amount to $257 \cdot t$. The dealer generates the NIZK proof of correctness of sharing, which constitutes 3 multiplicative group elements and two scalars of 381 bits each. GrothVSS uses the BLS12-381 curve, and hence the elements are 381 bits each. The dealer also generates proof of the correctness of chunking by showing that each 'chunk' is in a small range of values. For this, an approximate range proof is employed where the dealer forwards a set of elements, including $2\ell + 2$ group elements for a parameter $\ell$ and $\ell + n + 1$ masked values of the chunks. Taking a conservative estimate of 32 bits for the masked chunk value summations, we have the total bit-length of the approximate range proof to be $(2\ell + 2) \cdot 381 + (\ell + n + 1) \cdot 32$.

The total bit-length of the broadcast message (see Figure 6) for GrothVSS for a 150 party is 1.66Mb. This indicates a 5.6x improvement in total broadcast message length while using cgVSS when

compared to GrothVSS for a 150 party system. The comparison also indicates that the broadcast message length increases slower in cgVSS when compared to GrothVSS. In GrothVSS, for a 100 party system, the dealer takes 1.36sec for generating ciphertexts, 68msec for generating the proof of correct sharing, and 2.41sec for generating proof of correct chunking, whereas the corresponding numbers for a 150 party system are 2.03sec, 101msec and 3.92 sec respectively. For decrypting their share, each receiver decrypts all the corresponding chunks, which amounts to 338msec. For verification, in a 100 party system, the receiver takes 341msec for proof of correct sharing and 1.32sec for proof of correct chunking; for a 150 party system, the receiver takes 804msec for proof of correct sharing and 2.00sec for the proof of correct chunking.

To also give a sense of how the scheme compares to other existing state-of-the-art PVSS schemes, we briefly mention the timing reported by Gentry et al. [35] for their LWE-based PVSS scheme. We present their reported numbers, though their performance has been evaluated on a more powerful machine (with 32 cores and 250GB RAM) compared to our benchmarks (10 core 16GB RAM machine). For 128 parties, their system takes 4.2sec for generating ciphertexts and 22.9sec for generating the proof of correctness of sharing totaling 27.1sec of dealer time, whereas for 256 parties, the total dealer time is 28.1sec. The receiver takes 1.4msec to decrypt and 15.3sec to verify the dealing totaling 15.301sec. The total receiver time for 256 parties is 15.901sec.

**End-to-end Protocol Analysis**. We implement the cgDKG and GrothDKG protocols and compare them. Figure 8 compares the time taken by each node in each DKG instance; it is the time taken from the start of dealing to the computation of the system public key after verifying $t+1$ valid dealings. The nodes publish the encrypted shares and commitments using the HotStuff [56] SMR. The SMR is realized separately from the DKG nodes, which communicate with the SMR through RPC calls. For 10 nodes, GrothDKG takes 3.434 seconds, with cgDKG taking 2.656 seconds. For a 50 node network, GrothDKG takes 43.058 seconds while cgDKGtakes 17.950 seconds. From Figure 7 and Figure 8, it can be observed that the SMR takes significant time in the overall end-to-end scenario, and the optimizations in SMR usage (block rate, dummy blocks etc) would improve the performance.

Our performance analysis demonstrates that cgDKG protocol is efficient and continues to perform significantly better than the GrothDKG with an increasing number of nodes in the system. Moreover, as we improve class-group implementation in the future, we expect the performance of our cgDKG to improve further.

# 7 ASYNCHRONOUS VSS AND DKG

In the asynchronous communication setting, the adversary controls the communication links and may delay, or re-order messages between any two honest parties as long as it eventually delivers all the messages by honest parties. In this section, we discuss an easy extension of our VSS and DKG to the asynchronous communication setting. We first propose a new asynchronous VSS (AVSS) scheme using our NI-VSS and any reliable broadcast protocol [12] and then develop an asynchronous DKG (ADKG) using our AVSS scheme and asynchronous agreement ideas from the recent ADKG protocols by Das et al. [25, 27].

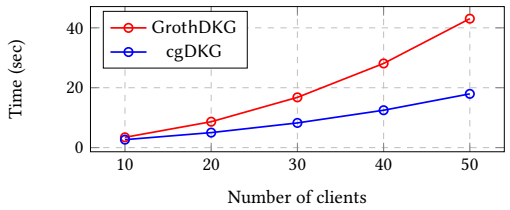



**Figure 8: Comparison of time taken to perform a DKG. GrothDKG is realized using GrothVSS where each party acts as a dealer and runs an instance of GrothVSS. The times reported are aggregates of time taken from starting of dealing and computation of public key by each node, across nodes; 10 such DKG runs are aggregated.**

## 7.1 Asynchronous VSS using Class Groups

In the asynchronous communication setting, Cachin et al. [14] proposed the first asynchronous verifiable secret sharing (AVSS) protocol with computational security relevant to threshold cryptography in 2002. Several works have reduced the communication complexity of AVSS process over the last two decades. [6, 26, 58] Relevant to threshold signing for state-machine replication (SMR) protocols, there also have been efforts to define high-threshold VSS schemes [4, 25, 27], where the secret sharing threshold $t$ can be doubled. Now, we describe an easy way to develop an AVSS protocol using NI-VSS.

The non-interactive nature of cgVSS makes the process of designing an AVSS significantly easy: A trivial approach of reliably broadcasting the NI-VSS vector is sufficient. Given the linear size of the vector, it is ideal to use the communication-balanced reliable broadcast primitives such as [3, 15, 26]. This will reduce the communication complexity of AVSS to $O(n^2\kappa)$ bits. In this straightforward approach, the nodes do not verify the correctness of sharing until they deliver the sharing in the deliver/output step of a reliable broadcast. However, in practice, it will be better not to leave the NI-VSS verification until the end. Instead, every node should verify the correctness of sharing the first time it receives/computes the entire NI-VSS vector and not proceed with the reliable broadcast instance if the verification fails. Notice that, in the asynchronous communication setting, similar to reliable broadcast, termination is not guaranteed for AVSS.

## 7.2 Asynchronous DKG using Class Groups

Kate et al. [39] combined AVSS by Cachin et al. [14] with the PBFT flow [22] towards developing a DKG beyond the bounded-synchronous setting. However, their approach makes the partial-synchrony communication assumption. While it is possible to employ a randomized Byzantine agreement primitive towards working in the asynchronous setting, generating common coins required for the randomized protocol itself requires DKG-like primitives. This seems to create a circular requirement.

Recently, in a seminal work, Kokoris-Kogias et al. [42] offer a novel efficient way towards breaking the circularity condition and propose a quartic communication complexity DKG protocol in the asynchronous communication setting. Improved asynchronous

DKG (ADKG) constructions are already available that reduce communication complexity to be cubic in the number of parties [2, 27] as well as to allow high-threshold secret sharing [25, 27].

These papers indeed make developing asynchronous DKG based on our NI-VSS significantly easy. A straightforward approach is to replace the employed AVSS (or its high-threshold version) with above mentioned AVSS based on class groups and then employ the agreement on a common subset procedure as it is from [25, 27]. This offers a cubic communication complexity ADKG. Nevertheless, in the future, it will be interesting to improve this agreement process and ADKG as well.

## ACKNOWLEDGEMENTS

We thank Adithya Bhat and Ioanna Karantaidou for participating in the early conversations on non-interactive VSS and DKG. We also thank Dan Boneh for the encouraging discussion on using the class-group setting for VSS.

## REFERENCES

[1] Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., and Stern, G. Bingo: Adaptively secure packed asynchronous verifiable secret sharing and asynchronous distributed key generation. *Cryptology ePrint Archive* (2022).

[2] Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., and Tomescu, A. Reaching consensus for asynchronous distributed key generation. In *PODC'21* (2021), pp. 363–373.

[3] Alhaddad, N., Das, S., Duan, S., Ren, L., Varia, M., Xiang, Z., and Zhang, H. Balanced byzantine reliable broadcast with near-optimal communication and improved computation. In *ACM PODC* (2022), A. Milani and P. Woelfel, Eds., ACM, pp. 399–417.

[4] Alhaddad, N., Varia, M., and Zhang, H. High-threshold AVSS with optimal communication complexity. In *FC* (2021), N. Borisov and C. Diaz, Eds., pp. 479–498.

[5] Bacho, R., and Loss, J. On the adaptive security of the threshold bls signature scheme. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (2022), pp. 193–207.

[6] Backes, M., Datta, A., and Kate, A. Asynchronous computational VSS with reduced communication complexity. In *CT-RSA* (2013), pp. 259–276.

[7] Backes, M., Kate, A., and Patra, A. Computational verifiable secret sharing revisited. In *Advances in Cryptology–ASIACRYPT* (2011), pp. 590–609.

[8] Blakley, G. R. Safeguarding cryptographic keys. In *1979 International Workshop on Managing Requirements Knowledge (MARK)* (1979), pp. 313–318.

[9] Boudot, F., and Traoré, J. Efficient publicly verifiable secret sharing schemes with fast or delayed recovery. In *Information and Communication Security* (1999), pp. 87–102.

[10] Boudot, F., and Traoré, J. Efficient publicly verifiable secret sharing schemes with fast or delayed recovery. In *Information and Communication Security: Second International Conference, ICICS'99, Sydney, Australia, November 9-11, 1999. Proceedings 2* (1999), pp. 87–102.

[11] Bouvier, C., Castagnos, G., Imbert, L., and Laguillaumie, F. I want to ride my bicycl: Bicycl implements cryptography in class groups. Cryptology ePrint Archive, Paper 2022/1466, 2022. https://eprint.iacr.org/2022/1466.

[12] Bracha, G. An asynchronous [(n - 1)/3]-resilient consensus protocol. In *ACM PODC* (1984).

[13] Braun, L., Damgård, I., and Orlandi, C. Secure multiparty computation from threshold encryption based on class groups. To appear at Crypto 2023, 2022.

[14] Cachin, C., Kursawe, K., Lysyanskaya, A., and Strobl, R. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security* (2002), pp. 88–97.

[15] Cachin, C., and Tessaro, S. Asynchronous veri.able information dispersal. In *IEEE SRDS* (2005), pp. 191–202.

[16] Camenisch, J., and Shoup, V. Practical verifiable encryption and decryption of discrete logarithms. In *Annual International Cryptology Conference* (2003), Springer, pp. 126–144.

[17] Canetti, R., Gennaro, R., Goldfeder, S., Makriyannis, N., and Peled, U. Uc non-interactive, proactive, threshold ecdsa with identifiable aborts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2020), Association for Computing Machinery.

[18] Cascudo, I., and David, B. Scrape: Scalable randomness attested by public entities. In *ACNS* (2017), pp. 537–556.

[19] Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., and Tucker, I. Two-party ecdsa from hash proof systems and efficient instantiations. In *Annual International Cryptology Conference* (2019), Springer, pp. 191–221.

[20] Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., and Tucker, I. Bandwidth-efficient threshold ec-dsa. In *23rd IACR International Conference on Practice and Theory of Public-Key Cryptography* (2020), pp. 266–296.

[21] Castagnos, G., and Laguillaumie, F. Linearly homomorphic encryption from ddh. In *Cryptographers' Track at the RSA Conference* (2015), Springer, pp. 487–505.

[22] Castro, M., and Liskov, B. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst. 20*, 4 (2002), 398–461.

[23] Chandramouli, A., Choudhury, A., and Patra, A. A survey on perfectly secure verifiable secret-sharing. *ACM Comput. Surv. 54*, 11s (2022), 232:1–232:36.

[24] Chor, B., Goldwasser, S., Micali, S., and Awerbuch, B. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *FOCS* (1985), p. 383–395.

[25] Das, S., Xiang, Z., Kokoris-Kogias, L., and Ren, L. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. Cryptology ePrint Archive, Paper 2022/1389. To appear at Usenix Sec 2023, 2022.

[26] Das, S., Xiang, Z., and Ren, L. Asynchronous data dissemination and its applications. In *ACM CCS* (2021), Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds., pp. 2705–2721.

[27] Das, S., Yurek, T., Xiang, Z., Miller, A., Kokoris-Kogias, L., and Ren, L. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy (SP)* (2022), IEEE, pp. 2518–2534.

[28] Desmedt, Y. G. Threshold cryptography. *European Transactions on Telecommunications 5*, 4 (1994), 449–458.

[29] D'Souza, R., Jao, D., Mironov, I., and Pandey, O. Publicly verifiable secret sharing for cloud-based key management. In *INDOCRYPT 2011: 12th International Conference on Cryptology 2011* (2011), Springer, pp. 290–309.

[30] Fouque, P.-A., and Stern, J. One round threshold discrete-log key generation without private channels. In *Public Key Cryptography* (2001), pp. 300–316.

[31] Fujisaki, E., and Okamoto, T. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In *Advances in Cryptology — EUROCRYPT'98* (1998), pp. 32–46.

[32] Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT'99* (1999), p. 295–310.

[33] Gennaro, R., Rabin, M. O., and Rabin, T. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *ACM PODC* (1998), p. 101–111.

[34] Gentry, C., Halevi, S., and Lyubashevsky, V. Practical non-interactive publicly verifiable secret sharing with thousands of parties. In *Advances in Cryptology—EUROCRYPT* (2022), pp. 458–487.

[35] Gentry, C., Halevi, S., and Lyubashevsky, V. Practical non-interactive publicly verifiable secret sharing with thousands of parties. In *Advances in Cryptology–EUROCRYPT 2022* (2022), Springer, pp. 458–487.

[36] Groth, J. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Paper 2021/339, 2021. https://eprint.iacr.org/2021/339.

[37] Groth, J., and Shoup, V. Design and analysis of a distributed ecdsa signing service. Cryptology ePrint Archive, Paper 2022/506, 2022. https://eprint.iacr.org/2022/506.

[38] Heidarvand, S., and Villar, J. L. Public verifiability from pairings in secret sharing schemes. In *Selected Areas in Cryptography* (Berlin, Heidelberg, 2009), R. M. Avanzi, L. Keliher, and F. Sica, Eds., Springer Berlin Heidelberg, pp. 294–308.

[39] Kate, A., Huang, Y., and Goldberg, I. Distributed key generation in the wild. *Cryptology ePrint Archive* (2012).

[40] Katz, J. Round optimal robust distributed key generation. Cryptology ePrint Archive, Paper 2023/1094, 2023. https://eprint.iacr.org/2023/1094.

[41] Kokoris Kogias, E., Malkhi, D., and Spiegelman, A. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. Association for Computing Machinery.

[42] Kokoris Kogias, E., Malkhi, D., and Spiegelman, A. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (2020), pp. 1751–1767.

[43] Komlo, C., Goldberg, I., and Stebila, D. A formal treatment of distributed key generation, and new constructions. *Cryptology ePrint Archive* (2023).

[44] Neji, W., Blibech, K., and Ben Rajeb, N. Distributed key generation protocol with a new complaint management strategy. *Security and Communication Networks 9*, 17 (2016), 4585–4595.

[45] Pedersen, T. P. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology—CRYPTO* (1991), pp. 129–140.

[46] Pedersen, T. P. A threshold cryptosystem without a trusted party. In *Workshop on the Theory and Application of of Cryptographic Techniques* (1991), Springer, pp. 522–526.

[47] Ruiz, A., and Villar, J. L. Publicly verifiable secret sharing from paillier's cryptosystem. In *WEWoRC 2005 – Western European Workshop on Research in Cryptology* (2005), pp. 98–108.

[48] Ruiz, A., and Villar, J. L. Publicly verifiable secret sharing from paillier's cryptosystem.
[49] Schnorr, C. P. Efficient identification and signatures for smart cards. In *Advances in Cryptology — CRYPTO' 89* (1990), G. Brassard, Ed., pp. 239–252.
[50] Schoenmakers, B. A simple publicly verifiable secret sharing scheme and its application to electronic. In *Advances in Cryptology—CRYPTO* (1999), pp. 148–164.
[51] Shamir, A. How to share a secret. *Communications of the ACM 22*, 11 (1979), 612–613.
[52] Stadler, M. Publicly verifiable secret sharing. In *Advances in Cryptology—EUROCRYPT* (1996), pp. 190–199.
[53] Thyagarajan, S. A. K., Castagnos, G., Laguillaumie, F., and Malavolta, G. Efficient cca timed commitments in class groups. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (2021), CCS '21.
[54] Wesolowski, B. Efficient verifiable delay functions. In *Advances in Cryptology–EUROCRYPT 2019* (2019), pp. 379–407.
[55] Wu, T.-Y., and Tseng, Y.-M. A pairing-based publicly verifiable secret sharing scheme. *Journal of systems science and complexity 24*, 1 (2011), 186–194.
[56] Yin, M., Malkhi, D., Reiter, M. K., Gueta, G. G., and Abraham, I. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing* (2019), pp. 347–356.
[57] Young, A., and Yung, M. A pvss as hard as discrete log and shareholder separability. In *Public Key Cryptography* (2001), pp. 287–299.
[58] Yurek, T., Luo, L., Fairoze, J., Kate, A., and Miller, A. K. hbacss: How to robustly share many secrets. In *NDSS* (2022).

# A GROTHS NI-VSS WITHOUT FORWARD SECRECY

[36] Groth et al. [36] present non-interactive VSS and DKG protocols that involve ElGamal encryption of share values. The authors propose a VSS protocol that offers forward secrecy using binary tree encryption. However, here we present a version of their VSS protocol without forward secrecy; we call it GrothVSS in this paper.

Let $pp$ be a set of public parameters everyone can access. $pp = \{\bar{g}_1, \bar{g}_2, \mathbb{G}_1, \mathbb{G}_2, H, H'\}$. Here, the generators $\bar{g}_1, \bar{g}_2$ are generators of prime order groups $\mathbb{G}_1, \mathbb{G}_2$ of order $q$ and $H, H' : \{0,1\}^* \rightarrow \mathbb{Z}_q$. The GrothVSS protocol consists of a tuple of algorithms (DLKeySetup, Share, ElShareEnc, ElShareDec for the share encryption mechanism and ElSharingProof, ElSharingVerify) for the generation and verification of proof of correct sharing. They are presented in Figure 10, Figure 9. The algorithms are for key generation, generating shares, encrypting and decrypting the shares, generating proof of correct sharing, and verification of all the sharing respectively. Before the start of the protocol, each party runs the DLKeySetup to sample a secret-public keys pair along with the proof of knowledge of the secret key corresponding to the public key. Each party $P_i$ runs the algorithm to generate $(sk_i, \bar{h}_i, \pi_i)$, and the proof $\pi_i$ is forwarded to all the parties before the start of the protocol. GrothVSS follows the same mechanics as cgVSS of Figure 3. Here, we present informally the variants of the algorithms used for GrothVSS in Figure 10, Figure 9.

The dealer runs the Share algorithm that generates the shares of each party $P_i$ as evaluations on a random $t$-degree polynomial $a(y) = \sum_{k=0}^{t} a_k y^k$. The shares of computed as $s_i = a(i) \in \mathbb{Z}_q$. The dealer 'exponentiated/lifted' ElGamal encrypts the share value $s_i$ of $P_i$ using the public key $pk_i = \bar{h}_i$ as $(\bar{g}_1^{r_i}, \bar{g}_1^{s_i} \bar{h}_i^{r_i})$. However, the discrete logarithm problem is intractable in the underlying group $\mathbb{G}$; hence, the receiver can not decrypt the value $s_i$ if it is forwarded in the exponentiated form as $\bar{g}_2^{s_i}$ directly. To overcome this, the dealer breaks the value $s_i$ into $m$ smaller 'chunks' $s_{i,u} < B, u \in [m]$ such that $\sum_u B^{u-1} s_{i,u} = s_i$. Essentially, the concatenation of bits of $s_{i,u}$ form the value $s_i$. The dealer encrypts each of the smaller chunks in the form $(\bar{g}_1^{r_u}, \bar{g}_1^{s_{i,u}} \bar{h}_i^{r_u}), i \in [n], u \in [m]$. The algorithm ElShareEnc realizes the chunking and the encryption procedure. The party $P_i$ uses the ElShareDec to decrypt their share. When the party $P_i$ receives the encryption of the value $\bar{g}_1^{s_{i,u}}$, decrypts it and uses a solver to compute the value $s_{i,u}$. They concatenate the values $s_{i,u}$ to compute the share $s_i$.

**GrothVSS-Proof of correct sharing**

Every party has access to the public parameters $pp$. Each party $P_i$ runs the key setup to generate the secret key - public key pairs $(sk_i, \bar{h}_i)$ and the NIZK proof of knowledge $\pi_{\text{DL},i}$.

- ElSharingProof$(pp, r, \{s_i, A_j\}_{i\in[n], j\in\{0,\cdots,t\}}) \rightarrow \pi_{\text{PoC}}$ :
  - Sample $\alpha, \rho \xleftarrow{\$} \mathbb{Z}_q$,
  - Compute and set
    - $W \leftarrow \bar{g}_1^{\rho}, X \leftarrow \bar{g}_2^{\alpha}$
    - $\gamma \leftarrow H(\{\bar{h}_i, A_j\}_{i\in[n], j\in\{0,\cdots,t\}})$
    - $Y \leftarrow (\prod_{i=1}^{n} \bar{h}_i^{\gamma^i})^{\rho} \bar{g}_1^{\alpha}$
    - $\gamma' \leftarrow H'(\gamma, W, X, Y)$
    - $z_r \leftarrow r\gamma' + \rho (\in \mathbb{Z}_q)$
    - $z_\alpha \leftarrow \gamma' \sum_{i=1}^{n} s_i \gamma^i + \alpha (\in \mathbb{Z}_q)$
    - $\pi_{\text{PoC}} \leftarrow (W, X, Y, z_r, z_\alpha)$
- ElSharingVerify $(pp, \pi_{\text{PoC}}, \{R_u, E_{i,u}\}_{i\in[n], u\in[m]}) \rightarrow 0/1$:
  - Compute and set
    - $c = \prod_{u=1}^{m} c_u^{B^{u-1}}$
    - $d_i = \prod_{u=1}^{m} d_{i,u}^{B^{u-1}}$
    - $\gamma \leftarrow H(\{\bar{h}_i, A_j\}_{i\in[n], j\in\{0,\cdots,t\}})$
    - $\gamma' \leftarrow H'(\gamma, W, X, Y)$
  - Verify
    - $c^{\gamma'} W \stackrel{?}{=} \bar{g}_1^{z_r}$
    - $(\prod_{j=0}^{t} A_j^{\sum_{i=1}^{n} i^j \gamma^i})^{\gamma'} X \stackrel{?}{=} \bar{g}_2^{z_\alpha}$
    - $(\prod_{i=1}^{n} d_i^{\gamma^i})^{\gamma'} Y \stackrel{?}{=} \prod_{i=1}^{n} (\bar{h}_i^{\gamma^i})^{z_r} \bar{g}_1^{z_\alpha}$

**Figure 9: Proof system of correct sharing of GrothVSS[36]. We do not present the proof and verification of correct chunking here, refer [36, Section 6.5] for it.**

## A.1 Proof of correct sharing

Here we present the proof of the correctness of sharing of the NI-DKG protocol by Groth et al. [36]. The dealer publishes the commitments $A_i = \bar{g}_2^{a_i}$ to coefficients of the polynomial $a(y) = \sum_{k=0}^{t} a_k y^k$ from which the shares of the nodes have been generated. The dealer generates the proof of correctness $\pi_{\text{PoC}}$ of sharing using the ElSharingProof algorithm. He proves the knowledge of the value $\sum_{i=1}^{n} s_i x^i$ and uses the relation: $s_i = a(i) = \sum_{k=0}^{t} a_k i^k \ \forall i \in [n]$. The algorithm samples two random values $\alpha, \rho \in \mathbb{Z}_q$ and using the relations $s_i = \sum_{j=1}^{m} s_{ij} B^{j-1}$, $r = \sum_{j=1}^{m} r_j B^{j-1}$ as a witness, provides Schnorr based proof using the relation: $\sum_{i=1}^{n} s_i x^i = \sum_{i=1}^{n} (\sum_{k=0}^{t} a_k i^k) x^i = \sum_{k=0}^{t} a_k (\sum_{i=1}^{n} i^k x^i)$ for $x \xleftarrow{\$} \mathbb{Z}_q$ . Each party $P_i$ verifies the proof of correct sharing using ElShareVerify before decrypting their share using ElShareDec.

**Correctness of relations being verified by the receivers**.

GrothVSS-Algorithms

Every party has access to the public parameters $pp$. Each party $P_i$ runs the key setup to generate the secret key - public key pairs $(sk_i, \bar{h}_i)$ and the NIZK proof of knowledge $\pi_i$.

- $\mathsf{DLKeySetup}(pp) \rightarrow (sk, h, \pi)$
  - $sk \xleftarrow{\$} \mathbb{Z}_q$
  - $\bar{h} \leftarrow \bar{g}_2^{sk}$
  - $\pi \leftarrow \mathsf{Prove}_{\mathsf{DL}}(sk, \bar{h})$
- $\mathsf{Share}(pp, s) \rightarrow (\{s_i\}_{i\in[n]}, \{A_j\}_{j\in\{0,\cdots,t\}})$ :
  - Sample $a_j \xleftarrow{\$} \mathbb{Z}_q, j \in \{0, \cdots, t\}$
  - Set $a_0 \leftarrow s$
  - Compute $s_i \leftarrow \sum_{j=1}^{t} a_j i^j, i \in [n]$
  - Set $A_j \leftarrow \bar{g}_2^{a_j}, j \in \{0, \cdots, t\}$
- $\mathsf{ElShareEnc}(pp, \{s_i, \bar{h}_i, \pi_i\}_{i\in[n]}) \rightarrow (\{R_u, E_{i,u}\}_{i\in[n],u\in[m]})$ :
  - $e_i \leftarrow \mathsf{Verify}(\pi_i, \bar{h}_i)$.
    - If $e_i = \perp$, abort.
  - Chunk $s_i$ into $s_{i,u}$ such that $s_i = \sum_{u=1}^{m} s_{i,u} B^{j-1}$ and $s_{i,u} \in [0, B-1]$.
  - Sample $r_u \leftarrow \mathbb{Z}_q, u \in [m]$
  - Compute $R_u \leftarrow \bar{g}_1^{r_u}, u \in [m]$
  - Compute $E_{i,u} \leftarrow \bar{g}_1^{s_{i,u}} \bar{h}_i^{r_u}, i \in [n], u \in [m]$.
- $\mathsf{ElShareDec}(pp, sk_i, \{R_u, E_{i,u}\}_{i\in[n],u\in[m]}) \rightarrow s_i$:
  - Compute and set
    - $\bar{g}_1^{s_{i,u}} \leftarrow \frac{E_{i,u}}{R_u^{sk_i}} \; \forall j \in [m]$.
    - $s_{i,u} \leftarrow \mathsf{Solve}_{\mathsf{DL}}(\bar{g}_1^{s_{i,u}})$.
    - $s_i \leftarrow \sum_{u=1}^{m} s_{i,u} B^{u-1}$

**Figure 10: Share generation, encryption and decryption algorithms of** GrothVSS**[36].**

Here we show the correctness of the relations being verified by the receivers. The receivers verify the correctness of secret sharing to accept the share. If the verification fails, the dealing is rejected.

- $c^{\gamma'} W = (\bar{g}_1^r)^{\gamma'} \cdot \bar{g}_1^{\rho} = \bar{g}_1^{r\gamma'+\rho} = \bar{g}_1^{z_r}$
- $(\prod_{j=0}^{t} A_j^{\sum_{i=0}^{n-1} i^k \gamma^i})^{\gamma'} X = \left(\bar{g}_2^{\sum_{j=0}^{t} a_j \cdot \sum_{i=0}^{n-1} i^k \gamma^i}\right)^{\gamma'} \cdot \bar{g}_2^{\alpha} = \left(\bar{g}_2^{\sum_{i=0}^{n-1} s_i \gamma^i}\right)^{\gamma'} \bar{g}_2^{\alpha} = \bar{g}_2^{\gamma' \cdot \sum_{i=0}^{n-1} s_i \gamma^i + \alpha} = \bar{g}_2^{z_\alpha}$
- $(\prod_{i=1}^{n} d_i^{\gamma^i})^{\gamma'} \cdot Y$
  $= (\prod_{i=1}^{n} (\prod_{u=1}^{m} d_{i,u}^{B^{u-1}})^{\gamma^i})^{\gamma'} \cdot Y$
  $= (\prod_{i=1}^{n} (\prod_{u=1}^{m} (\bar{g}_1^{s_{i,u}} \bar{h}_i^{r_u})^{B^{u-1}})^{\gamma^i})^{\gamma'} \cdot Y$
  $= (\prod_{i=1}^{n} (\prod_{u=1}^{m} (\bar{g}_1^{s_{i,u}} \bar{h}_i^{r_u})^{B^{u-1}})^{\gamma^i})^{\gamma'} \cdot Y$
  $= (\prod_{i=1}^{n} (\bar{g}_1^{s_i} \bar{h}_i^{r})^{\gamma^i})^{\gamma'} \cdot (\prod_{i=1}^{n} \bar{h}_i^{\gamma^i})^{\rho} \bar{g}_1^{\alpha}$
  $= \prod_{i=1}^{n} (\bar{h}_i^{\gamma^i})^{\gamma' r + \rho} \cdot (\bar{g}_1^{\gamma' \cdot \sum \gamma^i s_i + \alpha})^{\rho} \bar{g}_1^{\alpha}$
  $= \prod_{i=1}^{n} (\bar{h}_i^{\gamma^i})^{z_r} \cdot \bar{g}_1^{z_\alpha}$

Apart from the proof of correctness of sharing, the dealer provides zero-knowledge proof of correct chunking showing that $s_i = \sum_{j=1}^{m} s_{ij}$. He also proves that each such $s_{ij} < B$ using (approximate) range proofs. We refer the reader to [36, Section 6.5] for the proof of correct chunking.

# B MITIGATING THE BIASING PUBLIC KEY ATTACK

cgDKG (and Groth's NI-DKG) suffer from the same public key biasing attack as the one presented by Gennaro et al. [32]. This is because a rushing adversary can observe the first $t$ verified secret sharings and then perform a valid $t+1$st sharing to bias the public key while delaying the messages of the other honest parties in the system. The adversary can first compute the partial public of the $t$ honest parties and choose the $t+1^{st}$ party (which the adversary controls) to bias the public key.

To overcome this, we use an approach [44] where the knowledge of the commitments does not aid the adversary in biasing the public key. After verifying the dealings, the parties use the first set of $t+1$ verified dealers to compute their secret key share. Each party now publishes the public key computed as exponentiation of the secret key with a *different* generator $g' \in \mathbb{G}_1$ than $g_1$, the one used in the initial commitment phase. After computing the qualified set, each party $P_k$ broadcasts the value $(g')^{x_k}$ along with a NIZK proof that the exponent in $(g')^{x_k}$ is the same as the one computed using the verified dealings. The parties finally compute the public key of the DKG instance as $y = \prod_{k\in T} (g')^{x_k}$, where $T$ is the set of parties that have forwarded their public key, the set $T$ has at least $t+1$ parties as only a maximum of $t$ parties are corrupted by the adversary. This adds one round of communication to the DKG protocol. A previously suggested approach [32] to overcome the biasing attack is to use perfectly hiding Pedersen's commitments. These commitments are published in the initial commit phase while the public key is computed in the next phase (round) using discrete log commitments, which are published along with proof of the equality of the exponents (shared secret). This approach also needs an extra round for the parties to agree on the public key. However, the mentioned approach of using a different generator for the public key is more efficient as no blinding factors (and the corresponding exponentiations) are needed.

# C FORWARDED PROOFS

Theorem C.1. For any security parameters $\lambda, \lambda_{\mathsf{st}} \in \mathbb{N}$ and any modulus $q \in \mathbb{N}$, our construction, described in Fig. 1, satisfies the security definition given in Def. 3.2 assuming DLog is tractable in $F$, DDH is hard in $G$, the strong root and $\omega$-low order assumptions [53] hold in $G$ in the random oracle model.

Proof. We prove completeness, soundness and zero-knowledge in order.

**Completeness**. The completeness can be seen from checking the verification equations:

- $W \cdot R^{\gamma'} = g^{\rho + r\gamma'} = g^{z_r}$;
- $X \cdot (\prod_{j=0}^{t} A_j^{\sum_{i=1}^{n} i^k \gamma^j})^{\gamma'}$
  $= X \cdot \left(A_0^{(\gamma+\gamma^2+\ldots)} \cdot A_1^{(\gamma+2\gamma^2+\ldots)} \cdot A_2^{(\gamma+2^2\gamma^2+\ldots)} \ldots\right)^{\gamma'}$
  $= X \cdot \left(\bar{g}^{a_0(\gamma+\gamma^2+\ldots)} \cdot \bar{g}^{a_1(\gamma+2\gamma^2+\ldots)} \cdot \bar{g}^{a_2(\gamma+2^2\gamma^2+\ldots)} \ldots\right)^{\gamma'}$
  $= X \cdot \left(\bar{g}^{(a_0+a_1+\ldots)\gamma + (a_0+2a_1+2^2a_2+\ldots)\gamma^2+\ldots}\right)^{\gamma'}$
  $= X \cdot \left(\bar{g}^{s_1\gamma+s_2\gamma^2+\ldots}\right)^{\gamma'} = \bar{g}^{\alpha+\gamma' \sum_{i=1}^{n} s_i \gamma^i} = \bar{g}^{z_s}$;

- $(\prod_{i=1}^{n} E_i^{\gamma^i})^{\gamma'} \cdot Y$
  $= \left(f^{\gamma'(\sum_{i=1}^{n} s_i\gamma^i)} \cdot \prod_{i=1}^{n} h_i^{r\gamma'\gamma^i}\right) \cdot \left(f^{\alpha} \cdot \prod_{i=1}^{n} h_i^{\rho\gamma^i}\right)$
  $= f^{\alpha+\gamma'\sum_{i=1}^{n} s_i\gamma^i} \cdot \prod_i h_i^{(r\gamma'+\rho)\gamma^i} = f^{z_s} \cdot \prod_{i=1}^{n} (h_i^{\gamma^i})^{z_r}$

**Statistical Soundness**. The soundness argument is essentially the same as the one given by Groth [36](As mentioned in Groth's paper, we do not actually need simulation soundness.) but adjusted to our class group setting. The soundness holds unconditionally with overwhelming probability ($\geq 1 - \mathsf{negl}(\lambda_{\mathsf{st}})$) in the random oracle model.

We consider an unbounded adversary who attempts to produce a protocol instance $\{h_i, E_j, A_k, R\}_{i,j\in[n],k\in[t]}$ such that the shares $s_j \neq a(j)$. The adversary must return a well formed proof $\pi_{\mathsf{CS}} \leftarrow (W, X, Y, z_r, z_s)$ to avoid trivial detection.

Let $\gamma = H(\mathsf{inst})$ and $\gamma' = H(\gamma, W, X, Y)$ where $H$ is modeled as a random oracle. The adversary may try to generate a valid proof for invalid shares (i.e. $s_j \neq P(j)$) in three ways – (i) guess $\gamma$ and obtain $\gamma' \leftarrow H(\gamma, W, X, Y)$; (ii) obtain multiple $\gamma$ values such that $\sum_{j=1}^{n} s_j\gamma^j = \sum_{j=1} a_j\gamma^j$ for some $\gamma$ even when $s_j \neq a(j)$ for some $j$; (iii) while $\sum_{j=1}^{n} s_j\gamma^j \neq \sum_{j=1} a_j\gamma^j$, yet generate $z_r, z_s$ such that the verification succeeds by choosing $W, X, Y$ after obtaining $\gamma'$.

For strategy-(i), the adversary needs to guess the correct $\gamma$, the probability of which is bounded by $Q_H^2/2q$ where $Q_H$ is the number of RO queries to $H$. For strategy-(ii), it can be argues that for each uniform random $\gamma$ value, the probability of the equality holding is bounded by Schwartz-Zippel lemma over $\mathbb{Z}_q$, making the total success probability bounded by $O(t/q)$. Finally, for strategy-(iii), we assume that the adversary must produce the tuple $(z_r, z_s, Y)$ such that the verification passes even when $s_i \neq P(i)$. However, the first two verification checks ensure that $z_r, z_s$ are of the form $z_r = r\gamma' + \rho \in \mathbb{Z}_s$ and $z_s = \sum_{i=1}^{n} s_i\gamma^i + \alpha \in \mathbb{Z}_q$. The adversary may potentially choose a $Y$ carefully such that the third equality holds over $G$. We need to show that happens only with negligible probability. Now, let us express the third verification equation over $G$.

$$(\prod_{i=1}^{n} E_i^{\gamma^i})^{\gamma'} \cdot Y \stackrel{?}{=} f^{z_s} \cdot \prod_{i=1}^{n} (h_i^{\gamma^i})^{z_r}$$

Now, clearly since $\gamma$ is the output of random oracle, and $z_r = r\gamma' + \rho$ holds over $\mathbb{Z}_s$ (and therefore also holds over $\mathbb{Z}_{qs}$), the $Y$ value must be of the form $f^{\beta} \cdot \prod_{i=1}^{n} h_i^{\rho\gamma^i}$ for some $\beta \in \mathbb{Z}_q$ for the verification to hold. The equation indicates that,

$$\prod_{i=1}^{n} (f^{s_i} h_i^r)^{\gamma^i\gamma'} \cdot f^{\beta} \cdot \prod_{i=1}^{n} h_i^{\rho\gamma^i} = f^{z_s} \cdot \prod_{i=1}^{n} h_i^{\gamma^i(r\gamma'+\rho)}$$

$$\implies \prod_{i=1}^{n} (f^{s_i})^{\gamma^i\gamma'} \cdot \prod_{i=1}^{n} h_i^{(r\gamma'\gamma^i+\rho\gamma^i)} \cdot f^{\beta} = f^{z_s} \cdot \prod_{i=1}^{n} h_i^{\gamma^i(r\gamma'+\rho)}$$

Canceling out the $h_i$ terms results in,

$$\prod_{i=1}^{n} (f^{s_i})^{\gamma^i\gamma'} \cdot f^{\beta} = f^{z_s}$$

$$\implies \gamma' \sum_{i=1}^{n} (s_i)^{\gamma^i} + \beta = \gamma' \sum_{i=1}^{n} a_i\gamma^i + \alpha \in \mathbb{Z}_{qs}$$

For the verification to hold, the adversary should guess the correct $\beta$ such that $\gamma' \sum_{i=1}^{n} (s_i)^{\gamma^i} + \beta = \gamma' \sum_{i=1}^{n} a_i\gamma^i + \alpha$ when $s_i \neq P(i)$ for a set of $\alpha, \gamma'$ and $\{a_i\}_{i\in[t]}$, which would hold with probability $\leq 1/q$ over the random choice of $\gamma'$. Hence, taking union bound over the strategies, the overall probability is bounded by $Q_H^2/2q + t/q + 1/q$ which is negligible in $\lambda_{\mathsf{st}}$ as long as $q$ is chosen sufficiently larger than $Q_H$. For example, a reasonable choice can be $q = O(2^{256})$ and $Q_H = O(2^{100})$, then the overall probability is smaller than $2^{-40}$ (a typical choice of $\lambda_{\mathsf{st}}$ is 40).

**Statistical Zero-knowledge**. Following [36], we argue the statistical zero-knowledge of the proof of correct sharing in the programmable random oracle model. The simulator calls $O(\mathsf{inst}), O_{\mathsf{prog}}$ to obtain $\gamma, \gamma' \in \mathbb{Z}_q$. . Now the simulator uniform randomly samples $z_r \stackrel{\$}{\leftarrow} [q \cdot |\mathcal{D}_q| \cdot 2^{\lambda_{\mathsf{st}}}], z_s \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ and computes the unique values $(W, X, Y)$ that satisfy the three verification equations. The simulator programs the oracle $O_{\mathsf{prog}}$ such that $\gamma' \leftarrow O(\gamma, W, X, Y)$; unless the $(\gamma, W, X, Y)$ has been queried before. In such a case, the programming fails, and the simulator returns $\perp$. Otherwise, the simulator returns $(W, X, Y, z_r, z_s)$ as the proof.

The simulator obtains uniformly random $\gamma, \gamma'$ and hence $\rho \leftarrow z_r - r\gamma' \in [q \cdot |\mathcal{D}_q| \cdot 2^{\lambda_{\mathsf{st}}}]$, $\alpha \leftarrow \gamma' \sum_{i=1}^{n} s_i\gamma^i \in \mathbb{Z}_q$ are also uniformly random. In the real proof, $\alpha, \rho$ are randomly generated; hence in both simulated and the real proofs, $\alpha, \rho$ are uniformly random. Given $\alpha, \rho$ and $\gamma, \gamma'$, the values $W, X, Y$ are uniquely determined by the three verification equations making the real and simulated proof have the same distribution. It can be seen that since $z_r, z_s$ are random, $W, X$ are also random in their respective domains implying that the probability of the simulator already querying on $(\gamma, W, X, Y)$ is negligible. .

□