Non-interactive VSS using Class Groups and Application to DKG

Aniket Kate*† Easwar Vivek Mangipudi†§ Pratyay Mukherjee†
Hamza Saleem† Sri Aravinda Krishnan Thyagarajan‡

*Purdue University
†Supra Research
‡NTT Research

***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.**

This preliminary report presents 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 interesting. We implement our cgVSS protocol using the BICYCL library and compare its performance with the state-of-the-art NI-VSS by Groth. Our protocol reduces the message complexity and the bit length of the broadcast message by at least 5.6x for a 150 party system, with a 1.8x speed-up in the dealer's computation time and with similar receiver computation times.

I. Introduction

In a (threshold) secret sharing scheme [7], [38], 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 [18], 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 [22] 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 [5], [26], the validity of shares is assured only to the protocol participants through an interactive process; however, in publicly verifiable secret sharing (PVSS) [39], anybody can verify the correctness of the sharing. A typical VSS protocol [33] 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 [39] instead involves publishing encrypted shares and proving the correctness of the encryptions and sharing [27], [29], [37]. Multiple encryption schemes including ElGamal [29] and Lattice-based encryption schemes [28] 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 $\mathbb{Z}_\mathsf{q}$ 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

§Contact author: e.mangipudi@supraoracles.com

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. [29]). 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 use a class group-based encryption mechanism to achieve this.

A. Our Work

This work considers an encryption scheme in the class-group setting [16]. 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 [25], [29], [30]. 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 [25] similar to DKG protocols proposed earlier [29], [34]. To overcome this, we use an extension [32] 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).

We compare our cgVSS scheme with the closest existing scheme by Groth et al. [29] in terms of dealer, receiver times, and the total bit-length of the message broadcast by the dealer. To compare, we provide a simplified version of the VSS mechanism (referred to as GrothVSS) proposed by Groth et al. [29] without forward secrecy. Our implementation shows that the bit-length of the total broadcast message for a single VSS instance for 150 users is 296.51 Kb for the cgVSS compared to 1.66Mb in GrothVSS which is a 5.6x improvement. The comparison also indicates the increase in dealing size is slower for cgVSS when compared to GrothVSS with increasing number of nodes. The benchmarks also indicate that for the cgVSS, the dealer time is lower and the receiver time is similar, compared to GrothVSS. See Section VI-A for further details.

B. Related Work

In their seminal paper, Chor et al. [18] introduce the notion of verifiable secret sharing (VSS) and a scheme based on the intractability of factorization. Benaloh [6] introduced the notion of non-interactive VSS but with the existence of a mutually trusted party. Stadler [39] proposed publicly verifiable VSS and two constructions using verifiable ElGamal encryption, one for encrypting a share and the other proving that the encrypted value is a e-th root of a message. A long line of works [8], [24], [35], [37] proposed publicly verifiable secret-sharing (PVSS) schemes using factorization and pairing [42] for applications including electronic voting and key escrows.

Feldman [23] proposed a non-interactive VSS that employs randomized encryptions and exploits the homomorphic property of the encryption for verification. Pedersen [33] proposed

[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.

a non-interactive VSS where each party can verify their share and the dealing; however, they can not verify the shares of others. Recently, Groth et al. [29] proposed a non-interactive distributed key generation mechanism that does not involve any interaction between the parties. The proposed scheme uses ElGamal encryptions of shared values which can be publicly verified by all the parties from the commitments of the polynomial coefficients. This protocol has been particularly suitable for blockchain-based applications because of non-interactivity where the broadcast channel is typically the underlying blockchain. Since the blockchain stores the broadcast values, the message complexity becomes important. In this work, we improve the message complexity and the total bit-length of the information broadcast by each dealer using a class group encryption scheme. Gentry et al. [28] realized PVSS using a lattice-based encryption scheme based on LWE security. However, the scheme incurs huge costs for generating ZKP of correct sharing and verifying and is efficient with constant amortized ciphertext/plaintext rate only asymptotically. This makes the scheme suitable only when the number of parties is in the thousands.

Paper Outline. Rest of the paper is organized as follows: In Section II we introduce the preliminaries, including the class group setting with the multi-receiver encryption scheme. We propose and describe the non-interactive VSS protocol cgVSS in Section IV and the non-interactive DKG protocol cgDKG in Section V. We analyze and compare the cgVSS with GrothVSS in terms of message complexity, the dealer and receiver timings and present the experimental details in Section VI. Finally, we comment on achieving asynchronous non-interactive VSS and DKG from our proposed protocols in Section VII-A.

II. Preliminaries

A. Notation

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 λ (a typical value 128), and the statistical security parameter is denoted by λ_{st} (typical value 40). We say that a function is negligible in λ, if it grows as $2^{-\Omega(\lambda)}$.

B. Shamir Secret Sharing

We use Shamir's secret sharing [38]. 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_is as $s = \sum_{i=1}^{t+1} L_i s_i$. Due to linearity, this can be performed in the exponent without computing s.

C. Class Groups Setting

Castagnos and Laguillaumie [16] 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 [11], [14], [40], [41] including two-party ECDSA [14] and multi-party computation [11].

In this paper, we follow the presentation similar to [11]. 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 $G = \langle \tilde{g} \rangle$ of $\widehat{G}$, such that G has order $q \cdot s$ – s is known. Unlike $\widehat{G}$ the elements of G are not efficiently recognizable. $G^q = \langle \tilde{g}_q \rangle$ denotes the cyclic subgroup of G of the q-th power. So, G can be factored as $G \simeq G^q \times F$ and $\tilde{g} = \tilde{g}_q \cdot f$. We also consider two distributions $\mathcal{D}$ and $\mathcal{D}'$ over $\mathbb{Z}$ such that $\{\tilde{g}^x \mid x \leftarrow \mathcal{D}\}$ and $\{\tilde{g}_q^x \mid x \leftarrow \mathcal{D}_q\}$ induce distributions over G and G^q respectively, that are statistically close (within distance $2^{-\lambda_{\mathsf{st}}}$) to uniform distributions over respective sets. Going forward, we refer to $\tilde{g}_q$ simply as g for notational convenience.

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

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

Hardness assumptions on class groups. The employed encryption scheme uses the unknown order and the hard subgroup membership assumptions as described below.

Definition 1 (Unknown order assumption [11])**.** *For the security parameters $\lambda, \lambda_{\mathsf{st}} \in \mathbb{N}$, modulus $q \in \mathbb{Z}$ consider a set of public parameters $\mathsf{pp}_{\mathsf{CG}} := (q, \lambda, \lambda_{\mathsf{st}}, \bar{s}, f, g, \widehat{G}, F, \mathcal{D}, \mathcal{D}'; \rho) \leftarrow \mathsf{CG.ParamGen}(1^\lambda, 1^{\lambda_{\mathsf{st}}}, q)$ generated using a uniform random ρ. 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 λ.*

$$\Pr\left[h^e = 1 \mid (h, e) \leftarrow \mathcal{A}(pp_{\mathsf{CG}})^{\mathsf{CG.Solve}(\cdot)}\right]$$

Definition 2 (Hard subgroup membership assumption [11])**.** *For the security parameters $\lambda, \lambda_{\mathsf{st}} \in \mathbb{N}$, modulus $q \in \mathbb{Z}$ consider a set of public parameters $\mathsf{pp}_{\mathsf{CG}} :=$*

$(q, \lambda, \lambda_{\mathsf{st}}, \bar{s}, f, g, \widehat{G}, F, \mathcal{D}, \mathcal{D}'; \rho) \leftarrow \mathsf{CG.ParamGen}(1^\lambda, 1^{\lambda_{\mathsf{st}}}, q)$ generated using a uniform random ρ. Sample $x \leftarrow \mathcal{D}$ and $x' \leftarrow \mathcal{D}'$. Sample a bit $b^* \xleftarrow{\$} \{0,1\}$ uniformly at random. If $b = 0$, define $h^* \leftarrow \tilde{g}^x$, otherwise if $b^* = 1$ define $h^* \leftarrow g^{x'}$. 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 λ.

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

III. 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. Without going into a formal definition, we directly present the constructions.

A. Schnorr's NIZK for Knowledge of Exponent over class-groups. [15]

Our construction uses non-interactive zero-knowledge (NIZK) proof for knowledge of exponents over class groups. In particular, consider the class-group parameters $pp_{\mathsf{CG}} = (q, \lambda, \lambda_{\mathsf{st}}, \bar{s}, f, g, \widehat{G}, F, \mathcal{D}, \mathcal{D}'; \rho)$ an instance $\mathsf{inst} = (g, h)$ and witness $\mathsf{wit} = k$ such that $h \leftarrow g^k$. Also consider a hash function $H : \{0,1\}^* \to \mathcal{B}$ 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:

- $\mathsf{Kex.Prove}(pp_{\mathsf{Kex}}, \mathsf{inst}, \mathsf{wit}) \to \pi$. 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 $r \xleftarrow{\$} [\mathcal{B} \cdot |\mathcal{D}| \cdot 2^{\lambda_{\mathsf{st}}}]$
  - $\alpha \leftarrow g^r$;
  - $c \leftarrow H(g, h, \alpha) \in \mathcal{B}$;
  - $s \leftarrow r + k \cdot c \in \mathbb{Z}$;
  - Output the NIZK proof $\pi = (c, s)$
- $\mathsf{Kex.Ver}(pp_{\mathsf{Kex}}, \mathsf{inst}, \pi) \to 1/0$. 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 [36]. 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 that is 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.

B. 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 [16] 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 sub-group F where discrete log is easy. Let pp_{Enc} be the public parameters which is the same as the class-group parameters $pp_{\mathsf{CG}} := (q, \lambda, \lambda_{\mathsf{st}}, \bar{s}, f, g, \widehat{G}, F, \mathcal{D}, \mathcal{D}'; \rho)$. The multi-receiver encryption scheme is comprised of three algorithms $\mathsf{CGE.KeyGen}, \mathsf{CGE.mrEnc}$ and $\mathsf{CGE.Dec}$ for generating the keys, (multi-receiver) encryption and decryption, respectively:

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

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).

C. 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 [29] variant of Schnorr proof, adapted to our class-group setting. The overall idea, as we recall from [29], is to use a 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 basically 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 $\mathbb{G}$ of prime order q and a randomly chosen generator $\bar{g} \in \mathbb{G}$. Also, consider hash functions (modeled as random oracles) H, H' both mapping $\{0,1\}^* \to \mathbb{Z}_q$. The public parameter of the proof system is defined as $pp_{\mathsf{PoC}} := \{\bar{g}, \mathbb{G}, H, H'\} \cup pp_{\mathsf{Enc}}$. We use the generator $\bar{g}$ for commitments, on group $\mathbb{G}$, 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 $\mathsf{CGE.mrEnc}(pp_{\mathsf{Enc}}, \{h_i, s_i\}_{i\in[n]}; r)$ 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 $\mathfrak{R}$ that consists of an instance inst and a witness wit where:

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

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 $(R, \{E_i\}_{i\in[n]})$

The algorithms $\mathsf{SharingProof}$ and $\mathsf{SharingVerify}$ are described in Figure 1.

Proof of Correct Sharing

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

Fig. 1: Proof System of Correct Sharing.

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

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

Soundness. From a high level, the soundness an be argued from the two (in reality there are $n+1$ proofs, each of which is happening with a power of γ, from 0 to n) Schnorr's style proofs being run in parallel plus the additional check via the third equality, which ties them together. For soundness the third check becomes crucial. Without that, one may manipulate the Schnorr proofs, by adequately adjusting exponents to satisfy the first two equations. But the third equation ensures that such manipulations do not work. For a full proof we refer to Groth's original paper [29]

Zero-knowledge. Similar to the Schnorr proof (cf. Section III-A), the zero-knowledge arguments differ from Groth's as we compute z_r as an integer due to the unknown order of the corresponding group. So, a similar "statistical masking" technique is used to achieve zero-knowledge simulatability.

IV. 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.

A. System Model

For our non-interactive construction, we assume that all nodes 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 Δ. 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.

There are n nodes in the system. We consider a t-bounded static adversary who can corrupt at most t nodes. Each node P_i generates a secret-key-public-key pair of the form (sk_i, h_i) for public-key encryption. In the setup phase, each node broadcasts its public key and a proof the knowledge of secret key sk_i. In the online phase, node P_i forwards messages to P_j by encrypting to the public key h_j.

B. Definition: Non-Interactive Verifiable Secret Sharing

Definition 3 (Non-interactive Verifiable Secret Sharing (NI-VSS)). A non-interactive verifiable secret-sharing (NI-VSS) scheme is a protocol executed between n parties/nodes $P_1, \ldots, P_n$, among them P_n is the dealer (this is without loss of generality). Let pp be a set of public parameters that all algorithms have access to. Then a non-interactive verifiable secret-sharing protocol consists of a tuple of algorithms $(\mathsf{KeySetup}, \mathsf{KeyVer}, \mathsf{Share}, \mathsf{ShareEnc}, \mathsf{Verify}, \mathsf{ShareDec})$ with syntax:

- $\mathsf{KeySetup}(pp) \to (sk, pk, \pi)$. 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, π_i).
- $\mathsf{KeyVer}(pp, (pk, \pi)) \to 1/0$. 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 π.
- $\mathsf{Share}(pp, s) \to (\{s_i\}_{i\in[n]}, \mathsf{cmt})$. The sharing algorithm produces t out of n Shamir's secret-shares of a value s and the associated commitment cmt.
- $\mathsf{ShareEnc}(pp, \mathsf{cmt}, \{s_i, pk_i\}_{i\in[n]}) \to (R, \{E_i\}_{i\in[n]}, \pi_{\mathsf{PoC}})$. 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 π_{PoC}.
- $\mathsf{Verify}(pp, \mathsf{cmt}, R, \{E_i, pk_i\}_{i\in[n]}, \pi_{\mathsf{PoC}}) \to 1/0$. This algorithm verifies the entire ciphertext tuple with respect to the proof π_{PoC} and the commitment to output a decision bit.
- $\mathsf{ShareDec}(pp, sk_i, R, E_i) \to s_i$. 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).

In the protocol the dealer P_n and $n-1$ receivers $P_1, \ldots, P_{n-1}$ interact as described below in Figure 2.

The **correctness** requires that if all parties are honest, then for any pp, any share s, all receivers accepts their shares s_i, and the tuple $(s_1, \ldots, s_n)$ constitutes a t out of n Shamir's secret sharing of s.

We informally say that a NI-VSS scheme is **secure** if a malicious (who can behave completely arbitrarily) adversary even after corrupting up to t parties, can not learn s or can not make the honest parties accept inconsistent values (which does not form a valid secret-share, or is inconsistent with the public commitments).

cgVSS protocol

- **Key Generation**. Each party P_i runs $\mathsf{KeySetup}(pp)$ to generate (sk_i, pk_i, π_i) and broadcasts (pk_i, π_i).
- **Dealing**. The dealer P_n receives $\{(pk_i, \pi_i)\}_{i\in[n-1]}$. It then runs for all $i \in [n-1]$: $\mathsf{KeyVer}(pp, pk_i, \pi_i)$. If KeyVer returns 0 for any i, exit. Otherwise it executes the following steps.
  - Sample $s \xleftarrow{\$} \mathbb{Z}_{\mathsf{q}}$
  - $(\{s_i\}_{i\in[n]}, \{a_j\}_{j\in[t]}, \mathsf{cmt}) \leftarrow \mathsf{Share}(pp, s)$
  - Define own share to be s_n.
  - Compute $(R, \{E_i\}_{i\in[n]}, \pi_{\mathsf{PoC}}) \leftarrow \mathsf{ShareEnc}(pp, \{s_i, pk_i\}_{i\in[n]})$
  - Broadcast $(R, \{E_i\}_{i\in[n]}, \mathsf{cmt}, \pi_{\mathsf{PoC}})$ to all receivers $\{P_i\}_{i\in[n-1]}$.
- **Receiving**. Each party P_i for $i \in [n-1]$ 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.
  - $e \leftarrow \mathsf{Verify}(pp, \mathsf{cmt}, R, \{E_i\}_{i\in[n]}, \pi_{\mathsf{PoC}})$
  - If $e = 1$ then $s_i \leftarrow \mathsf{ShareDec}(pp, sk_i, R, E_i)$ and accept s_i as its share corresponding to the dealing. Otherwise, if $e = 0$ reject dealing.

Fig. 2: The general structure of an NI-VSS protocol.

C. Our NI-VSS Protocol

In this section we provide a concrete instantiation of our NI-VSS protocol based on the multi-receiver encryption scheme (cf. Section III-B), a corresponding proof of correct sharing (cf. Section III-C) and a Schnorr's proof for knowledge of exponent (cf. Section III-A. The instantiation is provided in Figure 3.

Correctness and Security. The correctness follows stright-forwardly from the underlying primitives. Arguing security is more intricate. While the full analysis relies on reducing to the security properties of the underlying primitives, here we provide a few intuitive arguments. Recall that our threat model considers at most t parties can be maliciously corrupt.

- If the dealer is corrupt (and possibly colluding with $t-1$ receivers), the honest parties should reject any invalid share. This can be argued by the *soundness* of proof of correct sharing. Furthermore, the soundness of Schnorr's proof guarantees that the public key is legitimate.
- If dealer is honest, and t receivers are corrupt and colluding, then the dealer would aim to protect the secrecy of s, which can be guaranteed by the Shamir's secret sharing, hardness of discrete log over cyclic groups, CPA-security of the encryption scheme, zero-knowledge property of the proof of correct-sharing, and also zero-knowledge property of the Schnorr proof.

V. 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 $\mathbb{G}$ of prime order. An NI-DKG protocol can be thought of as a symmetric version of NI-VSS, with the crucial difference that no one knows the secret in NI-DKG, in contrast to NI-VSS, the dealer knows the entire secret. Indeed, following prior works (e.g. [25], [29], [34]), we construct NI-DKG by augmenting our NI-VSS protocol naturally. 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 cgVSS-Algorithms

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

Construction

- $\mathsf{KeySetup}(pp) \rightarrow (sk, pk, \pi)$:
  - $(sk, h) \leftarrow \mathsf{CGE.KeyGen}(pp_{\mathsf{Enc}})$.
  - $\pi \leftarrow \mathsf{Kex.Prove}(pp_{\mathsf{Kex}}, h, sk)$.
  - 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 $a_j \xleftarrow{\$} \mathbb{Z}_{\mathsf{q}}, j \in [t]$.
  - 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 $\mathsf{cmt} \leftarrow \{A_0, \ldots, A_t\}$.
- $\mathsf{ShareEnc}(pp, \mathsf{cmt}, \{s_i, pk_i\}_{i\in[n]}) \rightarrow (R, \{E_i\}_{i\in[n]}, \pi_{\mathsf{PoC}})/\perp$.
  - 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 $r \xleftarrow{\$} \mathcal{D}$
    - Compute $(R, \{E_i\}_{i\in[n]}) \leftarrow \mathsf{CGE.mrEnc}(pp_{\mathsf{Enc}}, \{h_i, s_i; r\}_{i\in[n]})$.
    - Define:
      - $\mathsf{inst} = \big(\{h_i\}_{i\in[n]}, (R, \{E_i\}_{i\in[n]}), \mathsf{cmt}\big)$.
      - $\mathsf{wit} = \big((s_1, \ldots, s_n), r\big)$.
    - Compute $\pi_{\mathsf{PoC}} \leftarrow \mathsf{SharingProof}(pp_{\mathsf{PoC}}, \mathsf{inst}, \mathsf{wit})$.
- $\mathsf{Verify}(pp, \mathsf{cmt}, R, \{E_i, pk_i\}_{i\in[n]}, \pi_{\mathsf{PoC}}) \rightarrow 1/0$:
  - Parse $\mathsf{inst} \leftarrow \big(\{h_i\}_{i\in[n]}, (R, \{E_i\}_{i\in[n]}), \mathsf{cmt}\big)$.
  - Output $\mathsf{SharingVerify}(pp_{\mathsf{PoC}}, \mathsf{inst}, \pi_{\mathsf{PoC}})$.
- $\mathsf{ShareDec}(pp, sk_i, R, E_i) \rightarrow s_i$:
  - Compute $s_i \leftarrow \mathsf{CGE.Dec}(pp_{\mathsf{Enc}}, sk_i, R, E_i)$.

Fig. 3: Algorithms that constitute cgVSS

of z_i with shares of z_j received from other P_j. In Figure 4 we describe this protocol using the NI-VSS algorithms described in Definition 3.

Correctness and Security.. The correctness follows from the correctness of underlying NI-VSS, as long as the commitments cmt has a specific form realized by our construction (cf. Figure 2). We also note that, the linearity of Shamir's secret sharing is crucial to reconstruct individual shares. The only additional security aspect of NI-DKG, compared to NI-VSS is that any colluding set of t parties is unable to compute the secret k – this is easy to see due to the information theoretic security of Shamir's algorithm. Furthermore, this also ensures that any $t+1$ parties together uniquely holds the secret k.

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, π_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.
  - $z_i \xleftarrow{\$} \mathbb{Z}_q$.
  - $(\{s_{ij}\}_{j\in[n]}, \mathsf{cmt}_i) \leftarrow \mathsf{Share}(pp, z_i)$.
  - $(R_i, \{E_{ij}\}_{j\in[n]}, \pi_{\mathsf{PoC}i}) \leftarrow \mathsf{ShareEnc}(pp, \mathsf{cmt}, \{s_{ij}, pk_j\}_{j\in[n]})$
  - Broadcast $(R_i, \{E_{ij}\}_{j\in[n]}, \mathsf{cmt}_i, \pi_{\mathsf{PoC}i})$.
- **Receiving.** Each party P_i receives $n-1$ tuples:

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

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

$$e_j \leftarrow \mathsf{Verify}(pp, \mathsf{cmt}_j, R_j, \{E_{jj'}\}_{j'\in[n]}, \pi_{\mathsf{PoC}j}\})$$

  - 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 cmt_j as $\{A_{0j}, \ldots, A_{tj}\}$.
    - $y = y \cdot A_{0j}$.
  - Output y as system public key.

Fig. 4: cgDKG- Non-interactive distributed key generation using class groups

VI. Complexity and Performance 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. [29] 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 β 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.

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 α 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 $\mathbb{G}_1, \mathbb{G}_2$ and 2 group $\mathbb{Z}_\mathsf{q}$ 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 ℓ.

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.

A. Experimentation and Performance Analysis

We implement cgVSS in C++ using the BICYCL library [9]. For comparison, we realize a version of the implementation of GrothVSS without forward secrecy. We run the experiments on a Macbook pro machine with an Apple M1 Pro chip with 10 cores and 16GB RAM. All the reported timings are averages over 20 runs of the corresponding protocols.

In cgVSS, the dealer generates 256-bit shares for each party in the system and encrypts them. The encryption of each share consists of two elements (c, d), where c is the exponentiated randomness. In the multi-receiver encryption mechanism, the randomness can be reused across multiple receivers. Hence while encrypting the share values for n receivers, the dealer uses one element for randomness and n elements for the second element of the encryption tuples. Each element in the compressed form takes 1752 bits. The dealer commits to the t coefficients of the polynomial. Hence the total bit-length length for the multi-receiver encryption and commitments is $(1752) \cdot (n+1) + 384 \cdot t$. For the proof of correctness, the dealer also forwards 5 elements, including two class group elements, one elliptic curve element, and two scalars. Figure 5 shows the total bit-length of the dealing (the broadcast message). For 100 users, the dealer broadcasts a message (dealing) of length 201.55Kb whereas, for 150 users, it is 297.82Kb.

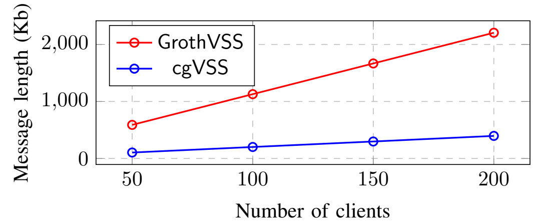
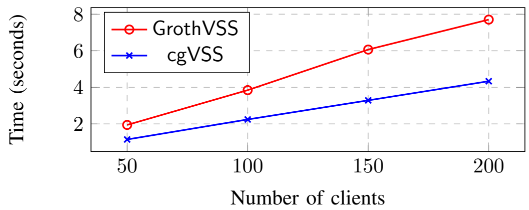


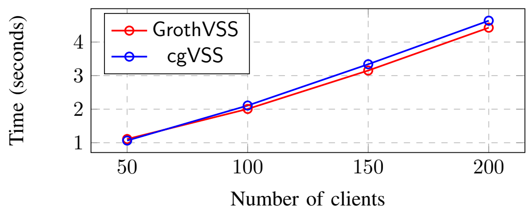
Fig. 5: Comparison of broadcast (dealing) message length where $n = 2t + 1$. cgVSS dealing consists of encryptions and proof of correct sharing, while GrothVSS also consists of proof of correct chunking.

Figure 6 shows the time taken by the dealer and the receiver in the cgVSS protocol. The dealer time includes the time to generate the multi-receiver ciphertext and the NIZK proof of correctness whereas the receiver time includes the decryption time and the time for proof verification. For a 100 party system, the dealer takes 1.22sec for generating the ciphertext and 1.02sec to generate the proof whereas for a 150 party system, it takes 1.80sec for encryption and 1.48sec for proof generation. The decryption takes 38msec, while the proof verification takes 2.07sec for a 100 user system and 3.30sec for a 150 party system (the decryption time stays the same irrespective of the number of parties). Figure 6b shows the total receiver times taken by the party to verify the sharing and decrypt their shares. Since the major portion of the time comes from the generation and verification of the proof of correct sharing, which involves multiple exponentiations, we expect the timings to come down when a multi-exponentiation technique is used for the same.

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 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 ℓ and $\ell+n+1$ masked values of the chunks. Taking a conservative estimate of 32 bits for the masked chunk value



(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.

Fig. 6: Comparison of dealer and receiver times for cgVSS and GrothVSS.

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 5) 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. [28] 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.

VII. 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 [10] and then develop an asynchronous DKG (ADKG) using our AVSS scheme and asynchronous agreement ideas from the recent ADKG protocols by Das et al. [19], [21].

A. Asynchronous VSS using Class Groups

In the asynchronous communication setting, Cachin et al. [12] 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. [4], [20], [43] Relevant to threshold signing for state-machine replication (SMR) protocols, there also have been efforts to define high-threshold VSS schemes [3], [19], [21], 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 [2], [13], [20]. 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.

B. Asynchronous DKG using Class Groups

Kate et al. [30] combined AVSS by Cachin et al. [12] with the PBFT flow [17] 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. [31] 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 quadratic in the number of parties [1], [21] as well as to allow high-threshold secret sharing [19], [21].

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 [19], [21]. This offers a quadratic 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] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. In *PODC'21*, pages 363–373, 2021.

[2] Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. Balanced byzantine reliable broadcast with near-optimal communication and improved computation. In Alessia Milani and Philipp Woelfel, editors, *ACM PODC*, pages 399–417. ACM, 2022.

[3] Nicolas Alhaddad, Mayank Varia, and Haibin Zhang. High-threshold AVSS with optimal communication complexity. In Nikita Borisov and Claudia Diaz, editors, *FC*, pages 479–498, 2021.

[4] Michael Backes, Amit Datta, and Aniket Kate. Asynchronous computational VSS with reduced communication complexity. In *CT-RSA*, pages 259–276, 2013.

[5] Michael Backes, Aniket Kate, and Arpita Patra. Computational verifiable secret sharing revisited. In *Advances in Cryptology—ASIACRYPT*, pages 590–609, 2011.

[6] Josh Cohen Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret (extended abstract). In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 251–260, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

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

[8] Fabrice Boudot and Jacques Traoré. Efficient publicly verifiable secret sharing schemes with fast or delayed recovery. In Vijay Varadharajan and Yi Mu, editors, *Information and Communication Security*, pages 87–102, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[9] Cyril Bouvier, Guilhem Castagnos, Laurent Imbert, and Fabien Laguillaumie. 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.

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

[11] Lennart Braun, Ivan Damgård, and Claudio Orlandi. Secure multiparty computation from threshold encryption based on class groups. Cryptology ePrint Archive, Paper 2022/1437, 2022. https://eprint.iacr.org/2022/1437.

[12] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 88–97, 2002.

[13] Christian Cachin and Stefano Tessaro. Asynchronous veri.able information dispersal. In *IEEE SRDS*, pages 191–202, 2005.

[14] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Two-party ecdsa from hash proof systems and efficient instantiations. In *Annual International Cryptology Conference*, pages 191–221. Springer, 2019.

[15] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold ec-dsa. In *Public-Key Cryptography–PKC 2020: 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4–7, 2020, Proceedings, Part II*, pages 266–296. Springer, 2020.

[16] Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from ddh. In *Cryptographers' Track at the RSA Conference*, pages 487–505. Springer, 2015.

[17] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[18] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 383–395, 1985.

[19] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. Cryptology ePrint Archive, Paper 2022/1389. To appear at Usenix Sec 2023, 2022.

[20] Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *ACM CCS*, pages 2705–2721, 2021.

[21] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2518–2534. IEEE, 2022.

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

[23] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438. IEEE, 1987.

[24] Eiichiro Fujisaki and Tatsuaki Okamoto. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In Kaisa Nyberg, editor, *Advances in Cryptology — EUROCRYPT'98*, pages 32–46, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[25] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, page 295–310, Berlin, Heidelberg, 1999. Springer-Verlag.

[26] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *ACM PODC*, page 101–111, 1998.

[27] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. Practical non-interactive publicly verifiable secret sharing with thousands of parties. In *Advances in Cryptology—EUROCRYPT*, pages 458–487, 2022.

[28] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. Practical non-interactive publicly verifiable secret sharing with thousands of parties. In *Advances in Cryptology–EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part I*, pages 458–487. Springer, 2022.

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

[30] Aniket Kate, Yizhou Huang, and Ian Goldberg. Distributed key generation in the wild. *Cryptology ePrint Archive*, 2012.

[31] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. 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*, pages 1751–1767, 2020.

[32] Wafa Neji, Kaouther Blibech, and Narjes Ben Rajeb. Distributed key generation protocol with a new complaint management strategy. *Security and Communication Networks*, 9(17):4585–4595, 2016.

[33] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology—CRYPTO*, pages 129–140, 1991.

[34] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 522–526. Springer, 1991.

[35] Alexandre Ruiz and Jorge L. Villar. Publicly verifiable secret sharing from paillier's cryptosystem. In Christopher Wulf, Stefan Lucks, and Po-Wah Yau, editors, *WEWoRC 2005 – Western European Workshop on Research in Cryptology*, pages 98–108, Bonn, 2005. Gesellschaft für Informatik e.V.

[36] C. P. Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 239–252, New York, NY, 1990. Springer New York.

[37] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In *Advances in Cryptology—CRYPTO*, pages 148–164, 1999.

[38] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[39] Markus Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology—EUROCRYPT'96: International Conference on the Theory and Application of Cryptographic Techniques Saragossa, Spain, May 12–16, 1996 Proceedings*, pages 190–199. Springer, 2001.

[40] Sri Aravinda Krishnan Thyagarajan, Guilhem Castagnos, Fabian Laguillaumie, and Giulio Malavolta. Efficient cca timed commitments in class groups. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 2663–2684, New York, NY, USA, 2021. Association for Computing Machinery.

[41] Benjamin Wesolowski. Efficient verifiable delay functions. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part III 38*, pages 379–407. Springer, 2019.

[42] Tsu-Yang Wu and Yuh-Min Tseng. A pairing-based publicly verifiable secret sharing scheme. *Journal of systems science and complexity*, 24(1):186–194, 2011.

[43] Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew K. Miller. hbacss: How to robustly share many secrets. In *NDSS*, 2022.

APPENDIX A
GROTHS NI-VSS WITHOUT FORWARD SECRECY [29]

Groth et al. [29] 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 $H, H' : \{0,1\}^* \rightarrow \mathbb{Z}_\mathsf{q}$. 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 7, Figure 8. 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 π_i is forwarded to all the parties before the start of the protocol.

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 π_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 $a_j \xleftarrow{\$} \mathbb{Z}_\mathsf{q}, j \in \{0, \cdots, t\}$
  - 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 $r_u \leftarrow \mathbb{Z}_\mathsf{q}, u \in [m]$
  - 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
    - $\bar{g}_1^{s_{i,u}} \leftarrow \frac{E_{i,u}}{R_u^{sk_i}} \; \forall j \in [m]$.
    - $s_{i,u} \leftarrow \mathsf{Solve}_{\mathsf{DL}}(\bar{g}_1^{s_{i,u}})$.
    - $s_i \leftarrow \sum_{u=1}^{m} s_{i,u} B^{u-1}$

Fig. 7: Share generation, encryption and decryption algorithms of GrothVSS [29].

GrothVSS follows the same mechanics as cgVSS of Figure 2. Here, we present informally the variants of the algorithms used for GrothVSS in Figure 7, Figure 8.

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 $s_i = a(i) \in \mathbb{Z}_\mathsf{q}$. 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.

A. Proof of correct sharing

Here we present the proof of the correctness of sharing of the NI-DKG protocol by Groth et al. [29]. The dealer

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 $\pi_{\mathsf{DL},i}$.

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

Fig. 8: Proof system of correct sharing of GrothVSS [29]. We do not present the proof and verification of correct chunking here, refer [29, Section 6.5] for it.

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 π_{PoC} of sharing using the $\mathsf{ElSharingProof}$ algorithm. He proves the knowledge of the value $\sum_{i=1}^{n} s_i x^i$ and uses the relation: $s_i = a(i) = \sum_{k=0}^{t} a_k i^k \quad \forall i \in [n]$. The algorithm samples two random values $\alpha, \rho \in \mathbb{Z}_{\mathsf{q}}$ 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 $x \xleftarrow{\$} \mathbb{Z}_{\mathsf{q}}$. Each party P_i verifies the proof of correct sharing using $\mathsf{ElShareVerify}$ before decrypting their share using $\mathsf{ElShareDec}$.

Correctness of relations being verified by the receivers.

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.

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

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 [29, Section 6.5] for the proof of correct chunking.

APPENDIX 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. [25]. 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 [32] 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 [25] 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.