Publicly Verifiable Generalized Secret Sharing Schemes and Their Applications

Liang Zhang[1], Dongliang Cai[2], Tao Liu[3], Xingyu Wu[3], Haibin Kan[2], Jiheng Zhang[1], and Moti Yung[4]

[1] The Hong Kong University of Science and Technology
[2] Fudan University
[3] Hainan University
[4] Google LLC and Columbia University

Abstract. Secret sharing under expressive access structures and public verifiability have largely been developed in separate lines of work. Generalized secret sharing (GSS) supports arbitrary monotone access structures, but relies on private channels and gives no public evidence that the dealer distributed a consistent sharing. Publicly verifiable secret sharing (PVSS), in contrast, supports public verification over insecure channels, but is fundamentally tailored to threshold policies. We introduce publicly verifiable generalized secret sharing (PVGSS), a primitive that combines arbitrary monotone access structures with non-interactive public verification of both encrypted shares and decrypted reconstruction shares. Our construction encrypts only the leaf shares of an underlying GSS instance and proves, via Schnorr-type NIZK proofs, that all ciphertexts are globally consistent with a single secret. The key technical ingredient is a GSS testing procedure that checks this global consistency from proof responses, without revealing the shares. We instantiate PVGSS from recursive Shamir secret sharing and from linear secret sharing schemes, obtaining verification complexity linear in the number of leaves of the access structure. We formalize correctness, encrypted-share verifiability, share-decryption verifiability, and IND1-secrecy, and prove security in the random oracle model under DDH, knowledge soundness of the proof systems, and privacy of the underlying GSS-on-group sharing. As an application, we show how PVGSS enables publicly accountable secret release in a decentralized exchange protocol.

Keywords: Access structure · Public verifiability · PVGSS · GSS Testing · Decentralized exchange · Blockchains

1 Introduction

Secret sharing (SS) [1] is a basic mechanism for distributing trust: a dealer encodes a secret into shares so that only authorized subsets of parties can reconstruct it. Verifiable secret sharing (VSS) [2–4] strengthens this primitive by making dealer misbehavior detectable, and publicly verifiable secret sharing (PVSS) [5–7] further allows external observers to verify encrypted shares over

public channels. This public-verification feature is essential in settings where there is no trusted delivery channel and where correctness must be checked without interaction.

Most efficient and widely used PVSS constructions are designed for threshold access structures. Threshold policies are useful, but they cannot naturally express hierarchical, weighted, compartmented, or composite authorization requirements. Generalized secret sharing (GSS) [8–10] addresses this expressiveness problem by supporting arbitrary monotone access structures, and linear secret sharing schemes (LSSS) [10–12] underlie many expressive access-control mechanisms, including ciphertext-policy attribute-based encryption (CPABE) [13, 14]. Classical GSS and LSSS, however, assume private share delivery and do not provide a public certificate that the dealer generated a globally consistent sharing. This motivates a construction that combines expressive sharing with non-interactive public verification over encrypted shares.

Simply composing existing PVSS instances along an access tree does not provide the desired guarantee. Consider a root threshold gate with two child subtrees. A malicious dealer may produce locally valid PVSS transcripts for the two subtrees using child secrets u and v, while u and v are not the evaluations of any root sharing polynomial with constant term s. Each subtree transcript verifies, but the published leaf ciphertexts are not globally tied to one root secret. Adding PVSS proofs for internal values would require publishing extra intermediate verification material and would change the intended leaf-share interface; opening those values would further weaken secrecy. The central problem is therefore to certify global consistency of an encrypted generalized sharing while keeping the shares hidden.

Table 1. Comparison of SS-related cryptographic primitives

Scheme	Accountable dealer (or encryptor)	Accountable shareholders (or authorities)	Insecure channel	Public verifiability	Expressiveness
SS	✗	✗	✗	✗	✗
VSS	✓	✗	✗	✗	✗
PVSS	✓	✓	✓	✓	✗
CPABE	✗	✗	✓	✗	✓
LSSS	✗	✗	✗	✗	✓
GSS	✗	✗	✗	✗	✓
PVGSS	✓	✓	✓	✓	✓

• Accountability indicates whether misbehavior by dealers (or encryptors) and shareholders (or authorities) can be detected.
• Insecure channel denotes that secret shares may be distributed over public channels.
• Public verifiability means that correctness can be checked by any external party.
• Expressiveness indicates native support for arbitrary monotone access structures.

We address this problem by studying publicly verifiable generalized secret sharing (PVGSS). A PVGSS scheme distributes a secret under an arbitrary monotone access structure and provides non-interactive public proofs for both encrypted sharing and decrypted reconstruction shares. It therefore brings public accountability to the sharing and reconstruction phases while preserving the expressiveness of GSS. Table 1 compares this functionality with related secret-sharing primitives.

Our construction starts from a GSS instance, instantiated by either recursive Shamir secret sharing or LSSS-based sharing. The dealer encrypts each leaf share as $C_i = \mathsf{pk}_i^{s_i}$ and attaches Schnorr-type Fiat-Shamir non-interactive zero-knowledge (NIZK) proofs [15, 16] whose challenge binds the ordered public keys, ciphertexts, and access structure. The key technical step is *response-vector testing*: instead of proving consistency through intermediate shares, the verifier treats the Fiat-Shamir responses as a masked linear view of the hidden leaf shares and runs *GSS testing* to check that this response vector is itself a valid sharing of the response secret. By linearity, acceptance certifies that the encrypted shares are globally consistent with the same dealer secret, up to the soundness error of the proof system and the instantiated testing procedure. During reconstruction, each shareholder publishes a decrypted group share together with a discrete logarithm equality (DLEQ) proof [17], so correctness of decryption can be checked independently of the original dealer and without exposing the shareholder's secret key.

1.1 Applications

PVGSS is intended as a public accountability layer for systems in which secrets are released only under structured authorization policies. Its role is not to replace application-specific consensus, availability, or economic mechanisms; rather, it provides a cryptographic way to publish encrypted shares, verify that they are globally consistent with a declared generalized access structure, and later verify the correctness of released reconstruction shares. This is useful whenever the sharing transcript must be checked by parties that do not possess private channels or secret reconstruction keys. Section 6 develops this use case in detail for a decentralized exchange (DEX), where PVGSS supports verifiable commitment, accountable secret release, and public arbitration without relying on a trusted third party.

Large-scale beacon protocols. Existing PVSS-based distributed randomness beacon protocols [18] are typically formulated for threshold committees. PVGSS can extend the publicly verifiable sharing component to expressive access structures supported by the instantiated GSS testing procedure. This allows a beacon design to model multi-tier committees, heterogeneous trust assumptions, and application-specific quorum rules, while preserving public auditability of the sharing and reconstruction transcripts. For example, by partitioning n distributed parties into subgroups of size c, one can define an access control policy of the form d-of$\big(\big(\frac{c}{3}$-of $P_1, \ldots, P_c\big), \big(\frac{c}{3}$-of $P_{c+1}, \ldots, P_{2c}\big), \ldots\big)$, where d controls the number of subcommittees required for reconstruction [19, 20]. Such a policy

does not by itself prove a complete beacon protocol secure, but it illustrates how PVGSS can make a non-threshold committee structure publicly checkable.

Attribute-based cryptosystems. Many attribute-based cryptosystems encode access policies by LSSS matrices [13, 14]. PVGSS is compatible with this style of policy representation when the corresponding sharing procedure is covered by the GSS testing method. It can therefore be used as an auditable subroutine for systems that need public evidence that policy-dependent shares or reconstruction values were generated consistently. This does not change the security definition of the underlying attribute-based encryption scheme; it adds a separately verifiable sharing layer for protocols that expose such computations to public scrutiny.

Blockchain ecosystems. Blockchain protocols naturally operate over public channels and often require disputes to be resolved from public evidence. PVGSS can serve as a public-verifiability layer in the following settings, provided that the application security proof accounts for the surrounding protocol logic. In *on-chain secret escrow* systems [21–23], PVGSS can provide an auditable commitment-and-release mechanism for witnesses used in applications such as e-voting and auctions, complementing timed-commitment techniques [24]. For *blockchain bridges* [25] that rely on committee-held secrets or authorization keys, PVGSS can make the distribution and later release of such material publicly auditable under non-threshold policies. In *relay networks* [26], PVGSS can support public accountability for committee-held authorization material and can express heterogeneous relay roles through generalized access structures. In *payment or state channels* [27], PVGSS can encode flexible quorum policies for dispute resolution, so that reconstruction is publicly verifiable only when an authorized set participates. In *blockchain oracle* [28] designs based on committee authorization or shared secrets, PVGSS can provide publicly verifiable sharing while supporting weighted or role-dependent privilege structures. In *compliance-aware asset* systems, PVGSS can express auditable approval policies involving regulated parties, e.g., 3-of$\big(Alice, Bob, (t\text{-of } Reg_1, \ldots, Reg_n)\big)$, while leaving policy enforcement and asset-transfer semantics to the application layer.

1.2 Contributions

• **A formal PVGSS primitive.** We define Publicly Verifiable Generalized Secret Sharing (PVGSS) as a non-interactive sharing primitive over public channels. The syntax separates encrypted-share verification, verifiable share decryption, and reconstruction, and the security model captures correctness, encrypted-share verifiability, share-decryption verifiability, and IND1-secrecy for unauthorized labelled sets.

• **Constructions via response-vector testing.** We construct PVGSS by encrypting the shares of an underlying GSS instance and proving, with Schnorr-type Fiat-Shamir proofs, that they form one globally consistent sharing. The core technique is *GSS testing*: the verifier treats the Fiat-Shamir response vector as a masked sharing and checks that it is a valid GSS sharing of the response secret. We give two PVGSS instantiations, one from recursive Shamir sharing and one

from LSSS-based sharing. In both constructions, we instantiate GSS testing with exact-interpolation and dual-code tests.

• **Modular security proofs.** We prove security in the random oracle model under DDH, zero-knowledge and knowledge soundness of the Fiat-Shamir proofs, DLEQ soundness, and privacy of the underlying GSS-on-group scheme. The proof separates the three core guarantees: encrypted-share verifiability, share-decryption verifiability, and IND1-secrecy, so that the role of the GSS testing procedure is explicit.

• **A ledger-hybrid DEX realization from PVGSS.** We use PVGSS as the cryptographic escrow layer in a decentralized exchange protocol with passive, publicly accountable watchers. We formalize the exchange task as an ideal functionality and prove that the concrete protocol realizes it in the ledger-hybrid model, where PVGSS verification binds commitments and arbitration shares to publicly checkable evidence. We also implement the DEX on Ethereum with concrete evaluation results.

2 Related Works

2.1 Access Structures in Secret Sharing

Many classical and widely used SS, VSS, and PVSS constructions [1, 2, 5–7] are designed for threshold access structures, where reconstruction depends only on the number of available shares. This model is simple and efficient, but it does not natively express role-dependent, hierarchical, weighted, or composite authorization policies. Ito et al. [8] and Benaloh and Leichter [9] initiated generalized secret sharing for arbitrary monotone access structures; in particular, Benaloh and Leichter showed how monotone formulas can be realized by composing threshold gates. Linear secret sharing schemes and monotone span programs [10–12] later became a standard algebraic language for representing such policies and are now widely used in expressive access-control systems, including attribute-based cryptography [13, 14].

Several structured subclasses of general access structures have also been studied because they capture common policy patterns more compactly than arbitrary monotone families. Simmons [29] emphasized that practical secret sharing often needs richer authorization semantics than a single threshold. Weighted secret sharing [30, 31] assigns a weight to each participant and authorizes sets whose total weight reaches a threshold. Multipartite secret sharing [32, 33] partitions participants into compartments and specifies how many participants are needed from each compartment. Hierarchical or multilevel secret sharing [29, 34, 35] organizes participants by levels of authority; variants include disjunctive, conjunctive, and distributive weighted threshold schemes [36]. These lines of work motivate expressive access structures, but they typically focus on share distribution and reconstruction rather than non-interactive public verification over encrypted shares.

Public verifiability under general access structures has received less attention. Mashhadi [37] proposed publicly verifiable and proactive secret sharing

schemes for general access structures using bilinear pairings and monotone span programs. More recently, Zhang et al. [38] introduced attribute-based publicly verifiable secret sharing, where a dealer distributes shares according to attributes rather than a fixed shareholder list and uses NIZK proofs to support public verification. Our work is complementary: PVGSS keeps the fixed-shareholder GSS interface, encrypts only leaf shares, and verifies global consistency through response-vector testing, so the public transcript remains aligned with the canonical leaf order of the access structure.

2.2 (Publicly) Verifiable Secret Sharing

A malicious dealer may distribute shares that are inconsistent with one another or with the claimed secret. Verifiable secret sharing (VSS) [2,39] addresses this problem by enabling shareholders to check their own shares. Feldman's scheme [2], for example, augments Shamir sharing with public commitments to the sharing polynomial. Modern VSS constructions often rely on polynomial commitments [39, 41, 42] to improve efficiency, robustness, or asynchrony. However, standard VSS still assumes that shares are delivered privately to shareholders, and verification is usually tied to the participants who receive those shares.

Publicly verifiable secret sharing (PVSS) strengthens this model by allowing anyone to verify the public sharing transcript. Galil and Yung [40] used partitioned encryption and NIZK techniques in an early public-verifiability-oriented construction, and Stadler [5] formalized PVSS by combining encrypted share delivery with publicly checkable proofs. Later instantiations used ElGamal-type encryption [5,43], RSA-based techniques [44], and Paillier encryption [45]. These schemes make public channels possible, but the main body of efficient PVSS work has remained threshold-oriented.

Scalability has been a central theme in threshold PVSS. For n shareholders and threshold t, early schemes typically incurred $O(nt)$ verification work. SCRAPE [6] reduced threshold PVSS verification to $O(n)$ exponentiations by using Reed-Solomon code checks to test consistency of encrypted shares. ALBA-TROSS [46] uses packed Shamir sharing and low-degree exponent interpolation for publicly attestable batched randomness. YOLO YOSO [47] introduced efficient PVSS and resharing tools in the YOSO model, including HEPVSS and DHPVSS. More recent class-group-based constructions such as qCLPVSS [7] and cgVSS [48] support reconstruction of the original field secret $s \in \mathbb{Z}_q$, which is useful for applications such as distributed key generation [49] and MPC [31].

PVGSS follows the PVSS goal of public accountability over public channels, but it targets a different policy layer. Instead of optimizing another threshold PVSS instance, it asks how to certify that encrypted leaf shares form one valid generalized sharing under an arbitrary monotone access structure. The response-vector testing technique used in this paper plays a role analogous to Reed-Solomon consistency testing in threshold PVSS, but it is applied to the GSS response vector and is instantiated for recursive Shamir and LSSS-based access structures.

3 Preliminaries

In this paper, $\mathbb{Z}_q$ denotes a finite field with prime order q, and $\mathbb{G}$ represents a cyclic group of the same order generated by g.

3.1 Lagrange Interpolation and Reed-Solomon Codes

We recall two algebraic notions used to reason about Shamir-type shares and polynomial consistency.

Definition 1 (Lagrange interpolation). *Let $\alpha_1, \ldots, \alpha_m \in \mathbb{Z}_q^*$ be distinct evaluation points. For an index set $I \subseteq [m]$, the Lagrange basis polynomial associated with $i \in I$ is $\ell_i^I(X) = \prod_{j \in I,\, j \neq i} \frac{X - \alpha_j}{\alpha_i - \alpha_j}$. Given values $\{y_i\}_{i \in I}$ with $|I| = t$, there is a unique polynomial $f \in \mathbb{Z}_q[X]$ of degree less than t satisfying $f(\alpha_i) = y_i$ for all $i \in I$, namely $f(X) = \sum_{i \in I} y_i \ell_i^I(X)$. In particular, Shamir reconstruction recovers the constant term as $f(0) = \sum_{i \in I} y_i \ell_i^I(0) = \sum_{i \in I} y_i \prod_{j \in I,\, j \neq i} \frac{-\alpha_j}{\alpha_i - \alpha_j}$. More generally, the same interpolation formula determines $f(\alpha_k)$ at any other evaluation point α_k.*

Definition 2 (Reed-Solomon code and dual code [6]). *For distinct evaluation points $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m) \in (\mathbb{Z}_q^*)^m$, the $[m, t]$ Reed-Solomon code over $\mathbb{Z}_q$ is $\mathsf{RS}_{\boldsymbol{\alpha}}(m, t) = \{(f(\alpha_1), \ldots, f(\alpha_m)) : \ f \in \mathbb{Z}_q[X],\ \deg(f) < t\}$. Its dual code is $\mathsf{RS}_{\boldsymbol{\alpha}}(m, t)^\perp = \{\boldsymbol{h} \in \mathbb{Z}_q^m : \ \langle \boldsymbol{h}, \boldsymbol{c} \rangle = 0$ for all $\boldsymbol{c} \in \mathsf{RS}_{\boldsymbol{\alpha}}(m, t)\}$. Consequently, a vector $\boldsymbol{y} \in \mathbb{Z}_q^m$ is a valid Reed-Solomon codeword if and only if $\langle \boldsymbol{h}, \boldsymbol{y} \rangle = 0$ for every $\boldsymbol{h} \in \mathsf{RS}_{\boldsymbol{\alpha}}(m, t)^\perp$. Equivalently, if $\boldsymbol{y} \notin \mathsf{RS}_{\boldsymbol{\alpha}}(m, t)$, then its syndrome is nonzero, and a uniformly random linear combination of dual parity checks accepts it with probability exactly $1/q$.*

3.2 Generalized Access Structure (GAS)

Definition 3 (GAS [9, 14]). *Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a finite set of parties. A generalized access structure (GAS) is a family $\mathbb{A} \subseteq 2^{\mathcal{P}}$ of authorized subsets satisfying:*

- *Non-triviality: $\emptyset \notin \mathbb{A}$ and $\mathbb{A} \neq \emptyset$.*
- *Monotonicity: if $B \in \mathbb{A}$ and $B \subseteq C \subseteq \mathcal{P}$, then $C \in \mathbb{A}$.*

If $B \in \mathbb{A}$, we say that B satisfies $\mathbb{A}$ or that B is authorized; otherwise B is unauthorized.

For example, Figure 1 represents the Boolean formula $(P_{1,1} \wedge P_{2,1}) \vee (P_{2,2} \wedge P_{3,1})$, and Figure 2 represents the threshold tree $2\text{-of}\big(2\text{-of}(P_{1,1}, P_{2,1}, P_{3,1}), 1\text{-of}(P_{2,2}, P_{4,1})\big)$. The label $P_{i,j}$ denotes the j-th authorization label associated with party P_i, and $P_{i,*}$ denotes any authorization label associated with P_i. Boolean access formulas can be represented as threshold trees: an OR gate over m children is a 1-of-m threshold gate, and an AND gate over m children is an m-of-m threshold gate. In the rest of the paper we use threshold-tree representations unless explicitly stated otherwise.

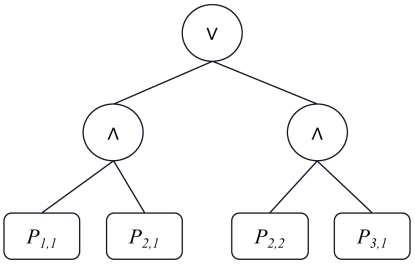

Fig. 1. An access structure represented by a Boolean formula. The sets $\{P_{1,1}, P_{2,1}\}$ and and $\{P_{2,*}, P_{3,1}\}$ are authorized.

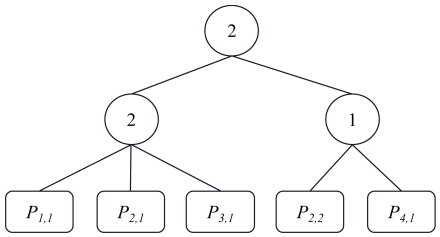

Fig. 2. An access structure represented by threshold gates. The sets $\{P_{3,1}, P_{2,*}\}$ and $\{P_{1,1}, P_{3,1}, P_{4,1}\}$ are authorized.

Corollary 1 *Any threshold-tree representation of $\mathbb{A}$ induces a deterministic membership predicate $\mathsf{Pred}_{\mathbb{A}} : 2^{\mathcal{P}} \to \{1, 0\}$, where $\mathsf{Pred}_{\mathbb{A}}(B) = 1$ if and only if $B \in \mathbb{A}$.* □

Corollary 2 *Let $\mathbb{A}$ be represented by a threshold tree. Every node x induces a sub-access structure $\mathbb{A}_x$ over the leaf labels in the subtree rooted at x. If x is a t_x-of-n_x internal node with children $x_1, \ldots, x_{n_x}$, then a set B satisfies $\mathbb{A}_x$ exactly when it satisfies at least t_x of the child substructures $\mathbb{A}_{x_1}, \ldots, \mathbb{A}_{x_{n_x}}$.* □

Corollary 3 *If a set $B \subseteq \mathcal{P}$ satisfies a threshold-tree access structure $\mathbb{A}$, then there exists a leaf-to-root path such that every induced sub-access structure on the path is satisfied by B. We call such a path a satisfied path.* □

3.3 Shamir Secret Sharing (SS) and LSSS

Definition 4 (Shamir SS). *Shamir secret sharing (SS) lets a dealer split a secret into n shares. Any $t \leq n$ shares reconstruct the secret, while fewer than t shares reveal nothing. It has two algorithms.*

- $\{s_1, s_2, \cdots, s_n\} \leftarrow \mathsf{Share}(s, n, t)$*: Inputs a secret $s \in \mathbb{Z}_q$, n and t, chooses a polynomial $f(x) = s + \sum_{i=1}^{t-1} a_i x^i$, where $a_i \xleftarrow{R} \mathbb{Z}_q$. Finally, the algorithm outputs n shares $f(i)$, $i \in [1, n]$.*
- $s \leftarrow \mathsf{Recon}(Q)$*: Inputs any t shares $Q \subseteq \{s_1, s_2, \cdots, s_n\}$, reconstructs the secret s via the Lagrange interpolation.*

Definition 5 (Shamir SS on Group $\mathbb{G}$). *We also use a group variant for sharing $S \in \mathbb{G}$ when the dealer need not know s such that $S = g^s$:*

- $\{S_1, S_2, \cdots, S_n\} \leftarrow \mathsf{GrpShare}(S, n, t)$*: Inputs a secret $S \in \mathbb{G}$, n and t, chooses a polynomial $f(x) = \sum_{i=1}^{t-1} a_i x^i$, where $a_i \xleftarrow{R} \mathbb{Z}_q$. Set $S_i = S \cdot g^{f(i)}$ for $i \in [1, n]$. The value $S_0 = S$ is only the interpolation target and is never output as a share.*

– $S \leftarrow \mathsf{GrpRecon}(Q)$*: Inputs any* t *shares* Q *with index set* I*, reconstructs* $S \leftarrow S_0 = \prod_{i \in I} S_i^{\prod_{j \in I, j \neq i} \frac{-j}{i-j}}$.

Definition 6 (LSSS [10, 12, 14]). *Let* $\mathcal{P}$ *be a set of shareholders. A linear secret sharing scheme over a ring* $\mathbb{Z}_q$ *for an access structure* $\mathbb{A}$ *is a tuple* (M, ρ)*, where* $M \in \mathbb{Z}_q^{|\mathbb{A}| \times l}$ *is a share-generating matrix with* $|\mathbb{A}|$ *rows and* l *columns* [5] *and* ρ *is a row-labeling function. The scheme consists of the following two algorithms:*

– $\{s_1, s_2, \cdots, s_{|\mathbb{A}|}\} \leftarrow \mathsf{LSSSShare}(s, \mathbb{A})$*: To share a value* $s \in \mathbb{Z}_q$ *under access structure* $\mathbb{A}$*, randomly choose* $v_2, \ldots, v_{|\mathbb{A}|} \xleftarrow{R} \mathbb{Z}_q$ *and set* $\boldsymbol{v} = [s, v_2, \ldots, v_{|\mathbb{A}|}]^\top$*. Then,* $[s_1, \cdots, s_{|\mathbb{A}|}] \leftarrow M\boldsymbol{v}$ *is the vector of shares, where* $s_i \in \mathbb{Z}_q$ *belongs to shareholder* $\rho(i)$ *for each* $i \in [1, \cdots, |\mathbb{A}|]$*.*

– $s \leftarrow \mathsf{LSSSRecon}(\mathbb{A}, Q)$*: Let* $Q = \{(i, s_i)\}_{i \in I_Q}$ *be a labelled set of available shares whose labels satisfy* $\mathbb{A}$*, where* I_Q *denotes the corresponding row indices of* M*. Let* $M_Q \in \mathbb{Z}_q^{|I_Q| \times l}$ *be the submatrix of rows indexed by* I_Q*. For authorized* Q*, there exists* $\boldsymbol{w}_Q \in \mathbb{Z}_q^{|I_Q|}$ *such that* $\boldsymbol{w}_Q^\top M_Q = [1, 0, \ldots, 0]$*, computable by Gaussian elimination. Further, the secret* s *can be reconstructed by* $\sum_{i \in I_Q} \boldsymbol{w}_i s_i$*.*

3.4 Sigma Protocol and NIZK Proofs

Let $\mathcal{R}$ be a relation, and define the corresponding language as $L_{\mathcal{R}} = \{x \mid \exists w : (x, w) \in \mathcal{R}\}$. The values x and w are called the statement and the witness, respectively. A Sigma protocol (P_1, P_2, V) is a three-move public-coin protocol [51]: the prover sends a first message a, the verifier samples a challenge e, and the prover answers with a response z. The verifier accepts if $V(x, a, e, z) = 1$. Applying the Fiat-Shamir transform [16], the challenge is computed as a hash of the public statement and the first message.

We use the standard security properties of Fiat-Shamir transformed Sigma protocols in the random oracle model, where H is modeled as a random oracle. **Completeness** states that honestly generated proofs for true statements are accepted. **Zero-knowledge** states that proofs can be simulated without the witnesses and therefore reveal no information beyond the truth of the statements. For a Fiat-Shamir proof system FS, we write $\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{FS}}(\mathcal{B}_{\mathsf{zk}})$ for the distinguishing advantage of a PPT distinguisher $\mathcal{B}_{\mathsf{zk}}$ between honestly generated proofs and simulated proofs for the same public statement in the random oracle model. **Knowledge soundness** states that an accepting proof for a statement implies efficient extraction of a witness for the underlying relation, except with the extraction failure probability of the proof system. Formally, for a Fiat-Shamir proof system FS for a relation $\mathcal{R}$, let $\mathsf{Adv}^{\mathsf{ks}}_{\mathsf{FS}}(\mathcal{B}_{\mathsf{FS}})$ denote the maximum probability, over

[5] We can consider $\mathbb{A}$ as an access control tree, as illustrated in Figure 1 and Figure 2, where $|\mathbb{A}|$ denotes the number of leaf nodes—or equivalently, the number of secret shares. The value of l, however, depends on the specific parsing algorithm introduced in [14, 50].

the random oracle and the coins of the extractor $\mathcal{B}_{\mathsf{FS}}$, that a PPT prover outputs an accepting Fiat-Shamir proof for a statement x, but $\mathcal{B}_{\mathsf{FS}}$ fails to extract a witness w such that $(x, w) \in \mathcal{R}$. This notation includes the standard random-oracle extraction loss, such as oracle programming, rewinding/forking failure, and hash-collision events. The Schnorr-type NIZK proof and the DLEQ NIZK proof below are the two Fiat-Shamir proof systems used in our constructions.

Schnorr-Type NIZK Proofs: The basic Schnorr relation [15] is $\mathcal{R}_{\mathsf{Sch}} = \{((B, U), w) : U = B^w\}$. The prover samples $r \xleftarrow{R} \mathbb{Z}_q$, computes $A = B^r, c = H(B, U, A)$, and returns $\pi = (A, c, z)$, where $z = r - cw$. The verifier recomputes c and accepts only if $A = U^c B^z$. Indeed, for an honestly generated proof, $U^c B^z = (B^w)^c B^{r-cw} = B^r = A$.

DLEQ NIZK Proofs: The DLEQ proof is the Fiat-Shamir transformed Chaum-Pedersen protocol for the discrete-logarithm equality relation [17] $\mathcal{R}_{\mathsf{DLEQ}} = \{((g, X, Y, Z), x) : X = g^x \wedge Z = Y^x\}$. A non-interactive DLEQ proof system consists of $\pi \leftarrow \mathsf{DLEQ}((g, X, Y, Z), x)$ and $0/1 \leftarrow \mathsf{DLEQVrf}(g, X, Y, Z, \pi)$, where the DLEQ algorithm uses a witness $x \in \mathbb{Z}_q$ satisfying $X = g^x$ and $Z = Y^x$. Concretely, the algorithm samples $r \xleftarrow{R} \mathbb{Z}_q$, computes $T_1 = g^r, T_2 = Y^r, e = H(g, X, Y, Z, T_1, T_2)$, and returns $\pi = (T_1, T_2, e, z)$, where $z = r - ex$. The $\mathsf{DLEQVrf}$ algorithm recomputes the challenge e and accepts only if $T_1 = X^e g^z$ and $T_2 = Z^e Y^z$. Indeed, $X^e g^z = (g^x)^e g^{r-ex} = g^r = T_1$ and $Z^e Y^z = (Y^x)^e Y^{r-ex} = Y^r = T_2$.

3.5 Decisional Diffie-Hellman (DDH) Assumption

The Decisional Diffie-Hellman (DDH) [6] assumption states that (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c) are computationally indistinguishable, where $a, b, c \xleftarrow{R} \mathbb{Z}_q$. For a PPT distinguisher $\mathcal{B}_{\mathsf{ddh}}$, define $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{B}_{\mathsf{ddh}}) = |\Pr[\mathcal{B}_{\mathsf{ddh}}(g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{B}_{\mathsf{ddh}}(g, g^a, g^b, g^c) = 1]|$. The DDH assumption requires $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{B}_{\mathsf{ddh}}) \leq \mathsf{negl}(\lambda)$ for every PPT $\mathcal{B}_{\mathsf{ddh}}$, where λ is the security parameter.

4 Generalized Secret Sharing (GSS)

We first define two types of GSS schemes, which differ based on the shared secret, $s \in \mathbb{Z}_q$ or $S \in \mathbb{G}$.

4.1 Definition

Definition 7 (GSS). *A generalized secret sharing (GSS) scheme [9] allows a dealer to share a secret $s \in \mathbb{Z}_q$ using arbitrary monotone access structure. The secret s can be recovered only given an authorized set. A GSS scheme is defined by the following two algorithms.*

- $\{s_1, s_2, \cdots, s_{|\mathbb{A}|}\} \leftarrow \mathsf{GSSShare}(s, \mathbb{A})$*: Inputs a value* $s \in \mathbb{Z}_q$ *and an access structure* $\mathbb{A}$*, outputs the labelled secret shares* $\{s_1, s_2, \cdots, s_{|\mathbb{A}|}\}$ *in the canonical leaf order of* $\mathbb{A}$*, where* $|\mathbb{A}|$ *is the number of leaf nodes under access tree* $\mathbb{A}$*.*
- $s \leftarrow \mathsf{GSSRecon}(\mathbb{A}, Q)$*: Inputs an access structure* $\mathbb{A}$ *and a labelled authorized share set* $Q = \{(i, s_i)\}_{i \in I_Q}$*, reconstructs the secret* s*.*

Definition 8 (GSS on Group $\mathbb{G}$). *Here we consider how to share secret* $S \in \mathbb{G}$ *and the dealer does not know the value* s *such that* $S = g^s$*. The GSS on group* $\mathbb{G}$ *of order* q *is defined as below:*

- $\{S_1, S_2, \cdots, S_{|\mathbb{A}|}\} \leftarrow \mathsf{GrpGSSShare}(S, \mathbb{A})$*: Inputs a value* $S \in \mathbb{G}$ *and an access structure* $\mathbb{A}$*, outputs the labelled group shares* $\{S_1, S_2, \cdots, S_{|\mathbb{A}|}\}$ *in the same canonical leaf order as Definition 7.*
- $S \leftarrow \mathsf{GrpGSSRecon}(\mathbb{A}, Q)$*: Inputs an access structure* $\mathbb{A}$ *and a labelled authorized group-share set* $Q = \{(i, S_i)\}_{i \in I_Q}$*, reconstructs the secret* S*.*

Similar to Shamir SS, GSS and GSS on Group $\mathbb{G}$ should also satisfy **correctness** and **secrecy**.

4.2 Shamir SS-based GSS

Then, we initialize each type (Definition 7 and Definition 8) using two approaches, i.e., Shamir SS-based and LSSS-based. Technically, the Shamir SS-based primitive requires polynomial evaluation in the secret sharing and Lagrange interpolation in the secret recovery; The LSSS-based primitive needs to handle matrix multiplication both in secret sharing and recovery.

(1) ***GSS based on Shamir SS:*** Benaloh et al. [9] had introduced the construction, which can be obtained by replacing GrpShare (and GrpRecon) with Share (and Recon) in Figure 3. Note that the sharing phase is carried out by traversing the access control structure tree $\mathbb{A}$ from top to bottom, while the recovery process is performed by traversing the tree from bottom to top. We omit the security proofs of **correctness** and **secrecy**, which is proven by Benaloh et al. [9].

(2) ***GSS on Group $\mathbb{G}$ based on Shamir SS:*** Without loss of generalization, an access structure $\mathbb{A}$ is a set of shareholders who are constraint by the threshold gate (refer to Figure 2 as an example). The non-leaf nodes are threshold gate values and the leaf nodes are identifiers of shareholders. The GSS scheme attaches a share for each (non-leaf or leaf) node and it is described by Figure 3.

For any node in GSS scheme on $\mathbb{G}$, the **correctness** and **secrecy** can be derived directly from those of Shamir SS scheme on $\mathbb{G}$, which is given by Lemma 1. Thus, the **correctness** and **secrecy** of GSS on group $\mathbb{G}$ based on Shamir SS (Figure 3) are satisfied.

Lemma 1. *Shamir SS on group* $\mathbb{G}$ *(Definition 5) satisfies* ***correctness*** *and* ***secrecy****.*

Functionality GSS on Group $\mathbb{G}$ using Shamir SS

$\{S_1, S_2, \cdots, S_{|\mathbb{A}|}\} \leftarrow \mathsf{GrpGSSShare}(S, \mathbb{A})$:

set $F_R \leftarrow S$ for the root node R.

For non-leaf node x with n_x children nodes and threshold-gate t_x:

$\{S_{x_1}, S_{x_2}, \cdots, S_{xn_x}\} \leftarrow \mathsf{GrpShare}(F_x, n_x, t_x)$

set $F_z \leftarrow S_{x_j}$, where z is the jth child of x

For the ith leaf node z in $\mathbb{A}$:

$S_i = S_{x_j}$, where z is the jth child of its parent node x

$(S) \leftarrow \mathsf{GrpGSSRecon}(\mathbb{A}, Q)$:

Denote *path* as the satisfied path (cf. Corollary 3)

For each node x in *path* from leaf to root:

$F_x \leftarrow \mathsf{GrpRecon}(Q_x)$, where $Q_x \subseteq \{F_{x_1}, \cdots, F_{x_{n_x}}\}$ generated at node x and $|Q_x| \geq t_x$. (**Remark:** if x is a leaf node, $F_x = Q_x$.)

$S \leftarrow F_R$, where R is the root node.

Fig. 3. GSS scheme on Group $\mathbb{G}$ based on Shamir SS

Proof. **Correctness**: The shares are $S_i = S \cdot g^{f(i)}$ for $i \in [1, n]$, and the interpolation target is $S_0 = S$. Following the correctness of Shamir SS scheme, inputs a set of any t shares Q, we can reconstruct the secret $\prod_{i \in I} S_i^{\prod_{j \in I, j \neq i} \frac{-j}{i-j}} = S \cdot g^{f(0)} = S$, where I is the set containing indexes of the shares Q. **Secrecy**: If less than t shares can reconstruct the secret, then we have $\sum_{i \in I} f(i) \prod_{j \in I, j \neq i} \frac{-j}{i-j} = f(0), |I| < t$, which violates the security properties of Shamir SS scheme. □

4.3 LSSS-based GSS

(1) ***GSS based on LSSS:*** It is obvious that LSSS and GSS schemes have the same algorithms, including inputs and ouputs. Figure 4 demonstrate that LSSS is an instance of GSS. Therefore, the **correctness** and **secrecy** are inherently guaranteed.

Functionality GSS using LSSS

$\{s_1, s_2, \cdots, s_{|\mathbb{A}|}\} \leftarrow \mathsf{GSSShare}(s, \mathbb{A})$:

$\{s_1, s_2, \cdots, s_{|\mathbb{A}|}\} \leftarrow \mathsf{LSSSShare}(s, \mathbb{A})$

$(s) \leftarrow \mathsf{GSSRecon}(\mathbb{A}, Q)$:

$s \leftarrow \mathsf{LSSSRecon}(\mathbb{A}, Q)$

Fig. 4. GSS based on LSSS

(2) ***GSS on Group $\mathbb{G}$ using LSSS:*** We first depict how to construct LSSS on group $\mathbb{G}$ by Figure 5. The construction resembles the $\mathsf{LSSSShare}$ and $\mathsf{LSSSRecon}$ algorithms of the LSSS scheme, as defined in Definition 6. We prove that it satisfies the **correctness** and **secrecy** by Lemma 2.

Functionality LSSS on Group $\mathbb{G}$

$\{S_1, S_2, \cdots, S_{|\mathbb{A}|}\} \leftarrow \mathsf{GrpLSSSShare}(S, \mathbb{A})$: denote $S = g^s$ and s is unknown.

parse $\mathbb{A}$ as (M, ρ) leveraging the algorithms in [14] or [50]

$\boldsymbol{v} = [1, v_2, \ldots, v_l]^\top$, where $v_2, \ldots, v_l \xleftarrow{R} \mathbb{Z}_q$.

$(s_1, \cdots, s_{|\mathbb{A}|}) \leftarrow M\boldsymbol{v}$

$S_i = S^{s_i} = g^{s \cdot s_i}$ for $i \in [1, \cdots, |\mathbb{A}|]$

$(S) \leftarrow \mathsf{GrpLSSSRecon}(\mathbb{A}, Q)$:

parse $\mathbb{A}$ as (M, ρ)

calculate I_Q as $I_Q = \{i | \rho(i) \in Q\}$

set $M_Q \in \mathbb{Z}_q^{|I_Q| \times l}$, where M_Q is the subset of rows in M that are indexed by I_Q

find $w_Q \in \mathbb{Z}_q^{|I_Q|}$ such that $w_Q^\top M_Q = [1, 0, \ldots, 0]$

$S \leftarrow \prod_{i \in I_Q} S_i^{w_i}$

Fig. 5. LSSS on Group $\mathbb{G}$

Lemma 2. *LSSS on group $\mathbb{G}$ (Figure 5) satisfies **correctness** and **secrecy**.*

Proof. **Correctness**: The shares are $S_i = S^{s_i} = g^{s \cdot s_i}$ for $i \in [1, \cdots, |\mathbb{A}|]$. Q is an authorized set with $I_Q = \{i | \rho(i) \in Q\}$ and $M_Q \in \mathbb{Z}_q^{|I_Q| \times l}$, which means there exists $w_Q \in \mathbb{Z}_q^{|I_Q|}$ such that $w_Q^\top M_Q = [1, 0, \ldots, 0]$. Following the correctness of LSSS scheme, we have $\sum_{i \in I_Q} w_i \cdot s_i = 1$, thus $S = \prod_{i \in I_Q} S_i^{w_i} = \prod_{i \in I_Q} g^{s \cdot s_i \cdot w_i} = (g^s)^{\sum_{i \in I_Q} w_i \cdot s_i} = g^s$. **Secrecy**: If unauthorized set Q can reconstruct $g^s = \prod_{i \in I_Q} S_i^{w_i}$, then it can find $w_Q \in \mathbb{Z}_q^{|I_Q|}$ such that $w_Q^\top M_Q = [1, 0, \ldots, 0]$, which violates the security properties of LSSS scheme, as only authorized sets can find such w_Q. □

Then, we apply the construction of Figure 5 to construct GSS on group $\mathbb{G}$, as Figure 6 presents. It can be observed that the algorithm $\mathsf{GrpGSSShare}$ (w.r.t. $\mathsf{GrpGSSRecon}$ directly executes $\mathsf{GrpLSSSShare}$ (w.r.t. $\mathsf{GrpLSSSRecon}$). Apparently, LSSS on group $\mathbb{G}$ can be regarded as an instance of GSS on group $\mathbb{G}$. Thus, **correctness** and **secrecy** are inherently guaranteed by Lemma 2.

Functionality GSS on Group $\mathbb{G}$ using LSSS

$\{S_1, S_2, \cdots, S_{|\mathbb{A}|}\} \leftarrow \mathsf{GrpGSSShare}(S, \mathbb{A})$:

$\quad \{S_1, S_2, \cdots, S_{|\mathbb{A}|}\} \leftarrow \mathsf{GrpLSSSShare}(S, \mathbb{A})$

$(S) \leftarrow \mathsf{GrpGSSRecon}(\mathbb{A}, Q)$:

$\quad S \leftarrow \mathsf{GrpLSSSRecon}(\mathbb{A}, Q)$

Fig. 6. GSS on Group $\mathbb{G}$ based on LSSS

4.4 GSS Testing

The reconstruction algorithm GSSRecon only needs an authorized subset of shares. This is sufficient for recovering the secret, but it is not sufficient for public verification in PVGSS: the verifier must check that the entire vector appearing in the proof response is one globally consistent sharing of a claimed response secret. Otherwise, a malicious dealer could make some authorized subsets reconstruct correctly while other leaves are inconsistent with the declared access structure. We capture this global consistency check as GSS testing.

Definition 9 (GSS Testing). *Let $\mathbb{A}$ be an access structure with $|\mathbb{A}|$ leaves in canonical order. A GSS testing algorithm $b \leftarrow \mathsf{GSSTest}(\mathbb{A}, \boldsymbol{x}, x)$ takes a candidate share vector $\boldsymbol{x} = (x_1, \ldots, x_{|\mathbb{A}|}) \in \mathbb{Z}_q^{|\mathbb{A}|}$ and a candidate secret $x \in \mathbb{Z}_q$, and outputs $b \in \{0, 1\}$. It is complete if $\mathsf{GSSTest}(\mathbb{A}, \boldsymbol{x}, x) = 1$ for every honestly generated vector $\boldsymbol{x} \leftarrow \mathsf{GSSShare}(x, \mathbb{A})$. It is sound with error ϵ if, except with probability at most ϵ, acceptance implies that $\boldsymbol{x}$ is a valid output of $\mathsf{GSSShare}(x, \mathbb{A})$.*

We use the following two testing procedures for threshold-gate access trees, using the notation of Definitions 1 and 2. By Corollary 2, each internal node of an access tree together with its children can be treated as a sub-access structure, which allows GSS testing to verify the sharing constraints recursively from the root to the leaves. Our polynomial/RS-code based GSS testing applies to Shamir-based threshold structures and recursively generated tree-based LSSS constructions, including the Liu et al. construction [50][6] used in our implementation. For an internal node v, let t_v be its threshold, let m_v be the number of its children, and let $\alpha_{v,1}, \ldots, \alpha_{v,m_v} \in \mathbb{Z}_q^*$ be their distinct evaluation points. Let r denote the root of $\mathbb{A}$, and let $Q = \{(i, x_i)\}_{i \in [|\mathbb{A}|]}$ denote the candidate leaf shares represented by $\boldsymbol{x}$.

① *Exact interpolation test.* Sample a minimal authorized subset $Q^* \subseteq Q$. Equivalently, for each internal node v, choose a set $I_v \subseteq [m_v]$ of size t_v that

[6] In that conversion, expanding a t-of-m gate appends a local Vandermonde block over $\mathbb{Z}_q$, so the values assigned to the children of each internal gate are still evaluations of a degree-$< t$ polynomial whose constant term is the value assigned to the parent. Hence the resulting LSSS share vector preserves the same local Reed-Solomon structure as the tree representation, and GSS testing can be performed on the canonical leaf order induced by the tree.

satisfies v, and recurse on the selected children until the selected leaves are exactly Q^*. The test assigns the claimed value $z_r := x$ to the root. For every internal node v, after the values $\{z_{v,j}\}_{j\in I_v}$ of the selected children have been obtained recursively, define the unique polynomial $f_v(X) = \sum_{j\in I_v} z_{v,j}\ell_j^{I_v}(X)$ with $\deg(f_v) < t_v$. Here the basis polynomials $\ell_j^{I_v}$ are defined with respect to the local evaluation points $\{\alpha_{v,j}\}_{j\in I_v}$. The local root consistency condition is $f_v(0) = z_v$. For every unselected child $j \notin I_v$, set $z_{v,j} := f_v(\alpha_{v,j})$ and verify the corresponding child subtree using $z_{v,j}$ as its claimed value; for a leaf child, this verification is the equality check $z_{v,j} = x_i$ against the corresponding entry of Q. The test accepts if and only if the consistency checks at all internal nodes and all leaf comparisons accept. Hence the accepted vector is a valid recursive Shamir sharing of x. The random choice of Q^* only fixes an interpolation basis and does not introduce soundness error.

② *Dual-code test.* The second instantiation uses the dual-code characterization of Reed–Solomon codewords in Definition 2. For every internal node v with children $v_1, \ldots, v_{m_v}$, consider the local word $\boldsymbol{w}_v = (z_v, z_{v,1}, \ldots, z_{v,m_v}) \in \mathbb{Z}_q^{m_v+1}$ over the extended evaluation domain $(0, \alpha_{v,1}, \ldots, \alpha_{v,m_v})$. The local Shamir constraint at v holds exactly when $\boldsymbol{w}_v$ is the evaluation of a degree-$< t_v$ polynomial on this domain, or equivalently when it is orthogonal to the corresponding dual Reed–Solomon code. Rather than explicitly interpolating the local polynomial, the verifier samples a random dual-code check at each internal node and tests the resulting orthogonality relation, together with the root condition $z_r = x$, through one random linear aggregation. Thus, an honest recursive sharing always passes verification. Conversely, if the candidate vector is invalid, then either $z_r - x \neq 0$ or at least one dual-code inner product is nonzero. In such a case, the random linear aggregation can conceal such a nonzero value only with probability $1/q$.

Lemma 3 (Soundness of GSS Testing). *Let $\mathbb{A}$ be a threshold-gate access tree over $\mathbb{Z}_q$ with distinct local evaluation points at every internal gate. For recursive Shamir GSS, and for the Liu et al. tree-to-LSSS realization used in this paper, the exact interpolation test is complete and perfectly sound. The randomized dual-code test is complete and sound with error at most $1/q$, where the probability is over the verifier's random dual-code check.*

Proof. Completeness follows from the local correctness of Shamir sharing at every threshold gate. In an honestly generated sharing, every internal t_v-of-m_v gate v is assigned a polynomial f_v of degree less than t_v, whose constant term is the value at v and whose evaluations at $\alpha_{v,1}, \ldots, \alpha_{v,m_v}$ are the values assigned to its children. Thus the exact interpolation test reconstructs the same local polynomials, and every local evaluation word is a Reed–Solomon codeword. By Definition 2, all dual-code checks also vanish, and both tests accept. The same reasoning applies to the underlying LSSS [50], as its tree-to-LSSS conversion preserves these local Vandermonde, equivalently Reed–Solomon, constraints.

We next prove perfect soundness of the exact interpolation test. Suppose the test accepts a candidate vector $\boldsymbol{x}$. At every internal gate v, the selected t_v child values determine a unique polynomial f_v of degree $< t_v$. Acceptance requires

$f_v(0)$ to equal the value assigned to v, and requires every unselected child value to equal the corresponding evaluation of f_v. Therefore all children of v, not only the selected ones, are consistent with the same local Shamir polynomial. Applying this argument recursively from the root to the leaves shows that the accepted leaf vector must have been generated by a recursive Shamir sharing whose root value is the claimed secret x. Hence the exact interpolation test has soundness error 0.

For the dual-code test, fix any candidate pair $(\boldsymbol{x}, x)$ such that $\boldsymbol{x}$ is not a valid recursive Shamir sharing of x. For each internal node v, let $\mathcal{C}_v$ be the Reed–Solomon code of dimension t_v over the extended evaluation domain $(0, \alpha_{v,1}, \ldots, \alpha_{v,m_v})$, and let $\boldsymbol{w}_v = (z_v, z_{v,1}, \ldots, z_{v,m_v})$ be the corresponding local word. Since the pair $(\boldsymbol{x}, x)$ is invalid, either the reconstructed root value differs from x, or there exists an internal node v such that $\boldsymbol{w}_v \notin \mathcal{C}_v$. In the latter case, Definition 2 implies that the linear functional $\boldsymbol{h} \in \mathcal{C}_v^{\perp} \mapsto \langle \boldsymbol{h}, \boldsymbol{w}_v \rangle$ over the dual code is nonzero. Hence an invalid pair defines a nonzero global error vector consisting of the root discrepancy $z_r - x$ and the local dual-code violations. The randomized dual-code test accepts exactly when a uniformly random linear functional vanishes on this vector. For every fixed nonzero vector over $\mathbb{Z}_q$, this happens with probability $1/q$. Therefore any fixed pair $(\boldsymbol{x}, x)$ that is not a valid recursive Shamir sharing passes the randomized dual-code test with probability at most $1/q$. □

5 Publicly Verifiable Generalized Secret Sharing

5.1 Definition

Definition 10 (PVGSS). *A publicly verifiable generalized secret sharing (PVGSS) scheme allows a dealer to split a secret using an arbitrary monotone access structure in a publicly verifiable manner. Shares are encrypted and can be delivered over a public channel. The correctness of encrypted shares and decrypted reconstruction shares can be verified by any external party. The ordered public-key list is always aligned with the canonical leaf order of the access structure. The PVGSS scheme includes the following algorithms.*

- $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{PVGSSSetup}(\lambda)$*: Inputs security parameter* λ*, outputs a key pair* $(\mathsf{sk}_i, \mathsf{pk}_i)$ *for shareholder* P_i*.*
- $(\{C_i\}, \mathsf{prf}_s) \leftarrow \mathsf{PVGSSShare}(s, \mathbb{A}, \{\mathsf{pk}_i\})$*: Inputs a secret* $s \in \mathbb{Z}_q$*, an access structure* $\mathbb{A}$*, and an ordered list of public keys* $\{\mathsf{pk}_i\}_{i \in [|\mathbb{A}|]}$*, outputs encrypted shares* $\{C_i\}$ *and the corresponding NIZK proof* prf_s*.*
- $(0/1) \leftarrow \mathsf{PVGSSVerify}(\{C_i\}, \mathsf{prf}_s, \mathbb{A}, \{\mathsf{pk}_i\})$*: Inputs encrypted shares* $\{C_i\}$*, proof* prf_s*, access structure* $\mathbb{A}$*, and ordered public keys, outputs 1 if the shares are correctly encrypted. Otherwise, outputs 0.*
- $(S_i, \mathsf{prf}_{S_i}) \leftarrow \mathsf{PVGSSPreRecon}(C_i, \mathsf{sk}_i)$*: Inputs ciphertext* C_i *and the corresponding secret key* sk_i*, outputs a decrypted share* S_i *and the attached NIZK proof* prf_{S_i}*.*

– $(0/1) \leftarrow \mathsf{PVGSSKeyVrf}(C_i, S_i, \mathsf{prf}_{S_i}, \mathsf{pk}_i)$*: Inputs the encrypted share* C_i*, the decrypted share* S_i*, the NIZK proof* prf_{S_i}*, and public key* pk_i*, outputs 1 if* S_i *is a correct decryption of* C_i*.*
– $(S) \leftarrow \mathsf{PVGSSRecon}(\mathbb{A}, Q)$*: Inputs the access structure* $\mathbb{A}$ *and a labelled authorized set of decrypted shares* $Q = \{(i, S_i)\}_{i \in I_Q}$*, outputs the recovered secret* $S(= g^s)$*.*

PVGSS scheme is required to achieve security properties, which are formally defined as follows.

Definition 11 (Correctness). *The PVGSS is correct if for every access structure* $\mathbb{A}$*, every labelled authorized index set* I_Q*, and each* $s \in \mathbb{Z}_q$*,*

$$\Pr\left[\begin{array}{l|l} \mathsf{PVGSSVerify}\left(\{C_i\}, \mathsf{prf}_s, \mathbb{A}, \{\mathsf{pk}_i\}\right) = 1 \wedge & \forall i \in [|\mathbb{A}|],\ (\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{PVGSSSetup}(\lambda); \\ \forall i \in I_Q: & (\{C_i\}, \mathsf{prf}_s) \leftarrow \mathsf{PVGSSShare}(s, \mathbb{A}, \{\mathsf{pk}_i\}) \\ \quad \mathsf{PVGSSKeyVrf}(C_i, S_i, \mathsf{prf}_{S_i}, \mathsf{pk}_i) = 1 \wedge & \forall i \in I_Q,\ (S_i, \mathsf{prf}_{S_i}) \leftarrow \mathsf{PVGSSPreRecon}(C_i, \mathsf{sk}_i) \\ S' = g^s & Q \leftarrow \{(i, S_i)\}_{i \in I_Q}; \\ & (S') \leftarrow \mathsf{PVGSSRecon}(\mathbb{A}, Q); \end{array}\right] = 1$$

Definition 12 (Verifiability of Encrypted Shares). *The PVGSS satisfies verifiability of encrypted shares if for every access structure* $\mathbb{A}$ *and every PPT* $\mathcal{A}$*, where* $m = |\mathbb{A}|$ *and* $\mathsf{Valid}_{\mathbb{A}}(s, \mathbf{s})$ *denotes that* $\mathbf{s} = (s_1, \ldots, s_m)$ *is a valid GSS sharing of* s *under* $\mathbb{A}$*, the advantage*

$$\mathsf{Adv}^{\mathsf{venc}}_{\mathsf{PVGSS}}(\mathcal{A}) = \Pr\left[\begin{array}{l|l} \mathsf{PVGSSVerify}(\{C_i\}, \mathsf{prf}_s, \mathbb{A}, \{\mathsf{pk}_i\}) = 1 \wedge & \forall i \in [m],\ (\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{PVGSSSetup}(\lambda), \\ \nexists (s, \mathbf{s}) \in \mathbb{Z}_q \times \mathbb{Z}_q^m \ s.t. & (\{C_i\}, \mathsf{prf}_s) \leftarrow \mathcal{A}(\mathbb{A}, \{\mathsf{pk}_i\}) \\ \quad \mathsf{Valid}_{\mathbb{A}}(s, \mathbf{s}) \wedge \forall i \in [m],\ C_i = \mathsf{pk}_i^{s_i} & \end{array}\right]$$

is negligible.

Definition 13 (Verifiability of Share Decryption). *The PVGSS satisfies verifiability of share decryption if for every PPT* $\mathcal{A}$ *and every polynomially bounded number* N *of registered keys, the advantage*

$$\mathsf{Adv}^{\mathsf{vdec}}_{\mathsf{PVGSS}}(\mathcal{A}) = \Pr\left[\begin{array}{l|l} \mathsf{PVGSSKeyVrf}(C, S, \mathsf{prf}_S, \mathsf{pk}) = 1 \wedge & \forall i \in [N],\ (\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{PVGSSSetup}(\lambda), \\ \nexists x \in \mathbb{Z}_q^* \ s.t.\ \mathsf{pk} = g^x \ and\ C = S^x & (\mathsf{pk}, C, S, \mathsf{prf}_S) \leftarrow \mathcal{A}(\{\mathsf{pk}_i\}),\ \mathsf{pk} \in \{\mathsf{pk}_i\} \end{array}\right]$$

is negligible.

Definition 14 (IND1-secrecy). *PVGSS satisfies IND1-secrecy if for every PPT adversary* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{\mathsf{ind1}}_{\mathsf{PVGSS}}(\mathcal{A}) = \left|\Pr[\mathsf{Exp}^{\mathsf{ind1}}_{\mathsf{PVGSS},\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right|$ *is negligible, where the experiment* $\mathsf{Exp}^{\mathsf{ind1}}_{\mathsf{PVGSS},\mathcal{A}}(\lambda)$ *is defined as follows.*

1. $\mathcal{A}$ *outputs a challenge access structure* $\mathbb{A}^*$ *and an index set* U*. Let* $m = |\mathbb{A}^*|$*. The challenger rejects unless* $U \subseteq [1, \ldots, m]$ *and the labelled set* $\{P_i : i \in U\}$ *is unauthorized for* $\mathbb{A}^*$*.*
2. *The challenger runs* $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{PVGSSSetup}(\lambda)$ *for every* $i \in [1, \ldots, m]$*, and gives the ordered public-key list* $\{\mathsf{pk}_i\}_{i \in [1,\ldots,m]}$ *to* $\mathcal{A}$*.*

3. *$\mathcal{A}$ outputs two distinct challenge secrets $x_0, x_1 \in \mathbb{Z}_q$.*
4. *The challenger samples $b \xleftarrow{R} \{0,1\}$, computes $(\{C_i^*\}_{i\in[1,\ldots,m]}, \mathsf{prf}_{x_b}) \leftarrow \mathsf{PVGSSShare}(x_b, \mathbb{A}^*, \{\mathsf{pk}_i\}_{i\in[1,\ldots,m]})$, and sends $(\{C_i^*\}_{i\in[1,\ldots,m]}, \mathsf{prf}_{x_b})$ to $\mathcal{A}$.*
5. *$\mathcal{A}$ may adaptively query decrypted shares for indices $i \in U$. For each such query, the challenger returns $(S_i^*, \mathsf{prf}_{S_i}^*) \leftarrow \mathsf{PVGSSPreRecon}(C_i^*, \mathsf{sk}_i)$.*
6. *$\mathcal{A}$ outputs $b' \in \{0,1\}$. The experiment outputs 1 iff $b' = b$.*

5.2 PVGSS Construction Based on GSS

Usually, PVSS schemes are constructed based on SS. Informally speaking, PVSS combines SS and public key encryption with verifiability. Intuitively, PVGSS can be constructed similarly by replacing SS with GSS. We leverage the original GSS scheme in the sharing phase and employ GSS on group $\mathbb{G}$ in the reconstruction phase. Figure 7 demonstrates the general PVGSS construction by taking advantage of GSS scheme and NIZK proofs. The sharing proofs are Schnorr-type NIZK proofs obtained through the Fiat-Shamir transform. The initiation phase comprises the PVGSSSetup algorithm. The sharing phase involves the PVGSSShare and PVGSSVerify algorithms. The reconstruction phase includes PVGSSPreRecon, PVGSSKeyVrf, and PVGSSRecon algorithms.

In the PVGSSSetup algorithm, each shareholder P_i generates its key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$, where $\mathsf{pk}_i = g^{\mathsf{sk}_i}$ and sk_i is randomly chosen from $\mathbb{Z}_q^*$.

In the PVGSSShare algorithm, the dealer first invokes GSSShare algorithm for secret $s \in \mathbb{Z}_q$ to generate secret shares based on the access structure $\mathbb{A}$. Note that a PVGSS is not merely a combination of multiple PVSS instances, since it outputs encrypted shares only for the leaf nodes. As previously introduced, $|\mathbb{A}|$ denotes the number of leaf nodes in the access structure $\mathbb{A}$. Then, it hides the $|\mathbb{A}|$ shares directly using the targeted shareholder's public key in the same leaf order, i.e., $\mathsf{pk}_i^{s_i}$. Next, it generates Schnorr-type Fiat-Shamir NIZK proofs. Specifically, $\{C_i'\}$, c, and $\hat{s}$ together with $\{\hat{s}_i\}$ are the commitment, the challenge value, and the response values, respectively. The challenge binds the ordered public keys and the access structure. Therefore, the NIZK proof prf_s is $(\{C_i'\}, c, \hat{s}, \{\hat{s}_i\})$. To this end, the dealer has successfully generated encrypted secret shares $\{C_i\}$ and the corresponding NIZK proof prf_s which can be adopted to prove the dealer's honesty.

In the PVGSSVerify algorithm, any (external) party can check whether the dealer has honestly performed the sharing phase. Due to the presence of $|\mathbb{A}|$ independent shares $\{s_i\}$, the verifier checks group membership, recomputes the Fiat-Shamir challenge, and then verifies two algebraic conditions. The first equation (executed $|\mathbb{A}|$ times) is a Schnorr-type proof of knowledge for each encrypted share s_i. The second check invokes GSSTest directly to verify that all response values $\{\hat{s}_i\}$ are valid GSS shares of the response secret $\hat{s}$. It binds the shares embedded in $\{C_i\}$ to the secret s under the access structure $\mathbb{A}$. Refer to Theorem 1 for details about the correctness.

In the PVGSSPreRecon algorithm, the shareholder, holding sk_i, decrypts C_i with its secret key sk_i and obtains $S_i = g^{s_i}$. Additionally, the shareholder em-

Functionality PVGSS Based on GSS

$(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{PVGSSSetup}(\lambda)$:

$\mathsf{sk}_i \xleftarrow{R} \mathbb{Z}_q^*$

$\mathsf{pk}_i \leftarrow g^{\mathsf{sk}_i}$

$(\{C_i\}, \mathsf{prf}_s) \leftarrow \mathsf{PVGSSShare}(s, \mathbb{A}, \{\mathsf{pk}_i\})$:

reject if any pk_i is invalid or the list order is ambiguous

$\{s_1, s_2, \cdots, s_{|\mathbb{A}|}\} \leftarrow \mathsf{GSSShare}(s, \mathbb{A})$

$C_i = \mathsf{pk}_i^{s_i} \ \forall i \in [1, \ldots, |\mathbb{A}|]$

$s' \xleftarrow{R} \mathbb{Z}_q$

$(s'_1, \ldots, s'_{|\mathbb{A}|}) \leftarrow \mathsf{GSSShare}(s', \mathbb{A})$

$C'_i = \mathsf{pk}_i^{s'_i} \ \forall i \in [1, \ldots, |\mathbb{A}|]$

$c \leftarrow H(\mathbb{A}, \{\mathsf{pk}_i\}, \{C_i\}, \{C'_i\})$

$\hat{s} \leftarrow s' - c \cdot s$

$\hat{s}_i \leftarrow s'_i - c \cdot s_i \ \forall i \in [1, \ldots, |\mathbb{A}|]$

$\mathsf{prf}_s \leftarrow (\{C'_i\}, c, \hat{s}, \{\hat{s}_i\})$

$(0/1) \leftarrow \mathsf{PVGSSVerify}(\{C_i\}, \mathsf{prf}_s, \mathbb{A}, \{\mathsf{pk}_i\})$:

reject if any pk_i, C_i, or C'_i is invalid

$c \stackrel{?}{=} H(\mathbb{A}, \{\mathsf{pk}_i\}, \{C_i\}, \{C'_i\})$

$C'_i \stackrel{?}{=} C_i^c \cdot \mathsf{pk}_i^{\hat{s}_i} \ \forall i \in [1, \ldots, |\mathbb{A}|]$

$1 \stackrel{?}{=} \mathsf{GSSTest}(\mathbb{A}, \{\hat{s}_i\}_{i \in [1, \ldots, |\mathbb{A}|]}, \hat{s})$

return 1 iff all checks pass

$(S_i, \mathsf{prf}_{S_i}) \leftarrow \mathsf{PVGSSPreRecon}(C_i, \mathsf{sk}_i)$:

reject if C_i is invalid

$S_i = C_i^{1/\mathsf{sk}_i} = g^{s_i}$

$\mathsf{prf}_{S_i} = \mathsf{DLEQ}((g, \mathsf{pk}_i, S_i, C_i), \mathsf{sk}_i)$

$(0/1) \leftarrow \mathsf{PVGSSKeyVrf}(C_i, S_i, \mathsf{prf}_{S_i}, \mathsf{pk}_i)$:

reject if pk_i, C_i, or S_i is invalid

$\mathsf{true} \stackrel{?}{=} \mathsf{DLEQVrf}(g, \mathsf{pk}_i, S_i, C_i, \mathsf{prf}_{S_i})$

$(S) \leftarrow \mathsf{PVGSSRecon}(\mathbb{A}, Q)$:

$S \leftarrow \mathsf{GrpGSSRecon}(\mathbb{A}, Q)$ where $S = g^s$ and $Q = \{(i, S_i)\}_{i \in I_Q}$ is a labelled authorized set of $\mathbb{A}$.

Fig. 7. PVGSS scheme based on GSS and NIZK proofs

ploys the DLEQ algorithm to generate a NIZK proof prf_{S_i}, demonstrating that $\log_g \mathsf{pk}_i = \log_{S_i} C_i = \mathsf{sk}_i$, i.e., $C_i = S_i^{\mathsf{sk}_i}$.

The $\mathsf{PVGSSKeyVrf}$ algorithm directly invokes the corresponding DLEQ verification algorithm, $\mathsf{DLEQVrf}$, to verify the correctness of S_i provided by the shareholder. Note that no information about sk_i is leaked during this process.

Given enough authorized decrypted shares, i.e., an authorized set Q, the $\mathsf{PVGSSRecon}$ algorithm can reconstruct a secret S using $\mathsf{GrpGSSRecon}$ algorithm. Note that $\mathsf{GrpGSSRecon}$, rather than $\mathsf{GSSRecon}$, is used to recover the secret, since in our PVGSS scheme the secret S resides in $\mathbb{G}$, consistent with many prior PVSS constructions [6, 46].

5.3 Security Analysis

Theorem 1 (Correctness). *The PVGSS construction satisfies correctness defined by Definition 11.*

Proof. Fix an arbitrary security parameter λ, access structure $\mathbb{A}$, secret $s \in \mathbb{Z}_q$, and ordered valid public-key list $\{\mathsf{pk}_i\}_{i\in[|\mathbb{A}|]}$, where $\mathsf{pk}_i = g^{\mathsf{sk}_i}$ and $\mathsf{sk}_i \in \mathbb{Z}_q^*$. Let $(\{C_i\}, \mathsf{prf}_s) \leftarrow \mathsf{PVGSSShare}(s, \mathbb{A}, \{\mathsf{pk}_i\})$ be an honest sharing. By the definition of $\mathsf{PVGSSShare}$, the dealer first samples a valid GSS sharing $(s_1, \ldots, s_{|\mathbb{A}|}) \leftarrow \mathsf{GSSShare}(s, \mathbb{A}), C_i = \mathsf{pk}_i^{s_i}$, and publishes $C_i = \mathsf{pk}_i^{s_i}$ for every leaf i. It also samples $s' \in \mathbb{Z}_q$, computes $(s'_1, \ldots, s'_{|\mathbb{A}|}) \leftarrow \mathsf{GSSShare}(s', \mathbb{A}), C'_i = \mathsf{pk}_i^{s'_i}$, sets $c = H(\mathbb{A}, \{\mathsf{pk}_i\}, \{C_i\}, \{C'_i\}), \hat{s} = s' - cs, \hat{s}_i = s'_i - cs_i$. We first show that $\mathsf{PVGSSVerify}$ accepts. Since all values are generated honestly, the verifier recomputes the same Fiat-Shamir challenge c. Moreover, for every i, $C_i^c \mathsf{pk}_i^{\hat{s}_i} = (\mathsf{pk}_i^{s_i})^c \mathsf{pk}_i^{s'_i - cs_i} = \mathsf{pk}_i^{s'_i} = C'_i$. Thus all Schnorr-type verification equations hold. It remains to check the GSS-consistency equation. Because the underlying GSS sharing algorithm is linear, $(\hat{s}_1, \ldots, \hat{s}_{|\mathbb{A}|}) = (s'_1, \ldots, s'_{|\mathbb{A}|}) - c(s_1, \ldots, s_{|\mathbb{A}|})$ is a valid GSS sharing of $\hat{s} = s' - cs$ under the same access structure $\mathbb{A}$. By completeness of $\mathsf{GSSTest}$, i.e., $\mathsf{GSSTest}(\mathbb{A}, \{\hat{s}_i\}_{i\in[|\mathbb{A}|]}, \hat{s}) = 1$. Therefore,

$$\mathsf{PVGSSVerify}(\{C_i\}, \mathsf{prf}_s, \mathbb{A}, \{\mathsf{pk}_i\}) = 1.$$

Next, consider any labelled authorized set $Q = \{(i, S_i)\}_{i\in I_Q}$ of $\mathbb{A}$. For every $i \in I_Q$, honest execution of $\mathsf{PVGSSPreRecon}$ gives $S_i = C_i^{1/\mathsf{sk}_i} = (\mathsf{pk}_i^{s_i})^{1/\mathsf{sk}_i} = (g^{\mathsf{sk}_i s_i})^{1/\mathsf{sk}_i} = g^{s_i}$. The proof prf_{S_i} generated by $\mathsf{PVGSSPreRecon}$ is an honest DLEQ proof for the statement $\log_g \mathsf{pk}_i = \log_{S_i} C_i = \mathsf{sk}_i$, since $C_i = S_i^{\mathsf{sk}_i}$. By completeness of the DLEQ proof system,

$$\mathsf{PVGSSKeyVrf}(C_i, S_i, \mathsf{prf}_{S_i}, \mathsf{pk}_i) = 1 \quad \text{for all } i \in I_Q.$$

Finally, since Q is authorized and the values supplied to reconstruction are exactly the group shares $S_i = g^{s_i}$ corresponding to a valid GSS sharing of s, correctness of the underlying GSS-on-group reconstruction algorithm gives

$$S' \leftarrow \mathsf{PVGSSRecon}(\mathbb{A}, Q) = \mathsf{GrpGSSRecon}(\mathbb{A}, Q) = g^s.$$

This correctness of $\mathsf{GrpGSSRecon}$ follows from the corresponding group-GSS instantiations proved in Sections 4.2 and 4.3.

Combining the three conclusions, for every honestly generated execution and every authorized labelled set Q, all verification algorithms accept and the reconstructed value equals g^s. Hence the PVGSS construction satisfies correctness by Definition 11. □

Theorem 2 (Verifiability of Encrypted Shares). *Assume that the Schnorr-type Fiat-Shamir proof used in* $\mathsf{PVGSSShare}$ *is knowledge-sound in the random oracle model, with knowledge-soundness advantage* $\mathsf{Adv}^{\mathsf{ks}}_{\mathsf{FS}}$*, and that the instantiated GSS testing procedure is sound with error* ϵ_{test}*. Then, for every PPT adversary* $\mathcal{A}$*, there exists a PPT knowledge-soundness adversary* $\mathcal{B}_{\mathsf{FS}}$ *such that*

$$\mathsf{Adv}^{\mathsf{venc}}_{\mathsf{PVGSS}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{ks}}_{\mathsf{FS}}(\mathcal{B}_{\mathsf{FS}}) + 2\epsilon_{\mathsf{test}}.$$

In particular, for recursive Shamir GSS, Lemma 3 gives $\epsilon_{\mathsf{test}} = 0$ *for the exact interpolation test and* $\epsilon_{\mathsf{test}} \leq 1/q$ *for the dual-code test.*

Proof. By Definition 12, the event counted by $\mathsf{Adv}^{\mathsf{venc}}_{\mathsf{PVGSS}}(\mathcal{A})$ occurs only if $\mathcal{A}$ produces $(\{C_i\}, \mathsf{prf}_s)$ such that $\mathsf{PVGSSVerify}$ accepts, but there is no secret s and no valid GSS sharing $\{s_i\}$ of s satisfying $C_i = \mathsf{pk}_i^{s_i}$ for all i.

Consider an accepting proof $\mathsf{prf}_s = (\{C_i'\}, c, \hat{s}, \{\hat{s}_i\})$, for the statement $(\mathbb{A}, \{\mathsf{pk}_i\}, \{C_i\})$. By knowledge soundness of the Fiat-Shamir Schnorr proof in the ROM, except with probability $\mathsf{Adv}^{\mathsf{ks}}_{\mathsf{FS}}(\mathcal{B}_{\mathsf{FS}})$, the extractor for $\mathcal{B}_{\mathsf{FS}}$ obtains a second accepting transcript with the same Sigma-protocol commitment and a distinct challenge: $(\{C_i'\}, \tilde{c}, \tilde{s}, \{\tilde{s}_i\})$ where $c \neq \tilde{c}$. Both transcripts pass the algebraic verification equation, and hence, for every leaf i, $C_i' = C_i^c \mathsf{pk}_i^{\hat{s}_i} = C_i^{\tilde{c}} \mathsf{pk}_i^{\tilde{s}_i}$. Since all public keys are validated as non-identity elements of the prime-order group and $c - \tilde{c} \neq 0$, define $s_i = \frac{\hat{s}_i - \tilde{s}_i}{c - \tilde{c}}$ and $s = \frac{\hat{s} - \tilde{s}}{c - \tilde{c}}$. From $C_i^c \mathsf{pk}_i^{\hat{s}_i} = C_i^{\tilde{c}} \mathsf{pk}_i^{\tilde{s}_i}$, we obtain ciphertext consistency: $C_i = \mathsf{pk}_i^{s_i}$ for every i.

It remains to prove that the extracted exponents form a valid GSS sharing of s. We do so by applying the soundness of the GSS testing procedure to the two accepting response vectors. Let $\hat{\mathbf{s}} = (\hat{s}_1, \ldots, \hat{s}_{|\mathbb{A}|}), \tilde{\mathbf{s}} = (\tilde{s}_1, \ldots, \tilde{s}_{|\mathbb{A}|})$. Since both transcripts are accepting, they also satisfy $\mathsf{GSSTest}(\mathbb{A}, \hat{\mathbf{s}}, \hat{s}) = 1$ and $\mathsf{GSSTest}(\mathbb{A}, \tilde{\mathbf{s}}, \tilde{s}) = 1$. By soundness of GSS testing, except with probability at most $2\epsilon_{\mathsf{test}}$, there exist sharing coins $\hat{\boldsymbol{r}}$ and $\tilde{\boldsymbol{r}}$ such that

$$\hat{\mathbf{s}} = \mathsf{GSSShare}(\hat{s}, \mathbb{A}; \hat{\boldsymbol{r}}), \quad \tilde{\mathbf{s}} = \mathsf{GSSShare}(\tilde{s}, \mathbb{A}; \tilde{\boldsymbol{r}}).$$

Because the underlying GSS sharing map is linear over $\mathbb{Z}_q$, the coordinate-wise linear combination of two valid sharings is again a valid sharing of the corresponding linear combination of the secrets. Therefore

$$\mathbf{s} := \frac{\tilde{\mathbf{s}} - \hat{\mathbf{s}}}{c - \tilde{c}} = \mathsf{GSSShare}\left(\frac{\tilde{s} - \hat{s}}{c - \tilde{c}}, \mathbb{A}; \frac{\tilde{\boldsymbol{r}} - \hat{\boldsymbol{r}}}{c - \tilde{c}}\right).$$

Hence $\mathsf{Valid}_{\mathbb{A}}(s, \mathbf{s}) = 1$, and the i-th coordinate of $\mathbf{s}$ is exactly the extracted exponent s_i. Combining this with $C_i = \mathsf{pk}_i^{s_i}$ for all i, the accepted ciphertext vector encrypts valid GSS shares of the secret s, contradicting the winning condition in Definition 12.

Consequently, any successful adversary must either break knowledge soundness of the Fiat-Shamir proof or cause at least one invalid response vector to pass GSS testing. Thus

$$\mathsf{Adv}^{\mathsf{venc}}_{\mathsf{PVGSS}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{ks}}_{\mathsf{FS}}(\mathcal{B}_{\mathsf{FS}}) + 2\epsilon_{\mathsf{test}}.$$

For recursive Shamir GSS, Lemma 3 gives $\epsilon_{\mathsf{test}} = 0$ for the exact interpolation test and $\epsilon_{\mathsf{test}} \leq 1/q$ for the dual-code test, yielding the claimed bound. □

Theorem 3 (Verifiability of Share Decryption). *Assume that the Fiat-Shamir DLEQ proof used in* PVGSSPreRecon *is knowledge-sound in the random oracle model, with knowledge-soundness advantage* $\mathsf{Adv}^{\mathsf{ks}}_{\mathsf{DLEQ}}$*. Then, for every PPT adversary* $\mathcal{A}$*, there exists a PPT knowledge-soundness adversary* $\mathcal{B}_{\mathsf{DLEQ}}$ *such that* $\mathsf{Adv}^{\mathsf{vdec}}_{\mathsf{PVGSS}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{ks}}_{\mathsf{DLEQ}}(\mathcal{B}_{\mathsf{DLEQ}})$.

Proof. By Definition 13, the event counted by $\mathsf{Adv}^{\mathsf{vdec}}_{\mathsf{PVGSS}}(\mathcal{A})$ occurs only if $\mathcal{A}$ produces $(\mathsf{pk}, C, S, \mathsf{prf}_S)$ such that $\mathsf{PVGSSKeyVrf}(C, S, \mathsf{prf}_S, \mathsf{pk}) = 1$, but there is no $x \in \mathbb{Z}_q^*$ satisfying $\mathsf{pk} = g^x$ and $C = S^x$.

Construct $\mathcal{B}_{\mathsf{DLEQ}}$ by running $\mathcal{A}$ in the experiment and forwarding any accepting output $(\mathsf{pk}, C, S, \mathsf{prf}_S)$ as a DLEQ statement-proof pair $((g, \mathsf{pk}, S, C), \mathsf{prf}_S)$. Since $\mathsf{PVGSSKeyVrf}$ validates pk, C, S and then invokes $\mathsf{DLEQVrf}(g, \mathsf{pk}, S, C, \mathsf{prf}_S)$, every accepting adversarial output is an accepting Fiat-Shamir DLEQ proof for the relation $\mathcal{R}_{\mathsf{DLEQ}} = \{((g, X, Y, Z), x) : X = g^x \wedge Z = Y^x\}$. By knowledge soundness of the DLEQ proof, except with probability $\mathsf{Adv}^{\mathsf{ks}}_{\mathsf{DLEQ}}(\mathcal{B}_{\mathsf{DLEQ}})$, the extractor obtains a witness $x \in \mathbb{Z}_q$ such that $\mathsf{pk} = g^x$ and $C = S^x$. The public key pk is one of the registered keys output by $\mathsf{PVGSSSetup}$, and registered public keys are validated as non-identity elements of the prime-order group. Hence $\mathsf{pk} = g^x \neq 1$, which implies $x \neq 0 \pmod q$, and therefore $x \in \mathbb{Z}_q^*$. Thus, except with the stated DLEQ knowledge-soundness advantage, the witness required by Definition 13 exists. Consequently, $\mathsf{Adv}^{\mathsf{vdec}}_{\mathsf{PVGSS}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{ks}}_{\mathsf{DLEQ}}(\mathcal{B}_{\mathsf{DLEQ}})$. □

Theorem 4 (IND1-secrecy). *The PVGSS constructions satisfy IND1-secrecy, defined by the selective unauthorized-set experiment in Definition 14, assuming (i) zero-knowledge of the Fiat-Shamir proofs in the random oracle model, (ii) DDH in* $\mathbb{G}$*, and (iii) privacy of the underlying GSS-on-group scheme against unauthorized labelled sets.*

Proof. Let $\mathcal{A}$ be a PPT adversary in $\mathsf{Exp}^{\mathsf{ind1}}_{\mathsf{PVGSS},\mathcal{A}}(\lambda)$. Let $\mathbb{A}^*$ and $U \subseteq [1, \ldots, m]$, where $m = |\mathbb{A}^*|$, be the selective access structure and unauthorized query set output by $\mathcal{A}$ before setup. By Definition 14, the labelled set $\{P_i : i \in U\}$ is unauthorized for $\mathbb{A}^*$. Let $x_0, x_1 \in \mathbb{Z}_q$ be the distinct challenge secrets output after seeing the ordered public keys. For each game G_j, let E_j denote the event that the experiment outputs 1, equivalently $b' = b$. We proceed by hybrids.

Game G_0**.** This is the real experiment of Definition 14. The challenger samples b, runs $(\{C_i^*\}_{i\in[1,\ldots,m]}, \mathsf{prf}_{x_b}) \leftarrow \mathsf{PVGSSShare}(x_b, \mathbb{A}^*, \{\mathsf{pk}_i\}_{i\in[1,\ldots,m]})$, and answers every adaptive query $i \in U$ with $(S_i^*, \mathsf{prf}_{S_i}^*) \leftarrow \mathsf{PVGSSPreRecon}(C_i^*, \mathsf{sk}_i)$. Thus $\mathsf{Adv}^{\mathsf{ind1}}_{\mathsf{PVGSS}}(\mathcal{A}) = |\Pr[E_0] - 1/2|$.

Game G_1: simulated sharing proof. This game is identical to G_0, except that the Fiat-Shamir proof prf_{x_b} attached to the sharing ciphertexts is generated by the zero-knowledge simulator for the same public statement. All decryption-query answers remain honestly generated. By zero-knowledge of the Fiat-Shamir proof in the random oracle model (Section 3.4), there exists a PPT distinguisher $\mathcal{B}_{\mathsf{zk}}$ such that $|\Pr[E_0] - \Pr[E_1]| \leq \mathsf{Adv}^{\mathsf{zk}}_{\mathsf{FS}}(\mathcal{B}_{\mathsf{zk}}) + \mathsf{negl}(\lambda)$.

Game G_2: randomized unopened ciphertexts. Let $T = [1, \ldots, m] \setminus U$ and write $T = \{i_1, \ldots, i_\ell\}$. Starting from G_1, replace the ciphertexts $\{C^*_i\}_{i \in T}$, whose decryption shares can never be queried, by independent uniform elements of $\mathbb{G}$. After the replacement, the sharing proof prf_{x_b} is simulated for the resulting public statement; this is possible because the proof is simulated and no witness is needed. The public keys $\{\mathsf{pk}_i\}_{i \in [1,\ldots,m]}$, the access structure $\mathbb{A}^*$, and the query-answer distribution for indices $i \in U$ are otherwise unchanged. The transition from G_1 to G_2 is justified by a standard coordinate-wise DDH hybrid for hidden group shares.[7] Since U is fixed before setup, the reduction knows the unopened set T when it generates public keys. In the j-th hybrid step, it embeds a DDH challenge (g, g^a, g^b, Z) in coordinate i_j: set $\mathsf{pk}_{i_j} = g^a$, regard g^b as the hidden unopened group share $S^*_{i_j}$, and set $C^*_{i_j} = Z$. The value $S^*_{i_j}$ is never returned to $\mathcal{A}$, and the sharing proof for the full ciphertext vector is simulated, so no scalar witness for this unopened coordinate is needed. If $Z = g^{ab}$, then $C^*_{i_j} = (S^*_{i_j})^a = \mathsf{pk}_{i_j}^{s^*_{i_j}}$, matching the real encrypted-share distribution in that coordinate; if Z is uniform, then $C^*_{i_j}$ is randomized as in the next hybrid. The opened coordinates $i \in U$ are generated and answered honestly, and the correlations required by the access structure are handled in the following GSS privacy step (in G_3). Thus, by summing over the ℓ coordinates, there exist PPT DDH distinguishers $\{\mathcal{B}^{(j)}_{\mathsf{ddh}}\}_{j \in [\ell]}$ such that $|\Pr[E_1] - \Pr[E_2]| \leq \sum_{j=1}^{\ell} \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{B}^{(j)}_{\mathsf{ddh}}) + \mathsf{negl}(\lambda)$.

Game G_3: simulated unauthorized opened view. In G_2, all ciphertexts for indices outside U are uniform and independent of b. For the query set U, the challenger may equivalently precompute the potential answers $\{(S^*_i, \mathsf{prf}^*_{S_i})\}_{i \in U}$ and reveal them only when $\mathcal{A}$ queries the corresponding index. Since U is unauthorized for $\mathbb{A}^*$, privacy of the underlying GSS-on-group scheme, established for our Shamir-based and LSSS-based instantiations in Section 4.2 and Lemma 2, implies that the opened group-share view $\{S^*_i\}_{i \in U}$ can be replaced by a simulated unauthorized view independent of whether the shared group secret is g^{x_0} or g^{x_1}. The challenger then recomputes $C^*_i = (S^*_i)^{\mathsf{sk}_i}$ and honest DLEQ proofs for $i \in U$, and simulates the sharing proof for the resulting public statement. Revealing any adaptively queried subset of these precomputed values is an effi-

[7] Unlike threshold PVSS schemes (such as SCRAPE [6], Albatross [46] and DH-PVSS [47]), PVGSS cannot in general fix a single β and interpolate it together with $t-1$ corrupted shares to obtain one global DDH reduction: the leaf shares are constrained by a generalized access structure rather than one threshold polynomial. We therefore use DDH only to randomize unopened encryptions, and rely on GSS-on-group privacy to handle the correlated opened view.

cient post-processing of the unauthorized group-sharing view. Hence there exists a PPT adversary $\mathcal{B}_{\mathsf{gss}}$ such that $|\Pr[E_2] - \Pr[E_3]| \leq \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GrpGSS}}(\mathcal{B}_{\mathsf{gss}}) + \mathsf{negl}(\lambda)$.

Therefore, the adversary's view consists only of simulated sharing proofs, uniform unopened ciphertexts, and a simulated unauthorized group-sharing view together with the corresponding ciphertexts and honest DLEQ proofs. This joint distribution is independent of b, and therefore $\Pr[E_3] = 1/2$. By the triangle inequality,

$$\mathsf{Adv}^{\mathsf{ind1}}_{\mathsf{PVGSS}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{zk}}_{\mathsf{FS}}(\mathcal{B}_{\mathsf{zk}}) + \sum_{j=1}^{\ell} \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{B}^{(j)}_{\mathsf{ddh}}) + \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{GrpGSS}}(\mathcal{B}_{\mathsf{gss}}) + \mathsf{negl}(\lambda),$$

which is negligible under the stated assumptions. □

6 Decentralized Exchange (DEX) Based on PVGSS

6.1 High-Level Overview

We use PVGSS as a public, non-interactive secret-escrow layer for fair token exchange without a trusted third party. The presentation focuses on ERC-20[8] token swaps, although the same mechanism applies to any asset whose transfer can be mediated by a ledger contract. The DEX has two types of participants: exchangers, who lock assets and exchange opening secrets, and watchers, who serve as passive arbiters only when a dispute occurs. Any party may act as an exchanger or a watcher, but every protocol message that affects fairness is publicly verifiable on the ledger.

Consider two exchangers, Alice and Bob, and n watchers $W_1, \ldots, W_n$. For each exchanger's secret, the PVGSS access structure is

$$\mathbb{A} = (2\ of\ (Alice, Bob, (t\ of\ (W_1, W_2, \cdots, W_n)))) \tag{1}$$

Thus $\{Alice, Bob\}$ is authorized, so in the optimistic case the two exchangers can reconstruct each other's secrets by exchanging only their own decrypted PVGSS shares. If one exchanger aborts after learning the counterparty's opening information, the honest exchanger can invoke the dispute path and combine its own share with t watcher shares. The watchers do not learn or release anything during the optimistic execution; they become relevant only after a publicly recorded complaint.

The role of PVGSS is twofold. First, PVGSSVerify makes the commitment phase publicly accountable: a malformed encrypted sharing is rejected before the swap can proceed. Second, PVGSSKeyVrf makes every later opening share publicly checkable, so an exchanger or watcher cannot bias the outcome by submitting an invalid decryption. The incentive mechanism then turns watcher participation into an accountable ledger action. Consequently, fairness does not require assuming that more than t watchers are honest. In the DEX instantiation, t may be set to 1: a single responsive watcher is enough to complete arbitration, while non-responsive or faulty watchers are penalized by the contract.

[8] `https://eips.ethereum.org/EIPS/eip-20`

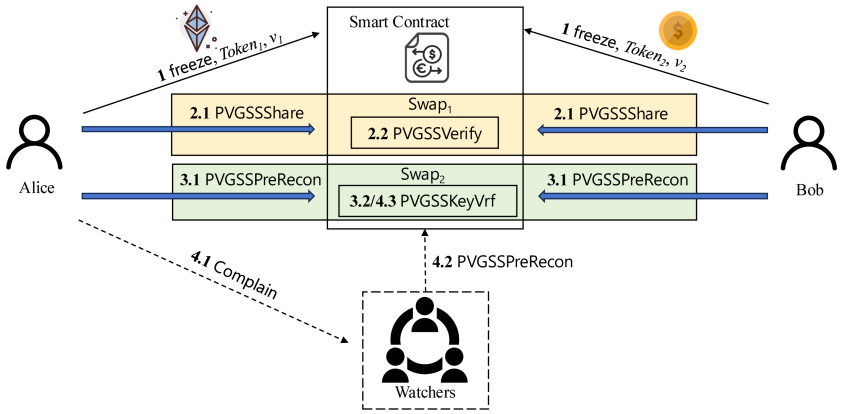

Fig. 8. Overview of the DEX between two exchangers (i.e., Alice and Bob) with passive watchers

Figure 8 summarizes the protocol. In the first round swap_1, each exchanger locks its asset and publishes a PVGSS commitment to its opening secret under the access structure in Equation (1); the contract accepts the commitment only if PVGSSVerify succeeds. In the second round swap_2, each exchanger publishes its decrypted share and proof, which are checked by PVGSSKeyVrf. If both exchangers open before the deadline, they reconstruct the two secrets by PVGSSRecon and the swap completes without watcher involvement. If exactly one exchanger fails to open, the counterparty files a complaint, eligible watchers publish verifiable decrypted shares, and the contract resolves the exchange from the publicly verifiable reconstruction evidence.

6.2 Concrete Design

Ideal functionality. Figure 9 specifies the ideal exchange task in the $\mathcal{F}_{\mathsf{Ledger}}$-hybrid model, where $\mathcal{F}_{\mathsf{Ledger}}$ denotes an ideal ledger functionality that maintains append-only public state, enforces contract time, and executes contract state transitions. The functionality deliberately exposes only the information that is unavoidable on a public ledger: registrations, deposits, commitments, openings, complaints, watcher openings, and the final outcome. It abstracts away the concrete PVGSS transcript and retains the semantic guarantees that the protocol must realize: escrowed assets are transferred only in a successful execution, a party cannot force success while withholding the opening information required from its side, and failure leads to refund rather than unilateral gain.

The deadlines are upper bounds on waiting, not mandatory delays. The first deadline Δt_1 separates the optimistic two-party opening phase from the dispute phase; the second deadline Δt_2 bounds arbitration and prevents indefinite asset locking. A session can terminate as soon as success evidence is available, namely when both exchangers have opened or when one exchanger has aborted and

Ideal Functionality $\mathcal{F}_{\mathsf{DEX}}$

The ideal functionality $\mathcal{F}_{\mathsf{DEX}}$ operates in the $\mathcal{F}_{\mathsf{Ledger}}$-hybrid model. It is parameterized by time bounds $\Delta t_1 < \Delta t_2$, and interacts with $\mathcal{F}_{\mathsf{Ledger}}$ to enforce time, store state, and manage asset escrow. Denote $\mathcal{A}$ as the adversary controlling corrupted parties. For each session id, it maintains registered keys, deposits, commitment flags, exchanger openings, complaints, and watcher opening sets. Upon activation, $\mathcal{F}_{\mathsf{DEX}}$ behaves as follows:

- **Register:** Upon receiving $(\mathsf{register}, \mathsf{id}, \mathsf{pid}, \mathsf{pk}_y)$ from a party y, stores pk_y in its local state via $\mathcal{F}_{\mathsf{Ledger}}$ and sends the public event to $\mathcal{A}$.
- **Deposit:** Upon receiving $(\mathsf{freeze}, \mathsf{id}, x, \mathsf{Token}_x, v_x)$ from exchanger x, instructs $\mathcal{F}_{\mathsf{Ledger}}$ to lock the tokens in escrow. Withdrawals are permitted only if the exchange either does not commence or terminates in $\mathsf{failure}$.
- **Commit (swap_1):** Upon receiving $(\mathsf{commit}, \mathsf{id}, x)$ from exchanger x, records that x has committed and sends the public event to $\mathcal{A}$. The timer starts when both exchangers have committed.
- **Reveal (swap_2):** Upon receiving $(\mathsf{reveal}, \mathsf{id}, x)$ from exchanger x before Δt_1, records that x has opened its commitment and sends the public event to $\mathcal{A}$.
- **Complain:** After Δt_1, if exactly one exchanger x has not opened, allow the counterparty $\bar{x}$ to submit $(\mathsf{complain}, \mathsf{id}, x)$ and record a complaint against x.
- **Watcher opening:** After a valid complaint against x, upon receiving $(\mathsf{arbitrate}, \mathsf{id}, x, i)$ from an assigned watcher W_i before Δt_2, records i in the watcher opening set for x and sends the public event to $\mathcal{A}$.
- **Determine:** Upon an explicit determine request:
  - If both exchangers have opened, instructs $\mathcal{F}_{\mathsf{Ledger}}$ to transfer tokens and output $(\mathsf{success})$.
  - If exactly one exchanger has not opened but at least t assigned watchers have opened for that exchanger, does the same as above.
  - Otherwise, after Δt_2, instructs $\mathcal{F}_{\mathsf{Ledger}}$ to refund and output $(\mathsf{failure})$.

Fig. 9. The ideal functionality $\mathcal{F}_{\mathsf{DEX}}$

enough watchers have opened for the missing commitment. Only unresolved sessions wait until Δt_2, where they are resolved as failure. Thus the ideal functionality captures fairness as a public, ledger-enforced state transition. For notation, $x \in \{A, B\}$ denotes an exchanger, $\bar{x}$ denotes its counterparty, and $K_y^{(x)}$ denotes party y's decrypted PVGSS share for exchanger x's commitment.

Real-world implementation. Figure 10 gives the real protocol Π_{DEX}. Each execution is bound to a session identifier id, the ledger contract address, the chain identifier, the ordered public-key list, and the role labels of the parties. These values are included in the contract state and in the verified PVGSS statements, so a transcript from one session cannot be replayed in another. The contract

Real Protocol Π_{DEX}

Any two exchangers (Alice and Bob) leverage the same access structure by Equation (1). The real-world protocol Π_{DEX} is executed on smart contracts and implements $\mathcal{F}_{\mathsf{DEX}}$ via PVGSS. The canonical leaf order is $(A, B, W_1, \ldots, W_n)$, and every PVGSS public-key list and ciphertext vector follows this order. For exchanger $x \in \{A, B\}$, write $C^{(x)} = (C_A^{(x)}, C_B^{(x)}, C_1^{(x)}, \ldots, C_n^{(x)})$ and let $K_y^{(x)}$ denote party y's decrypted share of $C_y^{(x)}$.

- **Register:** Any entity registers its public key pk_y via $(\mathsf{register}, \mathsf{id}, \mathsf{pid}, \mathsf{pk}_y)$.
- **Deposit:** Alice and Bob freeze tokens via $(\mathsf{freeze}, \mathsf{id}, A, \mathsf{Token}_A, v_A)$ and $(\mathsf{freeze}, \mathsf{id}, B, \mathsf{Token}_B, v_B)$, respectively. If the session later terminates in `failure`, either party may reclaim its tokens via $(\mathsf{withdraw}, \mathsf{id})$.
- **Commit (swap_1):** Each exchanger $x \in \{A, B\}$ samples a secret s_x and invokes $(\mathsf{swap}_1, \mathsf{id}, x, C^{(x)}, \mathsf{prf}_s^{(x)})$, where $(C^{(x)}, \mathsf{prf}_s^{(x)}) \leftarrow \mathsf{PVGSSShare}(s_x, \mathbb{A}, \mathsf{pks})$ and $\mathsf{pks} = (\mathsf{pk}_A, \mathsf{pk}_B, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$. The session identifier id, chain identifier, contract address, ordered public keys, and role label of x are recorded by the contract to prevent cross-session interference. The contract runs $\mathsf{PVGSSVerify}(C^{(x)}, \mathsf{prf}_s^{(x)}, \mathbb{A}, \mathsf{pks})$; if it fails, the transaction is aborted.
- **Reveal (swap_2):** Exchanger x invokes $(\mathsf{swap}_2, \mathsf{id}, x, x, K_x^{(x)}, \mathsf{prf}_{K_x}^{(x)})$, where $(K_x^{(x)}, \mathsf{prf}_{K_x}^{(x)}) \leftarrow \mathsf{PVGSSPreRecon}(C_x^{(x)}, \mathsf{sk}_x)$. The contract checks $\mathsf{PVGSSKeyVrf}(C_x^{(x)}, K_x^{(x)}, \mathsf{prf}_{K_x}^{(x)}, \mathsf{pk}_x)$; if valid, $K_x^{(x)}$ is stored. The counterparty $\bar{x}$ computes $K_{\bar{x}}^{(x)}$ locally and recovers $g^{s_x} \leftarrow \mathsf{PVGSSRecon}(\mathbb{A}, Q_x)$, where $Q_x = \{(x, K_x^{(x)}), (\bar{x}, K_{\bar{x}}^{(x)})\}$.
- **Complain:** If at time $t > \Delta t_1$ exchanger x has not revealed, the counterparty $\bar{x}$ may call $(\mathsf{complain}, \mathsf{id}, x)$, triggering a notification to all watchers to execute $(K_i^{(x)}, \mathsf{prf}_{K_i}^{(x)}) \leftarrow \mathsf{PVGSSPreRecon}(C_i^{(x)}, \mathsf{sk}_i)$ and publish $(\mathsf{swap}_2, \mathsf{id}, x, i, K_i^{(x)}, \mathsf{prf}_{K_i}^{(x)})$. The contract verifies each watcher response with $\mathsf{PVGSSKeyVrf}(C_i^{(x)}, K_i^{(x)}, \mathsf{prf}_{K_i}^{(x)}, \mathsf{pk}_i)$. The complainant combines its locally computed share $K_{\bar{x}}^{(x)}$ with any t accepted watcher shares to reconstruct g^{s_x}.
- **Determine:** Any user may call $(\mathsf{determine}, \mathsf{id})$. The contract checks:
  - Both exchangers revealed their own openings $\Rightarrow$ `success`.
  - Exactly one exchanger did not reveal, and at least t valid watcher openings for the missing exchanger's commitment are stored $\Rightarrow$ `success`.
  - Otherwise, after $\Delta t_2 \Rightarrow$ set `failure` and refund.

Fig. 10. The real-world protocol Π_{DEX} for implementing $\mathcal{F}_{\mathsf{DEX}}$

maintains a state variable for the lifecycle of the exchange, records the registered keys and deposits, and accepts only group elements and proofs that pass the corresponding public checks.

The implementation realizes the ideal transitions by replacing every ideal opening event with a publicly verifiable PVGSS operation. In swap_1, an ex-

changer publishes encrypted shares of its secret and the contract runs $\mathsf{PVGSSVerify}$, so a malformed commitment cannot enter the ledger state. In swap_2, an exchanger or watcher publishes a decrypted share together with a proof checked by $\mathsf{PVGSSKeyVrf}$, so invalid openings are ignored. If both exchangers open, the two-party authorized set reconstructs each secret; if one exchanger aborts, the counterparty combines its own share with t accepted watcher shares. Hence the real protocol implements the same success and failure conditions as $\mathcal{F}_{\mathsf{DEX}}$, with PVGSS providing the cryptographic link between ledger events and the hidden secrets. We assume that exchangers have access to external price oracles and therefore correctly offer v_A units of Token_A in exchange for v_B units of Token_B.

Watchers are selected from a registered and staked pool for each id, and the incentive layer makes watcher responses economically accountable. The concrete reward and penalty policy is deferred to Appendix B.2, where we enumerate the successful and failed termination cases and the corresponding transfers.

6.3 Security Properties

We prove in Appendix B.3 that the real-world protocol Π_{DEX} UC-securely realizes the ideal functionality $\mathcal{F}_{\mathsf{DEX}}$, and the DEX Π_{DEX} guarantees the security properties by Lemmas 4-10 and resists potential attacks by Claims 1-4.

7 Conclusion

We introduced publicly verifiable generalized secret sharing (PVGSS), which combines expressive monotone access structures with non-interactive public accountability. The construction encrypts only the leaf shares of an underlying GSS instance and uses Schnorr-type Fiat-Shamir proofs together with GSS testing to certify that all encrypted shares are globally consistent with a single dealer secret. We instantiated the framework from recursive Shamir secret sharing and from LSSS, and proved correctness, verifiability of encrypted shares, verifiability of share decryption, and IND1-secrecy under explicit assumptions on DDH, the proof systems, and the privacy of the underlying GSS-on-group sharing. We also demonstrated how PVGSS can serve as a cryptographic building block for accountable blockchain protocols through a decentralized exchange. In this application, PVGSS enables verifiable commitment, verifiable secret release, and public arbitration under a flexible access structure. Several directions remain open. First, PVGSS can be applied to other systems that require both expressive authorization and public auditability, including randomness beacons, bridges, channels, or oracle networks. Second, it is useful to explore additional PVGSS constructions with smaller proofs, lower verification cost, or post-quantum assumptions. Finally, dynamic variants such as proactive PVGS, q-PVGSS and aggregatable PVGSS, analogous to proactive SS [3], q-PVSS [7] and aggregatable PVSS [52], may further broaden the applicability of publicly verifiable sharing under general access structures.

References

1. Shamir, A. How to share a secret. *Comm. of the ACM*, 1979, 22(11), pp. 612-613.
2. Feldman, P. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, 1987, pp. 427-438.
3. Bagherpour, B., 2022. An efficient publicly verifiable and proactive secret sharing scheme. *Adv. in Math. of Comm.*, 16(4), pp. 671-691.
4. Alon, N., Galil, Z. and Yung, M. Efficient dynamic-resharing "verifiable secret sharing" against mobile adversary. In *EPA*, 1995, pp. 523-537.
5. Stadler, M. Publicly verifiable secret sharing. In *Eurocrypt*, 1996, pp. 190-199.
6. Cascudo, I., and David, B. SCRAPE: Scalable randomness attested by public entities. In *ACNS*, 2017, pp. 537-556.
7. Cascudo, I., and David, B. Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. In *Eurocrypt*, 2024, pp. 216-248.
8. Ito, M., Saito, A. and Nishizeki, T. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan*, 1989, 72(9), pp. 56-64.
9. Benaloh, J., and Leichter, J. Generalized secret sharing and monotone functions. In *Crypto*, 1990.
10. Beimel, A. Secret-Sharing Schemes for General Access Structures: An Introduction. *Cryptology ePrint Archive*. 2025.
11. Karchmer, M. and Wigderson, A. On span programs. In *Annual Structure in Complexity Theory Conference*, 1993, pp. 102-111.
12. Beimel, A. Secret-sharing schemes: A survey. In *ICC*, 2011, pp. 11-46.
13. Bethencourt, J., Sahai, A., and Waters, B. Ciphertext-Policy Attribute-Based Encryption. In *S&P*, 2007, pp. 321-334.
14. Lewko, A., and Waters, B. Decentralizing attribute-based encryption. In *Eurocrypt*, 2011, pp. 568-588.
15. Schnorr, C.P. Efficient identification and signatures for smart cards. In *Crypto*, 1990, pp. 239-252.
16. Fiat, A., and Shamir, A. How to prove yourself: Practical solutions to identification and signature problems. In *Eurocrypt*, 1986, pp. 186-194.
17. Chaum, D., & Pedersen, T. P. Wallet databases with observers. In *CRYPTO*, 1992, pp. 89-105.
18. Choi, K., Manoj, A. and Bonneau, J. Sok: Distributed randomness beacons. In *S&P*, 2023, pp. 75-92.
19. Syta, E., Jovanovic, P., Kogias, E.K., Gailly, N., Gasser, L., Khoffi, I., Fischer, M.J. and Ford, B. Scalable bias-resistant distributed randomness.In *S&P*,2017, pp. 444-460.
20. Schindler, P., Judmayer, A., Stifter, N. and Weippl, E., 2020, May. Hydrand: Efficient continuous distributed randomness. In 2020 IEEE Symposium on Security and Privacy (SP) (pp. 73-89). IEEE.
21. Benhamouda, F., Gentry, C., Gorbunov, S., Halevi, S., Krawczyk, H., Lin, C., Rabin, T. and Reyzin, L. Can a public blockchain keep a secret?. In *TCC*, 2020, pp. 260-290.
22. Goyal, V., Kothapalli, A., Masserova, E., Parno, B. and Song, Y. Storing and retrieving secrets on a blockchain. In *PKC*, 2022, pp. 252-282.
23. Cerulli, A., Connolly, A., Neven, G., Preiss, F.S. and Shoup, V. vetkeys: How a blockchain can keep many secrets. *Cryptology ePrint Archive*. 2023.
24. Tyagi, N., Arun, A., Freitag, C., Wahby, R., Bonneau, J. and Mazières, D. Riggs: Decentralized sealed-bid auctions. In *CCS*, 2023, pp.1227–1241.

25. Xie, T., Zhang, J., Cheng, Z., Zhang, F., Zhang, Y., Jia, Y., Boneh, D. and Song, D. zkbridge: Trustless cross-chain bridges made practical. In *CCS*, 2022, pp. 3003-3017.
26. Li, Y., Liu, H. and Tan, Y. POLYBRIDGE: A crosschain bridge for heterogeneous blockchains. In *ICBC*, 2022, pp. 1-2.
27. Dziembowski, S., Faust, S. and Hostáková, K. General state channel networks. In *CCS*, 2018, pp. 949-966.
28. Park, S., Bastani, O. and Kim, T. ACon2: Adaptive Conformal Consensus for Provable Blockchain Oracles. In *USENIX Security*, 2023, pp. 3313-3330.
29. Simmons, G.J., 1988, August. How to (really) share a secret. In *Eurocrypt*, 1988, pp. 390-448.
30. Beimel, A., Tassa, T., and Weinreb, E. Characterizing ideal weighted threshold secret sharing. In *TCC*, 2005, pp. 600-619.
31. Garg, S., Jain, A., Mukherjee, P., Sinha, R., Wang, M. and Zhang, Y. Cryptography with weights: MPC, encryption and signatures. In *CRYPTO*, 2023, pp. 295-327.
32. Chen, Q., Tang, C. and Lin, Z. Efficient explicit constructions of multipartite secret sharing schemes. *IEEE TIT*, 2021, 68(1), pp. 601-631.
33. Tassa, T., and Dyn, N. Multipartite secret sharing by bivariate interpolation. *JoC*, 2009, 22(2), pp. 227-258.
34. Tassa, T. Hierarchical threshold secret sharing. *JoC*, 2007, 20(2), pp. 237-264.
35. Chattopadhyay, A.K., Saha, S., Nag, A. and Nandi, S. Secret sharing: A comprehensive survey, taxonomy and applications. *Computer Science Review*, 2024, 51, 100608.
36. Drăgan, C.C. and Ţiplea, F.L., . Distributive weighted threshold secret sharing schemes. *Information sciences*, 2016, 339, pp. 85-97.
37. Mashhadi, S. Secure publicly verifiable and proactive secret sharing schemes with general access structure. *Information Sciences*, 2017, 378, pp. 99-108.
38. Zhang, L., Wu, X., Yue, Q., Kan, H., Zhang, J. Attribute-based publicly verifiable secret sharing. *Cybersecurity*, 2026, 9(1), article 123.
39. Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S. and Stern, G. Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In *Crypto*, 2023, pp. 39-70.
40. Galil, Z. and Yung, M. Partitioned encryption and achieving simultaneity by partitioning. *Infor. Proc. Letters*, 1987, 26(2), pp.81-88.
41. Kate, A., Zaverucha, G.M. and Goldberg, I., Constant-size commitments to polynomials and their applications. In *Asiacrypt*, 2010, pp. 177-194.
42. Zhang, Z., Li, W., Guo, Y., Shi, K., Chow, S.S., Liu, X. and Dong, J. Fast RS-IOP Multivariate Polynomial Commitments and Verifiable Secret Sharing. In *USENIX Security*, 2024, pp. 3187-3204.
43. Schoenmakers, B. A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic. In *Crypto*, 1999, pp. 148-164.
44. Fujisaki, E., and Okamoto, T. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In *Eurocrypt*, 1998, pp. 32-46.
45. Ruiz, A., and Villar, J. L. Publicly verifiable secret sharing from Paillier's cryptosystem. In *SAC*, 2005.
46. Cascudo, I., and David, B. ALBATROSS: publicly attestable batched randomness based on secret sharing. In *Asiacrypt*, 2020, pp. 311-341.
47. Cascudo, I., David, B., Garms, L., and Konring, A. YOLO YOSO: fast and simple encryption and secret sharing in the YOSO model. In *Asiacrypt*, 2022, pp. 651-680.

48. Kate, A., Mangipudi, E.V., Mukherjee, P., Saleem, H. and Thyagarajan, S.A.K. Non-interactive VSS using class groups and application to DKG. In *CCS*, 2024, pp. 4286-4300.
49. Das, S., Yurek, T., Xiang, Z., Miller, A., Kokoris-Kogias, L. and Ren, L. Practical asynchronous distributed key generation. In *S&P*, 2022, pp. 2518-2534.
50. Liu, Z., Cao, Z. and Wong, D.S. Efficient generation of linear secret sharing scheme matrices from threshold access trees. *Cryptology ePrint Archive.* 2010
51. Damgård, I. On Σ-protocols. *Lecture Notes, University of Aarhus, Department for Computer Science*, 2002.
52. Bacho, R. and Loss, J. Adaptively secure (aggregatable) pvss and application to distributed randomness beacons. In *CCS*, 2023, pp. 1791-1804.
53. Pagnia, H. and Gärtner, F.C. On the impossibility of fair exchange without a trusted third party. Technical Report TUD-BS-1999-02, Darmstadt University of Technology, Department of Computer Science, Darmstadt, Germany. 1999.
54. Erwig, A., Faust, S., Hostáková, K., Maitra, M. and Riahi, S. Two-party adaptor signatures from identification schemes. In *PKC*, 2021, pp. 451-480.
55. Aumayr, L., Ersoy, O., Erwig, A., Faust, S., Hostáková, K., Maffei, M., Moreno-Sanchez, P. and Riahi, S. Generalized channels from limited blockchain scripts and adaptor signatures. In *Asiacrypt*, 2021, pp. 635-664.
56. Thyagarajan, S.A., Malavolta, G. and Moreno-Sanchez, P. Universal atomic swaps: Secure exchange of coins across all blockchains. In *S&P*, 2022, pp. 1299-1316.
57. Zhang, L., Kan, H., Qiu, F. and Hao, F. A Publicly Verifiable Optimistic Fair Exchange Protocol Using Decentralized CPABE. *CJ*, 2024, 67(3), pp. 1017-1029.
58. Xu, J., Paruch, K., Cousaert, S. and Feng, Y. Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. *ACM Comp. Surv.*, 2023, 55(11), pp. 1-50.
59. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L. and Juels, A. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *S&P*, 2020, pp. 910-927.
60. Baum, C., David, B. and Frederiksen, T.K. P2DEX: privacy-preserving decentralized cryptocurrency exchange. In *ACNS*, 2021, pp. 163-194.
61. Heimbach, L., Schertenleib, E. and Wattenhofer, R. Risks and returns of uniswap v3 liquidity providers. In *AFT*, 2022, pp. 89-101.

A Use of AI tools.

During the preparation of this manuscript, the authors used ChatGPT to assist with language polishing, clarity improvement, and organization of technical exposition. The authors reviewed, edited, and verified all AI-assisted text and take full responsibility for the content, including the correctness of definitions, constructions, proofs, experiments, and references.

B Background, Incentives, and Security of the DEX

B.1 Motivation

The DEX application is a concrete instance of the fair-exchange problem. In a purely off-chain setting, fair exchange cannot be achieved without a trusted third

party [53]; on a public blockchain, the ledger can instead provide escrow, deadlines, and publicly auditable settlement. Centralized exchanges (CEXs), such as Binance[9] and Coinbase[10], offer liquidity and usability but reintroduce a trusted operator, exposing users to custody risk, censorship, and single points of failure. The goal of our DEX design is to retain public settlement and user custody while adding a verifiable arbitration path for non-cooperative executions.

Among non-custodial exchange designs, atomic-swap protocols and AMM-based DEXs provide complementary forms of public settlement but leave different limitations for fair pairwise exchange. Hash time-locked contracts (HTLCs)[11] and adaptor signatures [54, 55] enable atomic swaps [56, 57], but their fairness relies on timeout discipline and cooperative release of preimages or signatures. When a party refuses to cooperate, the protocol usually resolves by waiting for timeout rather than by producing publicly verifiable arbitration evidence. Automated market maker (AMM) DEXs, such as Uniswap[12], remove the need for pairwise negotiation by using liquidity pools [58], but this changes the exchange model: LPs face divergence loss, traders face slippage, and the transaction ordering layer creates front-running and MEV risks [59–61].

Our PVGSS-based DEX targets pairwise fair exchange with accountable arbitration. It keeps the swap off the AMM curve and avoids a custodial fairness provider. Watchers are invoked only in the pessimistic case, but they are not trust-free volunteers: they are selected from a registered pool, lock stake before serving swaps, and are rewarded or penalized according to the publicly verifiable validity of their responses. Table 2 summarizes this positioning at the protocol level. The comparison is not intended to replace a full market-microstructure analysis; rather, it isolates the properties relevant to this paper: settlement mechanism, auxiliary roles, absence of a trusted operator, verifiable arbitration, and avoidance of AMM-specific slippage.

Table 2. Protocol-level comparison of exchange mechanisms

Mechanism	Settlement mechanism	Auxiliary roles	No trusted operator	Verifiable arbitration	No AMM slippage
CEX	Custodial operator	Operator	✗	✗	✓
HTLC-based	Timeout contracts	None	✓	✗	✓
AMM-based	AMM contract	LPs / arbitrageurs	✓	✗	✗
Ours	PVGSS + staked watchers	Staked watchers	✓	✓	✓

[9] `https://www.binance.com/en`
[10] `https://www.coinbase.com`
[11] `https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts`
[12] `https://app.uniswap.org`

B.2 DEX Incentive Mechanism

The security proof treats watcher behavior through economic accountability rather than through an honest-majority or honest-threshold assumption. Every outcome-affecting action is public and attributable: commitments are checked by PVGSSVerify, exchanger and watcher openings are checked by PVGSSKeyVrf, and finalization is performed by an explicit on-chain determine call. A swap instance can therefore be finalized as either success or failure once the corresponding public evidence is available. In particular, success may be determined before Δt_2, while failure is used only for unresolved sessions after the arbitration deadline.

We assume that exchangers and watchers lock stake before participating. Watcher rewards are funded by the exchangers, for example through service fees deposited when a swap is created, while dispute-case compensation may be paid from slashable stake associated with attributable deviations. To keep the presentation focused on the cryptographic and UC aspects, we do not specify a concrete fee schedule or reward-splitting rule. A watcher's concurrent service capacity is proportional to its stake, which prevents a small stake from underwriting many unresolved obligations. The concrete stake amounts are application parameters, but the qualitative requirements are simple: rewards must cover the cost of valid arbitration, and penalties must dominate the benefit of withholding, rushing, or submitting invalid responses. Figure 11 enumerates the termination cases and the corresponding incentive policy.

A complaint is valid only if both exchangers have committed, Δt_1 has passed, the complainant has already revealed, the complained exchanger has not revealed, the complainant is the counterparty of the complained exchanger, and the session has not been finalized. Complaints that are premature, submitted by a non-counterparty, target an already revealed exchanger, are issued by a party that has not revealed, duplicate an existing valid complaint, or arrive after finalization are rejected and penalized as false complaints. A false complaint does not change the swap state; the session continues according to the ordinary success or failure rules. Similarly, watcher responses are accepted only after a valid complaint and only for the missing commitment named in that complaint. Premature, unsolicited, duplicate, or invalid watcher responses are ignored and may be penalized, while valid responses are rewarded.

The success outcomes have three forms. First, in the optimistic case, both exchangers commit and reveal before Δt_1; watchers are not invoked for arbitration, but may receive an availability tip for reserving capacity. Second, a false complaint is rejected and punished without changing the state of an otherwise successful swap, so the complaint interface cannot be used to grief an honest counterparty at no cost. Third, if one exchanger does not reveal before Δt_1, the counterparty raises a valid complaint and at least t valid watcher responses are posted before Δt_2; the swap still succeeds, while the faulty exchanger and non-responsive or invalid responders are punished.

The failure outcomes cover unresolved executions. If an exchanger fails to commit, the swap never starts and the faulty exchanger is punished after timeout. If both exchangers commit but neither side reveals or raises a valid complaint,

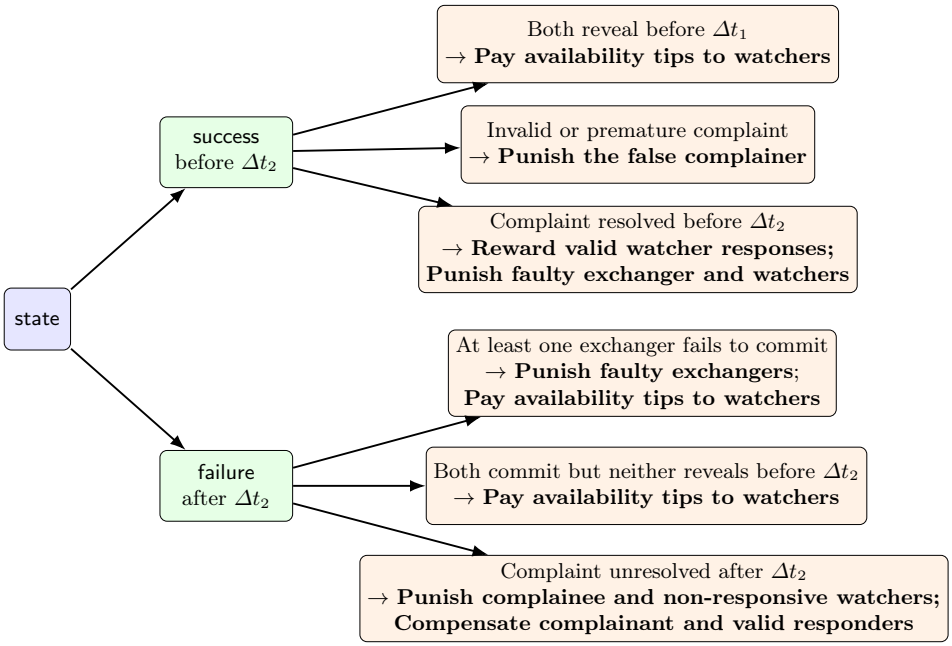

Fig. 11. The incentive and penalty mechanism

escrowed assets are refunded after Δt_2. If a valid complaint remains unresolved because fewer than t valid watcher responses are posted before Δt_2, the complainee and the non-responsive or invalid watchers are penalized to compensate the complainant and any valid responders. With $t = 1$, this last case captures failure of the economic response mechanism, not a cryptographic assumption that a threshold of watchers is honest.

B.3 UC-Based Security of the DEX

This appendix gives the UC argument for the concrete DEX protocol Π_{DEX}. The proof is intentionally interface-based: it follows exactly the contract calls exposed by the implementation, register, deposit, swap_1, complain, swap_2, and determine. These interfaces correspond to registration, asset escrow, commitment, complaint, reveal/arbitration, and settlement in $\mathcal{F}_{\mathsf{DEX}}$. This keeps the proof aligned with the ledger events that are visible to the environment.

Model and assumptions. We work in the $\mathcal{F}_{\mathsf{Ledger}}$-hybrid UC model. The ledger maintains append-only public state and executes the DEX contract deterministically. A PPT adversary $\mathcal{A}$ statically corrupts parties, controls message scheduling, and observes all public events, but cannot alter ledger state or contract code. The underlying PVGSS scheme satisfies correctness, encrypted-share verifiability, share-decryption verifiability, and IND1-secrecy (Theorems 1–4); all

group elements are validated; and the Fiat-Shamir and DLEQ proofs satisfy the zero-knowledge and knowledge-soundness properties used in Theorems 2 and 3. Separately, the DEX fairness argument uses economic accountability rather than an honest-watcher threshold. Every watcher response is checked on-chain by $\mathsf{PVGSSKeyVrf}$. Missing, invalid, or premature arbitration is attributable and is penalized by the incentive mechanism in Appendix B.2. Thus the implementation may set $t = 1$: one responsive watcher resolves a valid complaint, while faulty or non-responsive watchers lose stake.

Theorem 5 (UC Security of Π_{DEX}). *Assume ledger liveness for delivered finalization transactions and the economic accountability condition stated in Appendix B.2. Under the cryptographic assumptions above, for any PPT real-world adversary $\mathcal{A}$ interacting with Π_{DEX} and $\mathcal{F}_{\mathsf{Ledger}}$, there exists a PPT simulator $\mathcal{S}$ interacting with $\mathcal{F}_{\mathsf{DEX}}$ and $\mathcal{F}_{\mathsf{Ledger}}$ such that for every PPT environment $\mathcal{Z}$,*

$$\mathsf{EXEC}_{\Pi_{\mathsf{DEX}},\mathcal{A},\mathcal{Z}} \approx_c \mathsf{EXEC}_{\mathcal{F}_{\mathsf{DEX}},\mathcal{S},\mathcal{Z}}.$$

Here $\approx_c$ denotes computational indistinguishability.

Proof. We construct $\mathcal{S}$ by emulating the ledger and the DEX contract for $\mathcal{A}$, while forwarding the semantic effect of each accepted contract call to $\mathcal{F}_{\mathsf{DEX}}$.

Registration and deposits. For honest parties, $\mathcal{S}$ samples $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{PVGSSSetup}(\lambda)$ and registers pk_i. For corrupted parties, it forwards adversarially chosen public keys and deposits only when they satisfy the same validation and escrow checks as the contract.

Commitment via swap_1. For an honest exchanger x, $\mathcal{S}$ samples s_x, runs $(C^{(x)}, \mathsf{prf}_s^{(x)}) \leftarrow \mathsf{PVGSSShare}(s_x, \mathbb{A}, \mathsf{pks})$, records the ciphertext vector, and forwards $(\mathsf{commit}, \mathsf{id}, x)$ to $\mathcal{F}_{\mathsf{DEX}}$. For a corrupted exchanger, it first runs the contract verification $\mathsf{PVGSSVerify}(C^{(x)}, \mathsf{prf}_s^{(x)}, \mathbb{A}, \mathsf{pks})$. Rejected commitments are rejected in the simulation and produce the same public rejection event as in the real contract. For accepted corrupted commitments, Theorem 2 ensures, except with negligible probability, that the public ciphertext vector is bound to some valid PVGSS sharing under $\mathbb{A}$. Hence $\mathcal{S}$ can safely map the accepted swap_1 transaction to the ideal commitment event without needing to learn the committed secret.

Exchanger openings via swap_2. If an honest exchanger x opens, $\mathcal{S}$ computes $(K_x^{(x)}, \mathsf{prf}_{K_x}^{(x)}) \leftarrow \mathsf{PVGSSPreRecon}(C_x^{(x)}, \mathsf{sk}_x)$ from the recorded ciphertext and forwards $(\mathsf{reveal}, \mathsf{id}, x)$ to $\mathcal{F}_{\mathsf{DEX}}$ after the simulated contract accepts the proof. For a corrupted exchanger opening, $\mathcal{S}$ runs the same $\mathsf{PVGSSKeyVrf}$ check as the contract against the stored ciphertext and the registered public key; invalid openings produce no ideal reveal event. Accepted corrupted openings are correct decryptions of the corresponding stored ciphertext except with negligible probability by Theorem 3, and are therefore mapped to the same ideal reveal event.

Complaints via $\mathsf{complain}$. A complaint is forwarded to $\mathcal{F}_{\mathsf{DEX}}$ exactly when it is accepted by the simulated contract according to the complaint rules of Π_{DEX}.

Complaints that are rejected by the simulated contract produce the same public rejection or penalty event as in the real ledger, and no ideal complaint is sent. Thus the complaint interface changes only the public dispute state; it does not itself create an opening or reveal any PVGSS share.

Watcher arbitration via swap_2. Once a valid complaint is active, an honest watcher i is simulated by computing $(K_i^{(x)}, \mathsf{prf}_{K_i}^{(x)}) \leftarrow \mathsf{PVGSSPreRecon}(C_i^{(x)}, \mathsf{sk}_i)$ for the missing exchanger x. For a corrupted watcher response, $\mathcal{S}$ applies the contract's $\mathsf{PVGSSKeyVrf}$ check to the submitted decrypted share and proof. A valid watcher response is forwarded to $\mathcal{F}_{\mathsf{DEX}}$ as $(\mathsf{arbitrate}, \mathsf{id}, x, i)$. Premature, unsolicited, duplicate, or invalid watcher responses are rejected or penalized in both executions. By Theorem 3, every accepted corrupted watcher opening is consistent with the stored ciphertext except with negligible probability.

Settlement via $\mathsf{determine}$. When $\mathsf{determine}$ is invoked, $\mathcal{S}$ delivers the corresponding finalization request to $\mathcal{F}_{\mathsf{DEX}}$. The ideal functionality updates its state according to the same public predicates used by the contract: both exchangers opened, or one exchanger is missing and enough valid watcher openings have been recorded, or the session remains unresolved until Δt_2. The simulator mirrors the resulting $\mathsf{success}$ or $\mathsf{failure}$ event on the simulated ledger, including the same transfers, refunds, rewards, and slashing decisions.

Simulation consistency. The simulator accepts and rejects the same DEX messages as the real contract, except with negligible proof-system error.

The theorem follows because honest commitments and openings are generated by the real PVGSS algorithms, while every accepted corrupted commitment or opening is publicly bound to a valid sharing or a valid decryption except with negligible probability. Before an authorized opening set is formed, the adversary's decrypted-share view is an unauthorized labelled set under $\mathbb{A}$; therefore IND1-secrecy hides the honest exchanger's secret from the environment. Consequently the environment sees the same ledger events, the same acceptance and rejection behavior, and computationally indistinguishable cryptographic messages in the real and ideal executions. This gives the claimed UC realization by a standard hybrid argument. □

B.4 Security Properties of the DEX

The following properties are consequences of Theorem 5 and the public validation rules of the contract. They should be read under the same conditions as the UC theorem: ledger liveness for delivered finalization transactions, PVGSS security, and the economic accountability mechanism of Appendix B.2.

Lemma 4 (Decentralization). *Protocol Π_{DEX} does not rely on a trusted third-party operator.*

Proof. Registration, asset escrow, commitment verification, share-opening verification, complaint handling, and settlement are performed by the public contract and the PVGSS verification algorithms. No designated operator holds a special key, takes custody of the exchanged assets, or decides the outcome off-chain. □

Lemma 5 (Termination). *Every execution of Π_{DEX} can be finalized within bounded ledger time once a valid finalization transaction is delivered.*

Proof. The contract enforces $\Delta t_1 < \Delta t_2$. If the evidence for success is available, namely both exchangers have opened or one exchanger has opened and a valid complaint has accumulated enough watcher openings, determine can finalize success immediately. If no success evidence is available, then after Δt_2 any delivered valid determine transaction finalizes failure and refunds the escrowed assets. □

Lemma 6 (Fairness). *Protocol Π_{DEX} guarantees fairness: an honest exchanger either obtains the correct counterparty assets or can finalize a refund, assuming ledger liveness and the incentive mechanism of Appendix B.2.*

Proof. swap_1 binds each exchanger to publicly verifiable PVGSS shares before any transfer can occur. If both exchangers reveal, PVGSS correctness lets the parties reconstruct the required secrets and the contract finalizes success. If one exchanger withholds its reveal after the counterparty has opened, the counterparty can file a valid complaint. The complaint activates watcher arbitration; with $t = 1$, any one responsive watcher can submit a verified swap_2 response and complete the exchange. Malicious or rushing attempts do not create additional settlement power: a complaint is effective only after the public complaint predicate is satisfied, and watcher responses are accepted only for an active complaint and only if the accompanying decryption proof verifies. Thus false or premature complaints, invalid openings, and premature watcher responses are rejected or penalized, while missing, invalid, or premature watcher actions are attributable and slashable. If arbitration is not completed by Δt_2, the contract finalizes failure and refunds the escrowed assets. Hence an honest exchanger either receives the counterparty assets or recovers its own assets; this conclusion uses ledger liveness and incentive compatibility, not an assumption that more than t watchers are honest. □

Lemma 7 (Accountable Exchangers). *Every outcome-affecting exchanger deviation is publicly attributable.*

Proof. PVGSSVerify checks exchanger commitments in swap_1, and PVGSSKeyVrf checks decrypted shares in swap_2. Complaints are accepted only if the public complaint predicate is satisfied; false complaints are rejected and attributed to the caller. Thus malformed commitments, invalid openings, false complaints, and missed deadline-dependent actions are visible from the ledger state. □

Lemma 8 (Accountable Arbitration). *All watcher arbitration actions are publicly verifiable and attributable.*

Proof. After a complaint, each watcher response consists of a decrypted share and a DLEQ proof generated by PVGSSPreRecon. The contract verifies the response with PVGSSKeyVrf against the stored ciphertext, the registered public key, and the active complaint. Thus invalid, duplicate, premature, unsolicited, or missing arbitration is attributable from public ledger state and can be rewarded or penalized according to Appendix B.2. □

Lemma 9 (Optimistic and Stateless Arbitration). *Protocol Π_{DEX} supports optimistic and stateless arbitration.*

Proof. When both exchangers reveal on time, settlement uses only exchanger swap_2 messages and no watcher is invoked for arbitration. Upon a valid complaint, watchers compute responses from on-chain ciphertexts, public session data, and their long-term secret keys. Hence a watcher need not maintain per-swap private state beyond its registered key material. □

Lemma 10 (Incentive Compatibility). *Π_{DEX} admits a slashing-based parameter regime in which valid watcher arbitration is the rational strategy, allowing the protocol to set $t = 1$.*

Proof. By Lemmas 7 and 8, outcome-affecting deviations are detectable. Let B be an upper bound on the benefit of a watcher deviation, G the gas and opportunity cost of a valid response, R the reward for a valid response, and D the slashable stake. If $D > B$ and $R > G$, a rational watcher maximizes payoff by answering valid complaints and avoiding invalid, missing, duplicate, or premature responses. Therefore one economically accountable watcher is sufficient for dispute resolution, allowing $t = 1$. □

Claim 1 (Resistance of DoS attack) *DoS attempts can delay progress only until Δt_2, assuming ledger liveness for delivered finalization transactions. Missing or invalid required actions are attributable on-chain and are penalized under the stake bounds of Lemma 10.* □

Claim 2 (Resistance of Collusion/Coercion attack) *Public verification and slashing make collusive watcher behavior attributable and economically irrational under the incentive parameters of Lemma 10. Even with $t = 1$, a faulty watcher cannot make an honest exchanger lose escrowed assets by submitting invalid evidence: invalid responses are rejected, and an unresolved complaint is finalized as* failure *after Δt_2.* □

Claim 3 (Resistance of Sybil attack) *Watcher service capacity is proportional to locked, slashable stake. Splitting the same stake across Sybil identities does not increase total capacity or reduce the slashable amount needed for large-scale deviation.* □

Claim 4 (Resistance of Rushing attack) *By Lemma 6, premature complaints and premature watcher responses are rejected or penalized because arbitration is effective only after the complaint predicate is satisfied and the corresponding decryption proof verifies. Thus rushing does not change the settlement outcome for honest exchangers.* □

C Complexity Analysis of the PVGSS Scheme

Table 3 summarizes the cost of the PVGSS algorithms along three dimensions: dominant group operations in $\mathbb{G}$, public finite-field work over $\mathbb{Z}_q$, and

communication. We write Exp for one exponentiation in $\mathbb{G}$. The group-operation column captures the dominant cryptographic operations, while the finite-field column records algebraic routines such as GSS sharing, GSS testing, interpolation, and LSSS reconstruction-weight computation, which affect the concrete running time but are separate from group exponentiations.

Table 3. Complexity summary of PVGSS algorithms

Algorithm	Computation		Communication
	Dominant group operations	Finite-field work	
PVGSSShare	$2\|\mathbb{A}\|$Exp	GSSShare over $\mathbb{Z}_q$	$2\|\mathbb{A}\|\|\mathbb{G}\| + (\|\mathbb{A}\| + 2)\|\mathbb{Z}_q\|$
PVGSSVerify	$2\|\mathbb{A}\|$Exp	$\mathsf{GSSTest}_{\mathsf{Exact/Dual}}$	–
PVGSSPreRecon	3Exp per share	–	$3\|\mathbb{G}\| + 2\|\mathbb{Z}_q\|$ per share
PVGSSKeyVrf	4Exp per share	–	–
PVGSSRecon	$\|Q\|$Exp	Lagrange interpolation or calculation of $\boldsymbol{w}_Q$	$\|\mathbb{G}\|$

Group operations. In the sharing phase, PVGSSShare performs $2|\mathbb{A}|$ exponentiations, one for each C_i and C'_i. The verifier in PVGSSVerify performs $2|\mathbb{A}|$ exponentiations for the Schnorr-type equations. In the reconstruction phase, PVGSSPreRecon costs three exponentiations per decrypted share: one for $S_i = C_i^{1/\mathsf{sk}_i}$ and two for the DLEQ proof commitments. The PVGSSKeyVrf algorithm costs four exponentiations per share for DLEQ verification. Given prepared aggregate reconstruction coefficients, online PVGSSRecon is a group linear combination and uses $|Q|$ exponentiations.

Finite-field work. The finite-field work in Table 3 should be read as follows.

- For PVGSSShare, the finite-field work is the underlying GSS sharing of the secret and the random response value. In Shamir SS-based PVGSS this is polynomial evaluation over the access tree, while in LSSS-based PVGSS it is the matrix-vector sharing computation over $\mathbb{Z}_q$.
- For PVGSSVerify, the finite-field work is GSS testing over the response values. Its cost is linear in $|\mathbb{A}|$ once the local interpolation or dual-code weights for the fixed evaluation domains are prepared.
- For PVGSSRecon, the finite-field work depends on the instantiated sharing scheme. In Shamir SS-based PVGSS, reconstruction is performed along the selected path; the required Lagrange coefficients depend only on the public path and can be prepared for repeated paths, while computing them online costs $O(|Q|^2)$ field operations. In LSSS-based PVGSS, $\boldsymbol{w}_Q$ depends on the realized authorized row set Q, so a fixed policy alone only permits preprocessing of the public matrix and caching for repeated sets. Only when both $\mathbb{A}$ and Q are fixed can the weights be fully prepared in advance, which is the prepared/online path used for the reconstruction benchmark. Without prepared weights, generic LSSS reconstruction solves $M_Q^\top \boldsymbol{w} = \boldsymbol{e}_1$, e.g., by Gaussian elimination in $O(|Q|^3)$ field operations.

Communication. In the sharing phase, the dealer outputs $(\{C_i\}, \mathsf{prf}_s)$, consisting of $2|\mathbb{A}|$ elements of $\mathbb{G}$ and $(|\mathbb{A}|+2)$ elements of $\mathbb{Z}_q$. During reconstruction, each accepted decrypted share contributes one group element S_i and a DLEQ proof with two group elements and two field elements, i.e., $3|\mathbb{G}|+2|\mathbb{Z}_q|$ per share. The reconstruction algorithm outputs one group element in $\mathbb{G}$. Hence the sharing communication is $O(|\mathbb{A}|)$, while reconstruction communication is $O(|Q|)$.

D Experimental Evaluation

We implement our PVGSS scheme using Golang (off-chain) and Solidity (on-chain). The method in [14] supports only boolean formula-based access structures; therefore, we adopt the approach proposed in [50] to transform a general access structure into a LSSS matrix. To make on-chain and off-chain operations compatible, we choose BN128 as the asymmetric elliptic curve group, which is officially supported by Ethereum. The off-chain benchmarks of PVGSS schemes are executed on an Apple M4 with 16 GB RAM running macOS 26.3 and Golang 1.25.4. The smart contract codes are deployed and tested on a local Ethereum test network.

D.1 Off-Chain PVGSS Costs

We choose the access structure designed for the DEX (cf. Equation (1)) to evaluate the performance of the PVGSS scheme, where $t = n/2+1$ and n ranges from 100 to 1100. Figure 12-15 show the off-chain computational overheads of the PVGSS schemes.

Figure 12 reports the cost of PVGSSShare. Both constructions grow superlinearly because the sharing procedure generates one encrypted group share for each participant and constructs the corresponding Schnorr-type proof material. The Shamir SS-based curve is slightly higher at large n, mainly because its implementation evaluates the threshold polynomial over the full participant set, whereas the LSSS-based implementation shares along the access tree and then maps the resulting LSSS shares to group commitments.

Figure 13 reports the cost of PVGSSVerify. Verification first checks the NIZK equations for all encrypted shares and then invokes one GSS testing routine. The exact interpolation test evaluates the required values directly with precomputed Lagrange coefficients, rather than recovering full polynomial coefficients by generic Gaussian elimination. The dual-code test reuses precomputed barycentric weights for the fixed evaluation domain and reconstructs internal-node secrets by Lagrange interpolation at zero. The dual-code curves are therefore higher in this implementation, because the dual relation is checked over the relevant evaluation domain in addition to the per-share NIZK checks. These optimizations preserve the interpolation and dual-code verification relations defined in Definition 9.

Figure 14 reports n executions of PVGSSKeyVrf. The curves are close to linear because each decrypted share is verified independently by one DLEQ proof.

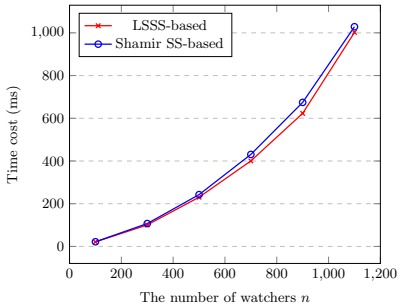

Fig. 12. Cost of PVGSSShare

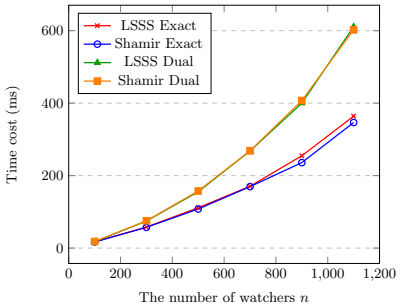

Fig. 13. Cost of PVGSSVerify

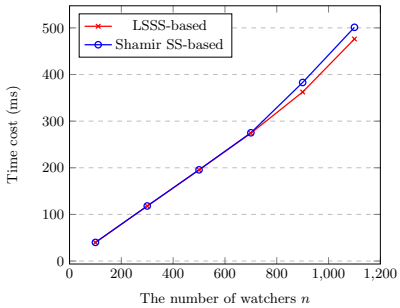

Fig. 14. Cost of n times of PVGSSKeyVrf

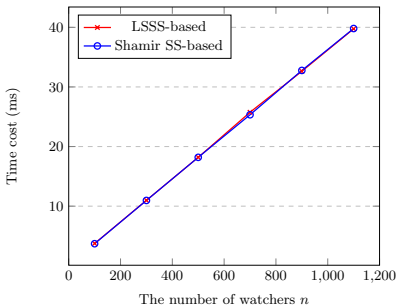

Fig. 15. Online PVGSSRecon with an authorized set size $t + 1$

The LSSS-based and Shamir SS-based variants almost overlap, since key verification depends only on $(C_i, K_i, \mathsf{pk}_i)$ and the DLEQ proof, not on how the access structure generated the original share.

Figure 15 reports online PVGSSRecon after preparing public reconstruction coefficients. For both constructions, the benchmark fixes the access policy and the authorized set of size $t + 1$, computes the public aggregate reconstruction coefficients before timing, and measures only the resulting group linear combination. The two curves therefore almost overlap and grow close to linearly with the authorized set size.

D.2 On-Chain DEX Costs

Further, we evaluate the gas consumption of the DEX. The on-chain prototype reports costs for $t = 1$ and n ranging from 1 to 10. We first introduce the cost in the normal case, where both exchangers honestly abide by the protocol. In swap_1, calculation of PVGSSVerify and storage of $\{C_i\}$ are included. The swap_2, which invokes PVGSSKeyVrf, roughly costs a constant amount of gas (about 140000) for each exchanger. The PVGSSKeyVrf algorithm is the same for both LSSS-based and Shamir SS-based PVGSS, hence they cost equal gas con-

sumption. Figure 16 depicts the gas consumption of the $\mathsf{swap}_1 + \mathsf{swap}_2$ on smart contract. The four curves separate exact interpolation and dual-code GSS testing for both LSSS-based and Shamir SS-based PVGSS, matching the off-chain verification variants. The on-chain implementation keeps the same verification relations but avoids generic polynomial-coefficient recovery: the exact test evaluates the interpolation polynomial directly at zero and at the remaining child points, while the dual-code test computes the fixed-domain dual weights using factorial denominators. Then, we depict the gas cost in exceptional scenarios where arbitration procedures are required. Particularly, an exchanger calls complain, then each watcher invokes swap_2 and finally anyone can execute determine. Figure 17 shows the gas cost of functionalities of complain, determine and swap_2.

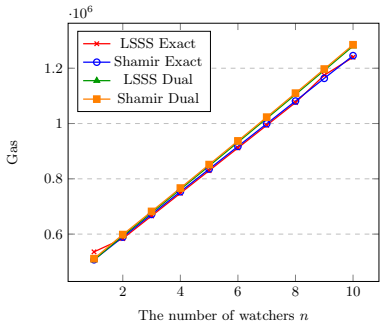

Fig. 16. On-chain gas cost of $\mathsf{swap}_1 + \mathsf{swap}_2$

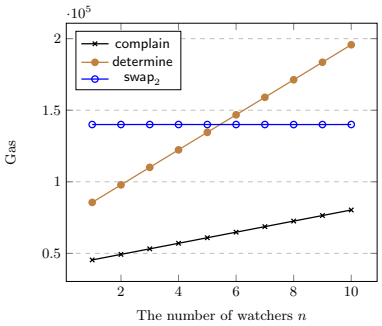

Fig. 17. On-chain cost of complain, determine, swap_2

As of the time of writing (May 9, 2026), the median gas price is 0.209 gwei, and the price of ETH is 2,326 USD.[13] Table 4 presents the monetary cost of each operation on the smart contract. For convenience, only the Shamir SS-based PVGSS is evaluated. The gray cells highlight the optimistic case, where the cost for each exchanger is only \$0.247/\$0.406/\$0.566 for $n = 1/5/9$ and $t = 1$. If watchers are involved in the pessimistic case, the complaint cost is less than 0.037 USD, and each watcher incurs a cost of approximately 0.068 USD (as swap_2 requires).

[13] `https://etherscan.io/gastracker`

Table 4. Monetary cost of the on-chain operations

Operation	Gas cost	USD
register	89985	\$0.044
freeze	195114	\$0.095
swap_1	n=1 367121	\$0.178
	n=5 695237	\$0.338
	n=9 1023451	\$0.498
swap_2	139999	\$0.068
total, i.e.,	n=1 507120	\$0.247
$\mathsf{swap}_1 + \mathsf{swap}_2$	n=5 835236	\$0.406
(Shamir SS-based)	n=9 1163438	\$0.566
complain	n=1 45399	\$0.022
	n=5 60911	\$0.030
	n=9 76423	\$0.037
determine	n=1 85604	\$0.042
	n=5 134552	\$0.065
	n=9 183500	\$0.089