Publicly Verifiable Generalized Secret Sharing and Its Application in Building Decentralized Exchange

Liang Zhang *HKUST* · Dongliang Cai *Fudan University* · Tao Liu *Hainan University* · Haibin Kan* *Fudan University* · Jiheng Zhang* *HKUST*

Abstract

Generalized secret sharing (GSS), which can offer more flexibility by accommodating diverse access structures and conditions, has been under-explored in distributed computing over the past decades. To address the gaps, we propose the publicly verifiable generalized secret sharing (PVGSS) scheme, enhancing the applicability of GSS in transparent systems. Public verifiability is a crucial property to gain trustworthiness for decentralized systems like blockchain. We begin by introducing two GSS constructions, one based on Shamir's secret sharing and the other on the linear secret sharing scheme (LSSS). Next, we present PVGSS schemes that combine GSS with non-interactive zero-knowledge (NIZK) proofs. Further, we construct a decentralized exchange (DEX) based on PVGSS scheme, where any users can participate in exchanges and engage in arbitrage. Specifically, users can fairly swap ERC-20 tokens with passive watchers, who earn profits by providing arbitration services. The critical property of "fairness" required by the DEX is ensured through a sophisticated access structure, supported by the PVGSS scheme. We provide a comprehensive evaluation on the performance of the PVGSS schemes and the monetary costs for users in the DEX. The results demonstrate the feasibility and practicality of this approach in real-world applications.

1 Introduction

Secret sharing (SS) [1] is widely adopted in decentralized and distributed systems [2], such as threshold cryptosystems [3], distributed storage [5], MPC [6], federated learning [4] and BFT consensus [7]. Various variant protocols of SS have been proposed, fostering the application of SS in cryptographic protocols and distributed systems.

On the one hand, linear secret sharing scheme (LSSS) [19, 21, 40] is put forwarded, since most SS schemes are linear. LSSS is proved to be equal to the notion of monotone span program [8]. On the other hand, to detect potential faulty behavior and enhance transparency for SS-based protocols, verifiable secret sharing (VSS) [9, 14] and publicly verifiable secret sharing (PVSS) [11, 12, 15] are proposed. While VSS enables shareholders to detect faulty dealers, PVSS allows for public verifiability by any party.

Generalized secret sharing (GSS) [16, 17] extends beyond threshold secret sharing, enabling the implementation of generalized access structures (GAS). Ciphertext-policy attribute-based encryption (CPABE) [18, 19] combines arbitrary monotone access structure with public key encryption, establishing a promising cryptographic primitive using attributes strings. The construction and broader potential applications of GSS are still being explored and developed.

In this paper, we put forward publicly verifiable generalized secret sharing (PVGSS) to enrich the family of SS protocols. We first provide two constructions of GSS scheme (i.e., Shamir SS-based and LSSS-based). Then, we formalize the definition of PVGSS and its security requirements. Further, we construct PVGSS scheme by hiding shares of GSS scheme and attaching non-interactive zero knowledge (NIZK) proofs to prove the correctness in hiding. Moreover, we build a decentralized exchange to highlight the practicality of PVGSS schemes. Table 1 depicts the distinguishes of cryptographic primitives that are related to secret sharing.

Table 1: Comparison of (PV)SS, CPABE and PVGSS

Scheme	accountable dealer or encryptor	accountable shareholders or authorities	insecure channel	public verifiability	GAS-supported
SS	✗	✗	✗	✗	✗
VSS	✓	✓	✗	✗	✗
PVSS	✓	✓	✓	✓	✗
GSS	✗	✗	✗	✗	✓
CPABE	✗	✗	✓	✗	✓
PVGSS	✓	✓	✓	✓	✓

1.1 Potential of Generalized Access Structure

Generalized access structure (GAS) [17, 19, 40] extends the traditional threshold-based access control to support more complex and flexible authorization requirements. Unlike threshold schemes, GAS enables the definition of arbitrary monotone access structures. Features of GAS include:

Monotonicity: GAS is defined by a monotone access structure, meaning if a set of participants is authorized to access a secret, any superset of that group is also authorized. This property ensures logical consistency in access control.

Expressiveness: GAS supports monotone access structure, enabling granular and precise control over who can reconstruct a secret.

Scalability: GAS efficiently supports large and dynamic participant sets, making it suitable for applications with evolving access needs, such as multi-organizational collaborations.

Adaptability: GAS can be tailored to various real-world applications, such as role-based access, weighted systems, or conditional hierarchies in decentralized environments.

1.2 Critical Role of Public Verifiability

Blockchain's momentum over the past decade stems from its ability to ensure transparency, decentralization and smart contracts. It revolutionizes industries by enabling trustless systems [22], empowering individuals [23], and fostering innovation in finance, healthcare, and global collaboration. Public verifiability [24, 25] therefore is critical when conducting privacy-preserving computation in a transparent blockchain or open environment.

Public verifiability is a fundamental property of certain cryptographic algorithms that allows any entity, regardless of its involvement in the initial operation, to independently verify the correctness of a computation or proof. This feature guarantees trust and transparency in cryptographic systems by enabling third-party validation without compromising the confidentiality of the underlying data. Therefore, it plays a crucial role in enforcing accountability by providing transparent validation and traceability.

Merkle tree [26] and accumulators [27] are often employed to prove membership of specific data in a set publicly. Signature schmes [28, 50] are leveraged to achieve authentication, i.e., enabling public verifiability of identities. NIZK systems, such as Fiat-Shamir heuristic based Σ protocol [35, 36] and zkSNARKs [29], are often adopted to prove knowledge of plaintext or correctness of operations in a public manner.

1.3 Contributions

- We introduce the concept of publicly verifiable generalized secret sharing (PVGSS), which extends the family of secret sharing (SS) schemes. GSS enhances SS by providing versatile access control as an atomic cryptographic primitive. Additionally, PVGSS enables public verifiability and accountability. Both GSS and PVGSS are valuable in the fields of cryptography and blockchain, as discussed in Section 8.

- For the first time, we explicitly demonstrate that LSSS is a specific instance of the GSS scheme. We also present two PVGSS constructions that combine the GSS scheme with NIZK proofs, where the GSS scheme is derived from either recursive Shamir secret sharing or LSSS.

- We construct a decentralized exchange (DEX) based on the proposed PVGSS scheme to highlight the application. The underlying access structure, i.e., $(2\ of\ (Alice, Bob, (t\ of\ (W_1, W_2, \cdots, W_n))))$, enables the DEX to achieve fairness for exchangers and optimistic arbitration with fault-tolerant arbitrators. Moreover, the incentive mechanism is carefully designed to ensure the practicality and usability of the DEX.

2 Related Works

2.1 (Publicly) Verifiable Secret Sharing

Dealer in traditional SS scheme may distribute invalid shares to deviate from the protocol. To mitigate this problem, VSS [9, 14] ensures that participants can verify the validity of the shares distributed by the dealer, while PVSS [12, 15] extends this by allowing any external observer to verify the integrity of the shares.

The first VSS scheme, proposed by Feldman [9], enhances Shamir's secret sharing by adding public commitments to ensure share validity. Shareholders verify their shares against these commitments, ensuring the dealer's honesty without revealing the secret. Recently, polynomial commitments [10, 13, 14] are widely adopted to design new VSS schemes. The secret shares of VSS should not be transferred publicly.

Stadler [15] put forward the first PVSS scheme in 1996. Intuitively, PVSS can be achieved by integrating public key encryption into VSS schemes, allowing secret shares to be delivered and verified on public channels. For example, ElGamal, RSA and Pailler cryptosystem are incorporated in [15, 44], [43] and [45], respectively. Early PVSS schemes require $O(nt)$ verification complexity, where n and t are the amount of shareholders and threshold value, respectively. In 2017, Cascudo et al. [12] introduced SCRAPE PVSS which for the first time achieves $O(n)$ verification complexity. As the core technique, Reed-Solomon code is employed to make all encrypted shares be verified in a linear operation.

Recently, Cascudo et al. successively proposed ALBATROSS [42], HEPVSS/DHPVSS [41] and qCLPVSS [11], which all maintain $O(n)$ verification complexity. ALBATROSS [42] leverages low-degree exponent interpolation to generate NIZK proof, which is essentially acquired from Σ protocol and Fiat-Shamir heuristic. HEPVSS [41] also uses ElGamal to encrypt secret share, while DHPVSS [41] leverages Shamir secret sharing on a group $\mathbb{G}$ to hide shares. qCLPVSS [11] for the first time enables users to recover the original secret $s \in \mathbb{Z}_q$ by leveraging class group. Recover-ing the original secret s can make PVSS applicable in more wider applications, such as distributed key generation [2, 11] and MPC [11, 41] protocols.

2.2 Access Structures in Secret Sharing

As is well known, traditional SS, VSS, and PVSS protocols [1, 9, 11, 12, 15] are designed around the threshold access structure (TAS). Benaloh et al. [17] introduced the concept of generalized secret sharing (GSS) based on recursive threshold secret sharing, where GAS is inherently implied. Beyond schemes based on TAS or GAS, some other protocols have explored specific types of access control mechanisms.

Particularly, Simmons [34] points out that real-world application require more versatile capabilities for secret sharings. In weighted secret sharing (WSS) [31, 37], each participant is assigned a positive weight. Only if the sum of weights of some participants exceeds the predefined threshold value, can the secret be reconstructed.

In multipartite secret sharing (MSS) [32, 33], an access structure defined over n participants is partitioned into disjoint $m(\ll n)$ compartments, where participants within the same compartment have equal weights. In MSS, each compartments must achieve a specific internal agreement before contributing to the reconstruction of the secret. Tassa et al. [33] point out that any GAS can be regarded as a multipartite access structure with singleton compartments. Hence, reconstructing the secret in MSS scheme depends only on the number of participants from each compartment, rather than their individual identities. In this way, practical applications can focus on m compartments, where TAS is applied, instead of n individuals. However, participants may be designed in different access structures in a single system, resulting in different divisions of disjoint compartments.

Hierarchical secret sharing (HSS) [38, 39], also referred as multilevel secret sharing [34, 38], place participants in disjoint levels according to their importance. HSS can be applied in managing staffs in big companies, where employees are assigned to different levels with different privileges. HSS is categorized into disjunctive and conjunctive types [30]. Further, Drăgan et al. [30] introduce the distributive weighted threshold secret sharing, where participants at the same level have the same weight, and the threshold values are 1.

Though GSS may not "ideal" [30–33], GSS can satisfy the application scenarios where MSS, WSS or HSS apply.

3 Preliminaries

3.1 Shamir Secret Sharing

Definition 1 (Shamir SS) *Shamir secret sharing (SS) scheme allows a dealer to split a secret into n shares for n shareholders. With only a threshold $t(\leq n)$ shares, the secret can be successfully recovered, while fewer than t shares reveal nothing about the secret. Shamir SS is defined with following 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 algorithm.*

Definition 2 (Shamir SS on a Group $\mathbb{G}$) *Definition 1 has introduced the original Shamir secret sharing scheme, where a secret $s \in \mathbb{Z}_q$ is shared and recovered. 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 Shamir secret sharing on a group of order q is defined as below:*

- $\{S_1, S_2, \cdots, S_n\} \leftarrow \mathsf{GrpShare}(S, n, t)$*: 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 $S_i = S \cdot g^{f(i)}$, $i \in [0, n]$. Obviously, $S_0 = S$.*
- $S \leftarrow \mathsf{GrpRecon}(Q)$*: Inputs any t shares Q, reconstructs the secret $S \leftarrow S_0 = \prod_{i \in I} S_i^{\prod_{j \in I, j \neq i} \frac{-j}{i-j}}$, where I is the set containing indexes of the shares Q.*

3.2 (Generalized) Access Structure

Definition 3 (AS/GAS [17, 19, 40]) *Let $\mathcal{P} = \{P_1, \cdots, P_n\}$ be a set of parties. A collection $\mathbb{A} \in 2^{\mathcal{P}}$ is a (generalized) access structure (AS/GAS), if it meets the following two conditions:*

- *non-triviality: if $B \in \mathbb{A}$, then $|B| \neq 0$.*
- *monotonicity: if $B \in \mathbb{A}$, $B \subseteq C$, then $C \in \mathbb{A}$.*

If $B \in \mathbb{A}$, we say $\mathbb{A}$ is satisfied by B or B is an authorized set; Otherwise, $\mathbb{A}$ is not satisfied by B or B is unauthorized.

For example, $(P_1 \wedge P_2) \vee (P_2 \wedge P_3)$ and $1\ of\ (2\ of\ (P_1, P_2, P_3), (1\ of\ (P_1, P_4)))$ are access structures, as shown by Figure 16 and Figure 17 in Appendix A. The disjunction (or conjunction) in a boolean formula based access structure is equal to gate value of 1 (or 2) in a threshold-gate based access structure. Hence, we only consider threshold-gate based access structure in this paper.

Corollary 1 *The boolean or threshold-gates based access structure can be regarded as a proposition, which outputs* true *given an authorized set and outputs* false *otherwise.*

Corollary 2 *An access structure $\mathbb{A}$ can be regarded as a recursive data structure, meaning that any node x in $\mathbb{A}$ with its children nodes is also an access structure, denoted by $\mathbb{A}_x$. For example, there are 8 sub access structures in $\mathbb{A}$ of Figure 17.*

Corollary 3 *If $B \in \mathbb{A}$, there exists a path (i.e., a leaf node to the root node) in which each sub access structure is satisfied by B. We call it a satisfied path.*

3.3 Linear Secret Sharing Scheme

Definition 4 (LSSS [19, 21, 40]) *Let $\mathcal{P}$ be a set of shareholders. A linear secret sharing scheme $\prod$ over a ring $\mathbb{Z}_q$ for $\mathcal{P}$ is a tuple (M, ρ), where $M \in \mathbb{Z}_q^{m \times l}$ is a share-generating matrix and ρ is a row-labeling function. The scheme $\prod$ consists of the following two algorithms:*

- $\{s_1, s_2, \cdots, s_m\} \leftarrow \mathsf{LSSSShare}(s, \mathbb{A})$*: To share a value $s \in \mathbb{Z}_q$, randomly choose $v_2, \ldots, v_n \xleftarrow{R} \mathbb{Z}_q$ and set $\vec{v} = [s, v_2, \ldots, v_n]^\top$. Then, $\boldsymbol{s} = M\vec{v}$ is the vector of shares, where $\boldsymbol{s}_i \in \mathbb{Z}_q$ belongs to shareholder $\rho(i)$ for each $i \in [m]$.*

- $s \leftarrow \mathsf{LSSSRecon}(\mathbb{A}, Q)$*: Let $Q \subseteq \mathcal{P}$ be a set of shareholders and let $I_Q = \{i \in [m] : \rho(i) \in Q\}$ be the row indices associated with Q. Let $M_Q \in \mathbb{Z}_N^{|I_Q| \times l}$ be the matrix formed by taking the subset of rows in M that are indexed by I_Q. Only if Q is an authorized set, then there exists a vector $w_Q \in \mathbb{Z}_N^{|I_Q|}$ such that $w_Q^\top M_Q = [1, 0, \ldots, 0]$. Further, the secret s can be reconstructed by $\sum_{i \in I_Q} w_i \boldsymbol{s}_i$.*

Remark: *Access control can be represented either as $\mathbb{A}$ [40] or (M, ρ) [19]. Denote $|\mathbb{A}|$ as the number of leaf nodes of an access structure $\mathbb{A}$, i.e., $|\mathbb{A}| = m$.*

3.4 Σ Protocol and NIZK Proof

Let $\mathcal{R}$ be a relation and $L_{\mathcal{R}} = \{x | \exists w : (x, w) \in \mathcal{R}\}$ be the corresponding language. A Σ protocol (P_1, P_2, V) is a three-move public-coin protocol between a prover and a verifier. Firstly, the prover calculates a commitment $a = P_1(x)$; Secondly, the verifier randomly selects a challenge value $e \xleftarrow{R} \{0,1\}^\lambda$, where λ is the security parameter; Lastly, the prover returns the response value $z = P_2(x, a, e)$. If $V(x, a, e, z) = 1$, the verifier is convinced that the prover has a witness w satisfying $(x, w) \in \mathcal{R}$.

A non-interactive zero-knowledge (NIZK) proof of knowledge of witness x is obtained by applying the Fiat-Shamir heuristic, with a random oracle. Model the random oracle using a random function, $H : \{0,1\}^* \rightarrow \{0,1\}^\lambda$. Consequently, the NIZK proof is denoted as $(a, e = H(x, a), z)$.

The NIZK proof has the properties of completeness, soundness and zero-knowledge. Completeness means that a valid proof always convinces the verifier; Soundness means that false statements cannot produce valid proofs (except negligibly); Zero-Knowledge means that the proof reveals nothing beyond the statement's truth.

3.5 Blockchain and Ethereum

Blockchain is a decentralized, distributed ledger technology that enables secure, transparent, and tamper-proof record-keeping. Ethereum is an open-source blockchain platform that facilitates the execution of smart contracts and decentralized applications. Users can monitor Ethereum events to receive notifications about specific data updates. The gas mechanism in Ethereum is a key component that governs the execution of transactions and smart contracts. Gas is a unit of measurement for computational work, used to prevent abuse of the network by requiring users to pay for resources consumed during transaction processing or contract execution. Ethereum request for comment 20 (ERC-20)[1] is a widely adopted standard for the creation and implementation of fungible tokens on the Ethereum blockchain. ERC-20 provides a standardized set of rules and interfaces that ensure interoperability of tokens within the Ethereum ecosystem. This standard defines essential functions such as `transfer()`, `allowance()`, and `transferFrom()`, which facilitate token transfers between accounts, as well as the allowance mechanism for spending tokens on behalf of others.

4 Generalized Secret Sharing

Definition 5 (GSS) *A generalized secret sharing (GSS) scheme [17] 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 secret shares $\{s_1, s_2, \cdots, s_{|\mathbb{A}|}\}$, where $|\mathbb{A}|$ is the size of $\mathbb{A}$.*

- $s \leftarrow \mathsf{GSSRecon}(\mathbb{A}, Q)$*: Inputs an access structure $\mathbb{A}$ and an authorized set $Q \subseteq \{s_1, s_2, \cdots, s_{|\mathbb{A}|}\}$, reconstructs the secret s.*

Definition 6 (GSS on Group $\mathbb{G}$) *Definition 5 has introduced the original Shamir secret sharing scheme, where a secret $s \in \mathbb{Z}_q$ is shared and recovered. 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 Shamir secret sharing on a group 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 secret shares $\{S_1, S_2, \cdots, S_{|\mathbb{A}|}\}$, where $|\mathbb{A}|$ is the size of $\mathbb{A}$.*

- $S \leftarrow \mathsf{GrpGSSRecon}(\mathbb{A}, Q)$*: Inputs an access structure $\mathbb{A}$ and an authorized set $Q \subseteq \{S_1, S_2, \cdots, S_{|\mathbb{A}|}\}$, reconstructs the secret S.*

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

Below, we introduce two types of GSS schemes, which differ based on the shared secret, $s \in \mathbb{Z}_q$ or $S \in \mathbb{G}$. And, each type is initialized using two primitives, i.e., Shamir SS and LSSS.

(1) *GSS based on Shamir SS:* Benaloh et al. [17] had introduced the construction, which can be obtained by replacing GrpShare (and GrpRecon) with Share (and Recon) in Figure 2. We do not provide additional elaboration here.

(2) *GSS based on LSSS:* It is obvious that LSSS is an instance of GSS scheme. Figure 1 presents the GSS construction using LSSS.

```
Functionality GSS using LSSS
$\{s_1, s_2, \cdots, s_{|\mathbb{A}|}\} \leftarrow \mathsf{GSSShare}(s, \mathbb{A})$ :
    $\{s_1, s_2, \cdots, s_m\} \leftarrow \mathsf{LSSSShare}(s, \mathbb{A})$
$(s) \leftarrow \mathsf{GSSRecon}(\mathbb{A}, Q)$ :
    $s \leftarrow \mathsf{LSSSRecon}(\mathbb{A}, Q)$
```

Figure 1: GSS based on LSSS

(3) *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 17 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 2.

```
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 $j$th child of $x$
    For the $i$th leaf node $z$ in $\mathbb{A}$:
        $S_i = S_{x_j}$, where $z$ is the $j$th 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.
```

Figure 2: GSS scheme on Group $\mathbb{G}$ based on Shamir SS

(4) *GSS on Group $\mathbb{G}$ using LSSS:* LSSS handles an access structure as input. Intuitively, an LSSS can be directly used to construct a GSS scheme. We first depict how to construct LSSS on group $\mathbb{G}$ by Figure 3. Then, we apply it to construct GSS on group $\mathbb{G}$, as Figure 4 presents. Apparently, LSSS (on group $\mathbb{G}$) is an instance of GSS (on group $\mathbb{G}$).

```
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.
    $(M, \rho) \leftarrow \mathbb{A}$
    $\vec{v} = [1, v_2, \ldots, v_n]^\top$, where $v_2, \ldots, v_n \overset{R}{\leftarrow} \mathbb{Z}_q$.
    $(s_1, \cdots, s_{|\mathbb{A}|}) \leftarrow M\vec{v}$
    $S_i = S^{s_i} = g^{s \cdot s_i}$ for $i \in [1, \cdots |\mathbb{A}|]$
$(S) \leftarrow \mathsf{GrpLSSSRecon}(\mathbb{A}, Q)$ :
    $I_Q = \{i \in [m] : \rho(i) \in Q\}$
    $M_Q \in \mathbb{Z}_N^{|I_Q| \times l}$
    find $w_Q \in \mathbb{Z}_N^{|I_Q|}$ such that $w_Q^\top M_Q = [1, 0, \ldots, 0]$
    $S \leftarrow \prod_{i \in I_Q} S_i^{w_i} = \prod_{i \in I_Q} g^{s \cdot s_i \cdot w_i} = (g^s)^{\sum_{i \in I_Q} w_i \cdot s_i} = g^s$
```

Figure 3: LSSS on Group $\mathbb{G}$

```
Functionality GSS on Group $\mathbb{G}$ using LSSS
$\{S_1, S_2, \cdots, S_{|\mathbb{A}|}\} \leftarrow \mathsf{GrpGSSShare}(S, \mathbb{A})$ :
    $\{S_1, S_2, \cdots, S_{|\mathbb{A}|}\} \leftarrow \mathsf{GrpLSSSShare}(S, \mathbb{A})$
$(S) \leftarrow \mathsf{GrpGSSRecon}(\mathbb{A}, Q)$ :
    $S \leftarrow \mathsf{GrpLSSSRecon}(\mathbb{A}, Q)$
```

Figure 4: GSS on Group $\mathbb{G}$ based on LSSS

Note: The GSS based on Shamir SS requires polynomial evaluation in the secret sharing and Lagrange interpolation in the secret recovery; The GSS based on LSSS needs to handle matrix multiplication both in secret sharing and recovery. Security proofs of GSS schemes are discussed in Appendix C.

5 Publicly Verifiable Generalized Secret Sharing (PVGSS)

5.1 Definition of PVGSS

Definition 7 (PVGSS) *Publicly verifiable generalized secret sharing (PVGSS) scheme allows a dealer to split a secret using arbitrary monotone access structure in a publicly verifiable manner. Hence, shares are encrypted and can be delivered in a public communication channel. Besides, the correctness of the encrypted shares can be verified by any external party. The PVGSS scheme includes following five algorithms.*

- $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{PVGSSSetup}(\lambda)$: *Inputs the system security parameter λ, outputs a key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ for each shareholder P_i.*

- $(\{C_i\}, \mathsf{prf}_s) \leftarrow \mathsf{PVGSSShare}(s, \mathbb{A}, \{pk_i\})$*: Inputs a value s, an access structure $\mathbb{A}$ and the public keys $\{\mathsf{pk}_i\}$, outputs the encrypted shares $\{C_i\}$ and the corresponding NIZK proofs* prf_s.
- $(0/1) \leftarrow \mathsf{PVGSSVerify}(\{C_i\}, \mathsf{prf}_s, \mathbb{A})$*: Inputs the encrypted shares $\{C_i\}$, NIZK proofs* prf_s *and the access structure $\mathbb{A}$, outputs 1 if the shares are correctly encrypted. Otherwise, outputs 0.*
- $(S_i) \leftarrow \mathsf{PVGSSPreRecon}(C_i, \mathsf{sk}_i)$*: Inputs the ciphertext C_i and the corresponding secret key* sk_i*, outputs a decrypted share S_i.*
- $(0/1) \leftarrow \mathsf{PVGSSKeyVrf}(C_i, S_i, pk_i)$*: Inputs the encrypted share C_i, the decrypted share S_i and the public key pk_i, outputs 1 if S_i is corrected decrypted from C_i.*
- $(S) \leftarrow \mathsf{PVGSSRecon}(\mathbb{A}, Q, \{C_i\})$*: Inputs the access structure $\mathbb{A}$ and a set of decrypted shares Q, outputs the recovered secret $S (= g^s)$.*

PVGSS scheme is required to achieve below security properties.

• **Correctness**: The scheme ensures that if the secret sharing process is correctly followed, any authorized subset of participants can reconstruct the secret S.

• **Verifiability**: Anyone can verify that the encrypted shares provided by the dealer are correct without knowing the actual secret. Besides, users can also check correctness of the decrypted shares in the reconstruction phase.

• **IND1-secrecy**: The secret remains hidden from unauthorized parties prior to the reconstruction phase.

Definition 8 (Correctness) *The PVGSS is correct if for all authorized set Q of $\mathbb{A}$, each s in the secret space,*

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

Definition 9 (Verifiability of Encrypted Shares) *The PVGSS satisfies verifiability of encrypted shares if for every PPT $\mathcal{A}$,*

$$\begin{aligned} &Pr[\mathsf{PVGSSVerify}(\{C_i\}, \mathsf{prf}_s, \mathbb{A}) = 1 \wedge \\ &\quad \nexists s \in S \text{ s.t. } (\{C_i\}_{i \in [n]}, *) \leftarrow \mathsf{PVGSSShare}(s, \mathbb{A}, \{pk_i\}) \mid \\ &\quad (\cdot, \{pk_i\}) \leftarrow \mathsf{PVGSSSetup}(\lambda), (\{C_i\}, \mathsf{prf}_s) \leftarrow \mathcal{A}((pk_i, \mathbb{A})] \leq negl(\lambda) \end{aligned}$$

Definition 10 (Verifiability of Share Decryption) *The PVGSS satisfies verifiability of share decryption if for every PPT $\mathcal{A}$,*

$$\begin{aligned} &Pr[\mathsf{PVGSSKeyVrf}(C, S, pk) = 1 \wedge \\ &\quad \nexists sk \in SK \text{ s.t. } S \leftarrow \mathsf{PVGSSPreRecon}(C, sk) \mid \\ &\quad (\cdot, \{pk_i\}) \leftarrow \mathsf{PVGSSSetup}(\lambda), (pk, C, S) \leftarrow \mathcal{A}(\{pk_i\}] \leq negl(\lambda) \end{aligned}$$

Definition 11 (IND1-secrecy) ***IND1-secrecy*** *standards for indistinguishability of secrets. PVGSS achieves* ***IND1-secrecy*** *if for any polynomial time adversary $\mathcal{A}$ without an authorized set, $\mathcal{A}$ win the following game played against a challenger with negligible advantage.*

- *Setup: The challenger initiates the* $\mathsf{PVGSSSetup}$ *algorithm of the PVGSS protocol as the dealer and transmits all public information to $\mathcal{A}$. Additionally, it generates both secret and public keys for all honest shareholders and sends the corresponding public keys to $\mathcal{A}$.*
- *Corrupt: $\mathcal{A}$ generates secret keys for the corrupted shareholders and forwards the respective public keys to the challenger.*
- *Challenge: The challenger randomly selects values x_0 and x_1 from the secret space. Further, it randomly selects $b \in \{0, 1\}$. The distribution phase of the protocol is then executed with x_0 as the secret. The challenger sends all public information produced in this phase, along with x_b, to $\mathcal{A}$.*
- *Output: $\mathcal{A}$ then outputs a guess $b' \in \{0, 1\}$.*

The $\mathcal{A}$'s advantage in the game is defined as $|Pr[b = b'] - \frac{1}{2}|$.

5.2 PVGSS Construction Based on GSS

As is known, 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. Note that we leverage the original GSS scheme in the sharing phase and employ GSS on group $\mathbb{G}$ in the reconstruction phase. Refer to Appendix B for an illustrative depiction of our concept from SS to PVGSS. Figure 5 demonstrates a PVGSS construction by taking advantage of GSS scheme and NIZK proofs. The NIZK proofs are acquired by incorporating Σ protocol [35] and Fiat-Shamir heuristic [36].

In the $\mathsf{PVGSSSetup}$ algorithm, each shareholder j 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 $\mathsf{PVGSSShare}$ algorithm, the dealer first invokes $\mathsf{GSSShare}$ algorithm to generate secret shares based on the access structure $\mathbb{A}$. Denote $|\mathbb{A}|$ as the number of leaf nodes of $\mathbb{A}$. Then, it hides the $|\mathbb{A}|$ shares directly using the targeted shareholder's public key, i.e., pk^{s_i}. Next, it executes the Σ protocol and Fiat-Shamir heuristic to generate NIZK proofs. Specifically, $\{C_i'\}$, c, $\{\hat{s}_i\}$ are the commitment, the challenge value and the response value, respectively. Therefore, the NIZK proof prf_s is $(\{C_i'\}, c, \{\hat{s}_i\})$. 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.

```
Functionality PVGSS Based on GSS
$(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{PVGSSSetup}(\lambda)$ :
    $\mathsf{sk}_i \xleftarrow{R} \mathbb{Z}_q$
    $\mathsf{pk}_i \leftarrow g^{\mathsf{sk}_i}$
$(\{C_i\}, \mathsf{prf}_s) \leftarrow \mathsf{PVGSSShare}(s, \mathbb{A}, \{\mathsf{pk}_i\})$ :
    $\{s_1, s_2, \cdots, s_{|\mathbb{A}|}\} \leftarrow \mathsf{GSSShare}(s, \mathbb{A})$
    $C_i = \mathsf{pk}_i^{s_i} \ \forall i \in [1, \ldots, |\mathbb{A}|]$
    $s' \xleftarrow{R} \mathbb{Z}_q$
    $(s'_1, \ldots, s'_{|\mathbb{A}|}) = \mathsf{GSSShare}(s', \mathbb{A})$
    $C'_i = \mathsf{pk}_i^{s'_i} \ \forall i \in [1, \ldots, |\mathbb{A}|]$
    $c \leftarrow H(\{C_i\}, \{C'_i\})$
    $\hat{s} \leftarrow s' - c \cdot s$
    $\hat{s}_i \leftarrow s'_i - c \cdot s_i \ \forall i \in [1, \ldots, |\mathbb{A}|]$
    $\mathsf{prf}_s \leftarrow (\{C'_i\}, c, \hat{s}, \{\hat{s}_i\})$
$(0/1) \leftarrow \mathsf{PVGSSVerify}(\{C_i\}, \mathsf{prf}_s, \mathbb{A})$ :
    $C'_i \stackrel{?}{=} C_i^c \cdot \mathsf{pk}_i^{\hat{s}_i} \ \forall i \in [1, \ldots, |\mathbb{A}|]$
    $\hat{s} \stackrel{?}{=} \mathsf{GSSRecon}(\mathbb{A}, \{\hat{s}_i\})$.
$(S_i) \leftarrow \mathsf{PVGSSPreRecon}(C_i, \mathsf{sk}_i)$ :
    $S_i = C_i^{1/\mathsf{sk}_i} = g^{s_i}$
$(0/1) \leftarrow \mathsf{PVGSSKeyVrf}(C_i, S_i, \mathsf{pk}_i)$ :
    $e(S_i, \mathsf{pk}_i) \stackrel{?}{=} e(C_i, g)$
$(S) \leftarrow \mathsf{PVGSSRecon}(\mathbb{A}, Q, \{C_i\})$ :
    $S \leftarrow \mathsf{GrpGSSRecon}(\mathbb{A}, Q)$ where $S = g^s$ and $Q \subseteq$
    $\{S_1, \cdots, S_{|\mathbb{A}|}\}$ is an authorized set of $\mathbb{A}$.
```

Figure 5: PVGSS scheme based on GSS and NIZK proofs

In the PVGSSVerify algorithm, any (external) party can check whether the dealer has honestly performed the sharing phase with two equations. The first equation is identical to the Schnorr signature [28], which allows the dealer to prove knowledge of $\{s_i\}$. The second equation implies that all the response values $\{\hat{s}_i\}$ are correctly generated using the challenge value c. Refer to Theorem 1 for details about the correctness.

In the PVGSSPreRecon algorithm, each shareholder P_i decrypts C_i with its secret key sk_i and obtains $S_i = g^{s_i}$. Note that original secret share s_i is still kept unknown.

In the PVGSSRecon algorithm, the user collects a set $Q = \{S_i\}$ and checks the correctness of each S_i. Given enough authorized decrypted shares, i.e., an authorized set Q, the user can reconstruct a secret S using GrpGSSRecon algorithm.

Computational Complexity: The PVGSSShare algorithm costs $2|\mathbb{A}|$ group exponentiations to calculate C_i and C'_i. The PVGSSVerify algorithm requires $|\mathbb{A}|$ exponentiations to verify the correctness of each C_i. The PVGSSPreRecon algorithm costs an exponentiation for each shareholder. The PVGSSVerify algorithm requires 2 bilinear pairing for each S_i. The PVGSSRecon algorithm takes $|I_Q|$ exponentiations, where I_Q denotes an authorized set of the access structure $\mathbb{A}$. Matrix multiplication in the LSSS-based construction, and polynomial evaluation and Lagrange interpolation in the Shamir SS-based construction, are omitted.

Communicational Complexity: In the PVGSS sharing phase, the dealer outputs $2|\mathbb{A}|$ $\mathbb{G}$ elements and $(|\mathbb{A}|+1)$ $\mathbb{Z}$ elements. During the reconstruction phase, $|\mathbb{I}_{\mathbb{Q}}|$ $\mathbb{G}$ elements are collected by the user.

5.3 Security Analysis

Theorem 1 (Correctness) *The PVGSS construction satisfies correctness defined by Definition 8.*

Proof. If the dealer is honest and outputs $\{C_i\}$ and prf_s using GSSShare based on the access structure $\mathbb{A}$, then $\mathsf{PVGSSVerify}(\{C_i\}, \mathsf{prf}_s, \mathbb{A}) = 1$. That is implied by Schnorr signature [28], where each $(C_i, c, (C'_i, \hat{s}))$ contained in prf_s comprise of a Σ protocol [28] and GSS correctness. If PVGSSPreRecon is applied to (C_i, sk_i) honestly, then $S_i = g^{s_i}$ and $\mathsf{PVGSSKeyVrf}(C_i, S_i, \mathsf{pk}_i) = 1, \forall i \in I_Q$. Given an authorized set Q of $\mathbb{A}$, we have the reconstructed value $S' \leftarrow \mathsf{GrpGSSRecon}(\mathbb{A}, Q)$ which in turn equals $S = g^s$ also due to GSS correctness.

GrpGSSRecon based on LSSS : $S' = \prod_{i \in I_Q} S_i^{w_i} = g^{\sum_{i \in I_Q} s_i \cdot w_i} = g^s$.

GrpGSSRecon based on Shamir SS: For each node x from leaf to root R: $F_x \leftarrow \mathsf{GrpRecon}(Q_x)$, where $Q_x \subseteq \{F_{x_1}, \cdots, F_{x_{n_x}}\}$ generated at node x and $|Q_x| \geq t_x$. Due to the correctness of Shamir SS, every F_x is correct. Consequently, $S' = F_R = g^s$.

Therefore, the PVGSS satisfies correctness defined by Definition 8.

Theorem 2 (Verifiability of Encrypted Shares) *If NIZK proof with soundness error negligible in λ, then PVGSS has verifiability of encrypted shares, defined by Definition 9.*

Proof. If $\mathsf{PVGSSVerify}(\{C_i\}, \mathsf{prf}_s, \mathbb{A}) = 1$, then except with soundness error $negl(\lambda)$ of NIZK, there exists s_i such that $C_i = \mathsf{pk}_i^{s_i}$ and $C'_i = C_i^c \cdot pk_i^{\hat{s}_i}$. Except with probability $1/q$, $\{\hat{s}_i\}$ are not GSS shares of $\hat{s}$ with $\hat{s} = \mathsf{GSSRecon}(\mathbb{A}, \{\hat{s}_i\})$ holding. Further, $\{s_i\}$ are not GSS shares of s with probability $1/q$, due to homomorphism. Therefore, the PVGSS satisfies verifiability of encrypted shares by Definition 9.

Theorem 3 (Verifiability of Share Decryption) *If bilinearity of bilinear map holds, then PVGSS has verifiability of share decryption.*

Proof. If $\mathsf{PVGSSKeyVrf}(C, S, pk) = 1$, then we have $e(S, \mathsf{pk}) = e(C, g)$. Obviously, $\mathsf{pk} = g^{\mathsf{sk}}$, and we can get following by bilinearity of bilinear map:

$$e(C, g) = e(S, \mathsf{pk}) = e(S, g)^{\mathsf{sk}} = e(S^{\mathsf{sk}}, g)$$

Then, we have $C = S^{\mathsf{sk}}$, thus $\exists sk \in SK$ s.t. $S \leftarrow \mathsf{PVGSSPreRecon}(C, sk)$. Therefore, the PVGSS satisfies verifiability of share decryption defined by Definition 10.

Theorem 4 (IND1-secrecy) *The PVGSS is IND1-secrecy, defined by Definition 11, under DDH assumption.*

Proof. Firstly, the NIZK proofs prf_s of $\mathsf{PVGSSShare}$ reveal nothing about the secret s or shares $\{s_i\}$, due to zero-knowledge property. Then, we argue that, if there exists an adversary $\mathcal{A}_{Priv}$ which can break the **IND1-secrecy** property of PVGSS, then there exists an adversary $\mathcal{A}_{DDH}$ which can use $\mathcal{A}_{Priv}$ to break DDH assumption.

As shown in Figure 5, the expression for PVGSS is the same whether based on Shamir SS or LSSS. For simplicity, we focus on the Shamir SS-based PVGSS, where there exists a node that cannot be satisfied by any unauthorized set, as stated in Corollary 3. At this node, we assume it is not satisfied by the maximum unauthorized set. That means, we can consider $\mathbb{A}$ as a (t,n) threshold access structure, and assume $\mathcal{A}_{Priv}$ can corrupts the $t-1$ first shareholders.

Let $(g, g^{\alpha}, g^{\beta}, g^{\gamma})$ be an instance of the DDH problem. Then $\mathcal{A}_{DDH}$ using $\mathcal{A}_{Priv}$ can simulate an IND game as follows:

- Setup: The challenger sets $h = g^{\alpha}$ and runs $\mathsf{PVGSSSetup}$. For $t \leq i \leq n$, $\mathcal{A}_{DDH}$ samples $r_i \overset{R}{\leftarrow} \mathbb{Z}_q$ and sends the values $\mathsf{pk}_i = g^{r_i}$ to $\mathcal{A}_{Priv}$.

- Corrupt: For $1 \leq i \leq t-1$, $\mathcal{A}_{Priv}$ samples $\mathsf{sk}_i \overset{R}{\leftarrow} \mathbb{Z}_q$ and sets $\mathsf{pk}_i = h^{\mathsf{sk}_i}$ and sends this to the challenger.

- Challenge: For $1 \leq i \leq t-1$, the challenger samples $s_i \overset{R}{\leftarrow} \mathbb{Z}_q$ and sets $C_i = \mathsf{pk}_i^{s_i}$. For $t \leq i \leq n$, the challenger generates g^{s_i} and $p(x)$ is the polynomial of degree at most t determined by $p(i) = s_i$ and $p(0) = \beta$. Note that $\mathcal{A}_{DDH}$ knows g^{β} (but does not know β) and g^{s_i} for $1 \leq i \leq t-1$. So $\mathcal{A}_{DDH}$ can use Lagrange interpolation to compute $g^{s_i} = g^{p(i)}$ for $t \leq i \leq n$ and it also generates shares $C_i = g^{s_i r_i} = \mathsf{pk}_i^{s_i}$. The challenger creates NIZK proof prf_s as the dealer does in the $\mathsf{PVGSSShare}$ algorithm. Finally, the challenger send $(\{C_i\}, \mathsf{prf}_s)$ to $\mathcal{A}_{Priv}$ with g^{γ} as x_b.

- Output: $\mathcal{A}_{Priv}$ make a guess b'.

  If $b' = 0$, $\mathcal{A}_{DDH}$ guess that $\gamma = \alpha \cdot \beta$, if $b' = 1$ $\mathcal{A}_{DDH}$ guess that γ is a random element. The challenge phase information $(\{C_i\}, \mathsf{prf}_s)$ sent to $\mathcal{A}_{Priv}$ is distributed exactly like a sharing of the value $h^{\beta} = g^{\alpha\beta}$. If g^{γ} sent to $\mathcal{A}_{Priv}$ is the secret shared by PVGSS, $\gamma = \alpha \cdot \beta$. So the advantage of $\mathcal{A}_{DDH}$ is the same as the advantage of $\mathcal{A}_{Priv}$.

6 Building a Decentralized Exchange

While centralized exchanges (CEXs), such as Binance [47] and Coinbase [46], offer high liquidity, advanced features, and ease of use, they also come with inherent risks. These include reliance on a central authority, which may lead to vulnerabilities such as security breaches, mismanagement, or censorship. Additionally, users must relinquish control of their private keys, creating potential single points of failure (SPOF). To address these challenges, decentralized exchanges (DEXs) have emerged as an alternative, providing a trustless and transparent trading environment where users retain full control over their assets and transactions are secured through blockchain technology.

Hash time-locked contract (HTLC) [48, 49] and adaptor signature [50, 53] have shown promising opportunity in implementing fair atomic swap [51, 52], which is the primary requirement for designing DEXs. However, these methods have flaws in fostering the DeFi markets, limiting their world-wide application in DEX. There is no straightforward arbitration mechanism, leading to long time waits or risks in the event of disputes or non-cooperation, such as a party refusing to reveal a preimage or signature. Besides, these methods focus primarily on ensuring fairness between the two trading parties through cryptographic mechanisms. [54] They often overlook the broader economic incentives for other participants, such as liquidity providers or arbitrageurs, who play a crucial role in fostering a thriving market ecosystem.

Automated market maker (AMM) based DEXs (such as Uniswap [55] and Curve [56]) eliminate the need for direct counterparties by using liquidity pools and mathematical algorithms to determine token prices [57]. This approach simplifies trading, improves accessibility, and removes the reliance on strict time-locks or preimage disclosures, making it an attractive option for both new and experienced users. Nevertheless, liquidity providers (LPs) have to store token pairs into DEX and face impermanent loss when asset prices in the pool change significantly. High slippage occurs during large trades, making AMMs less efficient for high-value transactions. Vulnerability to front-running and miner extractable value attacks exposes traders to unfair costs [58, 59]. AMMs also suffer from capital inefficiency, requiring large liquidity to reduce slippage, while locking funds with suboptimal returns [60].

Table 2: Comparisons of different exchanges

Exchanges	Fariness provider	Arbitrageurs	No SPOF	Arbitration	No slippage	Divergence Losslessness
CEX	CEX	N/A	✗	✗	✓	✓
HTLC-based	Miners	N/A	✓	✗	✓	✓
AMM-based	Smart contract	LPs	✓	✗	✗	✗
Ours	Smart contract	Watchers	✓	✓	✓	✓

Due to the existing obstacles, we propose a novel DEX built on a transparent and trustless protocol using the proposed PVGSS scheme. Trades are executed on smart contract without manipulation, ensuring equitable outcomes and preventing exploitation by any participant. Our design incentivizes arbitrageurs (wathers) by rewarding them for providing passive arbitration services, ensuring active participation and foster-

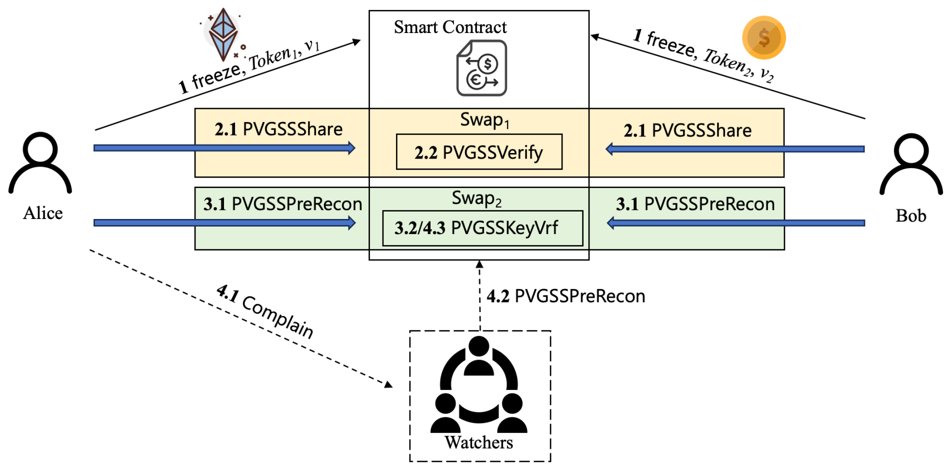


Figure 6: Overview of the DEX between two exchangers (i.e., Alice and Bob) with passive watchers

ing a robust and dynamic marketplace. Unlike AMM-based models, the arbitrageurs in our DEX has no impermanent loss and traders exchange at their expected prices without slippage. Table 2 compares the above mentioned exchanges.

6.1 High-Level Overview

Based on the proposed PVGSS scheme, we design a DEX to allow exchangers to express their willing fairly and simultaneously. In this research, we merely focus on ERC-20 token exchange. The DEX involves two roles: exchangers and watchers. Anyone can be exchangers and watchers. Each exchanger holds specific types of ERC-20 tokens, while multiple watchers collectively form a passive notary committee to address potential disputes.

Take two exchangers, i.e., Alice and Bob, and n watchers as an example. As shown by Figure 6, the DEX optimistically runs in two communication rounds for Alice and Bob. In the first round, swap_1, each exchanger commits to a secret using PVGSSShare, where all the $n+2$ entities are considered as shareholders. The correctness of the commitment is guaranteed by the PVGSSVerify algorithm. Refer to Section 6.2 for the detailed design of access structure. In the second round, swap_2, each exchanger reveals its decrypted share using PVGSSPreRecon. The correctness of share decryption is ensured by PVGSSKeyVrf. Then, both Alice and Bob jointly recover each other's secrets using PVGSSRecon.

In the pessimistic occasion where a player complains to the watchers, who will be involved to resolve dispute using PVGSSPreRecon. Note that the access structure of the DEX not only tolerates a faulty exchanger but also tolerates $n-t$ faulty watchers.

To promote the prosperity of the DEX, an incentive and penalty mechanism of all the entities should be fully considered. The fundamental assumption is that everyone is rational, and everyone acts in pursuit of profit. Hence, we devise different incentive polices for all occasions in Section 6.4.

6.2 Access Structure for the DEX

Suppose the two exchangers are Alice and Bob and the n watchers are $W_1, \cdots, W_n$. Figure 7 depicts the concrete access structure with threshold gates, i.e., $(2\ of\ (Alice, Bob, (t\ of\ (W_1, W_2, \cdots, W_n))))$. Based on Corollary 1, we can infer that the set $[Alice, Bob]$ is an authorized set. Apparently, if both Alice and Bob are honest, they can recover any secrets of PVGSS instance.

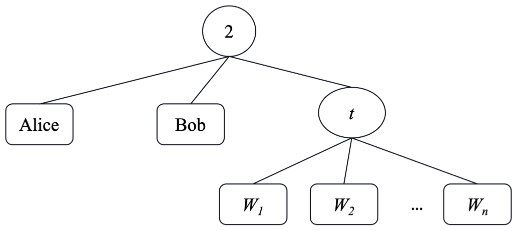


Figure 7: The access structure leveraged by Alice and Bob

Without loss of generality, if Bob is absent in a PVGSS instance, Alice and at least t watchers can collectively recover the secret. This shows the feature of fault tolerance, i.e., the access structure allows fault of one exchanger and $n-t$ watchers. By Lemma 1, we argue that t can be set to 1 without affecting fairness.

Figure 7 shows the simplest access structure in building a DEX. A DEX can, in fact, adopt a more sophisticated access structure, allowing different watchers to hold varying weights or privileges. This enables the application of Game Theory to incentivize profit-driven competition effectively. That will be future work and it is out of discussion in this paper.

6.3 Construction of the DEX

Figure 8 describes the functionality $\mathcal{F}_{DEX}$ defined in smart contract, which interacts with $n+2$ entities (i.e., 2 exchangers and $n \geq 1$ watchers) in session id. For convenience of description, denote the two exchangers as Alice and Bob. Both Alice and Bob leverages the same access structure $\mathbb{A}$, as described by Figure 7. Suppose the n watchers are randomly chosen from the registered arbitrageurs for each id. Two expiration time Δt_1 and Δt_2 are indicators for distinguishing the optimistic and pessimistic cases. Appendix D illustrates the primary data structures and global variables/events in implementing $\mathcal{F}_{DEX}$.

Suppose Alice, Bob and n watchers have registered their public keys via the $\mathsf{register}(\mathsf{id}, \mathsf{pk}_i)$ interface. Without loss of generality, Alice lists v_1 amount of $Token_1$ for sale and expects to exchange v_2 amount of $Token_2$.[2] Alice and Bob freeze their tokens by $\mathsf{freeze}(\mathsf{id}, Token_1, v_1)$ and $\mathsf{freeze}(\mathsf{id}, Token_2, v_2)$, respectively.

Bob invokes $\mathsf{swap}_1(\mathsf{id}, \{C_{iB}\}, \mathsf{prf}_{s_B})$, where $(\{C_{iB}\}, \mathsf{prf}_{s_B}) \leftarrow \mathsf{PVGSSShare}(s_B, \mathbb{A}, \{\mathsf{pk}_i\})$ and $s_B \overset{R}{\leftarrow} \mathbb{Z}_q$. If Bob is honest (i.e., $\{C_{iB}\}$ is stored in smart contract), Alice then sends $\mathsf{swap}_1(\mathsf{id}, \{C_{iA}\}, \mathsf{prf}_{s_A})$ and $\mathsf{swap}_2(\mathsf{id}, S_A)$ simultaneously to $\mathcal{F}_{DEX}$, where $S_A \leftarrow \mathsf{PVGSSPreRecon}(C_A, \mathsf{sk}_A)$. If Alice is honest (i.e., $\{C_{iA}\}$ and S_A are recorded in smart contract), Bob invokes $\mathsf{swap}_2(\mathsf{id}, S_B)$.

To this end, if both exchangers are honest, Alice (or Bob) could recover the Bob (or Alice) secret value hidden in $\{C_{iB}\}$ (or $\{C_{iA}\}$) by $g^{s_B} \leftarrow \mathsf{PVGSSRecon}(\mathbb{A}, (S_A, S_B), \{C_{iB}\})$ (or g^{s_A}). Hence, they reach a consensus on the swap and anyone can invoke $\mathsf{determine}(\mathsf{id})$ of $\mathcal{F}_{DEX}$ to terminate the exchange. Note that though S_A and S_B are published on blockchain and secrets g^{s_A} and g^{s_B} are computable by anyone, the exchange does not reveal individual private keys.

Without loss of generality, we consider Bob is malicious in the pessimistic cases.

- If Bob does not execute swap_1 honestly, Alice will wait to time Δt_2 with failure termination. The complain function in $\mathcal{F}_{DEX}$ guarantees that Bob cannot complain to watchers if he has not invoked swap_1 honestly.

- Or if Bob does not execute swap_2 honestly in Δt_1, Alice can initiate $\mathsf{complain}(\mathsf{id})$ to notify watchers. Each watcher P_i will generate his PVGSS share $S_i \leftarrow \mathsf{PVGSSPreRecon}(C_i, \mathsf{sk}_i)$ and upload it to $\mathcal{F}_{DEX}$ via $\mathsf{swap}_2(\mathsf{id}, S_i)$. With at least t pieces of $\{S_i\}$ collected, Alice can obtain $g^{s_B} \leftarrow \mathsf{PVGSSRecon}(\mathbb{A}, (S_A, \{S_i\}), \{C_{iB}\})$, representing a success termination. In case not enough watchers provide services, the swap will be failure and faulty watchers will be punished.

[2]Obviously, $v_1 \cdot Price_{Token_1} \approx v_2 \cdot Price_{Token_2}$. We assume that exchangers are rational and they can obtain Token prices from other exchanges or websites.

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

For session id, the contract manages two expiration time Δt_1 and $\Delta t_2 > \Delta t_1$, a termination variable *state*.

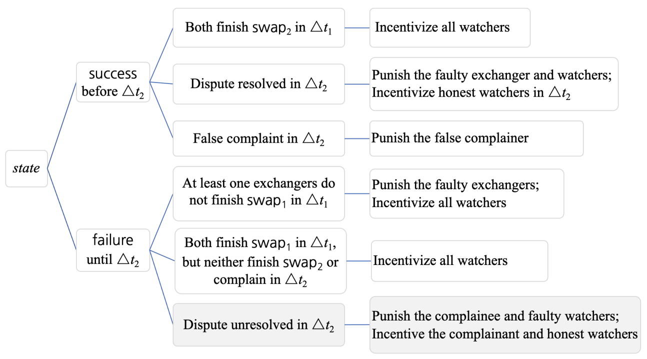
- Upon receiving $(\mathsf{register}, \mathsf{id}, \mathsf{pk}_x)$ from an entity, store the public key pk_x of the entity x.
- Upon receiving $(\mathsf{freeze}, \mathsf{id}, Token, v)$ from Alice or Bob, freeze v token $Token$ in smart contract.
- Upon receiving $(\mathsf{withdraw}, \mathsf{id})$ from Alice or Bob, if $state = \mathsf{failure}$ and current time $ts > \Delta t_2$, transfer the v frozen token $Token$ to the sender. Otherwise, abort.
- Upon receiving $(\mathsf{swap}_1, \mathsf{id}, \{C_i\}, \mathsf{prf}_s)$ from Alice or Bob, invoke $\mathsf{PVGSSVerify}(\{C_i\}, \mathsf{prf}_s, \mathbb{A})$. If it returns 0, abort. Otherwise, store $\{C_i\}$.
- Upon receiving $(\mathsf{swap}_2, \mathsf{id}, S_x)$ from an entity, invoke $\mathsf{PVGSSKeyVrf}(C_x, S_x, \mathsf{pk}_x)$. If it returns 0, abort. Otherwise, store S_x and notify exchangers.
- Upon receiving $(\mathsf{complain}, \mathsf{id})$ from a complainer (i.e., Alice or Bob), if current time $ts > \Delta t_1$, and both Alice and Bob have invoked swap_1 honestly, then notice all watchers. (Each watcher P_i sends the $(\mathsf{swap}_2, \mathsf{id}, S_i)$ message for the complainer.)
- Upon receiving $(\mathsf{determine}, \mathsf{id})$ by anyone, set $state = \mathsf{success}$ and transfer Alice/Bob's token to Bob/Alice' account, if one of below conditions is satisfied before Δt_2:
  - Both Alice and Bob have invoked swap_2 honestly.
  - Either Alice or Bob has invoked swap_2 honestly, and at least t watchers have invoked swap_2 honestly.

  Otherwise, set $state = \mathsf{failure}$ and send back tokens to Alice/Bob's account.

Figure 8: The functionality $\mathcal{F}_{DEX}$ in smart contract

6.4 Incentive Mechanism

Figure 9 presents the mindmap to illustrate all cases of results and the corresponding incentive policy a swap instance. A swap instance ends with either success or failure definitely. Specifically, the success can be determined before Δt_2 and the failure is determined only after Δt_2. All entities are required to make stakes in the system. The number of swap instances a watcher can serve simultaneously is proportional to the amount of its stakes.



state
success before $\triangle t_2$
failure until $\triangle t_2$
Both finish swap_2 in $\triangle t_1$
Incentivize all watchers
Dispute resolved in $\triangle t_2$
Punish the faulty exchanger and watchers; Incentivize honest watchers in $\triangle t_2$
False complaint in $\triangle t_2$
Punish the false complainer
At least one exchangers do not finish swap_1 in $\triangle t_1$
Punish the faulty exchangers; Incentivize all watchers
Both finish swap_1 in $\triangle t_1$, but neither finish swap_2 or complain in $\triangle t_2$
Incentivize all watchers
Dispute unresolved in $\triangle t_2$
Punish the complainee and faulty watchers; Incentive the complainant and honest watchers

Figure 9: The incentive and penalty mechanism

The success result includes three cases. The first case is optimistic scenario where both exchangers honestly invoke swap_1 and swap_2 before Δt_1. In this case, the watchers are not involved and each of them is rewarded with a small tip. The second case shows the scenario where one exchanger does not execute swap_2 before Δt_2. Then, the other exchanger notice the n watchers to resolve the dispute. In this case, faulty exchanger and watchers are punished to reward the honest watchers who send swap_2 before Δt_2. The third case introduces the scenario where some exchanger may propose false complaint. In this case, the complainer will be punished.

The failure results also contains three cases. The first case means at least one exchangers give up in Δt_1. Hence, the faulty exchanger will be punished and all watchers are rewarded. The second case means both exchangers negotiate to abort the swap instance before Δt_2, then all watchers are rewarded. The third case is an unexpected scenario, where honest complaint is raised but unresolved due to insufficient honest watchers. We impose penalties on the complainee and faulty watchers to compensate the complainant and honest watchers.

6.5 Property Analysis

Before describing the achieved properties, we argue the potential threats to the DEX has little impacts.

Lemma 1 (Dos attack) *An exchanger or (even up to n) watchers, who abort the protocol, can not bias the fairness.*

Proof. As described in Section 6.3, if an exchanger aborts without executing swap_1 or more than $n-t$ watchers abort without providing arbitration, the swap instance ends with failure; Otherwise, the instance ends with success. Fairness is guaranteed according to the protocol termination result and only faulty entities will bear specific loss. Thus, it is feasible to set $t=1$ with arbitrary $n \geq 1$ in practice.

Lemma 2 (Collusion/Coercion attack) *An exchanger colluding with (even up to n) malicious watchers cannot break the fairness.*

Proof. The exchanging tokens are locked in the smart contract until termination. Both the exchange protocol and smart contracts are transparent and publicly verifiable. The final *state* of an exchange instance is either success or failure, determined no later than Δt_2. If $state = \mathsf{success}$, the locked tokens are transferred to the counterparties' accounts; otherwise, they are returned to the respective exchangers' accounts. Consequently, even if an exchanger colludes with all n watchers, they cannot gain any advantage over the smart contract. Furthermore, inactive or malicious entities are penalized.

Lemma 3 (Sybil attack) *An arbitrageur cannot gain disproportionate profits through generating fake identities.*

*Proof.*An arbitrageur can be selected as a watcher in proportion to the amount of currency they have deposited. This implies that the nothing-at-stake attack is also eliminated. If a swap instance is ongoing, a proportional amount of deposited currency is locked, rendering a Sybil attack meaningless.

Below, we prove the security properties by Claims.

Claim 1 (Decentralization) *No trusted manager is required to execute the protocol or manage participants.*

Proof. As designed, anyone can register with deposited digital currency as an exchanger or a arbitrageur. Anyone is equal before the smart contract. By Lemma 3, we prove that it is infeasible for arbitrageurs to gain additional profits by generating fake identities.

Claim 2 (Fairness) *Either exchangers acquire the counterpart's token or neither acquires.*

*Proof.*According to the determine functionality in $\mathcal{F}_{DEX}$, the *state* of a swap instance is determined as either success or failure based on the actions of exchangers and watchers. A success results in a successful exchange, while a failure ensures the return of tokens to exchangers, guaranteeing fairness. Additionally, as demonstrated in Lemma 1 and Lemma 2, neither DoS attacks nor collusion attacks affect fairness.

Claim 3 (Termination) *An exchanging instance should be terminated at specific time point.*

Proof. A swap instance can terminate with success at any time before Δt_1 if both exchangers act honestly. The parameter Δt_1 allows exchangers to file complaints through the smart contract. And the complaints should be handled in Δt_2 by the watchers. Consequently, the swap instance is guaranteed to reach a definitive termination by Δt_2, regardless of the presence of faulty entities.

Claim 4 (Arbitration with Optimism and Statelessness) *The watchers can resolve dispute and they are not involved if no disputes arise.*

Proof. As the access structure designs (cf. Section 6.2), if an exchanger aborts in swap_2, at least t watchers can be of help in guaranteeing fairness with success termination. As describe in Section 6.3, the swap instance can normally ends with success if both exchangers are honest and watchers are not involved, showing the optimism of them. Moreover, watchers can take action according to smart contract's status without requiring additional storage for each swap instance, demonstrating their statelessness.

Claim 5 (Accountability) *Dishonest behaviors can be detected and faulty entities is punished.*

Proof. The whole protocol is designed through smart contract and all operations are publicly verifiable by any third party. Specifically, the public verifiability is ensured by Theorem 2 and Theorem 3. Therefore, it is feasible to find out faulty entities and punish them by confiscating a certain percentage of their deposits.

Claim 6 (Incentive Compatibility) *The protocol comprehensively considers interests of exchangers and watchers, forcing them to behave honestly.*

Proof. By Claim 3, we show that the *state* is determined no later than Δt_2. As illustrated in Section 6.4, all possible outcomes for the *state* values and the corresponding incentives are considered. Besides, the protocol considers both exchangers and arbitrageurs, who are accountable by Claim 5. Thus, we can design sophisticated incentive mechanisms to encourage honest behavior and penalize faulty actions, promoting the adoption of the DEX. Figure 9 illustrates the detailed incentive mechanisms.

7 Experimental Evaluation

We implement our PVGSS scheme using Golang (off-chain) and Solidity (on-chain). We implement the threshold-based LSSS matrix in Golang using the method provided by [70]. To make on-chain and off-chain operations compatible, we choose BN128 as the asymmetric elliptic curve group, which is the official supported by Ethereum. The on-chain exponentiation and pairing cost are about 40000[3] and $80000 * k + 100000$[4], respectively, where k is the number of pairing points. All the our benchmarks of PVGSS schemes are executed on a Intel (R) Core (TM) i7-11800H @2.30GHz with 4GB RAM running Ubuntu 22.04.5 LTS and Golang1.22.0. The source code of contract is deployed and available on Ethereum Sepolia testnet[5].

We choose the access structure designed for the DEX (cf. Figure 7) to evaluate the performance of the PVGSS scheme, where $t = n/2 + 1$ and n ranges from 100 to 1000. Figure 10-13 show the off-chain computational overheads of the PVGSS schemes. The cost of $\mathsf{PVGSSShare}$ and $\mathsf{PVGSSRecon}$ are super-linear, due to polynomial evaluation and Lagrange interpolation in Shamir SS-based approach and matrix multiplication in LSSS-based approach. Besides, the $\mathsf{PVGSSPreRecon}$ algorithm costs about 8.2ms. It can be seen all algorithms can be calculated within seconds given $n = 1000$, indicating practicality of the PVGSS schemes in large-scale systems.

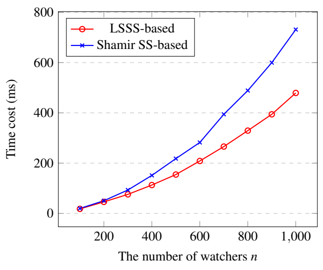


Figure 10: Cost of $\mathsf{PVGSSShare}$

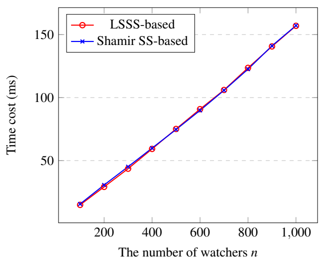


Figure 11: Cost of $\mathsf{PVGSSVerify}$

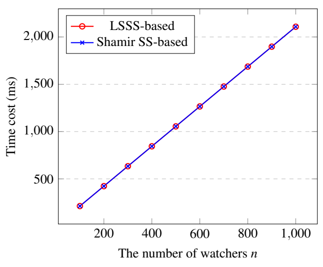


Figure 12: Cost of n times of $\mathsf{PVGSSKeyVrf}$

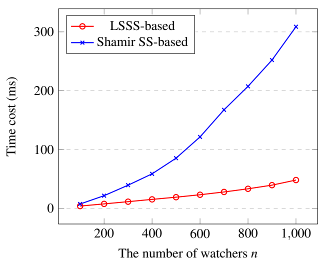


Figure 13: $\mathsf{PVGSSRecon}$ with an authorized set size $t + 1$

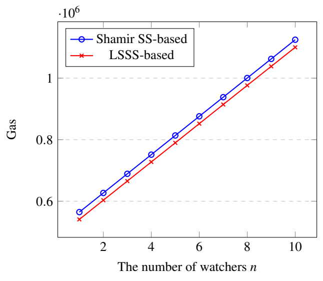


Figure 14: On-chain gas cost of $\mathsf{swap}_1 + \mathsf{swap}_2$

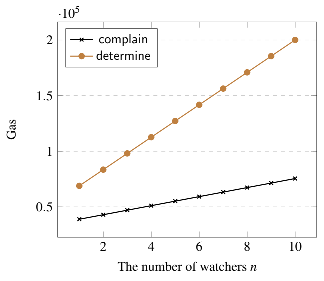


Figure 15: On-chain cost of $\mathsf{complain}$ and $\mathsf{determine}$

Further, we evaluate the gas consumption of the practice. The swap_2, which invokes $\mathsf{PVGSSKeyVrf}$, roughly costs a constant amount gas (i.e., 253784) for each exchanger. Figure 14 depicts the gas consumption of the $\mathsf{swap}_1 + \mathsf{swap}_2$ on smart contract. In swap_1, calculation of $\mathsf{PVGSSVerify}$ and storage of $\{C_i\}$ are included. It can be observed that the

[3] https://eips.ethereum.org/EIPS/eip-196#gas-costs
[4] https://eips.ethereum.org/EIPS/eip-197#gas-costs
[5] https://sepolia.etherscan.io/address/0xfc725a86fe14ee68b645f9d92e832b0d843045ee#code

PVGSSShare based on LSSS is slightly more efficient, as matrix multiplication is somewhat more computationally intensive than Lagrange interpolation in Solidity. Figure 15 shows the gas cost of functionality complain and functionality determine.

As of the time of writing (January 11, 2025), the gas price is 4.49 gwei, and the price of ETH is 3,260 USD. Table 3 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 \$8.3/\$11.9/\$15.6 for $n = 1/5/9$ and $t = 1$. If watchers are involved in the pessimistic case, the complaint cost is less than 1 USD, and each watcher incurs a cost of approximately 3.7 USD (as swap_2 requires).

Table 3: Monetary cost of the on-chain operations

Operation	Gas cost		USD
register	167767		\$2.5
freeze	71250		\$1.0
swap_1	n=1	310787	\$4.5
	n=5	559855	\$8.2
	n=9	808882	\$11.8
swap_2	253784		\$3.7
total, i.e., $\mathsf{swap}_1 + \mathsf{swap}_2$ (Shamir SS-based)	n=1	564571	\$8.3
	n=5	813639	\$11.9
	n=9	1062603	\$15.6
complain	n=1	38894	\$0.6
	n=5	55166	\$0.8
	n=9	71414	\$1.0
determine	n=1	68917	\$1.0
	n=5	127209	\$1.9
	n=9	185499	\$2.7

8 Other Applications

The novel GSS or PVGSS primitive can be applied in both cryptographic and blockchain fields. We list some of the topics, which are of independent interests.

• **Role-based access control (RBAC) [61].** (PV)GSS supports hierarchical RBAC by creating nested or multi-level access structures. Roles higher in the hierarchy can reconstruct secrets using fewer shares, while lower roles require stricter compliance.

• **Attribute-based cryptosystem.** PVGSS can significantly enhance attribute-based cryptosystems [18–20] by ensuring trust, transparency, and robustness in key distribution and access control. In attribute-based cryptosystems, access policies and attributes play a central role in granting decryption capabilities. Integrating PVGSS provides accountability, particularly in settings involving untrusted cloud servers or external authorities. Additionally, PVGSS can strengthen outsourced attribute-based encryption by ensuring the correctness of outsourced computations.

• **Fine-grained MPC.** Participants have equal rights in traditional SS or VSS based MPC protocols, such as distributed key generation [2] and federated learning [4]. With (PV)GSS, it will be possible to investigate MPC protocols with fine-grained access control policy.

• **Blockchain oracle.** Blockchains and smart contracts typically cannot access external information directly, so oracles [62] provide the necessary data feed, such as price feeds, weather data, or other real-world events. PVGSS-based solution can enhance decentralization and public verifiability.

• **Layer 2 (Bridge [63], Relay network [64] or Channel [65]).** Layer 2 [66] refers to a set of technologies built on top of a blockchain's base layer to address scalability issues, increase transaction throughput, and reduce costs without compromising the security and decentralization. Essentially, Layer 2 solutions help blockchain networks scale by processing transactions off-chain. PVGSS-based approaches can be incorporated to enhance trustworthiness and traceability.

• **On-chain secret escrow.** PVGSS can be effectively used in on-chain secret escrow systems [67–69] to provide secure and verifiable management of secrets while ensuring transparency and trust. In such systems, PVSS allows participants to securely share a secret (e.g., a decryption key, private key, or access credentials) without a trusted third party, ensuring that the secret can only be accessed under specified conditions.

9 Conclusion

We begin by introducing the concept of publicly verifiable generalized secret sharing (PVGSS) and propose two constructions: one based on Shamir secret sharing and the other on linear secret sharing schemes (LSSS). Furthermore, we design a decentralized exchange (DEX) leveraging the PVGSS schemes, fully integrated incentive mechanisms. Comprehensive experiments and detailed analysis demonstrate the feasibility of the DEX in real-world applications. Looking ahead, we plan to further explore (PV)GSS-related research. First, we aim to implement additional applications discussed in Section 8 and beyond. Second, we intend to formalize verifiable generalized secret sharing (VGSS, akin to VSS [14]) and examine its constructions. Lastly, we will explore variations of (PV)GSS schemes, such as proactive (PV)GSS [71] and q-PVGSS, inspired by [11].

Open Science

This research is committed to the principles of Open Science, promoting transparency, accessibility, and collaboration. All data, methodologies, and research findings will be made publicly available to foster reproducibility and further academic inquiry. The smart contract code which keeps anonymity for reviews has been deployed on the Ethereum Sepolia testnet. By adopting open science practices, we aim to enhance the integrity of research, increase knowledge dissemination, and contribute to the advancement of science in a more inclusive and equitable manner.

Ethical Considerations

This research adhered to ethical guidelines to ensure the rights, privacy, and well-being of all participants. Conflicts of interest were disclosed, and we acknowledge the contributions of prior works through proper citation. Our research aims to have a positive societal impact while adhering to the highest standards of academic integrity and responsibility.

References

[1] Shamir, A. How to share a secret. *Comm. of the ACM*, 1979, 22(11), 612-613.

[2] 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.

[3] Sahai, A., and Waters, B. Fuzzy identity-based encryption. In *Eurocrypt*, 2005, pp. 457-473.

[4] Rathee, M., Shen, C., Wagh, S. and Popa, R.A. Elsa: Secure aggregation for federated learning with malicious actors. In *S&P*, 2023, pp. 1961-1979.

[5] Dauterman, E., Rathee, M., Popa, R.A. and Stoica, I. Waldo: A private time-series database from function secret sharing. In *S&P*, 2022, pp. 2450-2468.

[6] Feneuil, T. and Rivain, M. Threshold linear secret sharing to the rescue of MPC-in-the-head. In *Asiacrypt*, 2023, pp. 441-473.

[7] Gao, Y., Lu, Y., Lu, Z., Tang, Q., Xu, J. and Zhang, Z. Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency. In *CCS*, 2022, pp. 1187-1201.

[8] Karchmer, M. and Wigderson, A. On span programs. In *Annual Structure in Complexity Theory Conference*, 1993, pp. 102-111.

[9] Feldman, P. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, 1987, pp. 427-438.

[10] 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.

[11] Cascudo, I., and David, B. Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. In *Eurocrypt*, 2024, pp. 216-248.

[12] Cascudo, I., and David, B. SCRAPE: Scalable randomness attested by public entities. In *ACNS*, 2017, pp. 537-556.

[13] Kate, A., Zaverucha, G.M. and Goldberg, I., Constant-size commitments to polynomials and their applications. In *Asiacrypt*, 2010, pp. 177-194.

[14] 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.

[15] Stadler, M. Publicly verifiable secret sharing. In *Eurocrypt*, 1996, pp. 190-199.

[16] Ito, M., Saito, A. and Nishizeki, T. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan*, 1989, 72(9), 56-64.

[17] Benaloh, J., and Leichter, J. Generalized secret sharing and monotone functions. In *Crypto*, 1990.

[18] Bethencourt, J., Sahai, A., and Waters, B. Ciphertext-Policy Attribute-Based Encryption. In *S&P*, 2007, pp. 321-334.

[19] Lewko, A., and Waters, B. Decentralizing attribute-based encryption. In *Eurocrypt*, 2011, pp. 568-588.

[20] Okamoto, T. and Takashima, K. Decentralized attribute-based signatures. In *PKC*, 2013, pp. 125-142.

[21] Garg, R., Lu, G., Waters, B. and Wu, D.J. Reducing the CRS size in registered ABE systems. In *Crypto*, 2024, pp. 143-177.

[22] Zamyatin, A., Harz, D., Lind, J., Panayiotou, P., Gervais, A. and Knottenbelt, W. Xclaim: Trustless, interoperable, cryptocurrency-backed assets. In *S&P*, pp. 193-210.

[23] Dunphy, P., Garratt, L. and Petitcolas, F. Decentralizing digital identity: Open challenges for distributed ledgers. In *EuroS&PW*, 2018, pp. 75-78.

[24] Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z. and Song, D., 2007, October. Provable data possession at untrusted stores. In *CCS*, 2007, pp. 598-609.

[25] Scafuro, A., Siniscalchi, L. and Visconti, I. Publicly verifiable proofs from blockchains. In *PKC*, 2019, pp. 374-401.

[26] Szydlo, M., 2004. Merkle tree traversal in log space and time. In *Eurocrypt*, 2004, pp. 541-554.

[27] Kemmoe, V.Y. and Lysyanskaya, A., 2024, December. RSA-Based Dynamic Accumulator without Hashing into Primes. In *CCS*, 2024, pp. 4271-4285.

[28] Schnorr, C.P., 1990. Efficient identification and signatures for smart cards. In *Crypto*, 1990, pp. 239-252.

[29] Setty, S. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Crypto*, 2020, pp. 704-737.

[30] Drăgan, C.C. and Ţiplea, F.L., . Distributive weighted threshold secret sharing schemes. *Information sciences*, 2016, 339, 85-97.

[31] Beimel, A., Tassa, T., and Weinreb, E. Characterizing ideal weighted threshold secret sharing. In *TCC*, 2005, pp. 600-619.

[32] Chen, Q., Tang, C. and Lin, Z. Efficient explicit constructions of multipartite secret sharing schemes. *IEEE TIT*, 2021, 68(1), 601-631.

[33] Tassa, T., and Dyn, N. Multipartite secret sharing by bivariate interpolation. *JoC*, 2009, 22(2), 227-258.

[34] Simmons, G.J., 1988, August. How to (really) share a secret. In *Eurocrypt*, 1988, pp. 390-448.

[35] Damgård, I. On Σ-protocols. *Lecture Notes, University of Aarhus, Department for Computer Science*, 2002.

[36] Fiat, A., and Shamir, A. How to prove yourself: Practical solutions to identification and signature problems. In *Eurocrypt*, 1986, pp. 186-194.

[37] 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.

[38] Tassa, T. Hierarchical threshold secret sharing. *JoC*, 2007, 20(2), 237-264.

[39] Chattopadhyay, A.K., Saha, S., Nag, A. and Nandi, S. Secret sharing: A comprehensive survey, taxonomy and applications. *Computer Science Review*, 2024, 51, 100608.

[40] Rouselakis, Y., and Waters, B. Efficient statically-secure large-universe multi-authority attribute-based encryption. In *FC*, 2015, pp. 315-332.

[41] 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.

[42] Cascudo, I., and David, B. ALBATROSS: publicly attestable batched randomness based on secret sharing. In *Asiacrypt*, 2020, pp. 311-341.

[43] 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.

[44] Schoenmakers, B. A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic. In *Crypto*, 1999, pp. 148-164.

[45] Ruiz, A., and Villar, J. L. Publicly verifiable secret sharing from Paillier's cryptosystem. In *SAC*, 2005.

[46] Coinbase, https://www.coinbase.com

[47] Binance, https://www.binance.com/en

[48] Hash Time Locked Contracts, https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts.

[49] Tsabary, I., Yechieli, M., Manuskin, A. and Eyal, I. MAD-HTLC: because HTLC is crazy-cheap to attack. In *S&P*, 2021, pp. 1230-1248.

[50] Erwig, A., Faust, S., Hostáková, K., Maitra, M. and Riahi, S. Two-party adaptor signatures from identification schemes. In *PKC*, 2021, pp. 451-480.

[51] 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.

[52] Zhang, L., Kan, H., Qiu, F. and Hao, F. A Publicly Verifiable Optimistic Fair Exchange Protocol Using Decentralized CP-ABE. *CJ*, 2024, 67(3), pp.1017-1029.

[53] 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.

[54] Augusto, A., Belchior, R., Correia, M., Vasconcelos, A., Zhang, L. and Hardjono, T. Sok: Security and privacy of blockchain interoperability. In *S&P*, 2024, pp. 3840-3865.

[55] Uniswap, https://app.uniswap.org.

[56] Curve, https://curve.fi.

[57] 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), 1-50.

[58] 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.

[59] Baum, C., David, B. and Frederiksen, T.K. P2DEX: privacy-preserving decentralized cryptocurrency exchange. In *ACNS*, 2021, pp. 163-194.

[60] Heimbach, L., Schertenleib, E. and Wattenhofer, R. Risks and returns of uniswap v3 liquidity providers. In *AFT*, 2022, pp. 89-101.

[61] Zhu, Y., Ahn, G.J., Hu, H., Ma, D. and Wang, S. Role-based cryptosystem: A new cryptographic RBAC system based on role-key hierarchy. *IEEE TIFS*, 2013, 8(12), 2138-2153.

[62] Park, S., Bastani, O. and Kim, T. ACon2: Adaptive Conformal Consensus for Provable Blockchain Oracles. In *USENIX Security*, 2023, pp. 3313-3330.

[63] 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.

[64] Li, Y., Liu, H. and Tan, Y. POLYBRIDGE: A crosschain bridge for heterogeneous blockchains. In *ICBC*, 2022, pp. 1-2.

[65] Dziembowski, S., Faust, S. and Hostáková, K. General state channel networks. In *CCS*, 2018, pp. 949-966.

[66] Gangwal, A., Gangavalli, H.R. and Thirupathi, A. A survey of Layer-two blockchain protocols. *Journal of Network and Computer Applications*, 2023, 209, 103539.

[67] 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.

[68] Goyal, V., Kothapalli, A., Masserova, E., Parno, B. and Song, Y. Storing and retrieving secrets on a blockchain. In *PKC*, 2022, pp. 252-282.

[69] Cerulli, A., Connolly, A., Neven, G., Preiss, F.S. and Shoup, V. vetkeys: How a blockchain can keep many secrets. *Cryptology ePrint Archive.* 2023.

[70] Liu, Z., Cao, Z. and Wong, D.S. Efficient generation of linear secret sharing scheme matrices from threshold access trees. *Cryptology ePrint Archive.* 2010

[71] Shoup, V. and Smart, N.P. Lightweight asynchronous verifiable secret sharing with optimal resilience. *JoC*, 2024, 37(3), p.27.

A Access Structure Examples

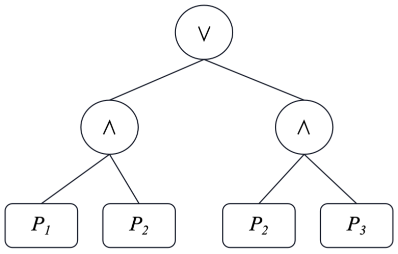


Figure 16: An example access structure using boolean formula, where $\{P_1,P_2\}$ and $\{P_2,P_3\}$ are authorized sets.

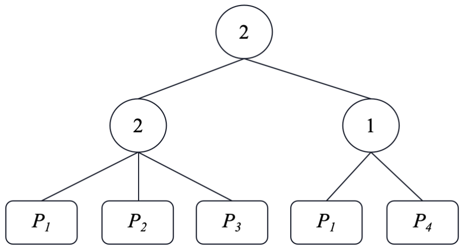


Figure 17: An example access structure using threshold gate, where $\{P_1,P_2\}$ and $\{P_2,P_3,P_4\}$ are authorized sets.

B Illustrative Overview of PVGSS

As described in Section 4, LSSS (on group $\mathbb{G}$) can be directly used as GSS (on group $\mathbb{G}$) scheme. Also, GSS (on group $\mathbb{G}$) can be constructed by recursively leveraging Shamir SS (on group $\mathbb{G}$). To generate NIZK proofs of knowledge of shares $s_i \in \mathbb{Z}$, we employ GSS in the sharing phase for PVGSS. Similar to most PVSS schemes [12, 41, 42], we recover a secret $S \in \mathbb{G}$, where GSS on group $\mathbb{G}$ is applied. Therefore, we derive two PVGSS constructions: one based on Shamir's SS and the other on LSSS. Figure 18 illustrate our concept vividly.

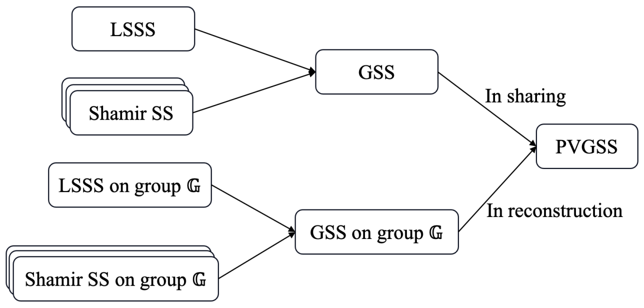

LSSS
Shamir SS
GSS
In sharing
PVGSS
LSSS on group $\mathbb{G}$
In reconstruction
GSS on group $\mathbb{G}$
Shamir SS on group $\mathbb{G}$

Figure 18: High-level overview of our PVGSS construction

C Security Proof of GSS Schemes

We first depict the security properties of GSS scheme, then argue the constructions in Section 4 satisfy the properties.

• **Correctness**: If the secret sharing process is correctly followed, any authorized set of participants can reconstruct the secret.

• **Secrecy**: Nothing information about the secret is revealed, given any unauthorized set of participants.

GSS based on LSSS: As the scheme is built directly on LSSS, the **correctness** and **secrecy** are self-evident.

Shamir SS scheme on Group $\mathbb{G}$:

- **Correctness**: The shares are $S_i = S \cdot g^{f(i)}$, $i \in [0,n]$. Following the correctness of Shamir SS scheme, inputs 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.

GSS on group $\mathbb{G}$ *based on Shamir SS*: For any node in GSS scheme on $\mathbb{G}$, the **correctness** and **secrecy** can be derived directly from those of Shamir SS scheme on $\mathbb{G}$. Thus the **correctness** and **secrecy** of GSS scheme on group $\mathbb{G}$ are satisfied.

LSSS on Group $\mathbb{G}$:

- **Correctness**: The shares are $S_i = S^{s_i} = g^{s \cdot s_i}$ for $i \in [1, \cdots |\mathbb{A}|]$. Q is a authorized set with $I_Q = \{i \in [m] : \rho(i) \in Q\}$ and $M_Q \in \mathbb{Z}_N^{|I_Q| \times l}$, which means there exists $w_Q \in \mathbb{Z}_N^{|I_Q|}$ 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 $S = \prod_{i\in I_Q} S_i^{w_i} = \prod_{i\in I_Q} g^{s \cdot s_i \cdot w_i} = (g^s)^{\sum_{i\in I_Q} w_i \cdot s_i} = g^s$ is correct.
- **Secrecy**: If unauthorized set Q can reconstruct $g^s = \prod_{i\in I_Q} S_i^{w_i}$, then it can find $w_Q \in \mathbb{Z}_N^{|I_Q|}$ 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 and reconstruct.

GSS on Group $\mathbb{G}$ *using LSSS*: As the scheme is built directly on LSSS on $\mathbb{G}$, the **correctness** and **secrecy** are inherently ensured.

D Structs and Global States in $\mathcal{F}_{DEX}$

```
struct Order {
    address seller;    //Order creator
    address tokenSell; // Token to sell (e.g., ETH)
    uint256 amountSell; // Amount to sell (e.g., 2 ETH)
    address tokenBuy; // Token to buy (e.g., USDT)
    uint256 amountBuy; // Amount to buy (e.g., 7000 USDT)
    bool isActive;     // Order state
}
// Store orders
mapping(uint256 => Order) public orders;

// values to track session state
enum SessionState { Active, halfSwap1, finishSwap1, halfSwap2, Complain, Success, Failure }
struct Session {
    SessionState state; // Session state
    address[] exchangers; // seller as exchanger[0], buyer as exchanger[1] in the session
    address[] watchers; // Watchers in the session
    mapping(address => G1Point) Cshares1; //shares from seller
    mapping(address => G1Point) Cshares2; //shares from buyer
    mapping(address => G1Point) shares;   //decrypted shares
    uint256 t1; // Expiration time t1
    uint256 t2; // Expiration time t2
    bool[2] seller_flag; // swap flag of seller
    bool[2] buyer_flag;  // swap flag of buyer
    mapping(address => bool) watcher_flag; //submit flag of watcher
}
// Store sessions
mapping(uint256 => Session) public sessions;

//Events in the smart contract
event TokensReceived(address token, address from, uint256 amount);
event TokensFrozen(address token, address from, uint256 amount, uint256 sessionId);
event TokensSwapped(address token, address from, uint256 amount, uint256 sessionId);
event ComplaintFired(address complainer, uint256 sessionId);
event SessionStateUpdated(uint256 sessionId, SessionState state);
event UserNotified(uint256 sessionId, address user);
event OrderCreated(uint256 orderId, address seller, address tokenSell, uint256 amountSell,
    address tokenBuy, uint256 amountBuy);
event SessionCreated(uint256 orderId, address seller, address buyer, address[] watchers,
    uint256 t1, uint256 t2);
event Incentivized(address user, uint256 amount);
event Penalized(address user, uint256 amount);
```