Traceable Verifiable Secret Sharing and Applications

Karim Baghery[1], Ehsan Ebrahimi[2],
Omid Mirzamohammadi[1], and Mahdi Sedaghat[1,3]

[1] COSIC, KU Leuven, Leuven, Belgium
[2] University of Luxembourg, Luxembourg
[3] Soundness Labs
karim.baghery@kuleuven.be, eebrahimi.pqc@gmail.com,
omid.mirzamohammadi@esat.kuleuven.be, ssedagha@esat.kuleuven.be

February 21, 2025

Abstract. A secret sharing scheme allows a trusted dealer to divide a secret among multiple parties so that a sufficient number of them can recover the secret, while a smaller group cannot. In CRYPTO'21, Goyal, Song, and Srinivasan introduced Traceable Secret Sharing (TSS), which enhances traditional secret sharing by enabling the identification of parties involved in secret reconstruction, deterring malicious behavior like selling shares. Recently, Boneh, Partap, and Rotem (CRYPTO'24) presented two more efficient TSS schemes. However, these existing TSS schemes assume that all distributed shares are valid and shareholders act honestly during the secret reconstruction phase. In this paper, we introduce Traceable Verifiable Secret Sharing (TVSS), a concept designed to ensure both traceability and verifiability in the face of malicious actions by either the dealer or shareholders. We propose a general strategy for transforming a Shamir-based, computationally secure Verifiable Secret Sharing (VSS) scheme into an efficient TVSS scheme. Building on this strategy, we construct two practical TVSS schemes in the honest-majority setting, based on well-known VSS schemes proposed by Feldman (SFCS'87) and Pedersen (CRYPTO'91). Our proposed TVSS schemes retain public shareholder indexes, enhancing flexibility in designing accountable threshold protocols (e.g., Distributed Key Generation protocols) using TVSS. Compared to the original VSS schemes, the individual share size in the new TVSS schemes increases by only a single field element and is just two or three times the size of the main secret. Motivated by a recent study on Accountable Threshold Cryptosystems (ATCs) by Boneh, Partap, and Rotem (CRYPTO'24), and by leveraging our proposed Feldman-based TVSS scheme, we also introduce an efficient ATC based on ElGamal cryptosystem. This new ATC enables a tracer to uniquely identify the parties involved in the decryption process while introducing minimal overhead to existing actively secure (and/or robust) threshold protocols built on the ElGamal cryptosystem.

Keywords: Verifiable Secret Sharing · Traceable Secret Sharing · Traceable Verifiable Secret Sharing · Shamir Secret Sharing

1 Introduction

An (n, t)-secret sharing scheme [26] allows a trusted dealer to distribute a secret among n parties such that any $t+1$ or more parties can recover the secret, while up to t parties learns nothing about the secret. These schemes are fundamental in cryptography and have numerous applications in multi-party computation and threshold cryptography, such as threshold decryption [11,12], threshold signatures [10,28], and threshold verifiable unpredictable functions [21]. They also enable distributed schemes, such as e-voting systems and anonymous credentials.

In this study, we focus exclusively on the well-known Shamir's Secret Sharing (SSS) protocol [26] and its verifiable variants, as it is currently a widely used secret sharing scheme. In a standard (n, t)-SSS, a trusted dealer shares a secret among n parties using a secret degree-t polynomial, with each party receiving a unique evaluation of the polynomial as their individual share of the main secret. Later, if needed, any $t+1$ shares can collectively reconstruct the secret polynomial and, consequently, the main secret or perform computations that require the main secret. However, standard SSS has two limitations: first, it does not protect against malicious dealers who may distribute invalid shares, nor against malicious shareholders who could provide invalid shares during the reconstruction process. Second, it fails to ensure accountability during the reconstruction phase and does not safeguard against untrustworthy shareholders who could be bribed to sell their shares without fear of being identified. To overcome these challenges, various extensions and modifications have been introduced to improve secret sharing schemes, particularly SSS. Verifiable Secret Sharing (VSS) protocols have been proposed to address the first issue, allowing parties to verify the shares received from the dealer as well as those broadcast by other parties during the secret reconstruction phase [2,9,15,24]. To tackle the second issue, recently Traceable Secret Sharing (TSS) has been developed which allows for tracing the shareholders involved in the reconstruction of the secret [6,19].

1.1 Verifiable Secret Sharing

The initial Verifiable Secret Sharing (VSS) was introduced by Chor, Goldwasser, Micali, and Awerbuch [9], by allowing parties to verify the shares. Later, Feldman [15] used the homomorphism property of Discrete Logarithm (DL) and proposed the first efficient, non-interactive VSS (in the happy path) based on SSS in the honest-majority setting which can achieve Information-Theoretic (IT) binding and computational hiding (a.k.a. unpredictability). In Feldman VSS [15] a dealer publishes homomorphic commitments (using discrete logarithm) to the coefficients of the secret polynomial used in SSS. These commitments enable the parties to verify the validity of their shares during the sharing phase, as well as the published shares during the reconstruction phase. Similar to the original SSS, one advantage of Feldman's VSS is that the individual shares are the same size as the main secret. However, one drawback of Feldman's VSS scheme is that the shared secret requires high entropy. Given the commitments, an adversary can learn the secret in the exponent, i.e., g^s, and if there is insufficient entropy,

they can solve the DL problem to recover the secret s. To address this concern, Pedersen [24] proposed his well-known homomorphic commitment scheme and subsequently revisited Feldman's construction, introducing a new variant that achieves IT hiding (a.k.a., statistical secrecy) and computational binding under the DL problem [24]. Compared to Feldman's scheme, Pedersen's VSS doubles the size of individual shares, increasing them from 1 field element to 2 elements, while eliminating the requirement for a high-entropy secret. Similar to the original SSS, Pedersen VSS can achieve statistical secrecy against up to t malicious shareholders. In a recent work [2], Atapoor, Baghery, Cozzo, and Pedersen (ABCP) proposed a practical Shamir-based VSS scheme in the honest-majority setting that operates with hash-based (or, more generally, non-homomorphic) commitments. Compared to Feldman's scheme, the ABCP VSS scheme triples the size of individual shares, increasing them from 1 field element to 3 elements, while accommodating low-entropy secrets and non-homomorphic (e.g., hash-based) commitments.

In the reconstruction phase of all VSS schemes, including [2, 15, 24], each party publishes their individual share. The published shares are then verified using the verification algorithm of the respective VSS scheme, and at least $t+1$ valid shares are used to interpolate the secret polynomial and reconstruct the main secret. It is important to note that, similar to the original Shamir scheme, in all the practical VSS schemes mentioned, given any $t+1$ individual shares, one can generate the shares of *all* parties. In other words, the secret reconstruction phase is not accountable, making it challenging to trace and identify the exact parties involved in the reconstruction phase of all VSS schemes.

1.2 Traceable Secret Sharing

In CRYPTO'21, Goyal, Song, and Srinivasan [19] introduced Traceable Secret Sharing (TSS), which enhances traditional secret sharing by enabling the identification of parties involved in secret reconstruction, thereby discouraging malicious behavior such as selling shares. In traditional secret sharing (and VSS) schemes, a secret is divided into shares, and any qualified set of shareholders can reconstruct the main secret. However, in TSS, the goal is not only to allow reconstruction by a sufficiently large number of shareholders but also to trace the shareholders who contribute their shares to the reconstruction process. This property is formally referred to as *traceability*, which is not guaranteed in standard secret sharing schemes. In addition to the requirements of a standard secret sharing scheme, a TSS scheme must satisfy *non-imputability*, which ensures that no malicious tracer can falsely accuse a particular (honest) shareholder, even if they have access to the shares of all other parties (except for the targeted individual). The motivation behind TSS is to prevent malicious actions, such as illicit reconstructions or leaks, by introducing accountability mechanisms while keeping the individual shares private.

The authors in [19] provide formal definitions of TSS and its security requirements, along with various constructions based on SSS. The main idea behind their construction is to partition each party's share into two parts: the first part

is a secret known only to that party and unknown to the dealer, while the second part is a share of a secret known only to the dealer (and unknown to any individual party). Intuitively, the first part, which is unknown to the dealer, allows to achieve non-imputability against a malicious dealer, while the whole shares enables the dealer (or a tracer) to identify the parties involved in the secret reconstruction [19]. However, their TSS scheme has some limitations: first, the size of the individual shares grows quadratically with the size of the secret; second, parties cannot verify the shares received from the dealer; and finally, their protocol assumes that the parties act honestly during the reconstruction phase.

Very recently, Boneh, Partap, and Rotem [6] in CRYPTO'24 presented two more efficient TSS schemes, one of which is constructed based on SSS. The authors focus on optimizing the share size within TSS schemes and, similar to SSS, assume that a trusted dealer generates the shares and distributes them among the parties, while separating a malicious tracer from the trusted dealer. In the TSS scheme by Goyal, Song, and Srinivasan [19], the parties participate two-by-two in a two-party protocol with the dealer to generate the shares. Although this is excessive for a TSS scheme, it could be accomplished by a trusted dealer who is separate from the tracer, as demonstrated in [6]. Main contribution in [6] is reducing the share size to a constant size, assuming the existence of a reconstruction box containing up to t (hardcoded honest) shares. Given the complementary set of shares, the box can reconstruct and return the main secret. In their Shamir-based TSS scheme, to trace the parties involved in the reconstruction of the secret, the tracer must query the reconstruction box twice. Moreover, their tracing approach involves querying invalid shares to the reconstruction box [6]. As a VSS scheme rejects invalid shares, their tracing strategy is not suitable for building a TVSS. This raises an interesting research question on developing traceable verifiable secret sharing schemes in [6], which we explore in this paper.

1.3 Our Contributions

Traceable Verifiable Secret Sharing (TVSS). In this paper, we introduce Traceable Verifiable Secret Sharing (TVSS) as an enhancement of both Verifiable Secret Sharing (VSS) and Traceable Secret Sharing (TSS) schemes, and we provide formal definitions for computationally secure TVSS schemes.

A TVSS scheme consists of six polynomial-time algorithms: Initial, Share, Verify, Reconstruct, VerifTrace, and Judge. The first four algorithms are inherited from a VSS scheme [15], while the last two are essential for implementing traceability in secret sharing [19]. Roughly speaking, a TVSS scheme simultaneously achieves the properties of both VSS [2, 15, 24] and TSS [6, 19] schemes, which are formally defined in Section 3. Similar to a VSS scheme, a TVSS scheme enables parties to use the Verify algorithm and check the validity of the distributed shares during the Sharing phase and the published shares during the reconstruction phase.

In common VSS schemes [2, 3, 15, 24], the Reconstruct algorithm typically assumes that parties broadcast the plain values of their shares. After verifying

these shares using the scheme's Verify algorithm, the Reconstruct algorithm returns either a verified reconstructed secret or a $\perp$ symbol if it fails to collect at least $t+1$ valid shares. In TVSS schemes, we extend the Reconstruct algorithm. First, inspired by current actively-secure threshold protocols, like [17, 24], we allow parties to broadcast a function of their shares instead of the plain values, also addressing scenarios where a malicious shareholder might sell a function of their shares, as explored in TSS schemes [19]. To maintain verifiability, when parties reveal a function of their shares, they must also provide proof that these values are derived from the original shares, and they know (or hold) the shares. Second, the Reconstruct algorithm is modified to ensure it produces proof of correct execution and valid outputs. When parties reveal their shares, this proof may simply be the shares themselves, as these are sufficient for an external party, such as a tracer or judge, to verify the validity of the reconstructed secret and identify the parties involved in the reconstruction phase. If shareholders reveal a function of their shares along with a valid proof, the reconstruction algorithm can return either the individual proofs provided by the parties or a single aggregated proof on behalf of all parties involved in the reconstruction. This latter option may lead to more efficient protocols in terms of communication and computation. Third, the Reconstruct algorithm is modified to return the valid collected shares along with individual proofs, even if they are fewer than the threshold value $t+1$ and insufficient to reconstruct the secret. It returns the $\perp$ symbol only if it fails to collect at least one valid share. This modification allows the output of the Reconstruct algorithm to be used to trace the parties involved in the reconstruction phase, even if their number is below the qualified threshold.

Similar to TSS schemes [6, 19], a TVSS scheme also ensures that parties involved in the reconstruction of the secret or a function of their shares can be uniquely traced and identified. We achieve this, using the Trace algorithm and the output of new Reconstruct algorithm. Additionally, as defined by Goyal et al. [19] for a TSS scheme, a TVSS scheme guarantees the *non-imputability* property, meaning that a malicious dealer (or tracer), even if colluding with all other shareholders, cannot produce valid evidence for the Judge algorithm to falsely implicate an honest participant. This feature is crucial for protecting honest shareholders from a malicious dealer's attempts to implicate them.

General Strategy for Building TVSS. We present a general strategy to convert a Shamir-based VSS scheme in the honest-majority setting into an efficient TVSS scheme with minimal computation and communication overhead. Like the original VSS scheme, we assume each party has access to a secure broadcast channel and private communication with the dealer. Our approach is inspired by the two-party protocol between the dealer and each shareholder in Goyal et al.'s TSS scheme [19], as well as the design of the ABCP VSS scheme [2], where part of the individual shares is specific to each party. The key idea is to take a Shamir-based VSS scheme and extend it by augmenting individual shares with additional shares specific to each party, as in [19]. However, in our case, these new shares are sampled by the parties themselves, while the dealer and other parties only have access to their commitments.

Practical TVSS Schemes. Using the proposed strategy, we build two practical TVSS schemes based on the well-known VSS schemes proposed by Feldman [15] and Pedersen [24] in the honest-majority setting. Our focus is on these established VSS schemes; however, we believe that our strategy is general enough to be applied to recently proposed VSS schemes, such as [2, 3], which are based on non-homomorphic commitments. Unlike the TSS schemes proposed in [6, 19], and similar to the standard SSS [26] and its verifiable variants [2, 3, 15, 24], our proposed TVSS schemes retain shareholder indexes (i.e., their unique evaluation points) as public information. This enhances their flexibility in building other accountable threshold protocols, particularly in the context of distributed VSS and Distributed Key Generation (DKG) [1, 17, 24] protocols, where parties need to know the indexes or evaluation points of all other participants. In terms of efficiency, in both of our proposed TVSS schemes, compared to their VSS variants [15, 24], the size of individual shares increases by only one additional field element. Refer to Table 1 for a compact and high-level comparison.

Table 1. Comparison of key features and share sizes in various relevant constructions. SS: Secret Sharing, VSS: Verifiable Secret Sharing, TSS: Traceable Secret Sharing, TVSS: Traceable Verifiable Secret Sharing. $|s|$ denotes the size of the main secret. ✓: Satisfied. ✗: Not satisfied.

Scheme	Verifiability	Traceability	Non-Imputability	Share Size
SS-Shmair [26]	✗	✗	✗	$1\|s\|$
VSS-Feldman [15]	✓	✗	✗	$1\|s\|$
VSS-Pedersen [24]	✓	✗	✗	$2\|s\|$
VSS-ABCP [2]	✓	✗	✗	$3\|s\|$
TSS-GSS [19]	✗	✓*	✓	$\mathcal{O}(\|s\|^2)$
TSS-BPR [6]	✗	✓*	✓	$2\|s\|$
TVSS-Feldman (Sec. 4.1)	✓	✓	✓	$2\|s\|$
TVSS-Pedersen (Sec. 4.2)	✓	✓	✓	$3\|s\|$

* In these schemes, the tracer must query the reconstruction box twice, and in our case only once.

Accountable Threshold Cryptosystems. Recently, Boneh, Partap, and Rotem [5] introduced and studied the concept of accountability in threshold decryption protocols, where dishonest parties may collude to sell their decryption shares without fear of identification. Motivated by their work, as a sample application of new TVSS schemes, we leverage our new Feldman-based TVSS scheme (from Section 4.1) and construct a practical Accountable Threshold Cryptosystem (ATC) based on the ElGamal cryptosystem [14].

To this end, we first present the syntax for an ATC that extends existing non-accountable threshold cryptosystems by incorporating a tracing mechanism. This mechanism allows a tracer to uniquely identify and trace the parties involved in the decryption process. In terms of efficiency, our proposed ATC adds minimal computational overhead compared to alternative actively secure (and/or robust) non-accountable threshold cryptosystems based on the ElGamal cryptosystem.

We provide detailed discussions on the efficiency of the new ATC and compare it with the protocol of Boneh, Partap, and Rotem [5] later in Section 5.

Comparison of Trust Assumptions in TSS and TVSS. Previous works on TSS [6,19] rely on two key assumptions regarding participants behavior: first, the dealer is honest and correctly distributes secret shares. Second, malicious parties who sell their shares act honestly (and naively) from the buyer's perspective. The second assumption is problematic for the following reason: These works assume that malicious parties provide a reconstruction box containing hard-coded honest shares, which answers secret reconstruction queries an arbitrary number of times. In their Shamir-based TSS schemes, tracing strategies [6, 19] rely on multiple (i.e., two) queries to this reconstruction box to identify malicious parties. However, if the reconstruction box were designed to self-destruct after a single query, this tracing strategy and model would fail. In such a scenario, a malicious party could sell shares once without being caught, making the tracing mechanism ineffective. Consequently, we remain skeptical that previous works fully address the risks posed by malicious share-selling.

In contrast, our work introduces traceability and verifiability to secret sharing, strengthening security and accountability in the following ways. First, our schemes ensure correct share distribution, eliminating reliance on the honest dealer assumption. Second, unlike previous works that depend on unrealistic reconstruction boxes for tracing, our approach guarantees verification and traceability in every execution of the protocol. We address malicious behavior in share trading through two key mechanism: 1) *Verification Before Use*: A buyer will not purchase an invalid share (e.g., an incomplete or non-verifiable share), as any end application (e.g., an Accountable Threshold Cryptosystem) enforces a verification procedure (i.e., the VerifTrace algorithm) to confirm both validity and legitimacy before use. 2) *Identity Binding to Shares*: Each share is cryptographically linked to the identity of shareholder, deterring shareholders from selling their shares. This ensures that the protocol maintains non-imputability while allowing any illicit share transaction to be directly traced back to the seller, creating a strong disincentive for malicious behavior. Practical implications of our TVSS schemes is that they provide strong guarantee that only valid shares can be traded and used. Additionally, they establish a direct mechanism to link shares to their sellers. Thus, our approach effectively mitigates the risks associated with malicious behavior in share trading and provides a more secure and accountable framework for real-world applications of T(V)SS.

1.4 Outline

In Section 2, we provide an overview of some preliminary concepts and definitions. In Section 3, we formally introduce the syntax of traceable verifiable secret sharing schemes and discuss their security properties. In Section 4, we propose two practical TVSS schemes based on well-known VSS schemes of Feldman and Pedersen. In Section 5, using our new Feldman-based TVSS scheme, we present a practical accountable threshold cryptosystem. Finally, in Section 6, we conclude the paper and discuss some potential future work.

2 Preliminaries

Notation. We let λ denotes a security parameter and 1^λ its unary representation. A function that is negligible in the security parameter λ is simply called negligible. We use the assignment operator $\leftarrow$ to denote uniform sampling from a set Ξ, e.g. $x \leftarrow \Xi$. We write $\mathbb{Z}_N := \mathbb{Z}/N\mathbb{Z}$ and $\mathbb{Z}_N[X]_t$ for polynomials of degree t in the variable X and with coefficients in $\mathbb{Z}_N$, i.e. the polynomial ring over $\mathbb{Z}_N$. For $n \in \mathbb{N}$, we write $[n] = \{1, \ldots, n\}$. PPT stands for probabilistic polynomial time and all algorithms are PPT, unless mentioned.

Definition 2.1 (Discrete Logarithm Problem (DLP) [13]). *Let $\mathbb{G}$ be a cyclic group of prime order p with generator g. The DLP is hard if for all PPT adversaries $\mathcal{A}$:* $\Pr[\forall\ G \leftarrow \mathbb{G} \mid x \leftarrow \mathcal{A}(g, G) : G = g^x] \leq \mathsf{negl}(\lambda)$.

We refer the reader to Appendix A for detailed preliminaries on Secret Sharing and Shamir's Scheme (App. A.1), Traceable Secret Sharing (App. A.2), Sigma Protocols and Non-Interactive Zero-Knowledge Proofs, the Fiat-Shamir Transform (App. A.3), Public-Key Cryptosystems and ElGamal's Scheme (App. A.4), Commitment Schemes (App. A.5), and Distributed Key Generation Protocols (App. A.6).

3 Traceable Verifiable Secret Sharing

In this section, we formally define Traceable Verifiable Secret Sharing (TVSS) as an extension of both VSS [2, 15, 24] and TSS [6, 19] and present the syntax. Our definition naturally combines the syntax of computationally secure VSS and TSS schemes in the honest-majority setting, as outlined in [2, 15, 19], with some subtle modifications and extensions.

In the reconstruction phase of typical VSS schemes, at least $t+1$ shareholders reveal the plain values of their shares, e.g., s_i. These revealed shares are then verified using the protocol's transcript and the VSS scheme's verification algorithm. Once a set of $t+1$ valid shares (that pass the verification) is identified, they are used to reconstruct the main secret, e.g., s. In our definitions and protocols, we extend the functionality of the reconstruction phase in two key directions. First, we extend the syntax to allow reconstruction with fewer than $t+1$ parties. This modification is inspired by recent works on traceable secret sharing protocols [6, 19], which permit an unqualified set of shareholders (fewer than $t+1$ parties) to reconstruct and potentially sell a function of their shares. Second, we enable shareholders participating in the reconstruction phase to reveal a function of their shares rather than the plain share values. This approach is commonly employed in actively secure real-world threshold protocols, such as distributed key generation (DKG) protocols [24] and threshold decryption schemes [11, 12], where shareholders reveal a function of their shares (e.g., g^{s_i}) and provide proof that the function was correctly computed using the original share s_i. Consequently, if the parties choose to reveal a function of their

shares, such as $z_i = F(s_i)$, they are required to attach a non-interactive (possibly zero-knowledge) proof demonstrating that z_i has been correctly computed from the original share s_i. This modification also allows for the formalization of the share-selling scenario as defined in traceable secret sharing (TSS) schemes, where shareholders can sell a function of their shares rather than the shares themselves [6, 19].

3.1 Syntax and Requirements

A TVSS scheme is a 6-tuple (Initial, Share, Verify, Reconstruct, VerifTrace, Judge) of polynomial-time algorithms. The first four algorithms (Initial, Share, Verify, Reconstruct) are inherited from a VSS scheme [15], while (VerifTrace, Judge) are defined specifically to enable traceability in secret sharing [19]. We formally define the syntax of TVSS schemes as follows:

Definition 3.1 (Traceable Verifiable Secret Sharing). *Given an injective function $F(\cdot)$, an (n,t)-TVSS consists of six PPT algorithms, defined as follows:*

1. $\mathsf{Initial}$*: During this phase, the initialization is completed, public parameters are generated, and then distributed to all parties.*
2. $\mathsf{Share}(1^\lambda, n, t, s) \to (\{s_i\}_{i=1}^n, \pi_{share}, tk, vk)$*: Given 1^λ, the integers (n,t), and a secret s as inputs, the algorithm shares the secret s and obtains the shares $s_1, \cdots, s_n$ along with a proof π_{share}, which is generated to verify the validity of these shares. Additionally, it generates a pair of tracing and verification keys (tk, vk). It then outputs $(\{s_i\}_{i=1}^n, \pi_{share}, tk, vk)$.*
3. $\mathsf{Verify}(n, t, \{s_i\}_{i=1}^n, \pi_{share}) \to$ `true/false`*: Given the integers (n,t), shares $\{s_i\}_{i=1}^n$, and proof π_{share} as inputs, the algorithm checks the validity of the shares and returns either* `true` *(accept) or* `false` *(reject).*
4. $\mathsf{Reconstruct}(\{s_i\}_{i\in Q}, \pi_{share}, F(\cdot)) \to \{F(s), \pi_{rec}\}/\{\{F(s_j)\}_{j\in V}, \pi_{rec}\}/\perp$*: Given the function $F(\cdot)$—which could also be the identity function $\mathsf{Iden}(\cdot)$—the proof π_{share}, and the shares $\{s_i\}_{i\in Q}$, the algorithm operates as follows: Each party $i \in Q$ publishes $(F(s_i), \pi_i)$, where π_i serves as a proof of the validity of $F(s_i)$. If $F(\cdot)$ is the identity function $\mathsf{Iden}(\cdot)$, then π_i is not required.*[4] *All parties then verify the published $\{(F(s_i), \pi_i)\}_{i\in Q}$ and gather the valid pairs. From the set of valid pairs, they construct an n-bit string who, where each bit indicates whether the corresponding party has provided a valid proof.*[5] *Let $\{(F(s_j), \pi_j)\}_{j\in V}$ denote the set of these valid pairs, where $V \subseteq Q$ is the set of their indexes. It acts as follows:*
   - *If $|V| = 0$, returns $\perp$.*

[4] Note that, in certain cases, such as round-robin threshold protocols [1], parties may update the output from the previous party using their shares, rather than independently publishing a function of their share. In this case, the party must provide proof to verify the correctness of the update.

[5] For example, if $\mathsf{who} = 010011$, it means that parties 1, 2, and 5 have provided valid proofs and participated in the reconstruction. As we will show later, the string who is defined solely for efficiency purposes.

- *If* $|V| \geq t+1$*, reconstructs* $F(s)$ *and generates an associated proof* π_{rec} *where* s *is the main secret.*[6] *The algorithm then returns* $(F(s), \pi_{rec})$*. Additionally, the string* who *is included in the proof* π_{rec}*.*
- *If* $0 < |V| < t+1$*, returns the set* $\{F(s_j)\}_{j\in V}$ *and an associated proof* π_{rec} *for their validity. Similarly, the string* who *is included in* π_{rec}*.*

5. $\mathsf{VerifTrace}(\{F(s), \pi_{rec}\}/\{\{F(s_j)\}_{j\in V}, \pi_{rec}\}, tk) \rightarrow \{b, I, \pi_{trace}\}$*: Given a function of the secret or its shares and the corresponding proof, i.e.* $\{F(s), \pi_{rec}\}$ *or* $\{\{F(s_j)\}_{j\in V}, \pi_{rec}\}$*, and a (public or private) tracing key* tk*, it checks the validity of* π_{rec}*. If the verification fails, it returns* $\{0, \emptyset, 0^n\}$*. Otherwise, it identifies a subset* $I \subseteq V$ *of valid identities. It then outputs* $1, I$ *along with a proof* π_{trace}*.*
6. $\mathsf{Judge}(I, \pi_{trace}, vk) \rightarrow$ `true/false`*: This is a deterministic algorithm that takes a set* I*, a proof* π_{trace}*, and a verification key* vk *as inputs. It outputs either* `true`*, indicating that the proof successfully verifies the parties in* I *are accurately traced, or* `false`*, indicating that the proof is rejected.*

3.2 Security Properties

A TVSS scheme must fulfill the properties of both VSS [2,15,24] and TSS [6,19] protocols. Specifically, a TVSS scheme should ensure the following properties: correctness, verifiability, unpredictability (or secrecy), traceability, and non-imputability, as defined below. We adapt some of these definitions from prior works [6,15,19,22].

Definition 3.2 (Perfect Correctness). *For any integers* $n > 1$ *and* $t < n$ *a TVSS is called correct, if we have:*

$$\Pr\left[\begin{array}{l}(\{s_i\}_{i=1}^n, \pi_{share}, tk, vk) \leftarrow \mathsf{Share}(1^\lambda, n, t, s),\\ (s', \pi_{rec}) \leftarrow \mathsf{Reconstruct}(\{s_i\}_{i\in V,\ |V|\geq t+1}, \pi_{share}, \mathsf{Iden}(\cdot)) : s' = s\end{array}\right] = 1 \ ,$$

$$\Pr\left[\begin{array}{l}(\{s_i\}_{i=1}^n, \pi_{share}, tk, vk) \leftarrow \mathsf{Share}(1^\lambda, n, t, s),\\ (b, Q', \pi_{trace}) \leftarrow \mathsf{VerifTrace}(\mathsf{Reconstruct}(\{s_i\}_{i\in Q}, \pi_{share}, F(\cdot)), tk) :\\ b = 1 \wedge Q' = Q\end{array}\right] = 1 \ ,$$

where V *is the set of honest parties,* Q *is set of honest parties participated in the reconstruction phase, where* $0 < |Q| \leq n$*, and* $\mathsf{Iden}(\cdot)$ *is the identity function.*

Definition 3.3 (Verifiability). *For any integers* $n \geq 2t+1$ *and* $t \geq 0$*, a TVSS is called verifiable if for any PPT adversaries* $\mathcal{A}$*, we have:*

$$\Pr\left[\begin{array}{l}(\{s_i\}_{i=1}^n, \pi_{share}, tk, vk) \leftarrow \mathcal{A}(1^\lambda, n, t);\\ \mathsf{Verify}(n, t, \{s_i\}_{i=1}^n, \pi_{share}) = \mathtt{true} :\\ \exists s, \forall V \in [n], |V| \geq t+1;\\ (s, \pi_{rec}) \leftarrow \mathsf{Reconstruct}(\{s_i\}_{i\in V}, \pi_{share}, \mathsf{Iden}(\cdot))\end{array}\right] \geq 1 - \mathsf{negl}(\lambda) \ ,$$

where V *is the set of honest parties, and* $\mathsf{Iden}(\cdot)$ *is the identity function.*

[6] It is important to note that, in this scenario, we assume any qualified set of parties can compute $F(s)$.

Informally, the verifiability property ensures that even if the dealer is dishonest, any execution of the sharing phase determines a unique value s which will be reconstructed at the reconstruction phase.

Definition 3.4 (Unpredictability). *For any integers $n > 1$ and $t < n$ a TVSS is called unpredictable if for all PPT adversaries $\mathcal{A}$ we have:*

$$\Pr\begin{bmatrix}(\{s_i\}_{i=1}^n, \pi_{share}, tk, vk) \leftarrow \mathsf{Share}(1^\lambda, n, t, s); Q \subset [n]; |Q| \leq t; \\ s' \leftarrow \mathcal{A}(1^\lambda, n, t, \{s_i\}_{i\in Q}, \pi_{Share}) : s' = s\end{bmatrix} \leq \mathsf{negl}(\lambda) \ .$$

Definition 3.5 (Statistical Secrecy). *For any integers $n > 1$ and $t < n$ a TVSS achieves the secrecy property if for all adversaries $\mathcal{A}$, we have:*

$$\Pr\begin{bmatrix}(m_0, m_1) \leftarrow \mathcal{A}(1^\lambda, n, t); b \leftarrow \{0,1\}; \\ (\{s_i\}_{i=1}^n, \pi_{share}, tk, vk) \leftarrow \mathsf{Share}(1^\lambda, n, t, m_b); Q \subset [n]; |Q| \leq t; \\ b' \leftarrow \mathcal{A}(1^\lambda, n, t, \{s_i\}_{i\in Q}, \pi_{share}, m_0, m_1) : b' = b\end{bmatrix} \leq \frac{1}{2} + \mathsf{negl}(\lambda) \ .$$

In some literature, this property is referred to as *IND2-Secrecy* [22], while in others, it is called *Privacy* [19]. In this work, we use the term *Secrecy*.

Definition 3.6 (Traceability). *For any integers $n > 1$ and $t < n$, a TVSS is called traceable if for all PPT adversaries $\mathcal{A}$, and any set $Q \subseteq [n]$, we have:*

$$\Pr\begin{bmatrix}(\{s_i\}_{i=1}^n, \pi_{share}, tk, vk) \leftarrow \mathsf{Share}(1^\lambda, n, t, s); \\ \{F(s), \pi_{rec}\}/\{\{F(s_j)\}_{j\in Q}, \pi_{rec}\} \leftarrow \mathcal{A}(\{s_i\}_{i\in Q}, \pi_{share}, F(\cdot)); \\ (b, I, \pi_{trace}) \leftarrow \mathsf{VerifTrace}(\{F(s), \pi_{rec}\}/\{\{F(s_j)\}_{j\in Q}, \pi_{rec}\}, tk) : \\ (b = 1 \wedge I \subseteq Q \ \wedge \ \mathsf{Judge}(I, \pi_{trace}, vk) = \mathtt{true}) \vee (b = 0)\end{bmatrix} \geq 1 - \mathsf{negl}(\lambda) \ .$$

Informally, the traceability property ensures that if the $\mathsf{Reconstruct}$ algorithm is executed maliciously by a PPT adversary, and it generates a verifiable output, the $\mathsf{VerifTrace}$ algorithm will identify the parties with valid shares.

Definition 3.7 (Non-imputability). *For any integers $n > 1$ and $t < n$, a TVSS is called non-imputable if for all PPT adversaries $\mathcal{A}$, we have:*

$$\Pr\begin{bmatrix}(\{s_i\}_{i=1}^n, \pi_{share}, tk, vk) \leftarrow \mathsf{Share}(1^\lambda, n, t, s); \\ (1, I, \pi_{trace}) \leftarrow \mathcal{A}(\{s_i\}_{i\in[n]\setminus i^*}, \pi_{share}, tk) : \\ \mathsf{Judge}(I, \pi_{trace}, vk) = \mathtt{true} \wedge i^* \in I\end{bmatrix} \leq \mathsf{negl}(\lambda) \ .$$

Informally, the non-imputability property guarantees that even if a PPT adversary has access to the shares of all honest parties except one, denoted as P_{i^*}, it cannot produce any valid proof that falsely implicates P_{i^*} in misbehavior.

3.3 Building a TVSS from a VSS Scheme

Next, we introduce a generic technique that allows a computationally secure Shamir-based Verifiable Secret Sharing (VSS) scheme to be converted into a Traceable VSS (TVSS) scheme.

Generally speaking, this approach redefines each party's share to consist of two components, i.e., $s_i := (f_i, \gamma_i)$. The first component (e.g., f_i) is sampled by the dealer using Shamir's secret sharing scheme with parameters (n, t), while the second component (e.g., γ_i) is an independently sampled random element chosen by each party P_i. Our technique is inspired by the traceable secret sharing scheme introduced by GSS'21 TSS [19], where the dealer engages in a two-party protocol with each shareholder. In the end, each party receives a share composed of two components, with the dealer only knowing one of those parts. After the sharing phase concludes, all parties learn the commitments to everyone's shares. However, in their TSS scheme, the size of the individual shares in GSS'21 is quadratic in secret size. In contrast, our proposed TVSS scheme requires shares that are only twice as large as the secret. By structuring each share to include both a dealer-generated component and a shareholder-generated component, we not only ensure the verifiability of the shared secret but also achieve traceability and non-imputability. Specifically, our constructed TVSS scheme achieves verifiability, unpredictability, and robustness against up to t malicious parties, while also ensuring non-imputability even when an adversary controls the dealer and up to $n-1$ other parties. The main idea is that, even if the adversary has control over the dealer and all but one party, they cannot fully determine the share of the remaining party. This is because the independently sampled component (e.g., γ_i) remains private to each party, and only commitments to these shares are made public. As a result, the adversary cannot gain complete knowledge of the target party's share. These commitments are later used in the $\mathsf{VerifTrace}$ algorithm to verify the *validity* of the reconstructed secret or shares.

Let $\Gamma' := (\mathsf{Initial}', \mathsf{Share}', \mathsf{Verify}', \mathsf{Reconstruct}')$ be a secure Shamir-based VSS scheme. Using the proposed technique, we construct a new TVSS scheme $\Gamma := (\mathsf{Initial}, \mathsf{Share}, \mathsf{Verify}, \mathsf{Reconstruct}, \mathsf{VerifTrace}, \mathsf{Judge})$ as outlined in Figure 1. This new scheme retains the foundational principles of the original VSS while incorporating additional functionalities, traceability and non-imputability, by extending the share generation and reconstruction processes to include individually sampled components by each party, as well as mechanisms for tracing the parties involved in the reconstruction phase.

New TVSS schemes extends the reconstruction phase of typical VSS schemes in two key ways, offering greater flexibility while still ensuring accountability. First, shareholders are permitted to run the reconstruction algorithm even with an unqualified set of parties. Second, they can publish a function of their shares rather than the plain values. These enhancements allow any set of parties to reconstruct a function of their shares, and if the number of parties exceeds t, forming a qualified set, they can reconstruct the main secret or a function of it.

To maintain verifiability in the reconstruction phase and ensure accountability, when parties choose to reveal a function of their shares, such as $z_i = F(f_i)$,

Initial: As in the $\mathsf{Initial}'$ algorithm of Γ', parties $\{P_i\}_{i=1}^n$ generate the necessary parameters for the commitment scheme $\mathcal{C}$ and the NIZK proof scheme Π_{NIZK}. We assume the existence of a dealer D and the set of parties $\{P_i\}_{i=1}^n$ who will receive the shares.

Share: Given (n, t) and the secret $s := f_0$, the dealer D proceeds as in the Share' algorithm of Γ' and privately distributes the shares, i.e., $\{f_i\}_{i=1}^n$, to the parties while publishing a proof π_D. Moreover, in addition to Γ', each party P_i samples a random secret, γ_i, and broadcasts a commitment to it, denoted $c'_i = \mathcal{C}(\gamma_i)$. At the end of this phase, parties obtain the proof $\pi_{share} = tk = vk := (\pi_D, c'_1, \ldots, c'_n)$, and each party P_i stores its individual share s_i consisting of two components, $s_i := (f_i, \gamma_i)$, jointly determined by the dealer and the shareholder itself.

Verify: the parties proceed with the verification phase of the underlying VSS scheme. Given the proof $\pi_{share} := (\pi_D, c'_1, \ldots, c'_n)$ and the shares, i.e., $s_i := (f_i, \gamma_i)$ for $i = 1, \ldots, n$, the parties proceed with the verification phase of the underlying VSS scheme. The parties also verify the well-formedness of $\{c'_i\}_{i=1}^n$ published by the parties. Any participant who fails to broadcast a valid commitment or submits a wrong commitment is disqualified. Ultimately, the parties reach a consensus either on an approved set Q of qualified participants or reject the final verification.

Reconstruct: Given a function $F(\cdot)$, which can also be the identity function, the proof $\pi_{share} := (\pi_D, c'_1, \ldots, c'_n)$, and the shares, i.e., $\{s_i := (f_i, \gamma_i)\}_{i \in Q}$, the parties proceed as follows.

- Each party P_i, for $i \in Q$, publishes $(z_i = F(f_i), \pi_i)$, where π_i is a non-interactive proof (or argument) generated by NIZK proof scheme Π_{NIZK} to ensure that $z_i = F(f_i)$ is computed correctly and the party knows the original $s_i := (f_i, \gamma_i)$, that are generated and committed in the sharing phase. Note that if the function $F(\cdot)$ is the identity function, then π_i will be $s_i := (f_i, \gamma_i)$, as s_i itself can be verified using π_{share}.
- Given π_{share}, the broadcasted values $\{z_i := F(f_i), \pi_i\}_{i \in Q}$ are verified and the valid ones are collected. Let $\{z_j = F(f_j), \pi_j)\}_{j \in V}$ be the set of valid pairs, and V denotes the set of their indexes. From the set of valid pairs, they construct an n-bit string who, where each bit indicates whether the corresponding party has provided a valid proof.
- If $|V| = 0$, the protocol returns $\perp$;
- If $t + 1 \leq |V|$: the protocol reconstructs $z := F(f_0)$ and generates associated proof $\pi_{rec} := \{\pi_j\}_{j \in V}$, to show that z is computed correctly by the parties indicated in who and they know $\{s_j\}_{j \in V}$. Finally, it returns $(z = F(f_0), \pi_{rec} := \{\mathsf{who}, \pi_j\}_{j \in V})$. If $F(\cdot)$ be the identity function, then $\{\pi_j\}_{j \in V}$ can be the set of shares $\{s_j\}_{j \in V}$.
- If $0 < |V| < t+1$: the protocol returns $\{z_j := F(f_j)\}_{j \in V}$ and a proof $\pi_{rec} := \{\pi_j\}_{j \in V}$ to show that $\{z_j\}_{j \in V}$ are computed correctly by the parties indicated in who and they know $\{s_j\}_{j \in V}$.

VerifTrace: Given $\{z := F(f_0), \pi_{rec} := \{\mathsf{who}, \pi_j\}_{j \in V}\}$ or $\{\{z_j := F(f_j)\}_{j \in V}, \pi_{rec} := \{\mathsf{who}, \pi_j\}_{j \in V}\}$, generated by $\mathsf{Reconstruct}$, and tracing key $tk := \pi_{share} = (\pi_D, c'_1, \ldots, c'_n)$, it verifies the proofs $\pi_{rec} := \{\mathsf{who}, \pi_j\}_{j \in V}$ using the tracing key tk and determines the set $I \subseteq V$ as the set of valid ones. Finally, it returns the set I and $\pi_{trace} := (z := F(f_0)$ or $\{z_j := F(f_j)\}_{j \in V}, \{\pi_j\}_{j \in I})$ if $|I| > 0$; otherwise, it returns $\{0, \emptyset, 0^n\}$.

Judge: Given an alleged subset I, and a proof $\pi_{trace} := (z = F(f_0)$ or $\{z_j = F(f_j)\}_{j \in I}, \{\pi_j\}_{j \in I})$, and $vk := \pi_{share} = (\pi_D, c'_1, \ldots, c'_n)$, it verifies if π_{trace} is valid and outputs `true` or `false`.

Fig. 1. A general framework for constructing a TVSS scheme Γ from a computationally secure Shamir-based VSS scheme Γ', a secure commitment scheme $\mathcal{C}(\cdot)$, and a secure NIZK proof scheme Π_{NIZK}.

they must provide a proof π_i, generated by a NIZK proof system Π_{NIZK}. This proof demonstrates that z_i was correctly computed from the original share f_i and that the party P_i knows the original share value $s_i := (f_i, \gamma_i)$, which was generated during the sharing phase. These individual proofs are then collected and combined into a final proof π_{rec}, which is published to verify that the reconstruction phase has been executed correctly. We highlight that in new TVSS schemes, malicious parties may attempt to sell only part of their shares to avoid identification. For example, they might try to sell only f_i instead of (f_i, γ_i). However, this would not be sufficient for the buyer to generate Π_{NIZK} (and subsequently π_{rec}). Thus, as mentioned before, a buyer will not purchase an incomplete share (e.g., just f_i instead of (f_i, γ_i)), since generating the proof Π_{NIZK} and passing the

VerifTrace algorithm require both secret values, (f_i, γ_i). As a practical example, consider an access control system based on our TVSS schemes. In such a system, parties would need to know two passwords, (f_0, γ_0), to gain access to a service. Therefore, even if a buyer manages to obtain and generate the first password, f_0, the system would still deny access, as the second component, γ_0, is required.

Efficiency and Security. In terms of efficiency, the new TVSS scheme Γ' has asymptotically the same computational costs as the underlying VSS scheme Γ. We will discuss the specific computational costs for the VerifTrace and Judge algorithms, as well as the security of the resulting TVSS scheme, later in the context of particular instantiations and the proposed protocols.

4 Practical TVSS Schemes

Our proposed simple strategy (in Section 3.3) is general enough to be used with various computationally secure VSS schemes. As to build a TVSS scheme Γ, it only requires a secure commitment scheme $\mathcal{C}$ and a VSS scheme Γ'. In this section, we employ the Feldman and Pedersen VSS schemes with DL-based commitment schemes for $\mathcal{C}$ and construct two efficient TVSS schemes based on Shamir secret sharing. The resulting TVSS schemes present various trade-offs in terms of efficiency and security.

4.1 An Efficient TVSS based on Feldman VSS

Feldman VSS. One of the well-known and widely used computationally secure VSS schemes is Feldman's protocol, which was proposed by Feldman in [15]. In his scheme, given (n, t) and a group generator g, to share a *high-entropy* secret $f_0 \in \mathbb{Z}_q$, the dealer acts as follows. It first does Shamir secret sharing using a secret polynomial $f(X) := f_0 + a_1 X + \cdots + a_t X^t$, and privately sends the shares $s_i := f_i = f(i)$ to each party P_i. Then, it publishes commitments $c_0 = g^{f_0}, c_1 = g^{a_1}, \ldots, c_t = g^{a_t}$ as the proof π_{share}. The proof π_{share} allow the parties to verify the validity of their shares in the sharing phase, as well as the published shares during the reconstruction phase.

To verify the share f_i, given, $\pi_{share} := (c_0, c_1, \ldots, c_t)$, party P_i checks if

$$g^{f_i} = \prod_{j=0}^{t} c_j^{i^j}, \tag{1}$$

and outputs `true` or `false`. For $n \geq 2t + 1$, if *all* n parties return `true`, then the final Verification returns `true`. Otherwise, any conflict between the dealer and the parties is resolved using a well-known conflict resolution approach, as employed in earlier VSS schemes such as Feldman and Pedersen.

Traceable Variant of Feldman VSS. Using our proposed general construction from Figure 1, we build a TVSS scheme based on Feldman VSS by instantiating the underlying VSS scheme Γ' with Feldman VSS and the commitment scheme

$\mathcal{C}(\cdot)$ using the Discrete Logarithm (DL). Specifically, we define the commitment to each party's random value $\gamma_i \leftarrow\!\!\$\, \mathbb{Z}_q$ as $c'_i := \mathcal{C}(\gamma_i) := g^{\gamma_i}$, where g is the same group generator used in Feldman VSS. This ensures that the commitments are compatible with the cryptographic assumptions of the original VSS scheme, while also enabling efficient verification and traceability. During the sharing phase, as in the original Feldman VSS scheme, each party also receives share f_i from the dealer, and sets his share $s_i := (f_i, \gamma_i)$. The share verification and conflict resolution phases are carried out almost identically to the original VSS scheme. The primary difference is that, in addition to the usual verification step, each party P_i must also verify the well-formedness of the commitments c'_j for $j = 1, \ldots, n, j \neq i$ published by all the other parties. If any party P_i fails to broadcast a well-formed commitment c'_i, it will be disqualified from the protocol. This additional step ensures that each party's commitment is well-formed, and can be opened latter in the reconstruction phase.

In the reconstruction phase of new TVSS scheme, any set of shareholders can compute either the plain value or a function of their shares. If the number of participating (honest) shareholders is at least $t + 1$, i.e., a qualified set, they can also compute the plain value or a function of the main secret, such as f_0 or $z = F(f_0)$. If a party P_i chooses to publish a function of their shares, such as $z_i = F(f_i)$, they must provide a non-interactive proof that z_i is correctly computed from f_i and they know the original share $s_i := (f_i, \gamma_i)$, sampled in the sharing phase. More formally, each party P_i needs to use a NIZK proof scheme, e.g., $\Pi^{(1)}_{\mathsf{NIZK}}$, and generate a proof for the relation $R^{(1)}_i$, defined as follows,

$$R^{(1)}_i = \{(g, F(\cdot), z_i, y_i, c'_i, (f_i, \gamma_i)) \mid z_i = F(f_i) \wedge y_i = g^{f_i} \wedge c'_i = g^{\gamma_i}\}, \quad (2)$$

where $y_i = g^{f_i} = \prod_{j=0}^{t} c_j^{i^j}$ can be publicly (pre)computed using (a valid) π_{share} and the *public index* i. Let π_i be the individual NIZK proof (or argument) published by party P_i, which is generated by a NIZK proof scheme $\Pi^{(1)}_{\mathsf{NIZK}}$. In practice, depending on the function $F(\cdot)$, the proof scheme $\Pi^{(1)}_{\mathsf{NIZK}}$ can be constructed using different families of zero-knowledge proof systems (possibly with non-malleable proofs), e.g., sigma protocols [25] and zk-SNARKs [20]. If party P_i reveals the plain values of their shares, i.e., $F(x)$ is the identity function, then π_i will be equal to $s_i := (f_i, \gamma_i)$, as the plain values of shares can be verified using π_{share} and intuitively in that case the shares themselves can serve as a sound (but non-zero-knowledge) proof. At the end of the reconstruction phase, the overall proof π_{rec} can be generated using the set of valid individual proofs, indicated by an n-bit string who, and their corresponding statements. In the general (non-optimized) case, π_{rec} consists of all valid proofs, i.e., $\pi_{rec} := \{\mathsf{who}, \pi_j\}_{j \in V}$ and its statement would be $\{z_j, c'_j, y_j\}_{j \in V}$ where V is the set of valid indices (i.e., the indices of parties who have published valid proofs) and can be deduced from the string who. In concrete applications, e.g., DL-based protocols, depending on the function $F(\cdot)$ and the underlying NIZK proof scheme $\Pi^{(1)}_{\mathsf{NIZK}}$ one might be able to generate a single and compact proof for all parties. The proof π_{rec} guarantees correctness of the revealed shares (or a function of them) and ensures that the

Initial: Let $\mathbb{G}$ be a group with generator g, where the DL problem is hard. The group generator g and the public parameters for the NIZK proof scheme $\Pi^{(1)}_{\mathsf{NIZK}}$ are shared with all the parties.

Share: Given (n, t), and a random secret $s := f_0 \in \mathbb{Z}_q$, the protocol proceeds as follows:

- The dealer samples a random degree-t polynomial $f(x) = f_0 + a_1 X + \ldots, a_t X^t$, with random coefficients from $\mathbb{Z}_q$, such that $f(0) = f_0$. Next, the dealer computes $f_i := f(i)$ for $i \in [n]$ and privately sends f_i to party P_i. Then, the dealer computes $c_0 = g^{f_0}, c_1 = g^{a_1}, \ldots, c_t = g^{a_t}$ and publishes a proof $\pi_D := (c_0, \ldots, c_t)$.
- In the same round, for $i \in [n]$, each party P_i samples a random secret $\gamma_i \in \mathbb{Z}_q$, and broadcasts the commitment $c'_i := g^{\gamma_i}$.
- At the end, each party P_i stores its individual share $s_i := (f_i, \gamma_i)$ and the public proof, the tracing and the verification keys $\pi_{share} = tk = vk := (\pi_D, c'_1, \ldots, c'_n)$ which are generated jointly by the dealer and the parties.

Verify: Given $\pi_{share} = (c_0, c_1, \ldots, c_t, c'_1, \ldots, c'_n)$ and the shares, i.e., $s_i := (f_i, \gamma_i)$ for $i \in [n]$, the parties proceed as follows:

1) Each party P_i checks if $g^{f_i} = \prod_{j=0}^{t} c_j^{i^j}$, and broadcasts a complaint against the dealer if the verification failed. Party P_i also checks if $c'_k \in \mathbb{G}$ for $k \in [n]$, and $k \neq i$. In case a commitment c'_k is missing or it is not well-formed, party P_i disqualify P_k and broadcasts a complaint against it.
2) If the number of parties complaining against party P_k exceeds a threshold value t, the party P_k is disqualified. Similarly, if the number of shareholders complaining against the dealer exceeds a threshold value t, the dealer will be disqualified, and the verification process will result in a `false` outcome.
3) If a shareholder P_i raises a complaint against the dealer, the dealer will respond by broadcasting f_i so that everyone can verify it using the verification equation. If the verification succeeds, the protocol continues as normal. However, if it fails, the dealer will be disqualified, resulting in a 'false' verification outcome. Since the disqualification decision is based solely on publicly shared information, all honest shareholders will eventually agree either on a qualified set of parties $Q \subseteq \{P_1, P_2, \ldots, P_n\}$ or on rejecting the final verification.

Reconstruct: Given a function $F(\cdot)$, which can be the identity function as well, the proof $\pi_{share} := (c_0, c_1, \ldots, c_t, c'_1, \ldots, c'_n)$, and the shares $\{s_i := (f_i, \gamma_i)\}_{i \in Q}$, the parties proceed as follows.

- Each party P_i, for $i \in Q$, publishes $(z_i = F(f_i), \pi_i)$, where π_i is a non-interactive proof (or argument) generated by $\Pi^{(1)}_{\mathsf{NIZK}}$ for the relation $R^{(1)}_i$ defined in equation (2). The proof π ensures that z_i is computed correctly and P_i knows the original share $s_i := (f_i, \gamma_i)$. Note that if $F(\cdot)$ is the identity function, then the proof π_i will be equal to $s_i := (f_i, \gamma_i)$, as the share s_i itself can be verified using π_{share}.
- Given π_{share}, the broadcasted values $\{z_i := F(f_i), \pi_i\}_{i \in Q}$ are verified using the verification algorithm of $\Pi^{(1)}_{\mathsf{NIZK}}$ and the valid ones are collected. Let $\{z_j = F(f_j), \pi_j)\}_{j \in V}$ be the set of valid pairs, V be the set of their indexes, and who be an n-bit string where each of its bit indicates whether the corresponding party has provided a valid proof.
- If $|V| = 0$, the protocol returns $\perp$;
- If $t + 1 \leq |V|$: the protocol reconstructs $z = F(f_0)$ and generates associated proof $\pi_{rec} := \{\mathsf{who}, \pi_j\}_{j \in V}$, to show that z is computed correctly and the parties indicated in who know $\{s_j\}_{j \in V}$. Finally, it returns $(z = F(f_0), \pi_{rec})$. Note that if $F(\cdot)$ is the identity function, then the proofs $\{\pi_j\}_{j \in V}$ would be the set of shares $\{s_j\}_{j \in V}$.
- If $0 < |V| < t + 1$: the protocol returns $\{z_j := F(f_j)\}_{j \in V}$ and a proof $\pi_{rec} := \{\mathsf{who}, \pi_j\}_{j \in V}$ to show that $\{z_j\}_{j \in V}$ are computed correctly by the parties indicated in who and they know $\{s_j\}_{j \in V}$.

VerifTrace: Given $\{z := F(f_0), \pi_{rec}\}$ or $\{\{z_j := F(f_j)\}_{j \in V}, \pi_{rec}\}$, generated by Reconstruct, and tracing key $tk := \pi_{share} = (c_0, c_1, \ldots, c_t, c'_1, \ldots, c'_n)$, it computes $y_j := \prod_{k=0}^{t} c_k^{j^k}$ for all the parties $j \in V$ (deducible from who) and then verifies the proofs $\pi_{rec} := \{\mathsf{who}, \pi_j\}_{j \in V}$ using the verification algorithm of $\Pi^{(1)}_{\mathsf{NIZK}}$ and commitment values $\{y_j, c'_j\}_{j \in V}$, and finally determines $I \subseteq V$ as the set of valid ones. Finally, it returns $1, I$ and $\pi_{trace} := (z = F(f_0)$ or $\{z_j = F(f_j)\}_{j \in I}, \{\pi_j\}_{j \in I})$ if $|I| > 0$; otherwise, it returns $\{0, \emptyset, 0^n\}$.

Judge: Given an alleged set I, and a proof $\pi_{trace} := (z = F(f_0)$ or $\{z_j = F(f_j)\}_{j \in I}, \{\pi_j\}_{j \in I})$, and $vk := \pi_{share} = (c_0, c_1, \ldots, c_t, c'_1, \ldots, c'_n)$, using the verification algorithm of $\Pi^{(1)}_{\mathsf{NIZK}}$, it verifies if π_{trace} is valid and outputs `true` or `false`.

Fig. 2. An efficient TVSS scheme based on the Feldman VSS scheme.

reconstruction was performed correctly by a set of valid shareholders. Figure 2 describes the algorithms of resulting TVSS scheme and we prove its security in the following theorem.

Theorem 4.1 (TVSS based on Feldman VSS). *Let* $\Pi^{(1)}_{\mathsf{NIZK}} := (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ *be a NIZK proof scheme for the relation* $R^{(1)}_i$ *defined in Eq. (2) that satisfies Correctness, and Soundness (or Knowledge Soundness), as defined in Section A.3. Then under the discrete logarithm assumption, the protocol presented in Figure 2 constitutes a secure TVSS scheme that satisfies the properties of Correctness, Verifiability, Unpredictability, Traceability and Non-imputability as defined in Definitions 3.2 to 3.4, 3.6 and 3.7.*

Proof. **Correctness of the Reconstruction Algorithm.** By the correctness of the Feldman VSS, the secret value s is constructed with a probability 1.

Correctness of the Trace Algorithm. We show the correctness by analyzing two cases based on the size of Q:

- When $0 < |Q| \leq t$, the reconstruction algorithm returns $\{F(f_j), \pi_j, \mathsf{who}\}_{j \in Q}$ where π_j is a non-interactive proof (or argument) generated by $\Pi^{(1)}_{\mathsf{NIZK}}$ for the relation $R^{(1)}_j$ defined in equation (2). The trace algorithm given $\{F(f_j), \pi_j, \mathsf{who}\}_{j \in Q}$ and the tracing key $tk := \pi_{share} = (c_0, c_1, \ldots, c_t, c'_1, \ldots, c'_n)$, computes $y_j := \prod_{k=0}^{t} c_k^{j^k}$ for $j \in Q$ (which can be deduced from the n-bit string who) and then verifies the proofs $\{\pi_j\}_{j \in Q}$ using the verification algorithm of $\Pi^{(1)}_{\mathsf{NIZK}}$ and commitment values $\{y_j, c'_j\}_{j \in Q}$. By the completeness property of $\Pi^{(1)}_{\mathsf{NIZK}}$, all proofs will be accepted. Finally, it returns Q and $\pi_{trace} := (\{F(f_j)\}_{j \in Q}, \{\pi_j\}_{j \in Q})$.
- When $|Q| \geq t + 1$, the reconstruction algorithm returns $\{F(s), \pi_j, \mathsf{who}\}_{j \in Q}$ where π_j is a non-interactive proof (or argument) generated by $\Pi^{(1)}_{\mathsf{NIZK}}$ for the relation $R^{(1)}_j$ defined in equation (2). Similarly, by the completeness property of $\Pi^{(1)}_{\mathsf{NIZK}}$, all proofs $\{\pi_j\}_{j \in Q}$ will be accepted. Therefore, the trace algorithm returns Q and $\pi_{trace} := (\{F(s)\}_{j \in Q}, \{\pi_j\}_{j \in Q})$.

Verifiability. It holds directly by the verifiability of the Feldman VSS. (Note that, the only difference between the verification in Figure 2 and Feldman scheme is that the parties check if c'_j are broadcast and well-formed. Since V contains honest parties, this extra check does not affect the output of verification algorithm in Figure 2.)

Unpredictability. The unpredictability of the protocol in Figure 2 holds clearly by the unpredictability of the Feldman VSS which is proven under the discrete logarithm assumption. The reason is that, for each i, the extra component of s_i, which is γ_i, is chosen by the party i randomly and, independently from f_i. Therefore, the adversary can sample them itself and broadcast g^{γ_i} for all $i \in Q$ instead of receiving them from the share algorithm. This reduces the unpredictability of Figure 2 scheme to the unpredictability of the Feldman VSS.

Traceability. The Share algorithm given $(1^\lambda, n, t, s)$, where s is a random secret from $\mathbb{Z}_q$, first samples a random degree-t polynomial $f(x) = s + a_1 X + \ldots, a_t X^t$,

with random coefficients from $\mathbb{Z}_q$. Then it outputs $s_i := (f_i, \gamma_i)$, where $f_i = f(i)$, and $\pi_{share} = tk = vk := (c_0, \ldots, c_t, c'_1, \ldots, c'_n)$ where $c_0 = g^s, c_1 = g^{a_1}, \ldots, c_t = g^{a_t}$, and for any $i \in [n]$, $c'_i = g^{\gamma_i}$. Then the adversary given $\{s_i\}_{i\in Q}, \pi_{share}, F(\cdot)$ for a set $Q \subseteq [n]$ returns either $\{z := F(s), \pi_{rec}\}$ or $\{\{z_j := F(f_j)\}_{j\in Q}, \pi_{rec}\}$ where $\pi_{rec} = \{\pi_j, \mathsf{who}\}_{j\in Q}$. The tracing algorithm $\mathsf{VerifTrace}$ in Fig. 2 given the tracing key $tk := \pi_{share} = (c_0, c_1, \ldots, c_t, c'_1, \ldots, c'_n)$, it computes $y_j := \prod_{k=0}^{t} c_k^{j^k}$ for $j \in Q$ (which can be deduced from the n-bit string who) and then verifies the proofs $\pi_{rec} := \{\pi_j, \mathsf{who}\}_{j\in Q}$ using the verification algorithm of $\Pi^{(1)}_{\mathsf{NIZK}}$ and commitment values $\{y_j, c'_j\}_{j\in[n]}$. For any statement $(c_i, c'_i, F(\cdot))$, which the verification algorithm of $\Pi^{(1)}_{\mathsf{NIZK}}$ on at least one of the proofs in $\pi_{rec} := \{\pi_j, \mathsf{who}\}_{j\in Q}$ returns 1, the tracing algorithm puts i in I. If there is no statement $(c_i, c'_i, F(\cdot))$ which has an accepting proof in $\pi_{rec} := \{\pi_j, \mathsf{who}\}_{j\in Q}$, it returns $\{0, \emptyset, 0^n\}$. If I is non-empty, the tracing algorithm returns $1, I, \pi_{trace} = \{\pi_i\}_{i\in I}$. Analyzing the following cases concludes the proof:

- If the tracing algorithm returns $\{0, \emptyset, 0^n\}$, nothing is left to prove.
- Assume that with a non-negligible probability ϵ, there is an index $i^* \in I$ such that $i^* \notin Q$. That is, the adversary does not know the corresponding witness $s_{i^*} := (f_{i^*}, \gamma_{i^*})$ for the statement $(c_{i^*}, c'_{i^*}, F(\cdot))$, however the verification algorithm of $\Pi^{(1)}_{\mathsf{NIZK}}$ on π_{i^*} returns 1 with a non-negligible probability ϵ. Since $\Pi^{(1)}_{\mathsf{NIZK}}$ is knowledge sound (with respect to the Definition A.10), there is an efficient extractor which returns the witness $s_{i^*} := (f_{i^*}, \gamma_{i^*})$ with a non-negligible probability, which breaks the Discrete Logarithm problem.
- It is trivial that Judge in Fig. 2 outputs `true` because it executes the verification algorithm of $\Pi^{(1)}_{\mathsf{NIZK}}$ on $\pi_{trace} = \{\pi_i\}_{i\in I}$ similar to the tracing algorithm.

Non-imputability. The Share algorithm given $(1^\lambda, n, t, s)$, where s is a random secret from $\mathbb{Z}_q$, first samples a random degree-t polynomial $f(x) = s + a_1X + \ldots, a_tX^t$, with random coefficients from $\mathbb{Z}_q$. Then it outputs $s_i := (f_i, \gamma_i)$, where $f_i = f(i)$, and $\pi_{share} = tk = vk := (c_0, \ldots, c_t, c'_1, \ldots, c'_n)$ where $c_0 = g^s, c_1 = g^{a_1}, \ldots, c_t = g^{a_t}$, and for any $i \in [n]$, $c'_i = g^{\gamma_i}$. Then the adversary given $\{s_i\}_{i\in[n]\setminus i^*}, \pi_{share}, F(\cdot)$ returns $(1, I, \pi_{trace} = \{\pi_i\}_{i\in I})$. The Judge algorithm in Figure 2 given the tracing key $vk := \pi_{share} = (c_0, c_1, \ldots, c_t, c'_1, \ldots, c'_n)$, it uses pre-computed values $y_j := \prod_{k=0}^{t} c_k^{j^k}$ for $j \in [n]$ and then verifies the proofs $\pi_{rec} := \{\pi_j\}_{i\in I}$ using the verification algorithm of $\Pi^{(1)}_{\mathsf{NIZK}}$ and commitment values $\{y_j, c'_j\}_{j\in[n]}$. If all the verification pass, it returns `true`. Otherwise, it returns `false`.

Assume that the Judge algorithm returns `true` and $i^* \in I$. That means the adversary has returned an accepting proof π_{i^*} for the statement $(c_{i^*}, c'_{i^*}, F(\cdot))$, without knowing the corresponding witness $s_{i^*} := (f_{i^*}, \gamma_{i^*})$. Since $\Pi^{(1)}_{\mathsf{NIZK}}$ is knowledge sound (with respect to the Definition A.10), there is an efficient extractor which returns the witness $s_{i^*} := (f_{i^*}, \gamma_{i^*})$ with a non-negligible probability, which breaks the Discrete Logarithm problem. This completes the proof. □

Efficiency. The resulting TVSS scheme introduces minimal overhead compared to the performance of the Feldman VSS. During the sharing phase, the dealer

performs the same computations as in the original Feldman VSS. Each party P_i only needs to sample a random secret γ_i and compute a single exponentiation to generate the commitment $c'_i = g^{\gamma_i}$. The size of the individual shares, (f_i, γ_i), is merely twice the size of the shares in the Feldman VSS, or equivalently, twice to the size of the main secret $s = f_0$.

In the verification phase, in the optimistic scenario, each party P_i needs to verify their share f_i using Eq. (1) and also check the well-formedness of commitments c'_k for $k = 1, ..., n$ and $k \neq i$. The former check requires $t + 1$ small exponentiations, while the latter is a straightforward group membership test that can be performed efficiently.

In the reconstruction phase of the new TVSS scheme, each participant may need to use a NIZK proof scheme $\Pi^{(1)}_{\mathsf{NIZK}}$ and publish a proof π_i for the relation $R^{(1)}_i$ (defined in Eq. (2)). However, if the parties reveal the plain values of their shares (as in typical VSS schemes), the proof π_i will be equal to the share $s_i := (f_i, \gamma_i)$. It is important to note that in practical threshold protocols based on VSS, such as DKG schemes or threshold signing or decryption protocols, parties typically do not reveal the plain values of their shares. Instead, they reveal a function of their shares along with a NIZK proof. The efficiency of proof π_i depends on the chosen function $F(\cdot)$ and the specific NIZK proof scheme used, i.e., $\Pi^{(1)}_{\mathsf{NIZK}}$. We will discuss concrete protocols later in this paper. The set of valid proofs constitutes the collective proof π_{rec}, which we will explore in detail in the context of specific protocols in subsequent sections.

During the tracing phase, a tracer needs to verify the proof π_{rec} using the verification algorithm of $\Pi^{(1)}_{\mathsf{NIZK}}$, which has computational cost linear in the number of valid proofs published during the reconstruction phase. Similarly, the efficiency of the Judge algorithm will primarily depend on the time required to verify each proof π_i multiplied by the number of parties in the alleged set I.

4.2 An Efficient TVSS based on Pedersen VSS

Similar to the Feldman VSS scheme, our proposed TVSS scheme in Section 4.1 requires a high entropy secret and achieves unpredictability (defined in Def. 3.4) which is a weaker security notion compared to Secrecy, defined in Def. 3.5.

Pedersen VSS. In [24], Pedersen introduced his well-known commitment scheme and subsequently proposed a variant of the Feldman VSS scheme that can function with a *low-entropy* secret f_0 as well and achieve information-theoretic secrecy. To share a given secret f_0, Pedersen's VSS follows a similar approach to Feldman's scheme, where the dealer commits to the coefficients of the secret polynomial $f(X) := f_0 + a_1X + \cdots + a_tX^t$ using a discrete logarithm-based homomorphic commitment. However, unlike Feldman's scheme, in this case the dealer uses Pedersen commitment. More precisely, the dealer additionally samples another secret polynomial $r(X) := r_0 + b_1X + \cdots + b_tX^t$ and privately sends the shares $s_i := (f_i, r_i) = (f(i), r(i))$ to each party P_i. Then, given two random group generators (g, h), the dealer publishes commitments

$c_0 = g^{f_0} h^{r_0}, c_1 = g^{a_1} h^{b_1}, \ldots, c_t = g^{a_t} h^{b_t}$ as the proof π_{share}. Similar to Feldman VSS, this proof π_{share} enables the parties to verify the validity of their shares during the sharing phase and to validate the published shares during the reconstruction phase. To verify the share $s_i := (f_i, r_i)$, given $\pi_{share} := (c_0, c_1, \ldots, c_t)$, party P_i checks if

$$g^{f_i} h^{r_i} = \prod_{j=0}^{t} c_j^{i^j}, \tag{3}$$

and outputs `true` or `false`, similar to the verification procedure in Feldman VSS. Compared to Feldman's scheme, Pedersen's VSS doubles the size of individual shares, but it allows the protocol to operate with low-entropy secrets and provides a stronger security guarantee. During the reconstruction phase, each party P_j must publish their share $s_i := (f_j, r_j)$. These, published shares are then verified using π_{share} and Eq. (3) and only the valid shares are used to reconstruct the secret f_0.

Traceable Variant of Pedersen VSS. Next, we build a TVSS scheme based on Pedersen VSS by instantiating the underlying VSS scheme Γ' in Figure 1 with Pedersen VSS and similarly the commitment scheme $\mathcal{C}(\cdot)$ using the discrete logarithm. Specifically, as in the Feldman-based TVSS scheme, we define the commitment to each party's random value $\gamma_i \leftarrow\!\$\, \mathbb{Z}_q$ as $c'_i = \mathcal{C}(\gamma_i) := g^{\gamma_i}$, where g is the same group generator used in Pedersen VSS. Compared to the original Pedersen VSS, this modification increases the size of each party's share by 1 filed element, but it facilitates efficient verification and traceability. At the end of sharing phase, as in the original Pedersen VSS scheme, each party also receives secrets (f_i, r_i) from the dealer, and sets their individual share as $s_i := (f_i, r_i, \gamma_i)$. The share verification and conflict resolution phases proceed similarly to the TVSS scheme outlined in Figure 2. The primary difference is that if party P_i raises a complaint against the dealer, the dealer broadcasts both (f_i, r_i) to allow everyone to verify the shares using the verification equation given in Eq. (3).

Similar to the reconstruction phase of our initial TVSS scheme in Figure 2, any set of shareholders can compute the plain value or a function of their shares. Moreover, any set of at least $t + 1$ honest shareholders can also compute the plain value or a function of the main secret, such as $s := f_0$ or $y_0 = F(f_0)$. A key distinction from the initial TVSS scheme (given in Figure 2), is that if parties reveal a function of their shares, such as $z_i = F(f_i)$, they must provide a non-interactive proof that z_i is correctly computed and they know the original share $s_i = (f_i, r_i, \gamma_i)$, generated in the sharing phase. Specifically, each party P_i is required to use a NIZK argument, e.g., $\Pi^{(2)}_{\mathsf{NIZK}}$, and generate a proof for the relation $R_i^{(2)}$, defined as follows:

$$\begin{aligned} R_i^{(2)} = \{(g, F(\cdot), z_i, y_i, c'_i, (f_i, r_i, \gamma_i)) \mid z_i = F(f_i) \wedge \\ y_i = g^{f_i} h^{r_i} \ \wedge \ c'_i = g^{\gamma_i}\}, \end{aligned} \tag{4}$$

where $y_i = g^{f_i} h^{r_i} = \prod_{j=0}^{t} c_j^{i^j}$ can be publicly (pre)computed using π_{share} and the public index i. Similar to the case of initial TVSS scheme from Figure 2, let π_i be the individual NIZK argument published by party P_i, which is generated

by a NIZK proof scheme $\Pi^{(2)}_{\mathsf{NIZK}}$. Depending on the specific function $F(\cdot)$ used, the proof scheme $\Pi^{(2)}_{\mathsf{NIZK}}$ can be constructed using various families of ZK proofs. Similarly, if party P_i reveals the plain values of their shares, then π_i will be equal to $s_i := (f_i, r_i, \gamma_i)$, as the plain values of shares can be verified directly using the proof π_{share}. As with our initial TVSS scheme depicted in Figure 2, at the conclusion of the reconstruction phase, the collective proof π_{rec} can be generated from the set of valid individual proofs and their associated statements. The proof π_{rec} ensures the integrity and correctness of the revealed shares (or a function thereof) and certifies that the reconstruction process was executed correctly by a set of valid shareholders.

Figure 3 outlines the algorithms of the new TVSS scheme based on Pedersen VSS. The security of this scheme is established in the subsequent theorem.

Theorem 4.2 (TVSS based on Pedersen VSS). *Let $\Pi^{(2)}_{\mathsf{NIZK}} := (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ be a NIZK argument for the relation $R^{(2)}_i$ defined in Eq. (4) that satisfies Correctness, and Soundness (or Knowledge Soundness), as defined in Section A.3. Then under the discrete logarithm assumption, the protocol presented in Figure 3 is a secure TVSS scheme that satisfies the properties of Correctness, Verifiability, Statistical Secrecy, Traceability and Non-imputability as defined in Definitions 3.2, 3.3 and 3.5 to 3.7.*

Proof. The proof is given in App. B.1. □

Efficiency. Our second TVSS scheme offers efficiency nearly identical to that of the first scheme presented in Figure 2, with only minimal overhead compared to the performance of the original Pedersen VSS.

In the sharing phase, the dealer operates as in Pedersen VSS, while each party P_i computes a single commitment $c'_i = g^{\gamma_i}$, which requires one exponentiation. The size of individual shares, $s_i := (f_i, r_i, \gamma_i)$, is only three times the size of main secret, increasing by only one field element compared to the original Pedersen VSS scheme. To verify the secrets (f_i, r_i) received from the dealer, each party P_i computes Eq. (3), which requires $t + 1$ small exponentiations. Each party P_i also checks the well-formedness of commitments c'_k for $k = 1, ..., n$ and $k \neq i$; this is a simple group membership test that can be executed efficiently. In the reconstruction phase, each participant may need to use a NIZK argument $\Pi^{(2)}_{\mathsf{NIZK}}$ and publish π_i for the relation $R^{(2)}_i$ (defined in Eq. (4)). The efficiency of $\Pi^{(2)}_{\mathsf{NIZK}}$ depends on the function $F(\cdot)$ and the specific NIZK proof scheme used. As in Figure 2, the set of valid proofs constitutes the collective proof π_{rec}, which we will examine in detail in the context of specific protocols in later sections. During the tracing phase, a tracer verifies π_{rec} using the verification algorithm of $\Pi^{(2)}_{\mathsf{NIZK}}$, which has a computational cost that is linear in the number of valid parties that participated in the reconstruction phase. As in the initial TVSS scheme (from Figure 2), the efficiency of the Judge algorithm will primarily depend on the time required to verify each proof π_i (using the verification algorithm of $\Pi^{(2)}_{\mathsf{NIZK}}$) multiplied by the number of parties in the alleged set I.

Initial: Let $\mathbb{G}$ be a group with hard DL, and two random generators (g, h). The group generators (g, h) and all necessary parameters for the NIZK argument $\Pi^{(2)}_{\mathsf{NIZK}}$ are shared with all the parties.

Share: Given (n, t), and a secret $s := f_0 \in \mathbb{Z}_q$ the protocol proceeds as follows:

- Given secret f_0, the dealer samples a random secret $r_0 \in \mathbb{Z}_q$ and a two random degree-t polynomials $f(x) = f_0 + a_1 X + \ldots, a_t X^t$ and $r(x) = r_0 + b_1 X + \ldots, b_t X^t$, with $f(0) = f_0$, $r(0) = r_0$ and random coefficients $\{a_j, b_j\}_{j=1}^{t}$ from $\mathbb{Z}_q$. Next, the dealer computes $f_i := f(i)$ and $r_i := r(i)$ for $i = 1, \ldots, n$ and privately sends (f_i, r_i) to party P_i. Then, the dealer computes commitments $c_0 = g^{f_0} h^{r_0}, c_1 = g^{a_1} h^{b_1}, \ldots, c_t = g^{a_t} h^{b_t}$ and publishes a proof $\pi_D := (c_0, \ldots, c_t)$.
- In the same round, for $i \in [n]$, each party P_i samples a random secret $\gamma_i \in \mathbb{Z}_q$, and broadcasts the commitment $c'_i := g^{\gamma_i}$.
- At the end, each party P_i stores its individual share $s_i := (f_i, r_i, \gamma_i)$ and the public proof, the tracing and the verification keys $\pi_{share} = tk = vk := (\pi_D, c'_1, \ldots, c'_n) = (c_0, c_1, \ldots, c_t, c'_1, \ldots, c'_n)$.

Verify: Given $\pi_{share} = (c_0, c_1, \ldots, c_t, c'_1, \ldots, c'_n)$, and the shares, i.e., $s_i := (f_i, r_i, \gamma_i)$ for $i = 1, \ldots, n$, the parties proceed as follows:

1) Each party P_i checks if $g^{f_i} h^{r_i} = \prod_{j=0}^{t} c_j^{i^j}$, and broadcasts a complaint against the dealer if the verification failed. Party P_i also check if $c'_k \in \mathbb{G}$ for $k = 1, \ldots, n$, and $k \neq i$. The rest of this step and step 2 are the same as in Figure 2.
3) In case a shareholder P_i complains against the dealer, the dealer will broadcast (f_i, r_i) to enable everyone to verify it using the verification equation from Eq. (3). The rest of this step is the same as in Figure 2.

Reconstruct: Given $F(\cdot)$, which can be the identity function as well, the proof $\pi_{share} := (c_0, c_1, \ldots, c_t, c'_1, \ldots, c'_n)$, and the shares $\{s_i := (f_i, r_i, \gamma_i)\}_{i \in Q}$, the parties act as below:

- Each party P_i, for $i \in Q$, publishes $(z_i = F(f_i), \pi_i)$, where π_i is a non-interactive argument generated by $\Pi^{(2)}_{\mathsf{NIZK}}$ for the relation $R_i^{(2)}$ in equation (4). The proof ensures that z_i is computed correctly and P_i knows the original share $s_i := (f_i, r_i, \gamma_i)$. Again, if $F(\cdot)$ is the identity function, then π_i can be equal to (f_i, r_i, γ_i), as the share itself can be verified using π_{share}.
- Given π_{share}, the published values $\{z_i := F(f_i), \pi_i\}_{i \in Q}$ are verified using the verification algorithm of $\Pi^{(2)}_{\mathsf{NIZK}}$ and the valid ones are collected. Let $\{z_j = F(f_j), \pi_j)\}_{j \in V}$ be the set of valid pairs, V be the set of their indexes, and who be an n-bit string where encodes the indexes in set V and each of its bit indicates whether the corresponding party has provided a valid proof.
- If $|V| = 0$, the protocol returns $\perp$;
- If $t + 1 \leq |V|$: the protocol reconstructs $z := F(f_0)$ and generates associated proof $\pi_{rec} := \{\pi_j, \mathsf{who}\}_{j \in V}$, to show that z is computed correctly the the parties indicated by who and they know $\{s_j\}_{j \in V}$. Finally, it returns $(z = F(f_0), \pi_{rec} = \{\pi_j, \mathsf{who}\}_{j \in V})$. Note that again if $F(\cdot)$ is the identity function, then the proofs $\{\pi_j\}_{j \in V}$ can be the set of shares $\{s_j\}_{j \in V}$.
- If $0 < |V| < t + 1$: the protocol returns $\{z_j = F(f_j)\}_{j \in V}$ and a proof $\pi_{rec} := \{\pi_j, \mathsf{who}\}_{j \in V}$ to show that $\{z_j\}_{j \in V}$ are computed correctly by the parties indicated by who and they know $\{s_j\}_{j \in V}$.

VerifTrace: Given $\{z := F(f_0), \pi_{rec}\}$ or $\{\{z_j := F(f_j)\}_{j \in V}, \pi_{rec}\}$, generated by Reconstruct, and tracing key $tk := \pi_{share} = (c_0, c_1, \ldots, c_t, c'_1, \ldots, c'_n)$, it computes $y_j := \prod_{k=0}^{t} c_k^{j^k}$ for $j \in V$ (which can be deduced from who) and then verifies the proofs $\pi_{rec} := \{\pi_j\}_{j \in V}$ using the verification algorithm of $\Pi^{(2)}_{\mathsf{NIZK}}$ and the commitment values $\{y_j, c'_j\}_{j \in V}$ and determines $I \subseteq V$ as the set of valid ones. Finally, it returns $1, I$ and $\pi_{trace} := (z = F(f_0)$ or $\{z_j = F(f_j)\}_{j \in I}, \{\pi_j\}_{j \in I})$ if $|I| > 0$; otherwise, it returns $\{0, \emptyset, 0^n\}$.

Judge: Given an alleged set I, and a proof $\pi_{trace} := (z = F(f_0)$ or $\{z_j = F(f_j)\}_{j \in I}, \{\pi_j\}_{j \in I})$, and $vk := \pi_{share} = (c_0, c_1, \ldots, c_t, c'_1, \ldots, c'_n)$, using the verification algorithm of $\Pi^{(2)}_{\mathsf{NIZK}}$, it verifies if π_{trace} is valid and outputs `true` or `false`.

Fig. 3. An efficient TVSS scheme based on the Pedersen VSS scheme.

In Section 5, we present concrete traceable threshold protocols based on our new TVSS schemes (given in Figure 2 and 3), detailing their specific reconstruction and tracing algorithms, along with the underlying NIZK proof schemes.

5 Accountable Threshold Cryptosystems

In threshold cryptography [11, 12], the single party (represented by the secret key) is distributed among a group of n parties, such that at least $t+1$ parties are required to recover the secret or perform computations involving the secret key. Particularly, in a threshold cryptosystem, the key difference from a standard cryptosystem $\Pi_{Enc} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, lies in the distribution of the key generation and decryption algorithms, i.e., (KeyGen, Dec). In a standard cryptosystem, a single party generates the key pair $(\mathsf{pk}, \mathsf{sk})$. One can encrypt messages using the public key pk, and decrypt ciphertexts using the private key sk. However, in an (n, t)-threshold cryptosystem, the key generation is distributed among n parties which results in $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i=1}^n)$, each party i receiving a share sk_i of the secret key sk. The decryption process is also collaborative, requiring at least $t+1$ parties to use their individual key shares to produce partial decryptions, which are then combined using the $\mathsf{Combine}$ algorithm to recover the original message. For actively secure (and/or robust) decryption, commitments to the individual shares of the parties are also required, i.e., $\mathsf{pk}_i = \mathcal{C}(\mathsf{sk}_i)$ for $i = 1, \ldots, n$, as these commitments allow the parties to prove that decryption is performed correctly. This distribution of the decryption process enhances both security and fault tolerance, as no single party holds the full secret key sk, reducing the risk of key compromise. Despite these modifications, the encryption algorithm Enc remains the same as in standard cryptosystems, allowing any party to encrypt using the public key pk without requiring access to the threshold-specific operations.

As mentioned above, in a typical (n, t)-threshold cryptosystem, any $t+1$ parties can jointly decrypt a ciphertext. However, like other non-accountable threshold protocols, canonical threshold cryptosystems are vulnerable when parties behave dishonestly. Specifically, due to the lack of accountability, parties might be incentivized to sell their key shares without fear of detection. This vulnerability poses significant risks in various applications of threshold cryptosystems, such as private voting, sealed-bid auctions, etc.

In this section, we discuss how our proposed Feldman-based TVSS scheme (cf. Figure 2) can be leveraged to build an Accountable Threshold Cryptosystem (ATC). Our approach builds on existing non-accountable threshold cryptosystems but incorporates a tracing mechanism that allows a tracer to uniquely identify and trace the parties involved in the decryption process. The proposed protocol is based on the standard ElGamal cryptosystem [14] and can be easily extended to its lifted variant, which supports additive homomorphism. Before presenting our construction, we outline the syntax and security requirements essential for these schemes.

5.1 Syntax and Requirements

An Accountable Threshold Cryptosystem (ATC) is defined as a 6-tuple $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Combine}, \mathsf{VerifTrace}, \mathsf{Judge})$ of PPT algorithms. The algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Combine})$ are inherited from a standard threshold decryption scheme [11, 12], while the algorithms $(\mathsf{VerifTrace}, \mathsf{Judge})$ are designed to

implement traceability in threshold decryption protocols, similar to Traceable Verifiable Secret Sharing (TVSS) schemes (cf. Section 3). We formally define the syntax of ATC schemes as follows, which is adapted from [5, 12],

Definition 5.1 (Accountable Threshold Cryptosystem (ATC)). *An (n, t)-accountable threshold cryptosystem Π_{ATC} over message space M and ciphertext space C consists of the following PPT algorithms:*

- $\mathsf{KeyGen}(1^\lambda, n, t) \to (\mathsf{pk}, \{\mathsf{sk}_i\}_{i=1}^n, \mathsf{vk}, \mathsf{tk})$*: Given the security parameter λ in its unary representation, and integers (n, t), this algorithm generates and returns a public key* pk*, individual secret shares $\{\mathsf{sk}_i\}_{i=1}^n$, a pair of verification and tracing keys* $(\mathsf{vk}, \mathsf{tk})$*.*
- $\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$*: Given a public key* pk *and a message $m \in M$ as inputs, this algorithm returns a ciphertext $\mathsf{ct} \in C$.*
- $\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct}) \to (d_i, \pi_i)$*: Given a secret key share* sk_i *and a ciphertext* ct*, the decryption algorithm returns a partial decryption d_i along with an associated proof π_i, which is generated to prove knowledge of s_i and the correctness of the partial decryption value d_i.*
- $\mathsf{Combine}(\{(d_i, \pi_i)\}_{i\in Q}, \mathsf{ct}, \mathsf{vk}) \to (\{m, \pi_{dec}\}/\{\{d_j\}_{j\in V}, \pi_{dec}\}/\perp)$*: Given the partial decryption values along with the associated proofs for a set of $Q \subseteq [n]$, i.e. $\{(d_i, \pi_i)\}_{i\in Q}$, the verification key* vk*, and the ciphertext* $\mathsf{ct} \in C$*, the algorithm first verifies the proofs $\{\pi_i\}_{i\in Q}$ based on $\{d_i\}_{i\in Q}$, and stores the set of valid pairs, i.e., $\{(d_i, \pi_i)\}_{i\in V\subseteq Q}$. From the set of valid pairs, it forms an n-bit string* who*, where each bit indicates whether the corresponding party has provided a valid proof.*[7] *It then acts as follows:*
  - *If $|V| = 0$, it returns $\perp$.*
  - *If $|V| \geq t + 1$, it uses $\{d_j\}_{j\in V}$ to decrypt the ciphertext* ct*, computes the message m, and generates an associated proof π_{dec}, and returns (m, π_{dec}). Additionally, the string* who *is included in the proof π_{dec}.*
  - *If $0 < |V| < t + 1$, it returns the set $\{d_j\}_{j\in V}$ along with an associated proof π_{dec} for their validity. Similarly, the string* who *is included in π_{rec}.*
- $\mathsf{VerifTrace}((m, \pi_{dec})/(\{d_j\}_{j\in V}, \pi_{dec}), \mathsf{ct}, \mathsf{tk}) \to (b, I, \pi_{trace})$*: Given a message $m \in M$, or a set of decryption shares $\{d_j\}_{j\in V}$ and the corresponding proof π_{dec} generated by the* $\mathsf{Combine}$ *algorithm, i.e., $(m, \pi_{dec})/(\{d_j\}_{j\in V}, \pi_{dec})$, the ciphertext $\mathsf{ct} \in C$, along with the tracing key* tk*, the algorithm first verifies π_{dec}. If the verification fails, it returns $\{0, \emptyset, 0^n\}$. Otherwise, it identifies a subset $I \subseteq V$ of valid identities and returns $(1, I, \pi_{trace})$, where π_{trace} is the proof of valid tracing.*
- $\mathsf{Judge}(I, \pi_{trace}, \mathsf{vk}) \to$ `true/false`*: Given a subset $I \in [n]$, a proof π_{trace}, and a verification key* vk*, the algorithm outputs either* `true`*, indicating that the proof confirms that the parties in I are accurately traced, or* `false`*, indicating rejection.*

[7] Similar to the case of TVSS (given in Def. 3.1), for instance, if $\mathsf{who} = 010011$, it means that parties 1, 2, and 5 have provided valid proofs and participated in the decryption. Similarly, the string who is defined solely for efficiency purposes.

1. $(Q, \mathsf{st}) \leftarrow \mathcal{A}\left(1^\lambda, n, t\right)$;
2. $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{vk}, \mathsf{tk}) \leftarrow \mathsf{KeyGen}(1^\lambda, n, t)$;
3. $(m_0, m_1) \leftarrow \mathcal{A}\left(\mathsf{st}, \{\mathsf{sk}_i\}_{i\in Q}, \mathsf{pk}, \mathsf{vk}, \mathsf{tk}\right)$;
4. $b \leftarrow \{0,1\},\ \mathsf{ct}_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$;
5. $b' \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{pk}, \mathsf{vk}, \mathsf{ct}_b)$;
6. **return** $(b == b' \ \wedge\ |Q| < t+1)$;

Fig. 4. $\mathbf{Game}_{\mathcal{A}}^{\mathrm{CPA}}$: The IND-CPA security game.

Security Requirements. An accountable threshold cryptosystem Π_{ATC} must satisfy the following security properties:

Definition 5.2 (Correctness). *For a given integer $n > 1$ and $t < n$, an ATC Π_{ATC} is called correct, if we have:*

$$\Pr\begin{bmatrix}\forall m \in M, Q \subseteq [n], |Q| \geq t+1; (\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{vk}, \mathsf{tk}) \leftarrow \mathsf{KeyGen}(1^\lambda, n, t);\\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m); (m', \pi_{dec}) \leftarrow \mathsf{Combine}(\{\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct})\}_{i\in Q}, \mathsf{ct}, \mathsf{vk}) : m' = m\end{bmatrix} = 1$$

Additionally,

$$\Pr\begin{bmatrix}\forall m \in M, Q \subseteq [n], (\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{vk}, \mathsf{tk}) \leftarrow \mathsf{KeyGen}(1^\lambda, n, t); \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m);\\ (1, Q', \pi_{trace}) \leftarrow \mathsf{VerifTrace}(\mathsf{Combine}\left(\{\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct})\}_{i\in Q}, \mathsf{ct}, \mathsf{vk}\right), \mathsf{ct}, \mathsf{tk}) :\\ Q' = Q\end{bmatrix} = 1$$

Definition 5.3 (IND-CPA Security). *Let the advange of an adversary $\mathcal{A}$ against the IND-CPA security $\mathbf{Game}_{\mathcal{A}}^{CPA}$ (cf. Figure 4), be defined as follows:*

$$\mathsf{Adv}_{\mathcal{A}}^{CPA}(\lambda) = \Pr\left[\mathbf{Game}_{\mathcal{A}}^{CPA} = 1\right] .$$

An ATC Π_{ATC}, is called IND-CPA secure if for all PPT adversaries $\mathcal{A}$, we have:

$$\mathsf{Adv}_{\mathcal{A}}^{CPA}(\lambda) \leq \tfrac{1}{2} + \mathsf{negl}(\lambda) .$$

Definition 5.4 (Traceability). *For $n > 1$ and $t < n$, an ATC Π_{ATC} satisfies the traceability if for all adversaries $\mathcal{A}$, and any set $Q \subseteq [n]$, we have:*

$$\Pr\begin{bmatrix}(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{vk}, \mathsf{tk}) \leftarrow \mathsf{KeyGen}(1^\lambda, n, t);\\ (\mathsf{ct}, (m, \pi_{dec})/(\{d_j\}_{j\in Q}, \pi_{dec})) \leftarrow \mathcal{A}(1^\lambda, n, t, \mathsf{pk}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i\in Q});\\ (b, I, \pi_{trace}) \leftarrow \mathsf{VerifTrace}((m, \pi_{dec})/(\{d_j\}_{j\in Q}, \pi_{dec}), \mathsf{ct}, \mathsf{tk}) :\\ (b = 1 \wedge I \subseteq Q \wedge \mathsf{Judge}(I, \pi_{trace}, \mathsf{tk}) = \mathtt{true}) \vee (b = 0)\end{bmatrix} \geq 1 - \mathsf{negl}(\lambda) .$$

Definition 5.5 (Non-imputability). *For any integers $n > 1$ and $t < n$, an ATC Π_{ATC} satisfies the non-imputability if for all adversaries $\mathcal{A}$ and ciphertext $\mathsf{ct} \in C$, we have:*

$$\Pr\begin{bmatrix}(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{vk}, \mathsf{tk}) \leftarrow \mathsf{KeyGen}(1^\lambda, n, t);\\ (1, I, \pi_{trace}) \leftarrow \mathcal{A}(1^\lambda, n, t, \mathsf{pk}, \mathsf{vk}, \mathsf{tk}, \{\mathsf{sk}_i\}_{i\in[n]\setminus i^*}, \mathsf{ct}) :\\ \mathsf{Judge}(I, \pi_{trace}, \mathsf{vk}) = \mathtt{true} \wedge i^* \in I\end{bmatrix} \leq \mathsf{negl}(\lambda) .$$

Setup: Given 1^λ, the parties sample (or agree on) a cyclic group $\mathbb{G}$ with hard DL and two random generators (g, h), and then share $pp := (g, h)$ with all the parties.

P: Given the public parameters $pp := (g, h)$, the statement $x := (d_i, c, \mathsf{pk}_i)$, and witness $w := (f_i, \gamma_i) \in \mathbb{Z}_q$, the prover acts as follows:

- Samples $(r_1, r_2) \leftarrow\!\$\, \mathbb{Z}_q$ and commits to them as $R_1 = c^{r_1}$ and $R_2 = g^{r_1} h^{r_2}$.
- Computes the challenge $e = \mathcal{H}(d_i, c, \mathsf{pk}_i, R_1, R_2)$, where $\mathcal{H} : \{0,1\}^\star \to \mathbb{Z}_q$ is modeled as a random oracle.
- Finally, computes $z_1 = r_1 + e f_i$ and $z_2 = r_2 + e\gamma_i$ and publishes the proof $\pi := (R_1, R_2, z_1, z_2)$.

V: Given the public parameters $pp := (g, h)$, the statement $x := (d_i, c, \mathsf{pk}_i)$, and the proof $\pi = (R_1, R_2, z_1, z_2)$, the verifier computes the challenge value $e = \mathcal{H}(d_i, c, \mathsf{pk}_i, R_1, R_2)$ and checks if:

$$c^{z_1} \stackrel{?}{=} R_1\ d_i^e \quad \text{and} \quad g^{z_1} h^{z_2} \stackrel{?}{=} R_2\ \mathsf{pk}_i^e$$

and returns `true` if the check passes, and `false` otherwise.

Fig. 5. $\Pi^{Dec}_{\mathsf{NIZK}}$: An efficient NIZK argument of knowledge for R_i^{Dec}, as defined in Eq. (5).

5.2 An Efficient Accountable Threshold Cryptosystem

Next, we introduce an efficient accountable threshold cryptosystem $\Pi^{ElGamal}_{ATC}$ based on the ElGamal cryptosystem [14]. This protocol leverages the TVSS scheme we previously defined and built in Section 4.1, and integrates accountability features into the threshold decryption process.

Concrete Example of $\Pi^{(1)}_{\mathsf{NIZK}}$ for Use in $\Pi^{ElGamal}_{ATC}$. Before presenting the full construction of new protocol $\Pi^{ElGamal}_{ATC}$, we first construct a concrete NIZK proof scheme, denoted as $\Pi^{Dec}_{\mathsf{NIZK}}$, which is essential for ensuring both the correctness and traceability of the decryption process. The NIZK argument $\Pi^{Dec}_{\mathsf{NIZK}}$ is a specific instantiation of $\Pi^{(1)}_{\mathsf{NIZK}}$ used in the reconstruction phase of our initial TVSS scheme. More precisely, $\Pi^{Dec}_{\mathsf{NIZK}}$ is tailored to a special case of the relation $R_i^{(1)}$ defined in Eq. (2), with two key modifications. First, the function $F(\cdot)$ is instantiated as the exponentiation function, meaning $d_i = c^{f_i}$, where $c \in \mathbb{G}$ and $f_i \in \mathbb{Z}_q$. Second, to achieve a better efficiency, the commitment values $y_i = g^{f_i}$ and $c'_i = g^{\gamma_i}$ are combined using an additional random group generator $h \in \mathbb{G}$, resulting in $\mathsf{pk}_i = g^{f_i} h^{\gamma_i}$.[8]

In summary, the new NIZK argument $\Pi^{Dec}_{\mathsf{NIZK}}$ allows a prover to generate a proof π for the relation R_i^{Dec}, defined as follows:

$$R_i^{Dec} = \{(g, h, c, d_i, \mathsf{pk}_i, (f_i, \gamma_i)) \mid d_i = c^{f_i} \ \wedge \mathsf{pk}_i = g^{f_i} h^{\gamma_i}\}, \tag{5}$$

where $x := (g, h, c, d_i, \mathsf{pk}_i) \in \mathbb{G}$ is the statement, and $w := (f_i, \gamma_i) \in \mathbb{Z}_q$ is the witness value.

[8] Note that this optimization can be avoided, though it would double the size of verification and tracing keys, i.e., $\{c'_i, y_i\}_{i\in[n]}$ instead of $\{\mathsf{pk}_i\}_{i\in[n]}$, and turn $\Pi^{Dec}_{\mathsf{NIZK}}$ into a proof of knowledge protocol, rather than an argument of knowledge.

We present the details of the NIZK argument $\Pi^{Dec}_{\mathsf{NIZK}}$ in Figure 5. This argument is constructed by combining a Schnorr Σ-protocol with an additional Σ-protocol to prove knowledge of the values (f_i, γ_i) in the Pedersen commitment $\mathsf{pk}_i = g^{f_i} h^{\gamma_i}$. The protocol is then converted into a non-interactive argument of knowledge using the Fiat-Shamir transform in the random oracle model.

Intuitively, the new NIZK argument enables each participant in the decryption protocol to prove that their partial decryption has been done correctly, while also showing that they possess knowledge of their individual shares $s_i := (f_i, \gamma_i)$.

Theorem 5.1 (NIZK Argument of Knowledge for R_i^{Dec}). *Let $\mathcal{H}$ be a random oracle, defined as $\mathcal{H} : \{0,1\}^\star \to \mathbb{Z}_q$. Under the discrete logarithm assumption, the proof scheme $\Pi^{Dec}_{\mathsf{NIZK}}$, as outlined in Figure 5, is a secure NIZK argument of knowledge for the relation R_i^{Dec} in the random oracle model.*

Proof. The proof is given in App. B.2. □

Construction of the New ATC. Figure 6 outlines the construction of our proposed ATC $\Pi^{ElGamal}_{ATC}$. Essentially, this scheme incorporates traceability into a standard ElGamal-based threshold decryption scheme. The integration is achieved through the use of the NIZK argument $\Pi^{Dec}_{\mathsf{NIZK}}$ defined earlier (cf. Figure 5), ensuring that only authorized participants can contribute to decryption while preserving the confidentiality of their inputs. This scheme combines the security of ElGamal encryption with enhanced traceability, providing both correctness and accountability throughout the decryption process.

Theorem 5.2 (Accountable Threshold Decryption). *Let $\Pi^{Dec}_{\mathsf{NIZK}} := (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ be a NIZK proof scheme for the relation R_i^{Dec}, defined in Eq. (5) that satisfies Correctness and Soundness (or Knowledge Soundness), as defined in Section A.3 and ElGamal cryptosystem guarantees correctness and IND-CPA security, as defined in Section A.4. Then, the proposed scheme in Figure 6 is a secure ATC scheme and satisfies correctness, IND-CPA, traceability and non-imputability properties, defined in Definitions 5.2 to 5.5.*

Proof. The proof is given in App. B.3. □

Distributed Key Generation (for $\Pi^{ElGamal}_{ATC}$). Our definition of ATC (as given in Definition 3.1) and the construction proposed in Figure 6 assume a centralized key generation process. However, just as a standard VSS scheme can be extended into a Distributed Key Generation (DKG) protocol, like the Pedersen DKG protocol [17, 24], our proposed TVSS schemes from Sections 4.1 and 4.2 can similarly be adapted to create a (biased or fully secure) DKG protocol for $\Pi^{ElGamal}_{ATC}$, executed by n decryptors.

Using a DKG protocol, the secret polynomial and the corresponding commitments to the shares are generated in a fully distributed manner by the parties themselves, rather than by a trusted authority. A standard approach, based on Feldman (or Pedersen) VSS schemes, is described in [23]. In this approach, each

KeyGen: Given 1^λ, and the integer values (n, t), the algorithm proceeds as follows:
- It samples a cyclic group $\mathbb{G}$ with a prime order q and two random generators (g, h), where DL is hard. It then sets $pp := (g, h)$ as the public parameters for the NIZK proof scheme $\Pi^{Dec}_{\mathsf{NIZK}}$.
- Given n, for $i = 1, \ldots, n$, it samples a random secret share $\gamma_i \in \mathbb{Z}_q$.
- Given (n, t), it samples a random secret key $f_0 \in \mathbb{Z}_q$, and then secret shares it using Shamir's scheme and obtains the shares $\{f_i\}_{i=1}^n$.
- For $i = 1, \ldots, n$, it sets the individual shares $\mathsf{sk}_i := (f_i, \gamma_i)$ and then computes Pedersen commitments $\mathsf{pk}_i := g^{f_i} h^{\gamma_i}$.
- It sets $\mathsf{pk} = g^{f_0}$, and $\mathsf{vk} := \mathsf{tk} := (pp, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, and returns $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i=1}^n, \mathsf{vk}, \mathsf{tk})$.

Note that, at the end of the key generation phase, the encryptor only stores the public key pk, while each shareholder i learns their individual share $\mathsf{sk}_i = (f_i, \gamma_i)$, in addition to the public parameters.

Enc: Given the public key pk and a message $m \in \mathbb{G}$, it first samples a randomizer $r \leftarrow \mathbb{Z}_q^*$ and then computes and returns the ElGamal ciphertext $\mathsf{ct} := (c_1, c_2) = (m \cdot \mathsf{pk}^r, g^r)$.

Dec: Given a ciphertext $\mathsf{ct} = (c_1, c_2)$ and a secret share $\mathsf{sk}_i = (f_i, \gamma_i)$, it first computes the partial decryption value $d_i := c_2^{f_i}$. Then, using $pp := (g, h)$, the statement $x := (d_i, c_2, \mathsf{pk}_i)$, and the witness value $w := (f_i, \gamma_i)$, it runs the NIZK argument of knowledge $\Pi^{Dec}_{\mathsf{NIZK}}$ (outlined in Figure 5) and obtains a proof π_i. Finally, it returns (d_i, π_i).

Combine: Given the partial decryption values and the proof $\{(d_i, \pi_i)\}_{i \in Q}$ for the set $Q \subseteq [n]$, the verification key $\mathsf{vk} := (pp, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, and the ciphertext $\mathsf{ct} = (c_1, c_2)$, it first verifies $\{(d_i, \pi_i)\}_{i \in Q}$ and stores the set of valid pairs, i.e., $\{(d_i, \pi_i)\}_{i \in V \subseteq Q}$. From the set of valid pairs, the algorithm constructs an n-bit string who, where each bit indicates whether the corresponding party has provided a valid proof. Then, the algorithm acts as follows:
- If $|V| = 0$, returns $\perp$.
- If $|V| \geq t + 1$, for each $i \in V$, it computes the Lagrange coefficients L_i. Then, it computes $m = c_1 / \left(\prod_{i \in V} d_i^{L_i}\right)$. Finally, it returns (m, π_{dec}), where π_{dec} consists of all the proofs provided by the decryptors in V and the string who, i.e., $\{\pi_i, \mathsf{who}\}_{i \in V}$.
- If $0 < |V| < t + 1$, it returns $(\{d_i\}_{i \in V}, \pi_{dec})$, where $\pi_{dec} = \{\pi_i, \mathsf{who}\}_{i \in V}$.

VerifTrace: Given a message $m \in M$ or a set of decryption shares $\{d_j\}_{j \in V}$ (where the set V can be deduced from the string who), the corresponding proof π_{dec} generated by the $\mathsf{Combine}$ algorithm, the ciphertext $\mathsf{ct} := (c_1, c_2)$, along with the tracing key $\mathsf{tk} := (pp, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, the algorithm first verifies $\pi_{dec} = \{\pi_i, \mathsf{who}\}_{i \in V}$ using the verification algorithm V (outlined in Figure 5). If the verification fails for all indexes $j \in V$, it returns $\{0, \emptyset, 0^n\}$. Otherwise, it identifies the set of valid ones $I \subseteq V$ and returns $(1, I, \pi_{trace})$, where $\pi_{trace} := \{\mathsf{ct}, d_k, \pi_k\}_{k \in I}$, and $I \subseteq V$.

Judge: Given a set of decryptors I, a proof $\pi_{trace} := \{\mathsf{ct}, d_k, \pi_k\}_{k \in I}$, and the verification key $\mathsf{vk} := (pp, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, it runs the verification algorithm V (outlined in Figure 5) for all $k \in I$ and returns `true` if all the checks passed. Otherwise, it returns `false`.

Fig. 6. $\Pi^{ElGamal}_{ATC}$: An efficient accountable cryptosystem based on ElGamal encryption.

party P_i acts as the dealer once, generating a polynomial $f^{(i)}$ of the correct degree. They privately send the share $f^{(i)}(j)$ to party P_j for $j \in \{1, \ldots, n\} \setminus \{i\}$. Each party then computes their own share by summing all the shares they have received. This process implicitly defines the secret polynomial $f = \sum_{i=1}^n f^{(i)}$, with each player's share $f(j) = \sum_{i=1}^n f^{(i)}(j)$ being the sum of their individual shares. The secret value is implicitly defined as $f_0 = f(0) = \sum_{i=1}^n f^{(i)}(0)$, which remains unknown to all players, assuming an honest-majority setting and proper protocol execution.

Since some parties (up to t) may behave maliciously, it is crucial that all parties verify the correctness of the shares they receive, ensuring they originate from a polynomial of degree at most t. If a malicious adversary uses a polynomial of a degree greater than t, both verification and reconstruction will fail. Once the verification process is completed using the Feldman VSS scheme, all (or a subset of non-disqualified) parties use their individual public keys $h_i = g^{s_i}$ for

$i = 1, \ldots, n$ to compute the final public key pk, given by

$$\mathsf{pk} = g^{f_0} = g^{f(0)} = \prod_{i \in Q} g^{L_{0,i}^{Q} s_i},$$

where $L_{0,i}^{Q} = \prod_{j \in Q,\ j \neq i} \frac{j}{j-i}$ is a Lagrange coefficient, and Q denotes a qualified set of parties.

Our analysis shows that this approach can also be applied to our proposed TVSS schemes (described in Figure 2 and Figure 3). In other words, the new TVSS schemes can similarly be used to construct a (biased or fully secure) robust DKG protocol for ATC $\Pi_{ATC}^{ElGamal}$ (as described in Figure 6), allowing the n decryptors to generate the parameters $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i=1}^{n}, \mathsf{vk}, \mathsf{tk})$ in a fully distributed manner. At the end of the DKG protocol, each party (i.e., decryptor) will learn their individual share $\mathsf{sk}_i := (f_i, \gamma_i)$ along with the public elements $(\mathsf{pk}, \mathsf{vk}, \mathsf{tk})$.

Efficiency and Comparisons. Although our proposed ATC based on ElGamal decryption, $\Pi_{ATC}^{ElGamal}$, introduces accountability, it adds minimal overhead compared to *a non-accountable*, actively secure, and robust threshold cryptosystem based on the ElGamal cryptosystem. Recall that in such a non-accountable threshold decryption protocol, parties still provide a NIZK proof for the correctness of decryption, but it is slightly (i.e., about 20%) more efficient than the NIZK proof scheme $\Pi_{\mathsf{NIZK}}^{Dec}$ used in our ATC. The reason is that in the non-accountable case, each party uses the Chaum-Pedersen DL equality NIZK protocol [7] and generates a proof to show that $d_i = c^{f_i} \wedge \mathsf{pk}_i = g^{f_i}$, instead of generating a NIZK proof for the relation R_i^{Dec}, defined in relation (5).

To be more precise, compared to an actively secure and robust threshold cryptosystem based on the ElGamal cryptosystem, during the key generation of our new ATC, the parties need to generate the elements $(pp, \mathsf{pk}, \mathsf{pk}_1 := g^{f_1} h^{\gamma_1}, \cdots, \mathsf{pk}_n := g^{f_n} h^{\gamma_n})$ instead of $(pp, \mathsf{pk}, \mathsf{pk}_1 := g^{f_1}, \cdots, \mathsf{pk}_n := g^{f_n})$, which has the same computational cost asymptotically. We see that the public key size remains the same in both cases. Note that in the new ATC, both $(\mathsf{vk}, \mathsf{tk})$ are equal to $(pp, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, and in both schemes, the encryptor only requires (pp, pk).

The encryption phase operates identically to the standard ElGamal scheme. In the decryption phase, each party P_i computes a partial decryption d_i, similar to the standard ElGamal scheme. However, as in the non-accountable actively secure threshold decryption protocol based on ElGamal, each party also generates a proof π_i using $\Pi_{\mathsf{NIZK}}^{Dec}$, which, as mentioned earlier, has asymptotically the same cost as in the non-accountable case, and concretely adds a constant number of exponentiations in group $\mathbb{G}$ to each party's cost.

When combining the partial decryptions, as in the non-accountable actively secure case, the validity of the proofs $\{\pi_i\}_{i \in Q}$ is verified using the public keys and the verification algorithm for $\Pi_{\mathsf{NIZK}}^{Dec}$. This again has asymptotically the same cost as in the non-accountable case, and concretely requires $5|Q|$ exponentiations instead of $4|Q|$ (in Chaum-Pedersen DL equality proof [7]). Once valid proofs $|V| \geq t+1$ are collected, similar to threshold ElGamal, each party computes the Lagrange coefficients and performs $|V|$ exponentiations and $|V|+1$ multiplications to obtain the message m.

In our case, to achieve traceability, this computation is done even if $|V| < t+1$, and parties cannot recover the message m. Finally, the trace and judge algorithms, similar to the verification part of the Combine algorithm, require to run the verification of $\Pi^{Dec}_{\mathsf{NIZK}}$ which requires at most $5|V|$ exponentiations, where $|V|$ is the number of valid parties participating in the decryption phase.

Related Work. As mentioned in introduction, in a recent related work by Boneh, Partap, and Rotem [5] at Crypto'24, the authors studied the threshold decryption protocols in the case where dishonest parties may collude to sell their decryption shares without fear of identification. This is quite relevant to our studied research question in this section, as in our proposed ATC protocol the decryption process is accountable and the participants in the decryption phase can be uniquely identified. In [5], the authors define a decoder function $D(\cdot)$, which is created by a quorum of $t+1$ or more parties and can decrypt ciphertexts. The tracing algorithm with a black-box access to this decoder function identifies some members of the malicious quorum. The paper adapts ideas from traditional traitor tracing schemes [8], where individual secret keys can be traced to those who create unauthorized decoders. To extend this idea to threshold setting, the authors present several new cryptographic techniques, including a confirmation algorithm that allows a verifier to confirm that a given subset of parties was involved in creating the malicious decoder. This is particularly relevant in practical applications like blockchain encrypted mempools, where validators might be bribed to reveal decryption keys prematurely.

A key technical difference between our ATC and their scheme lies in the tracing mechanisms. In their scheme, the tracing algorithm requires black-box access to a decoder box, which is queried with (at least two) malformed ciphertexts. The decoder successfully decrypts some ciphertexts and outputs "fail" on others. They, show that this success and failure pattern reveals at least one traitor whose key was used to create construct the decoder box. In contrast, our scheme does not rely on querying the decryption process with several (malformed) ciphertexts. Instead, our ATC protocol embeds accountability directly into each decryptor's decryption share by attaching a proof that validates both the correctness of the partial decryption and the identity of the decryptor. Intuitively, this can be seen as an accountable signature published by each party participated in the decryption protocol. As a result, our protocol is ready to use in practical applications and any set of decryptors involved in the decryption phase can be traced and held accountable without the need for additional tracing queries.

6 Conclusion

In conclusion, we introduced the concept of Traceable Verifiable Secret Sharing (TVSS), extending the traditional Verifiable Secret Sharing (VSS) and Traceable Secret Sharing (TSS) schemes. Our construction builds upon Shamir's Secret Sharing scheme, ensuring both the verifiability and traceability of shares, even in the presence of malicious dealers or shareholders. The concept of traceability provides accountability by enabling the identification of participants involved

in the secret reconstruction process, deterring malicious behavior such as share selling. We proposed a general strategy to build TVSS schemes from VSS protocols, and based on that, proposed new variants of the well-known VSS schemes by Feldman [15] and Pedersen [24] that can achieve traceability. The resulting TVSS schemes can be used to build various accountable threshold protocols. As an example, we demonstrated how one of our proposed TVSS schemes can be used to construct a practical accountable threshold cryptosystem based on ElGamal cryptosystem, illustrating the broader applicability of TVSS in threshold cryptography. We believe the underlying technique is general enough to be used by other cryptosystems as well.

Future work could focus on improving the efficiency of our proposed protocols and exploring further applications of new TVSS schemes. We believe that in some concrete cases, it may be possible to maintain accountability while aggregating the individual proofs generated during the reconstruction phase of TVSS schemes to create more compact proofs. We leave this as an interesting future research question. Our general strategy and constructed TVSS schemes are designed for honest-majority and asynchronous networks. Further research could explore the construction of similar protocols in asynchronous networks.

Acknowledgments

This work was supported by the Flemish Government through the Cybersecurity Research Program with grant number VOEWICS02 and through SolidLab Flanders (Flemish Government, EWI).

References

1. Shahla Atapoor, Karim Baghery, Daniele Cozzo, and Robi Pedersen. Practical robust DKG protocols for CSIDH. In Mehdi Tibouchi and Xiaofeng Wang, editors, *ACNS 23: 21st International Conference on Applied Cryptography and Network Security, Part II*, volume 13906 of *Lecture Notes in Computer Science*, pages 219–247, Kyoto, Japan, June 19–22, 2023. Springer, Cham, Switzerland.
2. Shahla Atapoor, Karim Baghery, Daniele Cozzo, and Robi Pedersen. VSS from distributed ZK proofs and applications. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part I*, volume 14438 of *Lecture Notes in Computer Science*, pages 405–440, Guangzhou, China, December 4–8, 2023. Springer, Singapore, Singapore.
3. Karim Baghery. A unified framework for computational verifiable secret sharing. In Tibor Jager and Jiaxin Pan, editors, *Public-Key Cryptography - PKC 2025 - 28th IACR International Conference on Practice and Theory of Public-Key Cryptography, Roros, Norway May 12-15, 2025, Proceedings*, Lecture Notes in Computer Science.
4. Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In Shafi Goldwasser, editor, *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 37–56, Santa Barbara, CA, USA, August 21–25, 1990. Springer, New York, USA.

5. Dan Boneh, Aditi Partap, and Lior Rotem. Accountability for misbehavior in threshold decryption via threshold traitor tracing. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part VII*, volume 14926 of *Lecture Notes in Computer Science*, pages 317–351, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.
6. Dan Boneh, Aditi Partap, and Lior Rotem. Traceable secret sharing: Strong security and efficient constructions. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part V*, volume 14924 of *Lecture Notes in Computer Science*, pages 221–256, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.
7. David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Berlin, Heidelberg, Germany.
8. Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In Yvo Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 257–270, Santa Barbara, CA, USA, August 21–25, 1994. Springer, Berlin, Heidelberg, Germany.
9. Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *26th Annual Symposium on Foundations of Computer Science*, pages 383–395, Portland, Oregon, October 21–23, 1985. IEEE Computer Society Press.
10. Ivan Damgård and Maciej Koprowski. Practical threshold RSA signatures without a trusted dealer. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 152–165, Innsbruck, Austria, May 6–10, 2001. Springer, Berlin, Heidelberg, Germany.
11. Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 120–127, Santa Barbara, CA, USA, August 16–20, 1988. Springer, Berlin, Heidelberg, Germany.
12. Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315, Santa Barbara, CA, USA, August 20–24, 1990. Springer, New York, USA.
13. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
14. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Berlin, Heidelberg, Germany.
15. Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science*, pages 427–437, Los Angeles, CA, USA, October 12–14, 1987. IEEE Computer Society Press.
16. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Berlin, Heidelberg, Germany.

17. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007.
18. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.
19. Vipul Goyal, Yifan Song, and Akshayaram Srinivasan. Traceable secret sharing and applications. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 718–747, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.
20. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326, Vienna, Austria, May 8–12, 2016. Springer, Berlin, Heidelberg, Germany.
21. Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Aggregatable distributed key generation. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 147–176, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland.
22. Somayeh Heidarvand and Jorge L. Villar. Public verifiability from pairings in secret sharing schemes. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *SAC 2008: 15th Annual International Workshop on Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 294–308, Sackville, New Brunswick, Canada, August 14–15, 2009. Springer, Berlin, Heidelberg, Germany.
23. Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract) (rump session). In Donald W. Davies, editor, *Advances in Cryptology – EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526, Brighton, UK, April 8–11, 1991. Springer, Berlin, Heidelberg, Germany.
24. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Berlin, Heidelberg, Germany.
25. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1990. Springer, New York, USA.
26. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.
27. Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
28. Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220, Bruges, Belgium, May 14–18, 2000. Springer, Berlin, Heidelberg, Germany.

A Preliminaries

A.1 Secret Sharing and Shamir's Scheme

Secret sharing schemes allow a secret to be divided into multiple shares, enabling any quorum of shareholders to reconstruct the secret, while smaller groups are unable to learn anything about the main secret.

Definition A.1 (Secret Sharing [27]). *For any two positive integers (n, t), an (n, t)-secret sharing scheme consists of two PPT algorithms,* Share *and* Reconstruct. Share *is a randomized function that takes a secret $s \in \mathbb{Z}_p$, and outputs* $\mathsf{Share}(n, t, s) \to (s_1, \ldots, s_n)$. *Two main properties are correctness and security, defined as follows:*

- ***Correctness****: For any s and a set of parties $Q \subseteq [n]$ s.t. $|Q| > t$, we have,*

$$\Pr\begin{bmatrix}\forall s, (s_1, \ldots, s_n) \leftarrow \mathsf{Share}(1^\lambda, n, t, s) \\ s' \leftarrow \mathsf{Reconstruct}\left(\{s_i\}_{i\in Q}\right) : s' = s\end{bmatrix} = 1 \ .$$

- ***Security****: For any secret s and a set of parties $Q \subseteq [n]$, for all unbounded adversaries $\mathcal{A}$ we have,*

$$\Pr\begin{bmatrix}Q \leftarrow \mathcal{A}(1^\lambda, n, t), (s_1, \ldots, s_n) \leftarrow \mathsf{Share}(1^\lambda, n, t, s) \\ s^* \leftarrow \mathcal{A}(\{s_i\}_{i\in Q}) : |Q| \leq t \wedge s^* = s\end{bmatrix} = 1/p \ .$$

Shamir Secret Sharing (SSS) scheme [26] is a widely used protocol, which is formally defined below.

Definition A.2 (Shamir Secret Sharing (SSS) [26]). *In an (n, t)-SSS, to distribute a secret $s \in \mathbb{Z}_p$ among n parties, the dealer creates a random polynomial $f(x) \in \mathbb{Z}_p[X]$ of degree t such that $f(0) = s$. The dealer then evaluates $f(x)$ at each shareholder's public index and securely provides each shareholder with their share $s_i = f(i)$ for $i \in [n]$. To reconstruct the secret, a subset $S \subseteq [n]$ can be used to compute the Lagrange coefficients $L_i := \prod_{j\in S, j\neq i} j/(j-i)$, which are then combined with the shares to recover the secret: $s = f(0) = \sum_{i\in S} s_i L_i$.*

A.2 Traceable Secret Sharing

As discussed in Section 1, Goyal et al. [19] introduced the concept of Traceable Secret Sharing (TSS), which enables traceability to identify the shareholders who participate in the secret reconstruction phase. Below, we recall the formal definition of TSS by synthesizing definitions from existing TSS schemes in the literature [6, 19] and slightly modify them.

Definition A.3 (Traceable Secret Sharing (TSS) schemes). *An (n, t)-TSS consists of five PPT algorithms of (Initial, Share, Reconstruct, Trace, Judge) defined as follows:*

- $\mathsf{Initial} \rightarrow (\mathsf{pp})$*: The initialization phase generates the set of common public parameters,* pp*.*
- $\mathsf{Share}(1^\lambda, n, t, s) \rightarrow (\{sh_i\}_{i=1}^n)$*: Given the security parameter* 1^λ*, the secret* s*, and integers* (n, t)*, the algorithm shares the secret* s *and obtains the shares* $s_1, \cdots, s_n$*.*
- $\mathsf{Reconstruct}(\{s_i\}_{i\in Q}) \rightarrow (s/\perp)$*: Given a set of shares* s_i *for shareholders* $i \in Q$*, it returns the secret* s *if* $|Q| > t$*, otherwise it outputs* $\perp$*.*
- $\mathsf{Trace}^{\mathsf{Rec}}(\{F(sh_j)\}_{j\in V}, F(s), tk) \rightarrow ((I, \pi_{trace})/\perp)$*: Given a function of main secret obtained from the* Reconstruct *algorithm and shares, and an optional (public or private) tracing key* tk*, this algorithm has access to a reconstruction oracle* Rec*. It then outputs either a subset* $I \subset [n]$ *along with an associated proof* π_{trace} *or* $\perp$*.*
- $\mathsf{Judge}(vk, I, \pi_{trace}) \rightarrow$ `true/false`*: It is a deterministic algorithm that, given a verification key* vk*, a subset* I*, and an associated proof* π_{trace}*, outputs either* `true`*, indicating acceptance of the proof that the parties in* I *are malicious, or* `false`*, indicating rejection.*

A.3 Sigma Protocols and Non-Interactive Zero-Knowledge Proofs

Given a security parameter λ, let $X = X(\lambda)$ and $W = W(\lambda)$ be two sets and $\mathbf{R}$ be a relation on $X \times W$ that defines the following language,

$$Ł = \{x \in X : \exists\ \omega \in W,\ \mathbf{R}(x, \omega) = 1\}.$$

For an element $x \in Ł$, a corresponding element $\omega \in W$ is referred to as a witness if $\mathbf{R}(x, \omega) = 1$. Let $\mathcal{R}(\lambda)$ be a PPT algorithm that generates pairs (x, ω) satisfying $\mathbf{R}$, i.e. $\mathbf{R}(x, \omega) = 1$.

A sigma-protocol (Σ-protocol) for the relation $\mathbf{R}$ is a 3-round interactive protocol between two PPT algorithms: a prover P and a verifier V. P holds a witness ω for $x \in Ł$ and V is given x. P first sends a value a (so called the commitment) to V, and then V answers with a challenge c , and finally P answers with z. V accepts or rejects the proof. The triple $\mathsf{trans} = (a, c, z)$ is called a transcript of the Σ-protocol. A Σ-protocol is supposed to satisfy *Completeness*, *Honest Verifier Zero-Knowledge* (HVZK), and *Special Soundness* defined below.

Definition A.4 (Completeness). *A* Σ*-protocol* Π_Σ *with parties* (P, V) *is* complete for $\mathcal{R}$*, if for all* $(x, \omega) \in \mathbf{R}$ *we have,*

$$\Pr\left[\mathsf{trans}(\mathsf{P}(\mathbf{R}, x, \omega) \leftrightarrow \mathsf{V}(\mathbf{R}, x)) \text{ is accepted by } \mathsf{V}\right] = 1\ ,$$

where trans *denotes the transcript of the protocol.*

Definition A.5 (Honest Verifier Zero-Knowledge). *A* Σ*-protocol* Π_Σ *satisfies* HVZK for $\mathcal{R}$*, if there exists a PPT algorithm* $\mathcal{S}$ *that given* $x \in X$*, can simulate the* trans *of the scheme, s.t. for all* $x \in Ł$*,* $(x, \omega) \in \mathbf{R}$ *we have,*

$$\mathsf{trans}(\mathsf{P}(\mathbf{R}, x, \omega) \leftrightarrow \mathsf{V}(\mathbf{R}, x)) \approx \mathsf{trans}(\mathcal{S}(\mathbf{R}, x) \leftrightarrow V(\mathbf{R}, x))$$

where $\mathsf{trans}(\mathsf{P}(\cdot) \leftrightarrow \mathsf{V}(\cdot))$ *indicates the transcript of* Π_Σ *with* (P, V)*, and* $\approx$ *denotes the indistinguishability of transcripts.*

Definition A.6 (Special Soundness). *A Σ-protocol Π_Σ with parties (P, V) is* special sound *for $\mathcal{R}$, if there exists a PPT extractor Ext, such that for any $x \in L$, given two valid transcripts (a, c, z) and (a, c', z') for the same message, a, but $c \neq c'$, then $\mathsf{Ext}(a, c, z, c', z')$ outputs a witness ω for the relation $\mathbf{R}$.*

Public-coin Σ-Protocols. A sigma protocol Π_Σ is called *public-coin* if the challenge value c is a public and random string rather than something derived in a complex or private way.

Non-Interactive Zero-Knowledge (NIZK) Proof Systems. As previously noted, zero-knowledge proofs are interactive protocols, while Non-Interactive Zero-Knowledge (NIZK) proofs [4,18] eliminate the need for interaction between the prover and the verifier. A NIZK for the relation $\mathbf{R}$ consists of three PPT algorithms $\Pi_{\mathsf{NIZK}} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$, that are defined as follows.

- $\mathsf{Setup}(1^\lambda) \to pp$: Given the security parameter λ in its unary representation, outputs public parameters pp.
- $\mathsf{P}(pp, x, w) \to \pi$: Given the public parameters pp, a statement x, and a witness w, it generates a non-interactive proof π.
- $\mathsf{V}(pp, x, \pi) \to \texttt{false}/\texttt{true}$: Given the public parameters pp, a statement x, and a proof π, it outputs either `true` (accept) or `false` (reject).

A NIZK proof scheme must satisfy Completeness, Soundness, and Zero-Knowledge (ZK), as defined below. The definition of completeness for NIZK is analogous to the completeness property of a Σ-protocol (as defined in Def. A.4), which can also be expressed as follows:

Definition A.7 (Completeness). *A NIZK proof system, Π_{NIZK}, satisfies completeness if for all $(x, w) \in \mathbf{R}$, honestly generated proofs are always accepted by the verifier, i.e.,*

$$\forall (x, w) \in \mathbf{R}, \quad \Pr\left[\mathsf{V}(pp, x, \mathsf{P}(pp, x, w)) = \texttt{true}\right] = 1.$$

Definition A.8 (Zero-Knowledge (ZK)). *A NIZK proof system Π_{NIZK} satisfies zero-knowledge if there exists a PPT simulator $\mathcal{S}$ such that for all $x \in L$, where L is the language corresponding to relation $\mathbf{R}$, the simulator can generate a proof that is indistinguishable from the real proof:*

$$\forall\ \mathsf{PPT}\,\mathcal{A}, \exists\,\mathcal{S}, \quad \{\mathcal{S}(pp, x)\} \approx \{\mathsf{P}(pp, x, w)\}\,.$$

Note that, Zero-Knowledge (ZK) is an amplified version of HVZK, defined in Def. A.5, as it guarantees privacy against even a malicious verifier.

Definition A.9 (Soundness). *A NIZK proof system, Π_{NIZK}, satisfies soundness if no PPT adversary $\mathcal{A}$ can generate a valid proof for an invalid statement $x \notin L$ with more than negligible probability. More formally, for all PPT adversaries $\mathcal{A}$ we have,*

$$\Pr\left[\mathsf{Setup}(1^\lambda) \to pp,\ \mathcal{A}(pp, x, w) \to \pi : \mathsf{V}(pp, x, \pi) = \texttt{true}\ \wedge x \notin L\right] \leq \mathsf{negl}(\lambda).$$

NIZK Proof of Knowledge. In addition to the standard soundness property, a NIZK proof system can satisfy a stronger notion known as *knowledge soundness*. When a NIZK proof system satisfies knowledge soundness, it is referred to as a NIZK proof of knowledge. Unlike standard soundness, which merely ensures that a prover cannot convince the verifier of a false statement, knowledge soundness guarantees that if a prover can generate a valid proof for a statement, then there exists an efficient extractor that can extract the corresponding witness from the prover. This strengthens the proof system by ensuring not only that a valid proof implies the truth of the statement, but also that the prover actually possesses the knowledge (i.e., the witness) required to construct the proof.

Definition A.10 (Knowledge Soundness). *A NIZK proof system Π_{NIZK} satisfies knowledge soundness if for any PPT adversary $\mathcal{A}$ that can produce a valid proof for some statement x, there exists an efficient extractor, $\mathsf{Ext}_{\mathcal{A}}$, that can extract the corresponding witness w. Formally, for all PPT adversaries $\mathcal{A}$, if:*

$$\Pr\left[\begin{array}{l}\mathsf{Setup}(1^\lambda) \to pp,\ \ \mathcal{A}(pp, x) \to \pi,\ \ \mathsf{Ext}_{\mathcal{A}}(pp, x, \pi) \to w : \\ \mathsf{V}(pp, x, \pi) = \mathtt{true}\ \ \wedge (x, w) \notin \mathbf{R}\end{array}\right] \leq \mathsf{negl}(\lambda)\ .$$

Fiat-Shamir Transform. The *Fiat-Shamir* transform [16] enables the conversion of a secure public-coin Σ-protocol for a relation $\mathbf{R}$ into a NIZK proof scheme for the same relation. This transformation is achieved by replacing the challenge issued by the verifier with a challenge sampled from a random oracle, thereby eliminating the interaction between the prover and the verifier. In a public-coin Σ-protocol, the verifier sends a random challenge c to the prover in the second round. The Fiat-Shamir Transform substitutes this interactive challenge with a hash function $H(\cdot)$, modeled as a random oracle, that takes the public parameters, the statement being proved, and the prover's previous messages as inputs. Specifically, the challenge is computed as $c = H(pp, x, a)$, where pp represents the public parameters, x is the statement, and a is the prover's initial commitment. In the random oracle model, this transformation ensures that the challenge is both unpredictable and uniformly random, thereby preserving the security properties of the original protocol and achieving ZK instead of HVZK.

A.4 Public Key Cryptosystems and ElGamal's Scheme

Public-Key Encryption (PKE) is a fundamental cryptographic technique that allows a user to encrypt a message using a recipient's public key, ensuring confidentiality. The corresponding secret key, which is held securely by the recipient, is used to decrypt the ciphertext and recover the original plaintext.

Definition A.11 (PKE Schemes). *For a given λ, message space M, and ciphertext space C, a PKE scheme is defined by the following PPT algorithms:*

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$*: Given the security parameter λ in its unary representation, it outputs the public/secret key pair* $(\mathsf{pk}, \mathsf{sk})$.

- $\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$*: Given a public key* pk*, a message* $m \in M$*, the encryption algorithm outputs a ciphertext* ct*.*
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to m/\perp$*: Given the secret key* sk*, a ciphertext* $\mathsf{ct} \in C$*, it returns either the message* m *or a failure symbol* $\perp$ *if the ciphertext is invalid.*

A PKE scheme must satisfy the following security properties.

Definition A.12 (Correctness). *A PKE scheme is called correct if we have,*

$$\Pr\left[\forall\ \lambda, m \in M, (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda) : \mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m\right] = 1\ .$$

Definition A.13 (IND-CPA **Security).** *A PKE scheme is said to be IND-CPA secure if for all PPT adversaries* $\mathcal{A}$ *we have,*

$$\Pr\left[\begin{matrix}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda), (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pk}), b \leftarrow \{0,1\} \\ \mathsf{ct}_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b), b' \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{pk}, \mathsf{ct}_b) : b' = b \wedge m_0 \neq m_1\end{matrix}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

ElGamal Cryptosystem. ElGamal cryptosystem (a.k.a., encryption) [14] is a well-known PKE scheme, defined over a cyclic group $\mathbb{G}$ of prime order p with generator g. The scheme operates as follows:

- $\mathsf{KeyGen}(1^\lambda)$: Choose a secret key $\mathsf{sk} \leftarrow \mathbb{Z}_p^*$ and compute the public key as $\mathsf{pk} = g^{\mathsf{sk}}$. Output $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}(\mathsf{pk}, m)$: To encrypt a message $m \in \mathbb{G}$ using the public key pk, pick a random $r \leftarrow\!\!\$\ \mathbb{Z}_p$ and compute and output the ciphertext $\mathsf{ct} := (c_1, c_2) = (m \cdot \mathsf{pk}^r, g^r)$.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: To decrypt a ciphertext $\mathsf{ct} = (c_1, c_2)$ using the secret key sk, compute and output: $m = c_1 / c_2^{\mathsf{sk}}$.

The correctness of ElGamal encryption is trivial and its IND-CPA security can be proved based on Decisional Diffie-Hellman (DDH) problem.

A.5 Commitment Schemes

Definition A.14 (Commitment Scheme). *A commitment scheme,* $\mathcal{C}$*, consists of the following PPT algorithms:*

- $\mathsf{Initial}(1^\lambda) \to pp_c$*: The initial algorithm outputs the set of public parameters* pp*.*
- $\mathsf{Com}(pp, m, \tau) \to \mathsf{cm}$*: The commitment algorithm takes as input* pp *and a message* m *along with a trapdoor* τ*, and outputs a commitment* cm*.*
- $\mathsf{Verify}(pp, \mathsf{cm}, m', \tau') \to$ `true`/`false`*: Verification is a deterministic algorithm which takes as input* pp*, a commitment* cm*, a message* m'*, along with an opening value* τ'*, and outputs either* `true` *(accept) or* `false` *(reject).*

Informally, the main security properties required for a commitment scheme are correctness, hiding, and binding. Correctness means that commitments generated correctly will pass the verification phase. The hiding property ensures that the commitment hides all information about the committed value. Meanwhile, binding guarantees that once a commitment is made, the committer cannot open it to reveal two different messages.

Pedersen Commitment Scheme [24]. Over a cyclic group $\mathbb{G}$ of prime order p with generator g, the Pedersen commitment scheme allows to commit to a scalar message $m \in \mathbb{Z}_p$ and is perfectly hiding and computationally binding. It consists of the following polynomial-time algorithms:

- $\mathsf{Initial}(1^\lambda)$: Sample $r \leftarrow\!\!\$\, \mathbb{Z}_p$ and set $h = g^r$. Output $pp := (g, h)$.
- $\mathsf{Com}(pp, m, \tau)$: Compute $\mathsf{cm} \leftarrow g^\tau h^m$. Output cm.
- $\mathsf{CVerify}(pp, \mathsf{cm}, m', \tau')$: Compute $\mathsf{cm}' \leftarrow g^{\tau'} h^{m'}$. Return $\mathsf{cm} == \mathsf{cm}'$.

A.6 Distributed Key Generation Protocols

Distributed Key Generation (DKG) protocols [24] allow multiple parties to collaboratively and securely generate key pairs without relying on a trusted third party. To carry out any operation involving the secret key, a sufficiently large subset of participants must cooperate, while any smaller group cannot gain any knowledge of the secret key. These generated keys can be used in various applications, particularly in threshold cryptosystems, such as threshold signatures and threshold encryption.

Definition A.15 (Distributed Key Generation (DKG) [24]). *More formally, an (n, t)-DKG is an interactive protocol among a set of parties $(P_1, \ldots, P_n)$ that generates a tuple of public keys $(\mathsf{pk}, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and a corresponding tuple of secret key shares $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ such that only party P_i knows the share sk_i.*

- *Consistency: A DKG is referred to as (t)-Consistent if, by the end of the protocol, all honest parties agree on a consistent public key pk and the vector of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, even when up to t parties are corrupted.*
- *Security: Additionally, as long as fewer than $t+1$ parties are corrupted, there exists a polynomial $f(x) \in \mathbb{Z}_p[X]_t$ of degree $d \leq t$ such that $\forall i \in [n] :\ \mathsf{sk}_i = f(i)$. In case of DKG for discrete logarithm, the global public key can then be computed as $\mathsf{pk} = g^{f(0)}$, where g is the generator of a cyclic group $\mathbb{G}$.*

B Security Proofs

B.1 Proof of Theorem 4.2

Proof. **Correctness of the Reconstruction Algorithm.** By the correctness of the Pedersen VSS, the secret value s is constructed with a probability 1.

Correctness of the Trace Algorithm. We show the correctness by analyzing two cases based on the size of Q:

- When $0 < |Q| \leq t$, the reconstruction algorithm returns $\{F(f_j), \pi_j, \mathsf{who}\}_{j \in Q}$ where π_j is a non-interactive proof (or argument) generated by $\Pi^{(2)}_{\mathsf{NIZK}}$ for the relation $R^{(2)}_j$ defined in equation (4). The trace algorithm $\mathsf{VerifTrace}$

given $\{F(f_j), \pi_j, \mathsf{who}\}_{j\in Q}$ and the tracing key $tk := \pi_{share} = (c_0, c_1, \ldots, c_t, c'_1, \ldots, c'_n)$, computes $y_j := \prod_{k=0}^{t} c_k^{j^k}$ for $j \in Q$ (which can be deduced from the n-bit string who) and then verifies the proofs $\{\pi_j\}_{j\in Q}$ using the verification algorithm of $\Pi^{(2)}_{\mathsf{NIZK}}$ and commitment values $\{y_j, c'_j\}_{j\in Q}$. By the completeness property of $\Pi^{(2)}_{\mathsf{NIZK}}$, all proofs will be accepted. Finally, it returns Q and $\pi_{trace} := (\{F(f_j)\}_{j\in Q}, \{\pi_j\}_{j\in Q})$.

- When $|Q| \geq t+1$, the reconstruction algorithm returns $\{F(s), \pi_j, \mathsf{who}\}_{j\in Q}$ where π_j is a non-interactive proof (or argument) generated by $\Pi^{(2)}_{\mathsf{NIZK}}$ for the relation $R^{(2)}_j$ defined in equation (4). Similarly, by the completeness property of $\Pi^{(2)}_{\mathsf{NIZK}}$, all proofs $\{\pi_j\}_{j\in Q}$ will be accepted. Therefore, the $\mathsf{VerifTrace}$ trace algorithm returns Q and $\pi_{trace} := (\{F(s)\}_{j\in Q}, \{\pi_j\}_{j\in Q})$.

Verifiability. It holds directly by the verifiability of the Pedersen VSS. (Note that, the only difference between the verification in Figure 3 and in Feldman scheme is that the parties check if c'_j are broadcast and well-formed. Since V contains honest parties, this extra check does not affect the output of verification algorithm in Figure 3.)

Statistical Secrecy. The statistical secrecy of the protocol in Figure 3 is guaranteed by the statistical secrecy properties of the Pedersen VSS, which stem from the perfect hiding property of the Pedersen commitment scheme. The reason is that, for each i, the additional component of s_i, denoted as γ_i, is chosen by party i randomly and independently of f_i. Therefore, the adversary can sample them itself and broadcast g^{γ_i} for all $i \in Q$ instead of receiving them from the share algorithm. This reduces the secrecy of Figure 3 scheme to the secrecy of the Pedersen VSS.

Traceability. The Share algorithm given $(1^\lambda, n, t, s)$, where s is a secret from $\mathbb{Z}_q$, samples a random secret $r_0 \in \mathbb{Z}_q$ and a two random degree-t polynomials $f(x) = s + a_1X + \ldots, a_tX^t$ and $r(x) = r_0 + b_1X + \ldots, b_tX^t$, and random coefficients $\{a_j, b_j\}_{j=1}^t$ from $\mathbb{Z}_q$. Then it outputs $s_i := (f_i, r_i, \gamma_i)$, where $f_i = f(i)$, $r_i = r(i)$, and $\pi_{share} = tk = vk := (c_0, \ldots, c_t, c'_1, \ldots, c'_n)$ where $c_0 = g^s h^{r_0}, c_1 = g^{a_1}h^{b_1}, \ldots, c_t = g^{a_t}h^{b_t}$, and for any $i \in [n]$, $c'_i = g^{\gamma_i}$. Then the adversary given $\{s_i\}_{i\in Q}, \pi_{share}, F(\cdot)$ for a set $Q \subseteq [n]$ returns either $\{z := F(s), \pi_{rec}\}$ or $\{\{z_j := F(f_j)\}_{j\in Q}, \pi_{rec}\}$ where $\pi_{rec} = \{\pi_j, \mathsf{who}\}_{j\in Q}$. The tracing algorithm $\mathsf{VerifTrace}$ in Figure 3 given the tracing key $tk := \pi_{share} = (c_0, c_1, \ldots, c_t, c'_1, \ldots, c'_n)$, computes $y_j := \prod_{k=0}^{t} c_k^{j^k}$ for $j \in Q$ (which can be deduced from the n-bit string who) and then verifies the proofs $\pi_{rec} := \{\pi_j, \mathsf{who}\}_{j\in Q}$ using the verification algorithm of $\Pi^{(1)}_{\mathsf{NIZK}}$ and commitment values $\{y_j, c'_j\}_{j\in[n]}$. For any statement $(c_i, c'_i, F(\cdot))$, which the verification algorithm of $\Pi^{(2)}_{\mathsf{NIZK}}$ on at least one of the proofs in $\pi_{rec} := \{\pi_j, \mathsf{who}\}_{j\in Q}$ returns 1, the tracing algorithm puts i in I. If there is no statement $(c_i, c'_i, F(\cdot))$ which has an accepting proof in $\pi_{rec} := \{\pi_j, \mathsf{who}\}_{j\in Q}$, it returns $\{0, \emptyset, 0^n\}$. If I is non-empty, the tracing algorithm returns $1, I, \pi_{trace} = \{\pi_i\}_{i\in I}$. Analyzing the following cases finishes the proof:

- If the tracing algorithm returns $\{0, \emptyset, 0^n\}$, nothing is left to prove.
- Assume that with a non-negligible probability ϵ, there is an index $i^* \in I$ such that $i^* \notin Q$. That is, the adversary does not know the corresponding witness $s_{i^*} := (f_{i^*}, r_{i^*}, \gamma_{i^*})$ for the statement $(c_{i^*}, c'_{i^*}, F(\cdot))$, however the verification algorithm of $\Pi^{(2)}_{\mathsf{NIZK}}$ on π_{i^*} returns 1 with a non-negligible probability ϵ. Since $\Pi^{(2)}_{\mathsf{NIZK}}$ is knowledge sound (with respect to the Definition A.10), there is an efficient extractor which returns the witness $s_{i^*} := (f_{i^*}, r_{i^*}, \gamma_{i^*})$ with a non-negligible probability, which breaks the Discrete Logarithm problem.
- It is trivial that Judge in Figure 3 outputs `true` because it executes the verification algorithm of $\Pi^{(2)}_{\mathsf{NIZK}}$ on $\pi_{trace} = \{\pi_i\}_{i \in I}$ similar to the tracing algorithm.

Non-imputability. The Share algorithm given $(1^\lambda, n, t, s)$, where s is a secret from $\mathbb{Z}_q$, samples a random secret $r_0 \in \mathbb{Z}_q$ and a two random degree-t polynomials $f(x) = s + a_1 X + \ldots, a_t X^t$ and $r(x) = r_0 + b_1 X + \ldots, b_t X^t$, and random coefficients $\{a_j, b_j\}_{j=1}^t$ from $\mathbb{Z}_q$. Then it outputs $s_i := (f_i, r_i, \gamma_i)$, where $f_i = f(i)$, $r_i = r(i)$, and $\pi_{share} = tk = vk := (c_0, \ldots, c_t, c'_1, \ldots, c'_n)$ where $c_0 = g^s h^{r_0}, c_1 = g^{a_1} h^{b_1}, \ldots, c_t = g^{a_t} h^{b_t}$, and for any $i \in [n]$, $c'_i = g^{\gamma_i}$. Then the adversary given $\{s_i\}_{i \in [n] \setminus i^*}, \pi_{share}, F(\cdot)$ returns $(1, I, \pi_{trace} = \{\pi_i\}_{i \in I})$. The Judge algorithm in Figure 3 given the tracing key $vk := \pi_{share} = (c_0, c_1, \ldots, c_t, c'_1, \ldots, c'_n)$, uses pre-computed values $y_j := \prod_{k=0}^t c_k^{j^k}$ for $j \in [n]$ and then verifies the proofs $\pi_{rec} := \{\pi_j\}_{i \in I}$ using the verification algorithm of $\Pi^{(2)}_{\mathsf{NIZK}}$ and commitment values $\{y_j, c'_j\}_{j \in [n]}$. If all the verification pass, it returns `true`. Otherwise, it returns `false`.

Assume that the Judge algorithm returns `true` and $i^* \in I$. That means the adversary has returned an accepting proof π_{i^*} for the statement $(c_{i^*}, c'_{i^*}, F(\cdot))$, without knowing the corresponding witness $s_{i^*} := (f_{i^*}, r_{i^*}, \gamma_{i^*})$. Since $\Pi^{(2)}_{\mathsf{NIZK}}$ is knowledge sound (with respect to the Definition A.10), there is an efficient extractor which returns the witness $s_{i^*} := (f_{i^*}, r_{i^*}, \gamma_{i^*})$ with a non-negligible probability, which breaks the DL problem. This complete the proof. □

B.2 Proof of Theorem 5.1

Proof. We present the proof for the interactive protocol, as the final NIZK argument is obtained by directly applying Fiat-Shamir transform on the Σ-protocol.

Completeness. When both P and V are honest, $R_1 = c^{r_1}$, $R_2 = g^{r_1} h^{r_2}$, $z_1 = r_1 + e f_i$ and $z_2 = r_2 + e\gamma_i$. Substituting P's response into V's checks:

$$c^{z_1} = c^{r_1 + e f_i} = c^{r_1} z_i^e = R_1 d_i^e,$$
$$g^{z_1} h^{z_2} = g^{r_1 + e f_i} h^{r_2 + e\gamma_i} = g^{r_1} h^{r_2}\ \mathsf{pk}_i^e = R_2\ \mathsf{pk}_i^e.$$

Both checks hold, so V will accept the proof, and completeness is satisfied.

Honest Verifier Zero-Knowledge (HVZK). HVZK means that an honest verifier learns nothing from the proof beyond the truth of the statement, and there exists a simulator that can generate a valid transcript without access to the witness. To prove HVZK, we need to construct a simulator that, given the challenge e, can create a valid transcript (R_1, R_2, z_1, z_2) without knowing (f_i, γ_i).

- *Simulation*: Given the challenge value e, the simulator picks random values z_1' and z_2' from $\mathbb{Z}_q$, and computes the commitments as $R_1' = c^{z_1'} d_i^{-e}$ and $R_2' = g^{z_1'} h^{z_2'} \ \mathsf{pk}_i^{-e}$. The simulator outputs the proof $\pi = (R_1', R_2', z_1', z_2')$.
- *Verification*: Using the challenge value e, the verifier checks:

$$c^{z_1'} \stackrel{?}{=} R_1' d_i^e \quad \text{and} \quad g^{z_1'} h^{z_2'} \stackrel{?}{=} R_2' \ \mathsf{pk}_i^e.$$

  Both conditions hold because of the simulator's construction, and the distribution of simulated proof is indistinguishable from the real one. Therefore, the verifier cannot distinguish a real proof from a simulated one and HVZK is satisfied.

Special Soundness. Special soundness means that given two valid transcripts for the same commitment (R_1, R_2) but with different challenges, it is possible to extract the witness f_i and γ_i. Let's assume there are two valid proofs:

$$\pi_1 = (R_1, R_2, z_1, z_2) \text{ with challenge } e,$$

$$\pi_2 = (R_1, R_2, z_1', z_2') \text{ with challenge } e', \text{ where } e \neq e'.$$

From the verifier's checks for both proofs:

$$c^{z_1} = R_1 d_i^e \quad \text{and} \quad c^{z_1'} = R_1 d_i^{e'}.$$

Dividing these two equations gives:

$$c^{z_1 - z_1'} = d_i^{e - e'} \quad \Rightarrow \quad d_i = c^{\frac{z_1 - z_1'}{e - e'}}.$$

Thus, we can extract f_i as: $f_i = \frac{z_1 - z_1'}{e - e'} \mod q$. Similarly, from the second check:

$$g^{z_1} h^{z_2} = R_2 \ \mathsf{pk}_i^e \quad \text{and} \quad g^{z_1'} h^{z_2'} = R_2 \ \mathsf{pk}_i^{e'},$$

dividing these gives:

$$g^{(z_1 - z_1')} h^{(z_2 - z_2')} = \mathsf{pk}_i^{e - e'} \quad \Rightarrow \quad \mathsf{pk}_i = g^{\frac{z_1 - z_1'}{e - e'}} \ h^{\frac{z_2 - z_2'}{e - e'}}.$$

Considering the fact that $\log_g(h)$ is (computationally) unknown for the prover, this allows us to extract γ_i as: $\gamma_i = \frac{z_2 - z_2'}{e - e'} \mod q$. Thus, given two valid transcripts with different challenges, we can extract the witness (f_i, γ_i), satisfying special soundness.

Since the Σ-protocol satisfies completeness, HVZK, and special soundness, and is a public-coin protocol, it can be transformed into a NIZK argument of knowledge using the Fiat-Shamir transform and a random oracle $\mathcal{H}$. □

B.3 Proof of Theorem 5.2

Proof. To prove this theorem, we apply a technique similar to the one used in the proof of Theorem 4.1.

Correctness of the Combine Algorithm. By the correctness of the threshold ElGamal encryption [11, 12], with a sufficiently large set of partial decryptions, the plaintext m can be retrieved with a probability 1.

Correctness of the Trace Algorithm. We show the correctness of the trace algorithm by analyzing two cases based on the cardinality of the set Q. W.l.o.g, we assume $Q = V = I$.

- When $0 < |Q| \leq t$, the combine algorithm returns $(\{d_j, \pi_j\}_{j\in Q}, \mathsf{who})$, where π_j is a non-interactive proof (or argument) generated by $\Pi^{Dec}_{\mathsf{NIZK}}$ for the relation R^{Dec}_j defined in Eq. (5) obtained from π_{dec}. The trace algorithm given $\{d_j, \pi_j\}_{j\in Q}$ and the tracing key $\mathsf{tk} = (pp, \{\mathsf{pk}_i\}_{i\in[n]})$, verifies the proofs $\{\pi_j\}_{j\in Q}$ using the verification algorithm of $\Pi^{Dec}_{\mathsf{NIZK}}$. By the completeness property of $\Pi^{Dec}_{\mathsf{NIZK}}$, all proofs will be accepted. Finally, it returns $\pi_{trace} := \{\mathsf{ct}, d_j, \pi_j\}_{j\in Q}$.
- When $|Q| \geq t + 1$, the combine algorithm returns (m, π_{dec}), where $\pi_{dec} := \{\pi_i, \mathsf{who}\}_{i\in Q}$ and π_j is a non-interactive proof (or argument) generated by $\Pi^{Dec}_{\mathsf{NIZK}}$ for the relation R^{Dec}_j defined in Eq. (5). Similarly, by the completeness property of $\Pi^{Dec}_{\mathsf{NIZK}}$, all proofs $\{\pi_j\}_{j\in Q}$ will be accepted. Therefore, the trace algorithm returns $(1, Q, \pi_{trace})$, where $\pi_{trace} := \{\mathsf{ct}, d_j, \pi_j\}_{j\in Q}$.

IND-CPA. Given the IND-CPA security of ElGamal encryption (cf. Section A.4) and information theoretical security of Shamir Secret sharing (cf. Section A.1), next we prove the IND-CPA security of the proposed ATC (cf. Figure 6).

Let there is an adversary $\mathcal{A}$ that can break the IND-CPA security of ATC, we will use $\mathcal{A}$ to build an adversary $\mathcal{B}$ that can break the IND-CPA security of standard ElGamal with the same probability. The adversary $\mathcal{B}$ interacts with the standard ElGamal challenger in the following way. First, the adversary $\mathcal{A}$ provides the list of corrupted parties Q to $\mathcal{B}$. Once $\mathcal{B}$ receives public the group description $(\mathbb{G}, g)$ and a public key pk' from the standard ElGamal challenger sets this the global public key of ATC $\mathsf{pk} := \mathsf{pk}'$ and simulates the public keys, $\{\mathsf{pk}_i\}_{i\in[n]}$, as follows:

- W.l.o.g., let $|Q| = t$. For all $i \in Q$, $\mathcal{B}$ samples random values $\mathsf{sk}_i \leftarrow \mathbb{Z}_q$, and defines the corresponding public key as $\mathsf{pk}_i := g^{\mathsf{sk}_i}$. Note that, due to the information-theoretic security of the Shamir secret sharing scheme, the uniformly random secret shares are indistinguishable from those generated by the standard Share algorithm.
- To generate the public key of the honest parties $k \in H, H := [n] \setminus Q$, $\mathcal{B}$ proceeds as follows:
  1. For all $i \in \tilde{T} := Q \cup \{0\}$, it computes the Lagrange coefficients, $L_{ki} = \prod_{j\in\tilde{T}, j\neq i} (j - k)/(j - i)$.

2. It takes the public keys of corrupted parties $\{\mathsf{pk}_i\}_{i\in Q}$ and the global public key pk and then computes $\mathsf{pk}_k := \mathsf{pk}^{L_{k0}} \prod_{i\in Q} \mathsf{pk}_i^{L_{ki}}$.

- $\mathcal{B}$ then samples $h \leftarrow \mathbb{G}$ and sets $\mathsf{vk} := \mathsf{tk} := (pp, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, where $pp := (g, h)$.

After receiving $(\{\mathsf{sk}_i\}_{i\in Q}, \mathsf{pk}, \mathsf{vk}, \mathsf{tk})$, $\mathcal{A}$ chooses two distinct messages m_0 and m_1 and sends them to $\mathcal{B}$. Then $\mathcal{B}$ forwards (m_0, m_1) to the standard ElGamal challenger, who encrypts one of them based on the random bit β, and returns the ciphertext ct_β to $\mathcal{B}$. $\mathcal{B}$ then passes this ciphertext to $\mathcal{A}$. Afterwards $\mathcal{A}$ returns the bit b', and $\mathcal{B}$ uses $\mathcal{A}$'s guess as its own guess in the standard ElGamal IND-CPA game.

Since the encryption process in both standard and threshold ElGamal is identical, the ciphertext that $\mathcal{A}$ sees is indistinguishable from what it would see in the ATC setting. As a result, if $\mathcal{A}$ can successfully distinguish between encryptions in IND-CPA game for the ATC, then $\mathcal{B}$ can also do so against the IND-CPA game for the standard ElGamal, thus breaking the IND-CPA security of standard ElGamal. This concludes the IND-CPA security of new ATC.

Traceability. In the traceability game, defined in Definition 5.4, the adversary given a set of secret keys $\{\mathsf{sk}_i\}_{i\in Q}$ returns either $\{m, \pi_{dec}\}$ or $\{\{d_j\}_{j\in Q}, \pi_{dec}\}$ where $\pi_{dec} = \{\pi_j, \mathsf{who}\}_{j\in Q}$. The tracing algorithm $\mathsf{VerifTrace}$ by taking the tracing key $\pi_{trace} := (\{d_j\}_{j\in Q}, \{\pi_j\}_{j\in Q})$, verifies the proofs $\pi_{dec} := \{\pi_j\}_{j\in Q}$ using the verification algorithm of $\Pi^{Dec}_{\mathsf{NIZK}}$ and partial ciphertext d_j and ciphertext ct. For any statement x, which the verification algorithm of $\Pi^{Dec}_{\mathsf{NIZK}}$ on at least one of the proofs in $\pi_{dec} := \{\pi_j\}_{j\in Q}$ accepts, the tracing algorithm updates $I = I \cup \{i\}$, otherwise it returns $\{0, \emptyset, 0^n\}$. If I is non-empty, the tracing algorithm returns $(1, I, \pi_{trace} = \{\pi_i\}_{i\in I})$. Analyzing the following cases concludes the proof:

- If the tracing algorithm returns $\{0, \emptyset, 0^n\}$, nothing is left to prove.
- Assume that with a non-negligible probability ϵ, there is an index $i^* \in I$ s.t. $i^* \notin Q$. That is, the adversary does not know the corresponding witness $\mathsf{sk}_{i^*} := (f_{i^*}, \gamma_{i^*})$ for the statement (d_{i^*}, ct), however the verification algorithm of $\Pi^{Dec}_{\mathsf{NIZK}}$ on π_{i^*} returns 1 with a probability ϵ. This breaks the knowledge-soundness of $\Pi^{Dec}_{\mathsf{NIZK}}$. Therefore, $I \subseteq Q$.
- It is trivial that Judge in Figure 6 outputs $\mathtt{true}$ because it executes the verification algorithm of $\Pi^{Dec}_{\mathsf{NIZK}}$ on $\pi_{trace} := \{\mathsf{ct}, d_j, \pi_j\}_{j\in I}$ similar to the tracing algorithm.

Non-imputability. In the non-imputability game, defined in Definition 5.5, the adversary given a set of secret keys of all parties except the challenger party i^*, i.e. $\{\mathsf{sk}_i\}_{i\in[n]\setminus\{i^*\}}$ returns $(1, I, \pi_{trace})$. The Judge algorithm in Figure 6 given the tracing key $\mathsf{tk} = \{\mathsf{pk}_i\}_{i\in[n]}$, verifies the proofs $\pi_{trace} := \{\mathsf{ct}, d_j, \pi_j\}_{j\in I}$ using the verification algorithm of $\Pi^{Dec}_{\mathsf{NIZK}}$ and ciphertext ct. If all the verifications pass, it returns $\mathtt{true}$. Otherwise, it returns $\mathtt{false}$.

Assume that the Judge algorithm returns $\mathtt{true}$ and a subset I s.t. $i^* \in I$. That means the adversary has returned an accepting proof π_{i^*} for the statement $(g, h, d_{i^*}, c, \mathsf{pk}_{i^*})$, without knowing the corresponding witness $\mathsf{sk}_{i^*} := (f_{i^*}, \gamma_{i^*})$ which breaks the knowledge-soundness of $\Pi^{Dec}_{\mathsf{NIZK}}$. This concludes the proof. □