# A Generic Framework for Practical Lattice-Based Non-interactive Publicly Verifiable Secret Sharing

Behzad Abdolmaleki<sup>1</sup>, John Clark<sup>1</sup>, Mohammad Foroutani<sup>2</sup>, Shahram Khazaei<sup>2</sup>, and Sajjad Nasirzadeh<sup>2</sup>

<sup>1</sup> University of Sheffield {behzad.abdolmaleki, john.clark}@sheffield.ac.uk <sup>2</sup> Sharif University of Technology m4utani@gmail.com, shahram.khazaei@sharif.edu, sajjadnasirzadeh@gmail.com

Abstract. Non-interactive publicly verifiable secret sharing (PVSS) schemes enable the decentralized (re-)sharing of secrets in adversarial environments, allowing anyone to verify the correctness of distributed shares. Such schemes are essential for large-scale decentralized applications, including committee-based systems that require both transparency and robustness. However, existing PVSS schemes rely on group-based cryptography, resulting them vulnerable to quantum attacks and limiting their suitability for post-quantum applications.

In this work, we propose the first practical, fully lattice-based, noninteractive PVSS scheme, grounded on standard lattice assumptions for post-quantum security. At the heart of our design is a generic framework that transforms vector commitments and linear encryption schemes into efficient PVSS protocols. We enhance vector commitments by incorporating functional hiding and proof of smallness, ensuring that encrypted shares are both verifiable and privacy-preserving. Our construction introduces two tailored lattice-based encryption schemes, each supporting efficient proofs of decryption correctness. This framework provides strong verifiability guarantees while maintaining low proof sizes and computational efficiency, making it suitable for systems with large numbers of participants.

# 1 Introduction

Publicly verifiable secret sharing (PVSS) schemes enable a dealer to distribute a secret among multiple participants, in such a way that anyone — not only the participants — can verify the correctness of the distributed shares [24]. PVSS protocols are foundational components in decentralized cryptographic systems, including distributed key generation, threshold signatures, and committee-based consensus protocols [8,3]. As decentralized systems continue to scale, efficient PVSS schemes become critical to maintain transparency, robustness against malicious dealers, and scalability to large committees.

A fundamental property of PVSS is *public verifiability*. This ensures that any party, not just the participants, can audit the distribution process and detect

any misbehavior by the dealer. Public verifiability is particularly crucial in open, decentralized environments where trust assumptions are minimal, and external observers must independently verify the integrity of the secret sharing process. However, many existing constructions rely heavily on pairing-based cryptography or group-based assumptions to achieve this property, which leads to substantial computational costs and large proof sizes [10, 23, 5, 22, 13, 11, 4, 7]. These inefficiencies hinder scalability, especially in systems with large committees.

Another critical requirement is *compactness of proofs*. In decentralized networks involving thousands of participants, the size of the proofs associated with each share directly impacts communication overhead and verification efficiency. Most recently, there have been efforts to design more efficient and compact PVSS schemes [8, 3]. However, these constructions still rely on group-based primitives and discrete logarithm assumptions, leaving them vulnerable to quantum attacks and limiting their suitability for post-quantum applications.

*Post-quantum security* is increasingly vital in the era of quantum computing. Cryptographic schemes based on traditional hardness assumptions, such as the discrete logarithm or factoring problems, are rendered insecure against quantum adversaries. Unfortunately, most PVSS constructions to date rely on these vulnerable foundations, leaving future decentralized systems exposed to potential quantum attacks. Lattice-based cryptography, founded on hard problems such as the Learning With Errors (LWE) and Integer Solution (SIS) problems [1, 20, 6, 15, 19, 17], provides a promising post-quantum secure alternative. However, designing efficient, publicly verifiable, and compact PVSS schemes based solely on lattice assumptions remains an open challenge. An important step towards lattice-based PVSS was made by Gentry et al. [11], who proposed a PVSS construction at EUROCRYPT 2022 that partially incorporates lattice-based primitives. Their scheme combines LWE-based encryption with a discrete-logarithmbased proof system. While this hybrid design improves efficiency and introduces lattice techniques into PVSS, it falls short of achieving full post-quantum security. Specifically, because the underlying proof system relies on DL-based assumptions, the security of the overall scheme is compromised in the presence of quantum adversaries. In their work, Gentry et al. explicitly identified the construction of an efficient, fully lattice-based PVSS scheme as an open problem.

Finally, modularity and flexibility of the construction are highly desirable features for PVSS schemes. A modular design enables flexible instantiation of building blocks, allowing protocol designers to adapt the scheme for varying performance and security requirements. Yet, achieving such modularity in the lattice setting while preserving efficiency and strong security properties poses significant technical difficulties.

With all of this in mind, it is interesting to ask the following question:

Can we design a non-interactive, PVSS scheme that simultaneously achieves post-quantum security, compact and efficient proofs, and modularity to support large-scale decentralized systems?

In this work, we answer this question affirmatively by introducing the first fully *lattice-based*, practical, non-interactive PVSS scheme. Our design departs

from traditional group-based approaches and instead builds a modular framework that composes vector commitments with linear encryption schemes. At the core of our framework lies a *generic framework* that transforms any compatible vector commitment and encryption scheme into a PVSS protocol, providing both correctness and public verifiability. We enhance vector commitments by introducing two crucial properties: *functional hiding* and *proof of smallness*. Functional hiding ensures that the commitment reveals no information beyond what is necessary for verification, while proof of smallness guarantees that the committed values remain within prescribed bounds, both essential for verifiability in the lattice setting. To instantiate our framework concretely, we propose two novel lattice-based encryption schemes, each supporting efficient zero-knowledge proofs of decryption correctness. These choices allow us to overcome the scalability barriers of existing PVSS constructions and deliver a solution well-suited for decentralized systems with thousands of participants.

### 1.1 Our Contributions

In this work, we make several contributions towards constructing a practical, fully lattice-based PVSS scheme. Our results advance the state of the art in both the theoretical understanding and practical deployment of PVSS protocols, particularly in the post-quantum setting. Our contributions are summarized as follows:

- A fully lattice-based, practical PVSS scheme. We design the first PVSS protocol based entirely on lattice assumptions, achieving postquantum security under standard hardness assumptions such as (Ring) LWE and SIS.
- A generic framework for PVSS construction. We develop a modular framework that generically composes vector commitments and linear encryption schemes into a PVSS protocol. Our framework requires the encryption scheme to support efficient proofs of correct decryption, and the vector commitment to satisfy linear opening, functional hiding, and proof of smallness properties. This modularity allows our scheme to flexibly accommodate different cryptographic primitives, making it adaptable to a variety of performance and security trade-offs.
- Two new lattice-based encryption schemes. To instantiate our framework, we propose two novel lattice-based encryption schemes designed to efficiently support the required proof systems. These encryption schemes not only ensure correctness and verifiability within our PVSS protocol but may also find applications in other cryptographic constructions where efficiency and post-quantum security are critical.
- Enhanced vector commitment with functional hiding and proof of smallness. We extend existing lattice-based vector commitments by introducing two crucial properties: functional hiding and proof of smallness. Functional hiding ensures that commitments leak no unnecessary information beyond what is required for verification. Proof of smallness enables the

prover to demonstrate that the committed values lie within a specific range, a property essential for ensuring both correctness and soundness in our PVSS construction.

### 1.2 Our Technical Overview

Our central contribution is a generic framework that transforms any suitable combination of a Vector Commitment (VC) scheme and a linear encryption scheme into a secure PVSS protocol. Leveraging this compiler, we present the first PVSS construction based entirely on standard lattice assumptions, offering a scalable and post-quantum secure solution for decentralized secret sharing. Below, we describe the essential components and technical choices in our framework, unpacking both the cryptographic primitives and the intuition behind their integration.

<u>Vector Commitments with Proof of Smallness</u>. At the heart of our design lies an enhanced vector commitment scheme, which the dealer uses to commit to a vector  $\mathbf{x}$  containing:

- The secret-sharing polynomial coefficients **a**,
- The computed shares  $s_i$ ,
- The encryption randomness values  $r_i$ .

The commitment ensures that the dealer cannot alter these values postcommitment (binding), and the opening proof attests that all entries of  $\mathbf{x}$  satisfy a consistency function M, verifying both the correct computation of shares and the encryption of these shares under participants' public keys.

To reinforce this commitment, we extend standard VC schemes with two critical properties:

- Proof of Smallness: In lattice-based constructions, bounding the norm of committed values is crucial to ensure decryption correctness and maintain soundness. We incorporate efficient proofs that all sensitive entries of  $\mathbf{x}$ , notably  $s_i$  and  $r_i$ , are bounded by a parameter  $\beta$ , ensuring:

$$\|\mathbf{x}_s\| \le \beta \quad \forall s \in S$$

where S denotes the index set of sensitive components.

- Functional Hiding: To preserve privacy, our scheme guarantees that the commitment to  $\mathbf{x}$  leaks no information beyond what is revealed by the evaluation of the opening function M. Specifically, for any linear function f, the opening proof  $\pi$  satisfies  $f(\mathbf{x}) = y$  while hiding all other information about  $\mathbf{x}$ . This is vital for maintaining *t-IND2 privacy* in the presence of active adversaries.

Our instantiation builds upon the VC scheme of Albrecht *et al.* [2], which we extend to support these properties efficiently within our framework.

Linear Encryption with Proof of Decryption Correctness. Next, we employ a linear encryption scheme that is fully compatible with our vector commitment structure and supports efficient zero-knowledge proofs of correct decryption. For any message m, randomness r, and public key pk, encryption takes the form:

 $\mathcal{E}$ .Encrypt( $\mathcal{E}$ .pp, pk, m, r) =  $m \cdot \mathbf{G} + r \cdot \mathbf{A}$ 

where **G** is a fixed public gadget matrix, and **A** is part of the public parameters. To enable public verification of decryption correctness, the scheme provides:

- $\mathcal{E}$ .ProveDecrypt(sk; (pk, m, C)) the decryptor generates a non-interactive proof that C decrypts to message m, without revealing the secret key sk.
- E.VerifyDecrypt(pk, m, C, pf) any verifier can efficiently check the validity of this decryption proof pf.

We instantiate this encryption component using two novel lattice-based schemes described in Sections 4.2 and 4.3:

- The first is a gadget-based scheme where the decryptor reveals r to prove decryption correctness via:

$$C - m \cdot \mathbf{G} = r \cdot \mathbf{A}.$$

- The second is a trapdoor-sampling-based scheme where the decryptor provides auxiliary values (B', e) to enable efficient public verification. Here:
  - $B' = c_2 p \cdot m$  is a blinded ciphertext component that hides the message,
  - $e = B' s^{\top} \cdot c_1$  is the decryption noise.

The verifier checks that  $||e|| < \beta$ , where  $\beta$  is the public smallness bound defined during setup. This ensures that the decryption is correct without revealing the secret key s.

Both schemes balance compactness, efficiency, and post-quantum security, while supporting public verifiability.

<u>Identification Protocols for Key Generation and Decryption Proofs</u>. To ensure trust in public keys and decryption proofs, we integrate *identification protocols* based on lattice assumptions, inspired by Lyubashevsky's identification scheme [15].

These protocols serve two roles:

- Key Generation Proofs: Each participant proves knowledge of their secret key  $\mathsf{sk}_i$  corresponding to the public key  $\mathsf{pk}_i = \mathbf{A} \cdot \mathsf{sk}_i$ . The proof is verified using  $\mathcal{E}$ .VerifyKey( $\mathcal{E}.\mathsf{pp},\mathsf{pk}_i,\mathsf{pf}_{\mathsf{Kev},i}$ ).
- Decryption Proofs: Each participant, upon decrypting their ciphertext  $C_i$ , proves that their decrypted share  $s_i$  is correct:

$$C_i - s_i \cdot \mathbf{G} = r_i \cdot \mathbf{A}$$

Verification is performed using  $\mathcal{E}$ .VerifyDecrypt(pk<sub>i</sub>, s<sub>i</sub>, C<sub>i</sub>, pf<sub>Dec.i</sub>).

These protocols ensure public verifiability at both key generation and decryption phases, without revealing secret information.

<u>Our Generic framework: From Building Blocks to PVSS</u>. We combine these primitives into a complete PVSS protocol through our generic framework. The workflow is as follows:

- 1. *Setup:* Initialize public parameters for Shamir secret sharing, vector commitments, and encryption schemes.
- 2. Key Generation: Each participant generates  $(\mathsf{pk}_i, \mathsf{sk}_i)$  along with proof  $\mathsf{pf}_{\mathsf{Key},i}$  of correct key generation.
- 3. Distribution:
  - The dealer defines a polynomial f(x) of degree t such that f(0) = s.
  - Compute shares  $s_i = f(i)$  and encrypt them:

$$C_i = \mathcal{E}.\mathsf{Encrypt}(\mathcal{E}.\mathsf{pp},\mathsf{pk}_i,s_i,r_i)$$

- Form the vector  $\mathbf{x} = (\mathbf{a}, s_1, \dots, s_n, r_1, \dots, r_n)$ , commit to  $\mathbf{x}$ , and generate an opening proof verifying:
  - Correctness of shares:  $s_i = f(i)$
  - Correctness of ciphertexts:  $C_i = \mathcal{E}.\mathsf{Encrypt}(\mathcal{E}.\mathsf{pp},\mathsf{pk}_i,s_i,r_i)$
- 4. *Distribution Verification:* Any party verifies the dealer's commitment and opening proof, ensuring correct distribution.
- 5. *Decryption:* Each participant decrypts their ciphertext and produces a proof  $pf_{Dec,i}$  of correct decryption.
- 6. Reconstruction: Using Lagrange interpolation on decrypted shares  $s_i$ , any set of at least t participants can reconstruct the secret s.

<u>Concrete Instantiation and Practicality.</u> Our concrete instantiation composes the above primitives into a fully lattice-based PVSS scheme. Specifically:

- We extend the vector commitment scheme of Albrecht *et al.* [2] to support proof of smallness and functional hiding.
- We design two linear encryption schemes optimized for efficiency and verifiable decryption, using either gadget-based techniques or trapdoor sampling.
- We integrate lattice-based identification protocols to guarantee public verifiability of key generation and decryption.

Our construction derives security from standard lattice assumptions (Ring) LWE and SIS. The result is a PVSS protocol that is compact, scalable to thousands of participants, and secure against quantum adversaries — making it suitable for deployment in decentralized, large-scale environments.

### 1.3 Comparison

Comparison with state-of-the-art works. We denote  $\mathbb{G}$  to be a cyclic group of order q and assume that each element in  $\mathbb{G}$  has  $\log q$  bits. The notation  $\mathrm{op}_{\mathbb{G}}$ 

Table 1. Comparisons.

Work	Communication	Computation	Secret	Assumptions
[7]	$O((n+\ell) \cdot \log q)$	$O((n^2 + n\ell + n\log^2 n) \cdot \operatorname{op}_{\mathbb{Z}_q})$	$G^{\ell}$	DLOG+ROM
[11]	$O(n \cdot (u+v) \cdot \log q)$	$O(n \cdot (u+v)\log q)$	$\mathbb{Z}_q$	LWE+DLOG+ROM
[9]	$O(n \cdot \log q)$	$O(n \cdot \mathrm{op}_G + n \log^2 n \cdot \mathrm{op}_{\mathbb{Z}_q})$	G	DLOG+ROM
[8]	$O(n \cdot \log q)$	$O(n \cdot \operatorname{op}_G + n \log^2 n \cdot \operatorname{op}_{\mathbb{Z}_q})$	$\mathbb{Z}_q$	DLOG+ROM
Ours 1	$O(n \cdot \log^2 q)$	$O(n^2 \log^3 q \cdot \operatorname{op}_{R_q})$	$\mathbb{Z}_q$	LWE+k-R-ISIS
Ours 2	$O(n \cdot \log q)$	$O(n^2 \log q \cdot \operatorname{op}_{R_q})$	$\mathbb{Z}_q$	R-LWE+k-R-ISIS

refers to the number of exponentiations in  $\mathbb{G}$ , and  $\operatorname{op}_{\mathbb{Z}_q}$  refers to the number of arithmetic operations in  $\mathbb{Z}_q$ . Table 1 summarizes the efficiency of our PVSS construction in terms of communication, computation, and underlying assumptions, benchmarked against prior state-of-the-art schemes [7, 11, 9, 8]. Our construction achieves communication complexity of  $O(n \cdot \log q)$ , matching the best known results, while offering substantially better computational efficiency. Specifically, the computation cost of our scheme is  $O(n^2 \log q \cdot \operatorname{op}_{R_q})$ , where  $\operatorname{op}_{R_q}$  denotes ring operations over the lattice modulus ring. This is enabled by our tightly integrated design combining lattice-based vector commitments, linear encryption schemes, and efficient proofs of decryption correctness. Unlike earlier works that rely on group-based cryptography, random oracles, or hybrid hardness assumptions, our scheme is built entirely from standard lattice assumptions, namely (R-)LWE and k-R-ISIS. This ensures strong post-quantum security and yields a more modular and scalable design, well-suited for deployment in large-scale decentralized systems.

We propose two distinct constructions building upon the established framework, each leveraging a specific combination of Vector Commitment (VC) and encryption schemes. Our first construction (Ours 1) employs a plain LWE based cryptosystem (as detailed in Section 4.2 or Section 4.3) in conjunction with the VC scheme from Section 3.1. The security of this construction is founded on the standard LWE assumption. Our second construction (Ours 2) incorporates the compact ring-based cryptosystem presented in Section 4.4. This alternative relies on the Ring-LWE (R-LWE) assumption and offers the advantage of faster computation.

**Concurrent Work.** In a concurrent work, Minh *et al.* [18] introduce a postquantum secure PVSS framework based on the LWE assumption and trapdoor  $\Sigma$ -protocols, with security proven in the standard model. While achieving these desirable properties, their construction exhibits significant computational and communication overheads. Specifically, as indicated in their work (cf. Table 1), the communication complexity is  $\Omega(n\lambda(u+v)\log q)$ , and the total computation cost is  $\Omega(\lambda(n^2 + nuv)) \cdot op_{\mathbb{Z}_q}$ , where  $\lambda$  denotes the security parameter, q is the modulus, and u, v represent lattice dimensions. A further contributing factor to this high cost is the reliance on binary-challenge  $\Sigma$ -protocols, which necessitate  $\lambda$  parallel repetitions to achieve a negligible soundness error, thereby magnifying the overall expenses. Moreover, their parameterization, as detailed in Table 1 of their paper, suggests a modulus q on the order of  $\lambda^{11}n$ . Such a polynomial dependency of the modulus on the security parameter  $\lambda$  is uncharacteristic of typical cryptographic constructions and presents practical challenges. If  $\lambda$  is chosen towards the lower end of cryptographically secure values to maintain manageable costs, the  $\lambda^{11}$  scaling might result in a modulus q that offers insufficient concrete security, particularly against adversaries equipped with substantial computational power (e.g., supercomputers). Conversely, selecting a  $\lambda$  large enough to guarantee robust security would lead to an exceptionally large modulus, further exacerbating the already considerable communication and computation costs, likely rendering the protocol impractical for many applications.

In contrast, our construction is non-interactive by design and leverages efficient lattice-based components. We achieve a much tighter communication complexity of  $O(n \cdot \log q)$  and computation complexity  $O(n^2 \cdot \log q \cdot \operatorname{op}_{R_q})$ , while supporting compact proofs and practical efficiency. Our design avoids  $\lambda$ -fold repetition and achieves negligible soundness error through direct integration of proof-of-smallness techniques within vector commitments and verifiable linear encryption. This makes our protocol more scalable and better suited for deployment in real-world post-quantum settings.

# 2 Preliminaries

Notation. Throughout this paper, we use bold lowercase letters such as  $\mathbf{x}$  to denote vectors and bold uppercase letters such as  $\mathbf{A}$  for matrices. We write  $\langle \cdot, \cdot \rangle$  to denote the standard inner product of vectors. Let  $\circ$  denotes component-wise multiplication. The set of integers modulo q is denoted by  $\mathbb{Z}_q$ , and for a ring R, we write  $R^{\times}$  to denote its multiplicative group of units. We use  $\mathcal{R}$  to denote the ring associated with the lattice-based construction (e.g.,  $\mathbb{Z}_q^n$ ), and  $\mathcal{K}$  to denote the base field. All algorithms are probabilistic polynomial-time (PPT) unless otherwise stated. When we write  $x \leftarrow \mathcal{D}$ , we mean that x is sampled from distribution  $\mathcal{D}$ . The security parameter is denoted by  $\lambda$ , and all negligible functions are implicit in  $\lambda$ .

### 2.1 Publicly Verifiable Secret Sharing

We first present the definitions of a PVSS and security properties, where we mainly adopt the definitions from [8].

**Definition 2.1 (Publicly Verifiable Secret Sharing).** A PVSS scheme consists of the following algorithms.

Setup:

-  $pp \leftarrow Setup(1^{\lambda}, 1^n, 1^t)$ : The setup algorithm generates the public parameters on input the security parameter  $\lambda \in \mathbb{N}$ , number of parties  $\lambda \in \mathbb{N}$  and reconstruction thresholds  $t \in \mathbb{N}$ . The public parameters include a description of spaces of secrets and shares S and spaces of private and public keys SK and PK and the relation  $R_{Key} \subseteq PK \times SK$  describing valid key pairs.

- $\begin{array}{l} (\mathsf{sk}_i,\mathsf{pk}_i,\mathsf{pf}_{Key,i}) \leftarrow \mathsf{KeyGen}(\mathsf{pp},i): \ The \ key \ generation \ algorithm \ generates \\ (\mathsf{pk}_i,\mathsf{sk}_i) \in R_{Key} \ and \ proof \ \mathsf{pf}_{Key,i} \ for \ identification \ of \ \mathsf{pk}_i. \end{array}$
- $-b \leftarrow \text{VerifyKey}(\text{pp}, i, \text{pk}_i, \text{pf}_{Key,i})$ : The key verification algorithm outputs a bit b deciding whether to accept or reject that  $pk_i$  is a valid identification.

### Distribution:

 $-((C_i)_{i\in[n]}, pf_D) \leftarrow Dist(pp, (pk_i)_{i\in[n]}, s)$  The distribution algorithm outputs encrypted shares  $C_i$  and a proof  $pf_D$  of sharing correctness on input the secret  $s \in S$ .

Distribution Verification:

 $-b \leftarrow \mathsf{VerifyDist}(\mathsf{pp}, (\mathsf{pk}_i, C_i)_{i \in [n]}, \mathsf{pf}_D)$ : The distribution verification algorithm outputs a bit b deciding whether to accept or reject that for each i share  $C_i$  is valid.

Reconstruction:

- $-(s_i, \mathsf{pf}_{Dec,i}) \leftarrow \mathsf{Decrypt}(\mathsf{pp}, i, \mathsf{pk}_i, \mathsf{sk}_i, C_i)$ : The decrypt share algorithm outputs a decrypted share  $s_i$  and a proof  $\mathsf{pf}_{Dec,i}$  of correct decryption.
- $-s' \leftarrow \text{Reconstruct}(\text{pp}, \{s_i : i \in T\})$ : The reconstruction algorithm for some  $T \subseteq [n]$  outputs an element of the secret space  $s' \in S$  or an error symbol  $\perp$ .
- $-b \leftarrow \text{VerifyDecrypt}(\text{pp}, i, \text{pk}_i, s_i C_i, \text{pf}_{Dec,i})$ : The decryption verification algorithm outputs a bit b deciding whether to accept or reject that  $s_i$  is a valid decryption of  $C_i$ .

Security properties We require a PVSS to satisfy correctness, verifiability and IND2-secrecy. We briefly summarize these here and defer the formal definitions to Appendix A.2.

Correctness. If all parties behave honestly, the protocol guarantees that:

- All verification procedures succeed, including key verification VerifyKey, distribution verification VerifyDist, and decryption verification VerifyDecrypt.
- Any set of at least t honest participants can successfully reconstruct the secret from their decrypted shares.

*Verifiability.* The scheme ensures that all cryptographic objects are publicly verifiable:

- Key verifiability: Any public key is certified to correspond to a valid secret key via VerifyKey.
- Distribution verifiability: The dealer's distribution of encrypted shares and proof  $pf_D$  certifies correct sharing of the secret using VerifyDist.
- Decryption verifiability: Each decrypted share is accompanied by a proof of correctness, verified using VerifyDecrypt.

Privacy We now define indistinguishability of secrets against an adversary corrupting t parties. We follow the notions from [2]. In this definition, the adversary is allowed to compute the public keys of the corrupted parties after seeing those of the honest parties. Then, provided two secrets  $(s_0, s_1)$  and a sharing of a random secret  $s_b$ , the adversary has negligible advantage in guessing which secret was shared. In this paper, we choose the IND2-privacy flavor where the adversary can choose  $s_0, s_1$ . This is stronger than IND1-privacy where the challenger chooses the secrets at random.

**Definition 2.2.** The PVSS is t-IND2-private if for any  $poly(1^{\lambda})$ -time adversary  $\mathcal{A}$  corrupting t parties (w.l.o.g.  $\mathcal{A}$  corrupts [n - t + 1, n]), we have

 $\Pr[\operatorname{Game}_{\mathcal{A}, \operatorname{PVSS}}^{\operatorname{ind-secrecy}, 0}(\lambda) = 1] - \Pr[\operatorname{Game}_{\mathcal{A}, \operatorname{PVSS}}^{\operatorname{ind-secrecy}, 1}(\lambda) = 1] = \operatorname{negl}(\lambda)$ 

where for b = 0, 1, Game<sup>ind-secrecy,b</sup><sub> $A,PVSS</sub>(<math>\lambda$ ) is the following game:</sub>

- The challenger runs  $pp \leftarrow Setup(1^{\lambda}, 1^n, 1^t)$  and sends pp to  $\mathcal{A}$ .
- For  $i \in [n t]$ , the challenger runs  $(\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{pf}_{Key,i}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i)$  and sends all created  $(\mathsf{pk}_i, \mathsf{pf}_{Key,i})$  to  $\mathcal{A}$ .
- For the corrupted parties,  $\mathcal{A}$  creates  $(\mathsf{pk}_i, \mathsf{pf}_{Key,i})_{i \in [n-t+1,n]} \leftarrow \mathcal{A}(\mathsf{pp}, (\mathsf{pk}_i, \mathsf{pf}_{Key,i})_{i \in [n-t]})$  and sends them to the challenger, together with two values  $s_0, s_1$  in S.
- The challenger runs  $\operatorname{VerifyKey}(\operatorname{pp}, i, \operatorname{pk}_i, \operatorname{pf}_{Key,i})$  for  $i \in [n-t+1, n]$ . If any of these output 0 (reject), the challenger sends  $\perp$  to  $\mathcal{A}$ .
- Otherwise, if all key verification proofs accept, the challenger runs  $(C_1, \ldots, C_n, \mathsf{pf}_D) \leftarrow \mathsf{Dist}(\mathsf{pp}, \{\mathsf{pk}_i : i \in [n]\}, s_b)$ , and sends  $(C_1, \ldots, C_n, \mathsf{pf}_D)$  to  $\mathcal{A}$ .
- $\mathcal{A}$  outputs a guess  $b' \in \{0, 1\}$ .

### 2.2 Linear Public Key Encryption

We describe here explicitly the encryption scheme as we use it in our protocol. Let  $\mathcal{E} = (\text{Setup}, \text{KeyGen}, \text{VerifyKey}, \text{Encrypt}, \text{Decrypt})$  be a public key encryption scheme (see Appendix A.1 for the detailed definition). The results in this paper require linear encryption schemes with proofs of decryption correctness.

*Proofs of Decryption Correctness [9].* We need proofs of decryption correctness, where of, of course the prover wants to keep their secret key hidden, i.e., proofs for the relation.

 $R_{\mathcal{E},\mathsf{Decrypt}} = \{(\mathsf{sk}; (\mathsf{pk}, m, c)) : \\ (\mathsf{pk}, \mathsf{sk}) \text{is a valid key-pair for Encrypt and } m = \mathsf{Decrypt}(\mathsf{sk}, c)\}$ 

**Definition 2.3.** These algorithms add to  $\mathcal{E}$ :

-  $pf_{Dec} \leftarrow ProveDecrypt(sk; (pk, m, C))$ : The proof of decryption correctness algorithm generates a proof  $pf_{Dec}$ .

 $-b \leftarrow \text{VerifyDecrypt}(pk, m, C, pf_{Dec})$ : The decryption verification algorithm outputs a bit b deciding whether to accept or reject that m is a valid decryption of C.

**Definition 2.4.** The public key encryption scheme  $\mathcal{E}$  satisfies verifiability of decryption if the following is satisfied: For every PPT  $\mathcal{A}$ ,

$$\begin{split} &\Pr\left[\mathcal{E}.\mathsf{VerifyDecrypt}(\mathsf{pk},m,C_i,\mathsf{pf}_{Dec})=1 \\ &\wedge \nexists\mathsf{sk} \in SK \ s.t. \ (m,\cdot) \leftarrow \mathcal{E}.\mathsf{Decrypt}(\mathsf{pp},\mathsf{pk},\mathsf{sk},C) \\ &\left| \ \mathsf{pp} \leftarrow \mathcal{E}.\mathsf{Setup}(1^{\lambda},p), \\ &\left(\mathsf{pk},m,C,\mathsf{pf}_{Dec}) \leftarrow \mathcal{A}(\mathsf{pp}) \right] \ is \ negligible \ in \ \lambda. \end{split}$$

If the prover knows the randomness under which the message was encrypted or recovers a valid randomness, the proving algorithm ProveDecrypt(sk; (pk, m, C)) can simply output that randomness as  $pf_{Dec}$ ; the verification  $VerifyDecrypt(pk, m, C, pf_{Dec})$  accepts if  $Encrypt(pk, m; pf_{Dec}) = C$ .

Alternatively, we leverage the identification protocol to prove the validity of decrypted shares. This approach ensures that the decrypted shares are consistent with the ciphertexts and the corresponding secret key, providing an additional layer of verifiability.

#### 2.3 Vector Commitments

A vector commitment (VC) scheme is a cryptographic primitive that allows a committer to fix a vector of values and subsequently prove the correctness of evaluations on the committed vector. Formally, a VC scheme consists of four PPT algorithms:

- Setup $(1^{\lambda}, 1^{v}, 1^{w}, 1^{o})$ : A setup algorithm that generates public parameters pp given the security parameter  $\lambda$  and the dimensions of the function family.
- Com(pp, x): A commitment algorithm that outputs a commitment c to a vector x along with auxiliary information aux.
- Open(pp,  $f, \mathbf{z}, aux$ ): An opening algorithm that generates a proof  $\pi$  that the committed vector  $\mathbf{x}$  satisfies  $f(\mathbf{z}, \mathbf{x}) = \mathbf{y}$  for a given public input  $\mathbf{z}$  and function f.
- Verify(pp,  $f, \mathbf{z}, \mathbf{y}, c, \pi$ ): A verification algorithm that checks whether the proof  $\pi$  correctly certifies the claimed evaluation result.

A VC scheme must satisfy the following security properties:

- *Correctness:* Any honestly generated commitment and proof must pass verification for correctly computed evaluations.
- *Binding:* An adversary must not be able to produce two valid openings of the same commitment to different outputs for the same function.
- Functional Hiding: The commitment and proof should reveal no more about the committed vector than what is implied by the evaluation outputs.

We defer to Appendix A.3 for the full formal definition.

# 3 New Framework: A Compiler from VC to PVSS

In this section, we present our new framework for constructing PVSS schemes from vector VC. Our goal is to achieve a generic, modular design that transforms any suitably structured vector commitment and linear encryption scheme into a secure and publicly verifiable PVSS protocol. We begin by revisiting the underlying vector commitment primitive and formalizing a specialized variant that we refer to as Linear Vector Commitments with Proof of Smallness (LVC-PoS). This variant enhances standard VC schemes by incorporating mechanisms for proving the smallness of certain parts of the committed vector, which is crucial for our later compiler construction. After defining LVC-PoS and its security properties, we proceed to describe our generic compiler. The compiler transforms any LVC-PoS scheme together with a compatible linear encryption scheme into a PVSS protocol that ensures correctness, verifiability, and privacy in the public setting. We provide the full details of this compiler in Algorithm 1 and rigorously analyze its security in the subsequent subsections. The generic nature of our design enables flexible instantiation with a variety of lattice-based primitives, which we will explore in Section 4.

#### 3.1 Linear Vector Commitments with Proof of Smallness

We now define our extended vector commitment scheme, *LVC-PoS*. This primitive extends conventional vector commitments by enforcing linearity of the supported function family and providing explicit proofs of smallness for specific components of the input vector. The smallness proofs are essential in scenarios where certain quadratic relations, such as ensuring the binary nature or bounded norm of vector components, must be enforced alongside linear function correctness. By embedding these proofs within the VC structure, we ensure that our framework preserves both soundness and efficiency. We formally define the LVC-PoS scheme and its security properties below.

**Definition 3.1 (LVC-PoS).** A LVC-PoS scheme is parameterized by the family

 $\mathcal{F} = \{\mathcal{F}_{v,w,o} \subseteq \{f : \mathcal{R}^v \times \mathcal{R}^w \to \mathcal{R}^o\}\}_{v,w,o \in \mathbb{N}}$ 

of linear functions over  $\mathcal{R}$ , and an input alphabet  $\mathcal{X} \subseteq \mathcal{R}$ . The parameters v, w, and o represent the dimensions of public inputs, secret inputs, and outputs of the function f, respectively. The LVC-PoS scheme consists of the PPT algorithms (Setup, Com, Open, Verify) defined as follows:

- $pp \leftarrow Setup(1^{\lambda}, 1^{v}, 1^{w}, 1^{o}, S, \beta_{S})$ : Generate public parameters given the security parameter  $\lambda \in \mathbb{N}$ , dimensions  $v, w, o \in \mathbb{N}$ , a smallness bound  $\beta_{S}$ , and an index set  $S = \{s_{1}, s_{2}, ..., s_{k}\}$  indicating the positions in the input vector **x** for which smallness proofs are required.
- $-(c, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{pp}, \mathbf{x})$ : Compute a commitment c to the vector  $\mathbf{x} \in \mathcal{X}^w$ , along with auxiliary opening information  $\mathsf{aux}$ .

- $-\pi \leftarrow \text{Open}(\text{pp}, f, \mathbf{z}, aux)$ : Generate a proof  $\pi$  for function  $f \in \mathcal{F}_{v,w,o}$  evaluated at public input  $\mathbf{z} \in \mathcal{X}^v$ , together with proofs that each indexed subvector  $\mathbf{x}_s$ satisfies  $\|\mathbf{x}_s\| \leq \beta_S$  for all  $s \in S$ .
- $b \leftarrow \text{Verify}(pp, f, \mathbf{z}, \mathbf{y}, c, \pi)$ : Given public parameters, function f, public input  $\mathbf{z}$ , claimed output  $\mathbf{y} \in \mathcal{X}^o$ , commitment c, and proof  $\pi$ , output a bit b that decides whether:

$$f(\mathbf{z}, \mathbf{x}) = \mathbf{y}$$
 and  $\forall s \in S, \|\mathbf{x}_s\| \leq \beta_S$ 

*Remark on Proof Structure.* The proof  $\pi$  generated by the **Open** algorithm can be conceptually partitioned into two components:

- $-\pi_f$ : proof of correctness of the linear function evaluation.
- $\{\pi_s\}$ : individual proofs of smallness for each subvector in S.

Thus, the proof can be expressed as  $\pi = (\pi_f, \{\pi_s\})$ . For efficiency, aggregation techniques can be employed to compress multiple proofs into a single compact proof, which is beneficial for reducing communication and verification overhead.

**Definition 3.2 (Correctness).** An LVC-PoS scheme is correct if for any  $\lambda, v, w, o \in \mathbb{N}$ , any  $pp \in \text{Setup}(1^{\lambda}, 1^{v}, 1^{w}, 1^{o}, S, \beta_{S})$ , any  $(f, \mathbf{z}, \mathbf{x}, \mathbf{y}) \in \mathcal{F}_{v,w,o} \times \mathcal{X}^{v} \times \mathcal{X}^{w} \times \mathcal{X}^{o}$  such that:

 $f(\mathbf{z}, \mathbf{x}) = \mathbf{y}$  and  $\forall s \in S, \|\mathbf{x}_s\| \leq \beta_S$ 

and for any  $(c, aux) \leftarrow Com(pp, x)$  and  $\pi \leftarrow Open(pp, f, z, aux)$ , it holds that:

 $\mathsf{Verify}(\mathsf{pp}, f, \mathbf{z}, \mathbf{y}, c, \pi) = 1$ 

The notions of weak binding and functional hiding closely follow the standard definitions for vector commitments, which we recall in Appendix A.3. For clarity and self-containment, we also define them explicitly here.

**Definition 3.3 (Weak Biniding).** Let  $\rho : \mathbb{N}^3 \to [0,1]$ . A LVC-PoS scheme for  $\mathcal{F}, \mathcal{X}, \mathcal{Y}$ ) is said to be weakly binding if for any pair of PPT adversary  $\mathcal{A}$ and any  $s, w \in poly(\lambda)$  it holds that the following expression is upper-bounded by  $\rho(\lambda, s, w)$ :

$$\Pr\left[\begin{array}{l}\forall i \in \{0,1\},\\ \mathsf{Verify}(\mathsf{pp}, f_i, \mathbf{z}_i, \mathbf{y}_i, c, \pi_i) = 1,\\ \wedge f_0(\mathbf{z}_0, \cdot) = f_1(\mathbf{z}_1, \cdot) \wedge \mathbf{y}_0 \neq \mathbf{y}_1\end{array}\right| \mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda}, 1^v, 1^w, 1^o)\\ (c, (f_i, \mathbf{z}_i, \mathbf{y}_i, \pi_i)_{i=0}^1) \leftarrow \mathcal{A}(\mathsf{pp}) \right]$$

**Definition 3.4 (Functional Hiding).** An LVC-PoS scheme satisfies functional hiding if, given a commitment c and an opening proof  $\pi_f$  for a function f, no adversary can distinguish between commitments to different vectors  $\mathbf{x}, \mathbf{x}' \in \mathcal{X}^w$ , beyond what is revealed by  $f(\mathbf{z}, \mathbf{x})$  and the smallness proofs.

### 3.2 Compiler from Vector Commitment to PVSS

Our compiler generically transforms a VC scheme and a linear encryption scheme into a PVSS protocol. The central idea is to commit to the dealer's secret shares, the associated encryption randomness, and the polynomial coefficients used for secret sharing, all within a single vector. This commitment, together with an opening proof, serves as a verifiable certificate of correct share generation and encryption.

Setup. The Setup procedure initializes the system parameters for the PVSS protocol. A prime modulus p is selected to define the finite field over which Shamir's secret sharing will operate. The encryption scheme is instantiated to accommodate plaintext messages up to  $p^3$ , ensuring sufficient capacity for the shares. The vector size w accounts for the secret, the polynomial coefficients, the shares, and the encryption randomness. Meanwhile, the output size o determines the capacity for verifying the correctness of both the shares and the ciphertexts. Finally, the vector commitment parameters are established to handle this configuration and to support efficient proof generation and verification.

Key Generation. In this step, each participant generates their own encryption key pair along with a corresponding proof of correct key generation. These proofs ensure that the public keys are correctly formed and trustworthy, allowing all parties to verify the validity of the public encryption keys before they are used in the distribution of shares.

Distribution procedure. During the Distribution phase, the dealer constructs a degree-t polynomial where the secret is embedded as the constant term, and the remaining coefficients are randomly sampled. Each participant's share is computed by evaluating this polynomial at their index. These shares are then encrypted under the corresponding public keys using fresh randomness. To bind all these values together, the dealer commits to a vector containing the polynomial coefficients, the shares, and the encryption randomness. The opening function  $M(\mathbf{x})$  enforces two consistency checks: it verifies that each share corresponds to the correct evaluation of the polynomial, and that each ciphertext encrypts the corresponding share correctly. The dealer then generates both a commitment and an opening proof with respect to  $M(\mathbf{x})$ , producing a publicly verifiable distribution proof.

<u>Distribution Verification</u>. The correctness of the dealer's behavior is verified through the Distribution Verification procedure. Verifiers recompute the opening function  $M(\mathbf{x})$  and check the validity of the commitment and opening proof against the published ciphertexts and public parameters. This step ensures that the dealer has honestly generated and encrypted the shares, providing public verifiability without requiring trust in the dealer.

Decrypt Share. After distribution, each participant decrypts their ciphertext using the Decrypt Share procedure to recover their individual share. In addition to recovering the share, participants generate a decryption proof that certifies the correctness of their decryption relative to the original ciphertext. This decryption proof is crucial for ensuring the integrity of the reconstruction process, allowing other parties to verify that decrypted shares are valid.

# Algorithm 1 PVSS Framework

```
procedure SETUP(1^{\lambda}, 1^{n}, 1^{t})
      Let p = \text{poly}(\lambda) such that n < p
      \mathcal{E}.\mathsf{pp} \leftarrow \mathcal{E}.\mathsf{Setup}(1^{\lambda}, p^3)
      w \leftarrow n(r+1) + t + 1
      o \leftarrow n(m+1)
      \mathcal{VC}.\mathsf{pp} \leftarrow \mathcal{VC}.\mathsf{Setup}(1^{\lambda}, 1^{v}, 1^{w}, 1^{o}, \{n+t+2, \dots, w\}, \mathcal{E}.\mathsf{pp}.\mathcal{R})
      return pp = (n, t, p, \mathcal{E}.pp, \mathcal{VC}.pp)
procedure Key GENERATION(pp, i)
      (\mathsf{pk}_i,\mathsf{sk}_i,\mathsf{pf}_{Key,i}) \leftarrow \mathcal{E}.\mathsf{KeyGen}(\mathcal{E}.\mathsf{pp})
      \mathbf{return} \; \mathsf{sk}_i, \mathsf{pk}_i, \mathsf{pf}_{Key,i}
procedure KEY VERIFICATION(pp, i, pk<sub>i</sub>, pf<sub>Key,i</sub>)
      return \mathcal{E}.VerifyKey(\mathcal{E}.pp, pk_i, pf_{Key,i})
procedure DISTRIBUTION(pp, (pk_i), s)
      (a_1,\ldots,a_t) \leftarrow \mathbb{Z}_p^t
      \mathbf{a} \leftarrow (s, a_1, \ldots, a_t)
      for i = 1 to n do
            \mathbf{b}_i \leftarrow (i^0 \mod p, \dots, i^t \mod p)
            s_i \leftarrow \langle \mathbf{b}_i, \mathbf{a} \rangle
            r_i \leftarrow \mathcal{R}
            C_i \leftarrow \mathcal{E}.\mathsf{Encrypt}(\mathcal{E}.\mathsf{pp},\mathsf{pk}_i,s_i,r_i)
      \mathbf{x} \leftarrow (\mathbf{a}, s_1, \ldots, s_n, r_1, \ldots, r_n)
      Define opening function:
       M(\mathbf{x}) = \begin{cases} s_i = \langle \mathbf{b}_i, \mathbf{a} \rangle & \forall i \in [n] \\ \mathcal{E}.\mathsf{Encrypt}(\mathcal{E}.\mathsf{pp}, \mathsf{pk}_i, s_i, r_i) = C_i & \forall i \in [n] \end{cases}
      (c, \mathsf{aux}) \leftarrow \mathcal{VC}.\mathsf{Com}(\mathcal{VC}.\mathsf{pp}, \mathbf{x})
      pf \leftarrow \mathcal{VC}.Open(\mathcal{VC}.pp, M, \bot, aux)
      return Encrypted shares: (C_1, \ldots, C_n)
      return Distribution proof: pf_D = (c, pf)
procedure DISTRIBUTION VERIFICATION(pp, (C_i), pf_D, (pk_i))
      Recompute M as in Distribution
      \mathbf{y} \leftarrow ([0]^n, C_1, \ldots, C_n)
      return \mathcal{VC}.Verify(\mathcal{VC}.pp, M, \bot, \mathbf{y}, c, pf)
procedure DECRYPT SHARE(pp, i, pk<sub>i</sub>, sk<sub>i</sub>, C_i)
      s_i \leftarrow \mathcal{E}.\mathsf{Decrypt}(\mathcal{E}.\mathsf{pp},\mathsf{sk}_i,C_i)
      pf_{Dec,i} \leftarrow \mathcal{E}.ProveDecrypt(sk_i; (pk_i, s_i, C_i))
      return s_i, pf_{Dec,i}
procedure RECONSTRUCTION(pp, (s_i))
      return Lagrange interpolation over (s_i \mod p)
procedure DECRYPTION VERIFICATION(pp, i, pk_i, s_i, C_i, pf_{Dec,i})
      return \mathcal{E}.VerifyDecrypt(pk<sub>i</sub>, s<sub>i</sub>, C<sub>i</sub>, pf<sub>Dec i</sub>)
```

Decryption Verification. The validity of each participant's decryption is checked using the Decryption Verification procedure. Verifiers use this step to confirm that the decrypted share corresponds correctly to the ciphertext and the participant's public key. This verification ensures that dishonest participants cannot inject invalid shares into the reconstruction process.

<u>Reconstruction</u>. Finally, once a sufficient number of correct decrypted shares have been collected, the secret is reconstructed using the Reconstruction procedure. Standard Lagrange interpolation is applied over the finite field defined by modulus p, recovering the original secret without interaction between participants.

This design ensures that all critical steps of the PVSS protocol—distribution, encryption, decryption, and reconstruction—are fully verifiable and publicly auditable. By leveraging the binding and hiding properties of vector commitments, alongside efficient encryption schemes, our compiler guarantees both correctness and robustness. In the next section, we discuss how to instantiate this framework concretely with suitable lattice-based components to achieve practical and post-quantum secure PVSS constructions.

### 3.3 Security Proofs

In this section, we prove the security properties of our generic PVSS framework. Specifically, we prove that our construction satisfies correctness, verifiability of key generation, verifiability of distribution, verifiability of share decryption, and privacy (IND2-privacy) under standard assumptions on the underlying encryption scheme and vector commitment scheme. Each proof relies on the modular structure of our compiler, allowing security to be inherited directly from the underlying primitives.

**Theorem 3.5 (Correctness).** The PVSS framework described in Algorithm 1 satisfies correctness with t-reconstruction, provided that the VC scheme is complete and the encryption scheme is correct.

We defer the proof to Appendix B.2.

**Theorem 3.6 (Verifiability of Key Generation).** The PVSS framework satisfies verifiability of key generation, provided that the underlying encryption scheme supports verifiable key generation.

*Proof.* The Key Verification procedure in Algorithm 1 directly corresponds to the key verification mechanism of the underlying encryption scheme. Since we assume the encryption scheme provides verifiable key generation, correctness of this verification implies that any participant's public key is valid only if the associated proof  $pf_{Key,i}$  verifies successfully.

**Theorem 3.7 (Verifiability of Distribution).** The PVSS framework satisfies verifiability of distribution, provided that the underlying vector commitment scheme is sound.

*Proof.* The Distribution Verification procedure in Algorithm 1 leverages the verification mechanism of the vector commitment scheme. If the commitment is sound, then the verification procedure guarantees that the shares and ciphertexts are consistent with the committed data. Therefore, verifiability of distribution follows from the soundness of the vector commitment.

**Theorem 3.8 (Verifiability of Share Decryption).** The PVSS framework satisfies verifiability of share decryption, provided that the underlying encryption scheme supports proofs of correct decryption.

*Proof.* The Decryption Verification procedure in Algorithm 1 corresponds to the verification algorithm for the decryption correctness proof provided by the encryption scheme. Thus, assuming the encryption scheme correctly supports proof of decryption, this procedure ensures that only valid decrypted shares will be accepted.

**Theorem 3.9 (Privacy).** The PVSS framework achieves t-IND2-privacy, provided that the vector commitment scheme satisfies functional hiding and the encryption scheme is semantically secure.

*Proof.* Assume an adversary  $\mathcal{A}$  corrupts up to t parties. Let

$$\Pr[\mathsf{Game}^{\mathsf{ind-secrecy},0}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Game}^{\mathsf{ind-secrecy},1}_{\mathcal{A}}(\lambda) = 1] = \epsilon$$

denote the adversary's distinguishing advantage between two secret distributions.

We construct a distinguisher  $\mathcal{D}$  against the functional hiding property of the vector commitment scheme. The reduction proceeds as follows:

- $\mathcal{D}$  samples a random bit  $b \in \{0, 1\}$  and runs the setup procedure of the functional hiding game to obtain public parameters **pp**.
- $\mathcal{D}$  generates public keys and proofs for the n-t honest parties and receives from  $\mathcal{A}$  the public keys, proofs, and challenge secrets  $s^0, s^1$  for the corrupted parties.
- $-\mathcal{D}$  verifies the key generation proofs and aborts if any verification fails.
- Following the PVSS protocol,  $\mathcal{D}$  constructs the instance (M, x, z) and submits it to the functional hiding challenger, receiving the simulated commitment c and proof pf.
- $\mathcal{D}$  forwards the resulting public values  $(C_1, \ldots, C_n, c, \mathsf{pf})$  to  $\mathcal{A}$  and outputs the adversary's guess  $b^*$  as its own guess.

If the adversary  $\mathcal{A}$  can distinguish between the distributions of  $s^0$  and  $s^1$  with advantage  $\epsilon$ , then  $\mathcal{D}$  breaks the functional hiding property of the vector commitment scheme with the same advantage, contradicting its assumed security.

# 3.4 Computation Cost of Our PVSS Compiler

The computational cost of our PVSS protocol is analyzed by considering each procedure defined in Algorithm 1. We express these costs in terms of operations performed by the underlying cryptographic primitives: a linear encryption scheme  $\mathcal{E}$  and a vector commitment scheme  $\mathcal{VC}$ . Let n be the number of participants and t be the threshold. The vector  $\mathbf{x} = (\mathbf{a}, s_1, \ldots, s_n, r_1, \ldots, r_n)$  contains the t + 1 coefficients of the polynomial  $\mathbf{a} = (s, a_1, \ldots, a_t)$ , the n shares  $(s_1, \ldots, s_n)$ , and the n randomness values  $(r_1, \ldots, r_n)$  used for encryption. Assuming that each randomness  $r_i$  is treated as r elements for length counting purposes, the total length of the committed vector  $\mathbf{x}$  is t + 1 + n + nr = O(nr).

The subscript  $p^3$  associated with the  $\mathcal{E}$  operations indicates that the scheme is instantiated to support a message space of size  $p^3$ , where p is the modulus for Shamir's secret sharing. This ensures that inner products used to compute shares  $s_i = \langle \mathbf{b}_i, \mathbf{a} \rangle$  do not overflow, i.e.,  $s_i < p^3$ . The parameters  $p^3, nr, n$  for the  $\mathcal{VC}$ operations refer to characteristics related to the bound of the elements involved, the length of the committed vector, and the number of outputs, respectively. The 2n relations checked by the opening function  $M(\mathbf{x})$  are implicitly handled by the  $\mathcal{VC}$  operations, whose costs depend primarily on nr and n.

<u>Setup</u>. This procedure is run once, typically by a trusted party or as a distributed process, to establish the common parameters. It involves initializing the encryption scheme parameters via  $\mathcal{E}$ .Setup $(1^{\lambda}, p^3)$ ; this has a computational cost of  $\operatorname{op}_{\mathcal{E}.\mathsf{Setup}_{p^3}}$ . Additionally, it sets up the vector commitment public parameters through  $\mathcal{VC}.\mathsf{Setup}(1^{\lambda}, 1^v, 1^w, 1^o, \ldots)$ . The crucial parameters for the VC scheme derived from our framework are the vector length nr, and the number of relations 2n. The cost of this VC setup is denoted as  $\operatorname{op}_{\mathcal{VC}.\mathsf{Setup}_{p^3,nr,n}}$ . The total computational cost for the Setup procedure is therefore  $\operatorname{op}_{\mathcal{E}.\mathsf{Setup}_{p^3}} + \operatorname{op}_{\mathcal{VC}.\mathsf{Setup}_{p^3,nr,n}}$ .

Key Generation. Each of the *n* participants executes this procedure once to generate their individual encryption key pair  $(\mathsf{pk}_i, \mathsf{sk}_i)$  and a corresponding proof of correct key generation  $\mathsf{pf}_{Key,i}$ . This is performed by calling  $\mathcal{E}.\mathsf{KeyGen}(\mathcal{E}.\mathsf{pp})$ . The computational cost incurred by each participant for Key Generation is  $\mathsf{op}_{\mathcal{E}.\mathsf{KeyGen}_{n^3}}$ .

Key Verification. To verify a participant's public key  $\mathsf{pk}_i$  using its associated proof  $\mathsf{pf}_{Key,i}$ , any interested party (e.g., the dealer, other participants) invokes  $\mathcal{E}$ .VerifyKey( $\mathcal{E}.\mathsf{pp},\mathsf{pk}_i,\mathsf{pf}_{Key,i}$ ). The computational cost for verifying a single public key is  $\mathsf{op}_{\mathcal{E}}.\mathsf{VerifyKey}_{n^3}$ .

<u>Distribution</u>. This procedure is executed once by the dealer to share the secret s.

- 1. Polynomial definition and evaluation: The dealer defines a degree-t polynomial  $P(x) = s + a_1 x + \cdots + a_t x^t$ . Then, n shares  $s_i = P(i)$  are computed. Each evaluation  $s_i = \langle \mathbf{b}_i, \mathbf{a} \rangle$  (where  $\mathbf{b}_i = (i^0, \dots, i^t)$ ) requires O(t) arithmetic operations in  $\mathbb{Z}_p$ . This step totals  $n \cdot O(t)$  operations.
- 2. Encryption of shares: Each share  $s_i$  is encrypted under the respective participant's public key  $\mathsf{pk}_i$  using fresh randomness  $r_i: C_i \leftarrow \mathcal{E}.\mathsf{Encrypt}(\mathcal{E}.\mathsf{pp},\mathsf{pk}_i,s_i,r_i)$ . This involves n encryptions, leading to a total cost of  $n \cdot \mathsf{op}\mathcal{E}.\mathsf{Encrypt}_{p^3}$ .
- 3. Vector Commitment: The dealer commits to the vector  $\mathbf{x} = (\mathbf{a}, s_1, \ldots, s_n, r_1, \ldots, r_n)$  of length nr. This operation,  $(c, \mathsf{aux}) \leftarrow \mathcal{VC}.\mathsf{Com}(\mathcal{VC}.\mathsf{pp}, \mathbf{x})$ , has a cost of  $\mathsf{op}_{\mathcal{VC}.\mathsf{Com}_{p^3, nr, n}}$ .

4. Opening Proof Generation: An opening proof  $\mathsf{pf}$  is generated for the relations defined in  $M(\mathbf{x})$ . These relations ensure that  $s_i = \langle \mathbf{b}_i, \mathbf{a} \rangle$  and  $\mathcal{E}.\mathsf{Encrypt}(\mathcal{E}.\mathsf{pp},\mathsf{pk}_i, s_i, r_i) = C_i$  for all  $i \in [n]$ . This step,  $\mathsf{pf} \leftarrow \mathcal{VC}.\mathsf{Open}(\mathcal{VC}.\mathsf{pp}, M, \bot, \mathsf{aux})$ , costs  $\mathrm{op}_{\mathcal{VC}.\mathsf{Open}(p^3, nr, n)}$ .

The overall computational cost for the Distribution procedure, approximating the  $n \cdot O(t)$  term as  $n^2$  (as t = O(n) is typical), is:

$$n(n + \mathrm{op}_{\mathcal{E}.\mathsf{Encrypt}_{p^3}}) + \mathrm{op}_{\mathcal{VC}.\mathsf{Com}_{p^3,nr,n}} + \mathrm{op}_{\mathcal{VC}.\mathsf{Open}(p^3,nr,n)}$$

<u>Distribution Verification</u>. This procedure can be performed by any party to verify the integrity of the dealer's actions using the public ciphertexts  $(C_1, \ldots, C_n)$ and the distribution proof  $pf_D = (c, pf)$ . It involves reconstructing the opening function M (which is a specification and has negligible computational cost) and then executing  $\mathcal{VC}$ . Verify $(\mathcal{VC}.pp, M, \perp, \mathbf{y}, c, pf)$ , where  $\mathbf{y} = ([0]^n, C_1, \ldots, C_n)$ . The dominant computational cost for Distribution Verification is  $op_{\mathcal{VC}.Verify_{p^3,nr,n}}$ . Decrypt Share. Each participant  $P_i$  who has received an encrypted share  $C_i$  performs this procedure.

- 1. Decryption: The participant decrypts  $C_i$  using their secret key  $\mathsf{sk}_i$  to obtain the share  $s_i \leftarrow \mathcal{E}.\mathsf{Decrypt}(\mathcal{E}.\mathsf{pp},\mathsf{sk}_i,C_i)$ . This costs  $\operatorname{op}_{\mathcal{E}.\mathsf{Decrypt}_{n^3}}$ .
- Proof of Decryption: A proof pf<sub>Dec,i</sub> is generated certifying that s<sub>i</sub> is the correct decryption of C<sub>i</sub> with respect to pk<sub>i</sub>. This is done via *E*.ProveDecrypt(sk<sub>i</sub>; (pk<sub>i</sub>, s<sub>i</sub>, C<sub>i</sub>)), costing op<sub>*E*.ProveDecrypt<sub>3</sub></sub>.

The total computational cost for a participant to decrypt their share and generate the accompanying proof is  $op_{\mathcal{E}.\mathsf{Decrypt}_{n^3}} + op_{\mathcal{E}.\mathsf{ProveDecrypt}_{n^3}}$ .

Decryption Verification. To confirm the validity of a decrypted share  $s_i$  (provided by participant  $P_i$  along with  $C_i$  and  $pf_{Dec,i}$ ), any party can perform this verification. It involves calling  $\mathcal{E}$ .VerifyDecrypt( $pk_i, s_i, C_i, pf_{Dec,i}$ ). This step is crucial before using the share  $s_i$  in the reconstruction phase. The computational cost for verifying one decrypted share is  $op_{\mathcal{E}.VerifyDecrypt_3}$ .

<u>Reconstruction</u>. Once a threshold of at least t + 1 valid decrypted shares  $(i, s_i)$  are collected, the original secret s (i.e., P(0)) is recovered using Lagrange interpolation over the field  $\mathbb{Z}_p$ . The computational cost for Reconstruction using standard interpolation algorithms is  $O(t^2)$  arithmetic operations in  $\mathbb{Z}_p$ . More advanced techniques (e.g., FFT-based algorithms) can reduce this to  $O(t \log^2 t)$  or  $O(t \log t)$ ; we conservatively state  $O(t^2)$ .

Table 2 below summarizes the computational cost for each procedure.

Procedure	Computation Cost
Setup	$\mathrm{op}_{\mathcal{E}.Setup_{p^3}} + \mathrm{op}_{\mathcal{VC}.Setup_{p^3,nr,n}}$
Key Generation (per participant)	$\operatorname{Op}_{\mathcal{E}}.KeyGen_{p^3}$
Key Verification (per key)	$\operatorname{op}_{\mathcal{E}}.\operatorname{VerifyKey}_{p^3}$
Distribution	$n(n + \operatorname{op}_{\mathcal{E}.Encrypt_{p^3}}) + \operatorname{op}_{\mathcal{VC}.Com_{p^3,nr,n}} + \operatorname{op}_{\mathcal{VC}.Open_{p^3,nr,n}}$
Distribution Verification	$\operatorname{Op}_{\mathcal{VC}}.\operatorname{Verify}_{p^3,nr,n}$
Decrypt Share (per share)	$\operatorname{op}_{\mathcal{E}.Decrypt_{p^3}} + \operatorname{op}_{\mathcal{E}.ProveDecrypt_{p^3}}$
Decryption Verification (per share)	$\operatorname{Op}_{\mathcal{E}}.\operatorname{VerifyDecrypt}_{n^3}$
Reconstruction	$O(t^2)$ operations in $\mathbb{Z}_p$

**Table 2.** Computation costs for the core procedures in our PVSS protocol. All encryption-related costs are parameterized by the message space size  $p^3$ . VC costs are driven by the committed vector length O(nr) and output size n.

# 4 Instantiations: PVSS Constructions

In this section, we provide three instantiations of the proposed framework. First, we instantiate our extended vector commitment (LVC-PoS) using the VC scheme from [2]. Then, we construct two linear encryption schemes inspired by [21, 15], both tailored to support publicly verifiable decryption within our framework. Our third instantiation utilizes the compact ring-based encryption scheme by Lyubashevsky *et al.* [16], offering a more efficient structure in the ring setting. Together with our compiler, these components yield three fully lattice-based PVSS constructions.

# 4.1 Instantiation of LVC-PoS

In this section, we present our extended vector commitment, *Proof-of-Smallness* Vector Commitment (LVC-PoS), which serves as a foundational component in our PVSS construction. Starting from the VC scheme of Albrecht *et al.* [2], we make two essential modifications:

- We restrict its evaluation to linear and multi-output linear functions, suitable for our framework.
- We extend it to support proofs of smallness, including a specific focus on binary satisfiability of committed vectors.

These extensions are crucial. In our PVSS, the committed vectors include secret sharing coefficients, shares, and encryption randomness — all of which must be proven to be well-formed and small. Since the vectors in our construction are inherently binary and of fixed length, this guarantees smallness automatically (bounded by  $2^{\text{length of randomness}}$ ). For general cases with larger input lengths, we refer to techniques such as those proposed by Libert *et al.* [14].

We formally define our construction in two stages:

#### Algorithm 2 Vector Commitment [2]

procedure SETUP $(1^{\lambda}, 1^{v}, 1^{w}, 1^{o})$  $\mathbf{v} \leftarrow (R_q^{\times})^w, h \leftarrow R_p^o$  $(\mathbf{A}, td) = \mathsf{TrapGen}(1^{\eta}, 1^{l}, q, R, \beta)$  $t \leftarrow T$  $u_{i,j} = \mathsf{SampPre}(td, \frac{v_i}{v_j}t, \beta), \forall i, j \in \mathbb{Z}_w | i \neq j$ return  $(\mathbf{A}, t, \{u_{i,j}\}, \mathbf{v}, h)$ procedure COM(pp, x)  $c = \langle \mathbf{v}, \mathbf{x} \rangle \mod q$ for i = 1 to w do  $u_i = \sum_{j \neq i} x_j \cdot u_{j,i}$  $\operatorname{aux} = (u_i)_{i \in \mathbb{Z}_w}$ return (c, aux)**procedure** OPEN(pp, f, z, aux)  $\begin{aligned} \pi = \sum_{i \in \mathbb{Z}_o} \sum_{j \in \mathbb{Z}_w} h_i f_{i,j}(\mathbf{z}) u_j \\ \text{return } \pi \end{aligned}$ **procedure** VERIFY(pp,  $f, \mathbf{z}, \mathbf{y}, c, \pi$ )  $a_{1} = \left(\mathbf{A} \cdot \mathbf{u} \stackrel{?}{=} \left(\sum_{k \in \mathbb{Z}_{o}} h_{k} \left(\sum_{j \in \mathbb{Z}_{w}} f_{k,j}(\mathbf{z}) \frac{c}{v_{j}} - y_{k}\right)\right) \cdot t \mod q\right)$  $a_{2} = \left(\|\mathbf{u}\| \stackrel{?}{\leq} \delta_{0}\right)$ return  $a_1 \wedge a_2$ 

- First, we recall the base VC scheme [2] (Algorithm 2).

- Second, we extend it into a lattice-based LVC-PoS scheme (Algorithm 3).

**Base Vector Commitment (Algorithm 2).** The goal of the VC scheme is to allow a dealer to commit to a vector  $\mathbf{x}$  and later provide succinct, non-interactive proofs that linear relations over  $\mathbf{x}$  hold, without revealing  $\mathbf{x}$  itself. We recall the VC scheme [2] in Algorithm 2. For more details see Appendix A.4.

**LVC-PoS Scheme (Algorithm 3).** We now extend the VC scheme to support proof of smallness and binary satisfiability.

Setup: The setup is similar to the base VC but with a refined commitment key:

$$\mathbf{v}' \leftarrow (R_p^{\times})^w, \quad \mathbf{v} = \frac{1}{\mathbf{v}'} \mod q$$

This transformation simplifies the commitments required for verifying quadratic relations in the binary proof.

<u>Commit</u>: Commitment is delegated to the base VC. Additionally, we define an internal function  $\overline{\text{Com}}$  for auxiliary commitments:  $\overline{c} = \langle \mathbf{x}, \frac{1}{\mathbf{v}} \rangle \mod q$ . Open and Verify: We combine three proof components:

- Base proof  $\pi$  (as in VC).
- Proof  $\pi_{eq}$  ensuring consistency of auxiliary commitment  $\bar{c}_{h\circ\mathbf{x}}$  with  $h\circ\mathbf{x}$ .
- Inner product proof  $\pi_{ip}$  verifying the binary relation.

The final verification checking  $c_{\mathbf{x}} \cdot \overline{c}_{h \circ (\mathbf{x}-1)} \cdot t \stackrel{?}{=} \langle \mathbf{A}, \pi_{ip} \rangle$  confirms that the committed vector  $\mathbf{x}$  is binary.

# Algorithm 3 LVC-PoS

procedure SETUP $(1^{\lambda}, 1^{v}, 1^{w}, 1^{o}, S, \beta_{S})$ Let  $\mathbf{v}' \leftarrow (R_p^{\times})^w$ Let  $\mathbf{v} = \frac{1}{\mathbf{v}'} \mod q$   $h \leftarrow R_p^o$   $(\mathbf{A}, td) = \mathsf{TrapGen}(1^\eta, 1^l, q, R, \beta)$  $t \leftarrow T$  $\triangleright \text{ Where } G = \left\{ \frac{v_i}{v_i} | i \neq j \right\}$  $u_g = \mathsf{SampPre}(td, g(v)t, \beta), \forall g \in G$ return Public Parameters:  $(\mathbf{A}, t, (u_g)_{g \in G}, v, h, S, \beta_S)$ procedure  $\overline{\text{COM}}(pp, \mathbf{x})$  $\triangleright$  Inner function Let  $c = \langle \frac{1}{\mathbf{v}}, \mathbf{x} \rangle \mod q$ for  $i \in \{1, \dots, w\}$  do Let  $u_i = \sum_{i \neq j=1}^w x_j u_{\frac{v_i}{v_j}}$ Let  $aux = (u_i)_{i \in \{1,...,w\}}$ return Commitment: (c, aux) procedure COM(pp, x) return Commitment: VC7.Com(pp, x) **procedure** OPEN(pp, f, z, aux) Let  $\pi = \mathcal{VC}.\mathsf{Open}(\mathsf{pp}, f, \mathbf{z}, \mathsf{aux})$ Let  $\overline{c}_{\mathbf{h}\circ\mathbf{x}}$ ,  $\mathsf{aux}_{\mathbf{h}\circ\mathbf{x}} = \overline{\mathsf{Com}}(\chi_S(\mathbf{h})\circ\mathbf{x})$ Let  $\pi_{eq} = \mathcal{VC}.\mathsf{Open}(\mathsf{pp}, \frac{\chi_S(\mathbf{h})}{\mathbf{v}}, \mathbf{z}, \mathsf{aux_{hox}})$ Let  $\pi_{ip} = \sum_{i,j \in \mathbb{Z}_w: i \neq j} x_i((\chi_S(h))_j(x_j - 1))\mathbf{u}_{i,j}$ **return** Proof:  $(\pi, \overline{c}_{\mathsf{hox}}, \pi_{eq}, \pi_{ip})$ **procedure** VERIFY(**pp**, f, **z**, **y**, c,  $\pi$ ) Let  $b_1 = \mathcal{VC}$ . Verify(pp,  $f, \mathbf{z}, \mathbf{y}, c, \pi$ ) Let  $b_2 = \mathcal{VC}$ . Verify(pp,  $\frac{\mathbf{h}}{\mathbf{v}}, \mathbf{z}, \overline{c}_{\mathbf{hox}}, c_{\mathbf{x}}, \pi_{eq}$ ) Let  $\overline{c}_{\mathbf{h} \circ (\mathbf{x}-1)} = \overline{c}_{\mathbf{h} \circ \mathbf{x}} - \langle \mathbf{h}, \frac{1}{\mathbf{y}} \rangle$ Let  $b_3 = (c_{\mathbf{x}} \times \overline{c}_{\mathbf{h} \circ (\mathbf{x}-1)} \times t \stackrel{?}{=} < \mathbf{A}, \pi_{ip} > \times t)$ return  $\bigwedge_{i=1}^3 b_i$ 

Binary Satisfiability Proof. Finally, we prove that a committed vector  $\mathbf{x}$  is binary. The prover constructs:

$$\pi_{ip} = \sum_{i,j \in \mathbb{Z}_w, i \neq j} x_i \left( h_j (x_j - 1) \right) u_{i,j}$$

The verifier reconstructs  $\overline{c}_{h\circ(\mathbf{x}-1)} = \overline{c}_{h\circ\mathbf{x}} - \langle h, \frac{1}{\mathbf{v}} \rangle$  and checks  $c_{\mathbf{x}} \cdot \overline{c}_{h\circ(\mathbf{x}-1)} \cdot t \stackrel{?}{=} \langle \mathbf{A}, \pi_{ip} \rangle$ . Thus, we conclude that  $\mathbf{x}$  is a binary vector.

**Theorem 4.1 (Security Properties of LVC-PoS).** Assuming the correctness, functional hiding, and binding properties of the underlying VC scheme (Algorithm 2), the LVC-PoS (Algorithm 3) achieves the following properties:

- Correctness: Algorithm 3 correctly verifies both the evaluation of linear functions and the smallness of specified indices in the committed vector.
- Functional Hiding: Algorithm 3 preserves functional hiding for linear function openings, meaning that the commitment and proof reveal no information about the committed vector beyond the output of the opened function and smallness verification.
- Weak Binding: Algorithm 3 satisfies weak binding, ensuring that after committing to a vector, it is computationally infeasible to produce two distinct valid openings.

We defer the proof to Appendix B.1.

### 4.2 Instantiation of Linear Encryption Scheme I

In this section, we present our first instantiation of the linear encryption scheme, which we refer to as *Linear Encryption Scheme I*. This scheme is built upon lattice-based assumptions and is designed to work seamlessly with our vector commitment construction and PVSS framework. It supports both encryption of messages (shares in our setting) and efficient public proofs of correct decryption, an essential feature for enabling verifiability in distributed systems.

We detail the design of this scheme in Algorithm 4, and we now explain each component of the construction, step by step, highlighting the technical rationale and the role of each operation.

<u>Setup</u>: The scheme begins with the parameter setup, as specified in the Setup procedure of Algorithm 4. We select appropriate cryptographic parameters:

- Define  $k = \lfloor \log(p) \rfloor$  to capture the bit length of the modulus.
- Select dimensions: l (rows of public keys), m = 2lk (columns of public keys), and gadget dimension d.
- Define the error bound parameter  $\alpha = \frac{\sqrt{p}}{4}$ , which limits noise growth in encryption and ensures correctness of decryption and proofs.

These parameters balance security and efficiency, and set the environment for trapdoor generation and gadget operations.

Key Generation: In the Key Generation procedure, we generate both the public and secret keys:

– Use the trapdoor generation function to sample a public matrix  $\mathbf{A} \in \mathbb{Z}_p^{l \times m}$ along with its associated trapdoor td:

$$(\mathbf{A}, td) = \operatorname{TrapGen}(1^{\mathrm{l}}, 1^{\mathrm{m}}, 1^{\mathrm{p}}, \alpha)$$

- Generate a key proof:  $pf = \text{SampPre}(td, \text{Hash}(\mathbf{A}), \alpha)$ .

The proof pf provides public verifiability of the key generation without revealing td, which is crucial for the non-interactive verifiability of our PVSS scheme. Key Verification. It checks the correctness of the public key using the proof pf:

$$\mathbf{A} \cdot \mathsf{pf} \stackrel{?}{=} \operatorname{Hash}(\mathbf{A}) \land \|\mathsf{pf}\| \stackrel{?}{<} \alpha$$

Algorithm 4 Linear Encryption Scheme I

```
procedure SETUP(1^{\lambda}, p)
       k = \lceil \log(p) \rceil
       Select l \in \mathbb{N}, m = 2lk, d \in \mathbb{N}, and \alpha = \frac{\sqrt{p}}{4}
       return (p, l, m, d, \alpha)
procedure Key GENERATION(p, l, m, d, \alpha)
       (\mathbf{A}, td) = \operatorname{TrapGen}(1^{\mathrm{I}}, 1^{\mathrm{m}}, 1^{\mathrm{p}}, \alpha)
       pf = SampPre(td, Hash(\mathbf{A}), \alpha)
       return Secret Key: td, Public Key: A, Key Proof: pf
procedure Key VERIFICATION(A, pf)
       \mathbf{a} = \mathbf{A} \cdot \mathsf{pf}
       \mathbf{b} = \text{Hash}(\mathbf{A})
       return a \stackrel{?}{=} b and \|\mathbf{pf}\| \stackrel{?}{<} \alpha
procedure ENCRYPTION(\mathbf{A}, m, (\mathbf{B}, \mathbf{f}, \mathbf{e}, \mathbf{e}'))
       \mathbf{U}=\mathbf{A}\cdot\mathbf{B}
       \begin{split} \mathbf{h}^\top &= \mathbf{f}^\top \cdot \mathbf{A} + \mathbf{e}^\top \\ \mathbf{C}^\top &= \mathbf{f}^\top \cdot \mathbf{U} + \mathbf{e}'^\top + m \cdot \mathbf{g}^\top \end{split}
       return (\mathbf{U}, \mathbf{h}, \mathbf{C})
procedure DECRYPTION(td, (\mathbf{U}, \mathbf{h}, \mathbf{C}))
       \mathbf{B}' = \operatorname{SampPre}(\operatorname{td}, \mathbf{U}, \alpha)
       m = \text{GadgetSolve}(\mathbf{C} - \mathbf{h}^{\top} \cdot \mathbf{B}')
       return m
procedure PROVE DECRYPTION(td, \mathbf{A}, m, (\mathbf{U}, \mathbf{h}, \mathbf{C}))
       \mathbf{B}' = \operatorname{SampPre}(\operatorname{td}, \mathbf{U}, \alpha)
       \mathbf{e} = \mathbf{h}^\top \cdot \mathbf{B}' + m \cdot \mathbf{g}^\top - \mathbf{C}
       return (\mathbf{e}, \mathbf{B}')
procedure VERIFY DECRYPTION(\mathbf{A}, m, (\mathbf{U}, \mathbf{h}, \mathbf{C}), (\mathbf{e}, \mathbf{B}'))
       a_1 = \mathbf{A} \cdot \mathbf{B}'
       b_1 = \mathbf{U}
       a_2 = \mathbf{h}^\top \cdot \mathbf{B}' + \mathbf{e}^\top + m \cdot \mathbf{g}^\top
       b_2 = \mathbf{C}
       return a_1 \stackrel{?}{=} b_1 and a_2 \stackrel{?}{=} b_2 and \|\mathbf{e}\| \stackrel{?}{<} \alpha and \|\mathbf{B'}\| \stackrel{?}{<} \alpha
```

If both conditions hold, the key is deemed valid. This ensures honest parameter generation and prevents malicious setup attacks.

Encryption: To encrypt a message m, as described in Encryption, the sender performs the following:

- Sample randomness:  $\mathbf{B} \in \mathbb{Z}_p^{m \times d}$ ,  $\mathbf{f} \in \mathbb{Z}_p^l$ ,  $\mathbf{e} \in \mathbb{Z}_p^m$  (with  $\|\mathbf{e}\| < \alpha$ ), and  $\mathbf{e}' \in \mathbb{Z}_p^d$  (with  $\|\mathbf{e}'\| < \alpha$ ).
- Compute  $\mathbf{U} = \mathbf{A} \cdot \mathbf{B}$ ,  $\mathbf{h}^{\top} = \mathbf{f}^{\top} \cdot \mathbf{A} + \mathbf{e}^{\top}$ , and  $\mathbf{C}^{\top} = \mathbf{f}^{\top} \cdot \mathbf{U} + \mathbf{e}'^{\top} + m \cdot \mathbf{g}^{\top}$ . Where  $\mathbf{g}$  is the public gadget vector.

The ciphertext is the tuple  $(\mathbf{U}, \mathbf{h}, \mathbf{C})$ , which encapsulates both the randomized encryption and the structured form enabling efficient decryption and proof generation.

Decryption: As shown in the Decryption procedure, the decryptor uses the trapdoor td to sample:  $\mathbf{B}' = \text{SampPre}(\text{td}, \mathbf{U}, \alpha)$ . With this, the message is recovered by solving:  $m = \text{GadgetSolve}(\mathbf{C} - \mathbf{h}^{\top} \cdot \mathbf{B}')$ . The correctness of this step follows from the structured form of the ciphertext.

Prove Decryption: To enable public verifiability of decryption, the decryptor runs **Prove Decryption**. Using *td*, the decryptor computes:  $\mathbf{B}' = \text{SampPre}(\text{td}, \mathbf{U}, \alpha)$ and  $\mathbf{e} = \mathbf{h}^{\top} \cdot \mathbf{B}' + m \cdot \mathbf{g}^{\top} - \mathbf{C}$ . The proof consists of the tuple  $(\mathbf{e}, \mathbf{B}')$ , which shows that the ciphertext decrypts to *m* correctly.

Verify Decryption: Finally, anyone can run Verify Decryption to check the correctness of the decryption proof. Specifically, the verifier checks:

- 1. Consistency of  $\mathbf{B}'$  with  $\mathbf{U}: \mathbf{A} \cdot \mathbf{B}' \stackrel{?}{=} \mathbf{U}$ .
- 2. Consistency of **e** with ciphertext **C**:  $\mathbf{h}^{\top} \cdot \mathbf{B}' + \mathbf{e}^{\top} + m \cdot \mathbf{g}^{\top} \stackrel{?}{=} \mathbf{C}$ .
- 3. Norm bounds:  $\|\mathbf{e}\| \stackrel{?}{<} \alpha$ ,  $\|\mathbf{B'}\| \stackrel{?}{<} \alpha$ .

If all checks succeed, the decryption proof is accepted.

**Lemma 4.2.** Let Regev encryption scheme be IND-CPA, then the encryption scheme I in Algorithm 4 satisfies IND-CPA security.

Proof. The Regev encryption scheme is provably IND-CPA secure under the LWE assumption. Algorithm 4 presents a variant of Regev's scheme where:

- The key generation outputs  $(\mathbf{A}, td) = \text{TrapGen}(1^1, 1^m, 1^p, \alpha)$  and  $\mathsf{pf} = \text{SampPre}(td, \text{Hash}(\mathbf{A}), \alpha)$ . The view of the attacker is  $(\mathbf{A}, \mathsf{pf})$  which is pseudorandom [12].
- The encryption algorithm outputs  $\mathbf{U} = \mathbf{A} \cdot \mathbf{B}$  and  $\mathbf{h}^{\top} = \mathbf{f}^{\top} \cdot \mathbf{A} + \mathbf{e}^{\top}$  and  $\mathbf{C}^{\top} = \mathbf{f}^{\top} \cdot \mathbf{U} + \mathbf{e}'^{\top} + m \cdot \mathbf{g}^{\top}$ . The view of the attacker is  $(\mathbf{U}, \mathbf{h}, \mathbf{C})$ , where  $\mathbf{U}$  is pseudorandom under the SIS assumption and  $(\mathbf{h}, \mathbf{C})$  is pseudorandom under the LWE assumption.

Consequently, Algorithm 4 satisfies IND-CPA security.

**Lemma 4.3.** The encryption scheme 4 satisfies verifiability of key generation and verifiability of decryption.

Proof. The verifiability of key generation follows directly from SampPre definition from [12]. The verifiability of decryption follows from the fact that the ProveDecrypt outputs randomness as  $pf_{Dec}$ , and the verification algorithm checks consistency of the Encrypt algorithm.

**Theorem 4.4 (Security of PVSS Instantiation with Linear Encryption Scheme I).** Assuming the hardness of the LWE and SIS problems, and the security properties of the LVC-PoS and Linear Encryption Scheme I, our instantiated PVSS construction achieves correctness, verifiability, and t-IND2 privacy.

We defer the proof to Appendix B.3.

Algorithm 5 Linear Encryption Scheme II

procedure SETUP $(1^{\lambda}, p)$  $k = \lceil \log(p) \rceil$ . Select  $l \in \mathbb{N}, m = 2lk, d \in \mathbb{N}, \alpha = \frac{\sqrt{p}}{4}$ , and  $\mathbf{A} \leftarrow \mathbb{Z}_p^{l \times m}$ return  $(p, l, m, d, \alpha, \mathbf{A})$ **procedure** Key GENERATION $(p, l, m, d, \alpha, \mathbf{A})$  $\mathsf{sk} \leftarrow \{0,1\}^m, \, \mathsf{pk} = \mathbf{A} \cdot \mathsf{sk}$  $pf_{Key} = id.prove(\mathbf{A}, pk, sk)$  [15] return Secret Key: sk, Public Key: pk, Key Proof: pf<sub>Key</sub> procedure Key Verification(A, pk, pf) **return** id.verify( $\mathbf{A}, \mathsf{pk}, \mathsf{pf}_{Kev}$ ) [15] procedure ENCRYPTION( $\mathbf{A}, \mathsf{pk}, m, (\mathbf{B}, \mathbf{e}, \mathbf{e}')$ )  $C = B \cdot A + e$  $\mathbf{C}' = \mathbf{B} \cdot \mathsf{pk} + \mathbf{e}' + m \cdot \mathbf{g}$ return  $(\mathbf{C}, \mathbf{C}')$ procedure Decryption(sk, (C, C')) $\mathbf{B}' = \mathbf{C} \cdot \mathsf{sk}$  $m = \text{GadgetSolve}(\mathbf{C}' - \mathbf{B}')$ return m**procedure** PROVE DECRYPTION( $\mathsf{sk}, \mathsf{pk}, m, (\mathbf{C}, \mathbf{C}')$ )  $\mathbf{B}' = \mathbf{C} \cdot \mathsf{sk}$  $\mathbf{e} = \mathbf{B}' + m \cdot \mathbf{g} - \mathbf{C}'$  $pf_{Dec} = id.prove(\mathbf{C}, \mathbf{B}', \mathsf{sk})$  [15] return  $(\mathbf{e}, \mathbf{B}', \mathsf{pf}_{Dec})$ **procedure** DECRYPTION VERIFICATION( $pk, m, C, C', e, B', pf_{Dec}$ )  $a = \mathbf{B}' + \mathbf{e} + m \cdot \mathbf{g}$  $b = \mathbf{C}'$ return  $a \stackrel{?}{=} b$  and  $\|\mathbf{e}\| \stackrel{?}{<} \alpha$  and id.verify( $\mathbf{C}, \mathbf{B}', \mathsf{pf}_{\mathsf{Dec}}$ ) [15]

# 4.3 Instantiation Linear Encryption Scheme II

We now present our second instantiation of the linear encryption scheme, which we refer to as *Linear Encryption Scheme II*. Unlike the first scheme, this design leverages direct lattice-based encryption using secret keys sampled from binary spaces, and public keys validated via lattice-based identification protocols [15]. This approach offers strong structural simplicity and compatibility with our compiler and verification mechanisms. Algorithm 5 outlines the full construction.

We explain each step below, highlighting both the cryptographic intuitions and operational flow.

Setup: As defined in Setup of Algorithm 5, we initialize public parameters:

- Security parameter  $\lambda$ , and modulus p.
- Dimension parameters:  $k = \lceil \log(p) \rceil$ ,  $l \in \mathbb{N}$  (rows of public keys), m = 2lk (columns of public keys), and gadget dimension  $d \in \mathbb{N}$ .
- Error bound  $\alpha = \frac{\sqrt{p}}{4}$ , ensuring correctness and noise control.

We then sample a random matrix  $\mathbf{A} \in \mathbb{Z}_p^{l \times m}$  as part of the public parameters, which is shared by all parties.

Key Generation: As shown in Key Generation, the secret key sk is sampled uniformly at random from binary space sk  $\leftarrow \{0, 1\}^m$ . The corresponding public key is computed as  $\mathsf{pk}_{Key} = \mathbf{A} \cdot \mathsf{sk}$ . To allow public verifiability of the key, we generate a key proof  $\mathsf{pf}_{Key}$  using a lattice-based identification protocol [15]:

 $\mathsf{pf}_{Key} \leftarrow \mathrm{id.prove}(\mathbf{A},\mathsf{pk},\mathsf{sk})$ 

This zero-knowledge proof attests to the correctness of pk without revealing sk. <u>Key Verification</u>: Anyone can run Key Verification to check the validity of the public key and its proof, id.verify( $\mathbf{A}, pk, pf_{Key}$ ) [15]. This ensures trust in the public key without exposing secret material.

Encryption: In Encryption, given public parameters and public key pk, the sender encrypts message m using fresh randomness:

- Sample:  $\mathbf{B} \in \mathbb{Z}_p^{d \times l}$ ,  $\mathbf{e} \in \mathbb{Z}_p^{d \times m}$  with  $\|\mathbf{e}\| < \alpha$ , and  $\mathbf{e}' \in \mathbb{Z}_p^d$  with  $\|\mathbf{e}'\| < \beta < \alpha$ . - Compute  $\mathbf{C} = \mathbf{B} \cdot \mathbf{A} + \mathbf{e}$  and  $\mathbf{C}' = \mathbf{B} \cdot \mathbf{pk} + \mathbf{e} + m \cdot g$ , where g is the public gadget vector.

The ciphertext consists of  $(\mathbf{C}, \mathbf{C}')$ . This structure preserves linearity and enables efficient decryption and verification.

Decryption: As described in Decryption, the decryptor computes:  $\mathbf{B}' = \mathbf{C} \cdot \mathbf{s}k$ . The message is then recovered by solving  $m = \text{GadgetSolve}(\mathbf{C}' - \mathbf{B}')$ . The correctness follows from the ciphertext structure and proper randomness bounds. Prove Decryption: To publicly verify the decryption, the decryptor runs Prove Decryption:

- Compute  $\mathbf{B}' = \mathbf{C} \cdot \mathsf{sk}$  and  $\mathbf{e} = \mathbf{B}' + m \cdot g \mathbf{C}'$ .
- $\text{ Generate the proof of correctness of } \mathbf{B'[15]: \, pf}_{\mathit{Dec}} \leftarrow \mathrm{id.prove}(\mathbf{C},\mathbf{B'},\mathsf{sk}) \quad .$

The proof  $(\mathbf{e}, \mathbf{B}', \mathsf{pf}_{Dec})$  enables anyone to verify the decryption. Verify Decryption: Finally, Decryption Verification checks the validity of the decryption proof:

- 1. Ciphertext consistency:  $\mathbf{B}' + \mathbf{e} + m \cdot g \stackrel{?}{=} \mathbf{C}'$ .
- 2. Norm bound on noise:  $\|\mathbf{e}\| < \alpha$ .
- 3. Proof of correct  $\mathbf{B}'$ : id.verify( $\mathbf{C}, \mathbf{B}', \mathsf{pf}_{\mathsf{Dec}}$ ) [15].

If all checks succeed, the ciphertext is correctly decrypted.

**Lemma 4.5.** Let Regev encryption scheme be IND-CPA, then the encryption scheme II in Algorithm 5 satisfies IND-CPA security.

Proof. The Regev encryption scheme is provably IND-CPA secure under the LWE assumption. Algorithm 5 presents a variant of Regev's scheme where:

- The key generation outputs  $sk \leftarrow \{0,1\}^m$  and  $pk = \mathbf{A} \cdot sk$  and  $pf = id.prove(\mathbf{A}, pk, sk)$ . The view of the attacker is (pk, pf), where pk is pseudorandom under the SIS assumption and pf is pseudorandom under [15].

- The encryption algorithm outputs  $\mathbf{C} = \mathbf{B} \cdot \mathbf{A} + \mathbf{e}$  and  $\mathbf{C}' = \mathbf{B} \cdot \mathbf{pk} + \mathbf{e} + m \cdot \mathbf{g}$ . The view of the attacker is  $(\mathbf{C}, \mathbf{C}')$ , which is pseudorandom under the LWE assumption.

Consequently, Algorithm 5 satisfies IND-CPA security.

**Lemma 4.6.** The encryption scheme 5 satisfies verifiability of key generation and verifiability of decryption.

Proof. This follows directly from public key identification from [15].

**Theorem 4.7 (Security of PVSS Instantiation with Linear Encryption Scheme II).** Assuming the hardness of the LWE and SIS problems, and the security properties of the LVC-PoS and Linear Encryption Scheme II, our instantiated PVSS construction achieves correctness, verifiability, and t-IND2 privacy.

We defer the proof to Appendix B.4.

# 4.4 Instantiation Linear Encryption Scheme III

Our third instantiation utilizes the compact ring-based encryption scheme by Lyubashevsky *et al.* [16], which is established as IND-CPA secure. For key verification and the generation of decryption proofs, this instantiation, similar to our second one, leverages the identification protocol from Lyubashevsky [15]. The full specification of this construction is detailed in Algorithm 6.

Corollary 4.8 (Security of PVSS Instantiation with Linear Encryption Scheme III). Assuming the hardness of the LWE and SIS problems, and the security of the LVC-PoS and Linear Encryption Scheme III, our PVSS instantiation achieves correctness, verifiability, and t-IND2 privacy. This follows from Theorems in Section 3.3, as Scheme III meets the compiler's requirements.

Algorithm 6 Linear Encryption Scheme III

procedure SETUP $(1^{\lambda}, p)$  $k = \lceil \log(p) \rceil$ Select  $\lambda$ -bit prime q s.t.  $q = 1 \mod k$ Let  $R = \mathbb{Z}[x]/(x^k + 1)$  $a \leftarrow R_q$ return (a, k, p, q, R)**procedure** Key GENERATION(a, k, p, q, R)Let  $\mathsf{sk}, e \leftarrow R$  be small and  $\mathsf{pk} = a \cdot \mathsf{sk} + e$  $\mathsf{pf}_{Key} = \mathrm{id.prove}((a, 1), \mathsf{pk}, (\mathsf{sk}, e))$  [15] return Secret Key: sk, Public Key: pk, Key Proof:  $pf_{Key}$ **procedure** KEY VERIFICATION(*a*, pk, pf) **return** id.verify((a, 1),  $pk, pf_{Kev}$ ) [15] **procedure** ENCRYPTION $(a, \mathsf{pk}, m, (r, e, e'))$ Let  $\hat{m}$  be a polynomial such that its *i*-th coefficient is *m*'s *i*-th bit  $c = r \cdot a + e$  $c' = r \cdot \mathsf{pk} + e' + \hat{m} \left| \frac{q}{2} \right|$ return (c, c')**procedure** Decryption(sk, (c, c'))  $\begin{array}{l} b=c\cdot\mathsf{sk}\\ m'=[\frac{c'-b}{q/2}] \mod 2 \end{array}$ return m = m'(2)**procedure** PROVE DECRYPTION( $\mathsf{sk}, \mathsf{pk}, m, (c, c')$ )  $b = c \cdot \mathsf{sk}$ 
$$\begin{split} e &= b + \hat{m} \left\lfloor \frac{q}{2} \right\rfloor - c' \\ \mathsf{pf}_{Dec} &= \mathrm{id.prove}(\mathbf{c}, \mathbf{b}, \mathsf{sk}) \ [15] \end{split}$$
**return**  $(e, b, pf_{Dec})$ **procedure** DECRYPTION VERIFICATION ( $pk, m, c, c', e, b, pf_{Dec}$ )  $a = b + e + \hat{m} \left| \frac{q}{2} \right|$ **return**  $a \stackrel{?}{=} c'$  and  $||e|| \stackrel{?}{<} \alpha$  and id.verify(c, b, pf<sub>Dec</sub>) [15]

# References

- Ajtai, M.: Determinism versus non-determinism for linear time RAMs (extended abstract). pp. 632–641 (1999). https://doi.org/10.1145/301250.301424
- Albrecht, M.R., Cini, V., Lai, R.W.F., Malavolta, G., Thyagarajan, S.A.K.: Lattice-based SNARKs: Publicly verifiable, preprocessing, and recursively composable - (extended abstract). pp. 102–132 (2022). https://doi.org/10.1007/978-3-031-15979-4 4
- Bacho, R., Loss, J.: Adaptively secure (aggregatable) PVSS and application to distributed randomness beacons. pp. 1791–1804 (2023). https://doi.org/10.1145/3576915.3623106
- 4. Baghery, K.:  $\pi :$  A unified framework for verifiable secret sharing. Cryptology ePrint Archive (2023)
- 5. Boudot, F., Traoré, J.: Efficient publicly verifiable secret sharing schemes with fast or delayed recovery. In: Information and Communication Security: Second Interna-

tional Conference, ICICS'99, Sydney, Australia, November 9-11, 1999. Proceedings 2. pp. 87–102. Springer (1999)

- Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. pp. 575–584 (2013). https://doi.org/10.1145/2488608.2488680
- Cascudo, I., David, B.: ALBATROSS: Publicly AttestabLe BATched Randomness based On Secret Sharing. pp. 311–341 (2020). https://doi.org/10.1007/978-3-030-64840-4 11
- Cascudo, I., David, B.: Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. pp. 216–248 (2024). https://doi.org/10.1007/978-3-031-58740-5 8
- Cascudo, I., David, B., Garms, L., Konring, A.: YOLO YOSO: Fast and simple encryption and secret sharing in the YOSO model. pp. 651–680 (2022). https://doi.org/10.1007/978-3-031-22963-3 22
- Fujisaki, E., Okamoto, T.: A practical and provably secure scheme for publicly verifiable secret sharing and its applications. pp. 32–46 (1998). https://doi.org/10.1007/BFb0054115
- Gentry, C., Halevi, S., Lyubashevsky, V.: Practical non-interactive publicly verifiable secret sharing with thousands of parties. pp. 458–487 (2022). https://doi.org/10.1007/978-3-031-06944-4 16
- Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing. p. 197–206. STOC '08, Association for Computing Machinery, New York, NY, USA (2008). https://doi.org/10.1145/1374376.1374407, https://doi.org/10.1145/1374376.1374407
- Heidarvand, S., Villar, J.L.: Public verifiability from pairings in secret sharing schemes. In: International Workshop on Selected Areas in Cryptography. pp. 294– 308. Springer (2008)
- Libert, B.: Vector commitments with proofs of smallness: Short range proofs and more. In: Tang, Q., Teague, V. (eds.) Public-Key Cryptography – PKC 2024. pp. 36–67. Springer Nature Switzerland, Cham (2024)
- Lyubashevsky, V.: Lattice-based identification schemes secure under active attacks. pp. 162–179 (2008). https://doi.org/10.1007/978-3-540-78440-1\_10
- Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-LWE cryptography. pp. 35–54 (2013). https://doi.org/10.1007/978-3-642-38348-9\_3
- Micciancio, D., Regev, O.: Lattice-based cryptography. In: Post-quantum cryptography, pp. 147–191. Springer (2009)
- Minh, P.N., Nguyen, K., Susilo, W., Nguyen-An, K.: Publicly verifiable secret sharing: Generic constructions and lattice-based instantiations in the standard model. arXiv preprint arXiv:2504.14381 (2025)
- 19. Peikert, C., et al.: A decade of lattice cryptography. Foundations and trends (**R**) in theoretical computer science **10**(4), 283–424 (2016)
- Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. pp. 84–93 (2005). https://doi.org/10.1145/1060590.1060603
- Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM 56(6) (Sep 2009). https://doi.org/10.1145/1568318.1568324, https://doi.org/10.1145/1568318.1568324
- Ruiz, A., Villar, J.L.: Publicly verifiable secret sharing from paillier's cryptosystem. In: WEWoRC 2005–Western European Workshop on Research in Cryptology. pp. 98–108. Gesellschaft für Informatik eV (2005)

- Schoenmakers, B.: A simple publicly verifiable secret sharing scheme and its application to electronic. pp. 148–164 (1999). https://doi.org/10.1007/3-540-48405-1 10
- Stadler, M.: Publicly verifiable secret sharing. pp. 190–199 (1996). https://doi.org/10.1007/3-540-68339-9\_17

# A Omitted Preliminaries

### A.1 Basic Notions on Public Key Encryption

In this section we introduce well-known concepts on public key encryption.

**Definition A.1.** A public key encryption scheme consists of five polynomial time algorithms (Setup, KeyGen, VerifyKey, Encrypt, Decrypt) as follows:

- $pp \leftarrow Setup(1^{\lambda}, p)$ : The setup algorithm generates the public parameters on input the security parameter  $\lambda \in \mathbb{N}$  and modulus p.
- (sk, pk, pf<sub>Key</sub>) ← KeyGen(pp): The key generation algorithm generates a pair (sk, pk) consisting of a secret key and a public key along with proof pf<sub>Key</sub> for identification of pk on input the public parameters pp.
- $-b \leftarrow \text{VerifyKey}(pp, pk, pf_{Key})$ : The key verification algorithm outputs a bit b deciding whether to accept or reject that pk is a valid identification.
- $-C \leftarrow \mathsf{Encrypt}(\mathsf{pp},\mathsf{pk},m,r)$ : The encryption algorithm generate ciphertext Con input the public parameters  $\mathsf{pp}$ , the public key  $\mathsf{pk}$ , the plaintext  $m \in \mathcal{M}$ and randomness  $r \in \mathcal{R}$ .
- $-m' \leftarrow \mathsf{Decrypt}(\mathsf{pp},\mathsf{sk},C)$ : The decrypt algorithm outputs a decrypted message m' on input the public parameters  $\mathsf{pp}$ , the secret key  $\mathsf{sk}$  and encrypted message C.

and which satisfy that for every (pk, sk) output by KeyGen, and for every  $m \in \mathcal{M}$ and  $r \in \mathcal{R}$ ,

 $\Pr[\mathsf{Decrypt}(\mathsf{pp},\mathsf{sk},\mathsf{Encrypt}(\mathsf{pp},\mathsf{pk},m,r))] = 1$ 

The most widely recognized security notion for public key encryption is IND-CPA security, which demands that encryptions of any two messages under a public key pk remain computationally indistinguishable without the knowledge of the corresponding sk. Here we consider the notion of  $\ell$ -multi-key IND-CPA security. This requires that the encryptions of two vectors of messages of the same length, where each coordinate is encrypted under a public key  $pk_i$ , are indistinguishable. The notions are equivalent as long as  $\ell$  is polynomial in the security parameter.

**Definition A.2.** A public key encryption scheme  $\mathcal{E}$  satisfies  $\ell$ -multi-key IND-CPA security if for any poly $(1^{\lambda})$ -time adversary  $\mathcal{A}$ , we have

$$\Pr\left[\operatorname{Game}_{\mathcal{A},\mathcal{E}}^{\ell\text{-}IND\text{-}CPA,0}(\lambda)=1\right] - \Pr\left[\operatorname{Game}_{\mathcal{A},\mathcal{E}}^{\ell\text{-}IND\text{-}CPA,1}(\lambda)=1\right] \le \operatorname{negl}(\lambda)$$

where for b = 0, 1,  $\operatorname{Game}_{\mathcal{A}, \mathcal{E}}^{\ell-IND-CPA, b}(\lambda)$  is the following game against a challenger:

- The challenger run  $\mathcal{E}$ .Setup $(1^{\lambda}, p)$  and then  $\forall i \in [\ell]$  runs  $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathcal{E}$ .KeyGen $(\mathsf{pp}, i)$  and sends  $(\mathsf{pk}_i)_{i \in [\ell]}$  to  $\mathcal{A}$ .
- The attacker return two vectors of messages of the same length  $(m_1^0, m_2^0, \cdots, m_{\ell}^0), (m_1^1, m_2^1, \cdots, m_{\ell}^1) \leftarrow \mathcal{A}(\mathsf{pp}, (\mathsf{pk}_i)_{i \in [\ell]}).$
- The challenger  $\forall i \in [\ell]$  choose random  $r_i \in \mathcal{R}$  and run  $C_i = \mathcal{E}.\mathsf{Encrypt}(\mathsf{pp},\mathsf{pk}_i,m_i^b,r_i)$  and sends  $(C_i)_{i\in[\ell]}$  to  $\mathcal{A}$ .
- The attacker  $\mathcal{A}((C_i)_{i \in [\ell]})$  outputs a guess  $b' \in \{0, 1\}$ .

The case  $\ell = 1$  is the usual IND-CPA definition and for  $\ell = \text{poly}(\langle)$  a standard hybrid argument shows that a scheme is  $\ell$ -multi-key IND-CPA if and only if it is IND-CPA.

**Definition A.3.** A public key encryption scheme  $\mathcal{E}$  satisfies verifiability of key generation for  $R_{Key}$  if for all PPT  $\mathcal{A}$ ,

$$\begin{split} &\Pr\left[\mathcal{E}.\mathsf{VerifyKey}(\mathsf{pp},\mathsf{pk},\mathsf{pf}_{Key}) = 1 \\ &\wedge \nexists\mathsf{sk} \in SK \ s.t. \ (\mathsf{pk},\mathsf{sk}) \in R_{Key} \\ &\left| \ \mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda},p), \\ &\left(\mathsf{pk},\mathsf{pf}_{Key}) \leftarrow \mathcal{A}(pp) \right] \ is \ negligible \ in \ \lambda. \end{split}$$

#### A.2 Security Definitions of PVSS

A PVSS scheme should satisfy correctness, verifiability and IND2-secrecy.

*Correctness* The correctness with r-reconstruction requirement ensures that if everybody is honest, then all proofs involved pass and any set of at least r participants can reconstruct the secret from their shares (by first having each party decrypt their share and then jointly applying the reconstruction algorithm Rec).

**Definition A.4.** For a set  $T \subseteq [n]$ , and a probability distribution  $\mathcal{D}_s$  over the secret space, define the following experiment  $\operatorname{ExpCorr}_{T,\mathcal{D}_s}(1^{\lambda})$ .

 $\begin{array}{l} - \ \mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda}, 1^n, 1^t) \\ - \ \forall i \in [n], \ (\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{pf}_{Key,i}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i) \\ - \ s \leftarrow \mathcal{D}_s \\ - \ ((C_i)_{i \in [n]}, \mathsf{pf}_D) \leftarrow \mathsf{Dist}(\mathsf{pp}, \{\mathsf{pk}_i : i \in [n]\}, s) \\ - \ \forall i \in T, \ (s_i, \mathsf{pf}_{Dec,i}) \leftarrow \mathsf{Decrypt}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{sk}_i, C_i) \\ - \ s' \leftarrow \mathsf{Reconstruct}(\mathsf{pp}, \{s_i : i \in T\}), \ where \ s' \in S \cup \{\bot\} \\ - \ Output \ (\mathsf{pp}, (\mathsf{pk}_i, \mathsf{pf}_{Key,i}, C_i)_{i \in [n]}, \mathsf{pf}_D, (\mathsf{pf}_{Dec,i})_{i \in T}, s, s') \end{array}$ 

We say that the PVSS is correct with r-reconstruction if for all  $T \subseteq [n]$  of size at least r, any probability distribution  $\mathcal{D}_s$  over the secret space,

$$\Pr\left[\mathsf{VerifyKey}(\mathsf{pp}, i, \mathsf{pk}_i, \mathsf{pf}_{Key, i}) = 1 \,\forall i \in [n]\right]$$

г

$$\wedge \operatorname{VerifyDist}(\operatorname{pp}, (C_i)_{i \in [n]}, \operatorname{pf}_D, (\operatorname{pk}_i)_{i \in [n]}) = 1 \wedge \operatorname{VerifyDecrypt}(\operatorname{pp}, i, \operatorname{pk}_i, s_i, C_i, \operatorname{pf}_{Dec,i}) = 1 \forall i \in T \wedge s' = s \mid (\operatorname{pp}, (\operatorname{pk}_i, \operatorname{pf}_{Key,i}, C_i)_{i \in [n]}, \operatorname{pf}_D, (\operatorname{pf}_{Dec,i})_{i \in T}, s, s') \leftarrow \operatorname{ExpCorr}_{T, \mathcal{D}_s}(1^{\lambda}) \rceil = 1$$

*Verifiability* The verifiability properties assert that passing the verification procedures VerifyKey, VerifyDist, and VerifyDecrypt guarantee respectively that the key pairs are well constructed, that the set of encrypted shares is indeed a correct sharing of a secret, and that the shares have been correctly decrypted.

**Definition A.5.** The PVSS satisfies verifiability of key generation if underlying encryption scheme satisfies verifiability of key generation A.3.

**Definition A.6.** The PVSS satisfies verifiability of sharing distribution if for every PPT A,

$$\begin{split} &\Pr\left[\mathsf{VerifyDist}(\mathsf{pp},(C_i)_{i\in[n]},\mathsf{pf}_D,(\mathsf{pk}_i)_{i\in[n]})=1 \\ &\wedge \nexists s\in S \ s.t. \ ((C_i)_{i\in[n]},\cdot) \leftarrow \mathsf{Dist}(\mathsf{pp},\{\mathsf{pk}_i:i\in[n]\},s) \\ &\left| \ \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda,1^n,1^t), \\ & ((C_i)_{i\in[n]},\mathsf{pf}_D) \leftarrow \mathcal{A}(\mathsf{pp}) \right] is \ negligible \ in \ \lambda. \end{split}$$

**Definition A.7.** The PVSS satisfies verifiability of share decryption if the underlying encryption scheme satisfies verifiability of decryption 2.4.

Privacy We now define indistinguishability of secrets against an adversary corrupting t parties. We follow the notions from [2]. In this definition, the adversary is allowed to compute the public keys of the corrupted parties after seeing those of the honest parties. Then, provided two secrets  $(s_0, s_1)$  and a sharing of a random secret  $s_b$ , the adversary has negligible advantage in guessing which secret was shared. In this paper, we choose the IND2-privacy flavor where the adversary can choose  $s_0, s_1$ . This is stronger than IND1-privacy where the challenger chooses the secrets at random.

**Definition A.8.** The PVSS is t-IND2-private if for any  $poly(1^{\lambda})$ -time adversary  $\mathcal{A}$  corrupting t parties (w.l.o.g.  $\mathcal{A}$  corrupts [n - t + 1, n]), we have

$$\Pr[\operatorname{Game}_{\mathcal{A}, \operatorname{PVSS}}^{\operatorname{ind-secrecy}, 0}(\lambda) = 1] - \Pr[\operatorname{Game}_{\mathcal{A}, \operatorname{PVSS}}^{\operatorname{ind-secrecy}, 1}(\lambda) = 1] = \operatorname{negl}(\lambda)$$

where for b = 0, 1, Game<sup>ind-secrecy,b</sup><sub> $\mathcal{A}, PVSS$ </sub>  $(\lambda)$  is the following game against a challenger:

- The challenger runs  $pp \leftarrow Setup(1^{\lambda}, 1^n, 1^t)$  and sends pp to  $\mathcal{A}$ .
- For  $i \in [n t]$ , the challenger runs  $(\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{pf}_{Key,i}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i)$  and sends all created  $(\mathsf{pk}_i, \mathsf{pf}_{Key,i})$  to  $\mathcal{A}$ .

- For the corrupted parties,  $\mathcal{A}$  creates  $(\mathsf{pk}_i, \mathsf{pf}_{Key,i})_{i \in [n-t+1,n]} \leftarrow \mathcal{A}(\mathsf{pp}, (\mathsf{pk}_i, \mathsf{pf}_{Key,i})_{i \in [n-t]})$  and sends them to the challenger, together with two values  $s_0, s_1$  in S.
- The challenger runs  $\operatorname{VerifyKey}(\operatorname{pp}, i, \operatorname{pk}_i, \operatorname{pf}_{Key,i})$  for  $i \in [n-t+1, n]$ . If any of these output 0 (reject), the challenger sends  $\perp$  to  $\mathcal{A}$ .
- Otherwise, if all key verification proofs accept, the challenger runs  $(C_1, \ldots, C_n, \mathsf{pf}_D) \leftarrow \mathsf{Dist}(\mathsf{pp}, \{\mathsf{pk}_i : i \in [n]\}, s_b)$ , and sends  $(C_1, \ldots, C_n, \mathsf{pf}_D)$  to  $\mathcal{A}$ .
- $-\mathcal{A} \text{ outputs a guess } b' \in \{0,1\}.$

### A.3 Vector Commitments

We recall a non-interactive variant of vector commitments [2].

**Definition A.9 (Vector Commitments (VC)).** A vector commitment (VC) scheme is parameterised by the families

$$\mathcal{F} = \{\mathcal{F}_{v,w,o} \subseteq \{f : \mathcal{R}^v \times \mathcal{R}^w \to \mathcal{R}^o\}\}_{v,w,o \in \mathbb{N}}$$

of functions over  $\mathcal{R}$  and an input alphabet  $\mathcal{X} \subseteq \mathcal{R}$ . The parameters v, w, and o are the dimensions of public inputs, secret inputs, and outputs of f respectively. The VC scheme consists of the PPT algorithms (Setup, Com, Open, Verify) defined as follows:

- $pp \leftarrow Setup(1^{\lambda}, 1^{v}, 1^{w}, 1^{o})$ : The setup algorithm generates the public parameters on input the security parameter  $\lambda \in \mathbb{N}$ , the size parameters  $v, w, o \in \mathbb{N}$ .
- $(c, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{pp}, \mathbf{x})$ : The commitment algorithm generates a commitment c of a given vector  $\mathbf{x} \in \mathcal{X}^w$  with some auxiliary opening information  $\mathsf{aux}$ .
- $-\pi \leftarrow \text{Open}(\text{pp}, f, \mathbf{z}, \text{aux})$ : The opening algorithm generates a proof  $\pi$  for  $f(\mathbf{z}, \cdot)$  for the public input  $\mathbf{z}$  and function  $f \in \mathcal{F}_{v,w,o}$ .
- $-b \leftarrow \text{Verify}(pp, f, \mathbf{z}, \mathbf{y}, c, \pi)$ : The verification algorithm inputs public parameters pp, linear function  $f \in \mathcal{F}_{v,w,o}, \mathbf{z} \in \mathcal{X}^v, \mathbf{y} \in \mathcal{X}^o$  and a commitment c, and an opening proof  $\pi$ . It outputs a bit b deciding whether to accept or reject that the vector  $\mathbf{x}$  committed in c satisfies  $f(\mathbf{z}, \mathbf{x}) = \mathbf{y}$ .

**Definition A.10 (Correctness).** A VC scheme for  $(\mathcal{F}, \mathcal{X})$  is said to be correct if for any  $\lambda, v, w, o \in \mathbb{N}$ , any  $pp \in \text{Setup}(1^{\lambda}, 1^{v}, 1^{w}, 1^{o})$ , any  $(f, \mathbf{z}, \mathbf{x}, \mathbf{y}) \in \mathcal{F}_{v,w,o} \times \mathcal{X}^{v}, \mathcal{X}^{w} \times \mathcal{X}^{o}$  satisfying  $f(\mathbf{z}, \mathbf{x}) = \mathbf{y}$ , any  $(c, \mathsf{aux}) \leftarrow \text{Com}(\mathsf{pp}, \mathbf{x})$ , any  $\pi \in \text{Open}(\mathsf{pp}, f, \mathbf{z}, \mathbf{y}, \mathsf{aux})$ , it holds that

$$\mathsf{Verify}(\mathsf{pp}, \mathbf{z}, \mathbf{y}, c, \pi) = 1.$$

**Definition A.11 (Biniding).** Let  $\rho : \mathbb{N}^3 \to [0,1]$ . A VC scheme for  $\mathcal{F}, \mathcal{X}, \mathcal{Y}$ ) is said to be weakly  $\rho$ -binding if for any pair of PPT adversary  $\mathcal{A}$  and any  $s, w \in poly(\lambda)$  it holds that the following expression is upper-bounded by  $\rho(\lambda, s, w)$ :

$$\Pr \begin{bmatrix} \forall i \in \{0, 1\}, \\ \mathsf{Verify}(\mathsf{pp}, f_i, \mathbf{z}_i, \mathbf{y}_i, c, \pi_i) = 1, \\ \wedge f_0(\mathbf{z}_0, \cdot) = f_1(\mathbf{z}_1, \cdot) \wedge \mathbf{y}_0 \neq \mathbf{y}_1 \end{bmatrix} \mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda}, 1^v, 1^w, 1^o) \\ (c, (f_i, \mathbf{z}_i, \mathbf{y}_i, \pi_i)_{i=0}^1) \leftarrow \mathcal{A}(\mathsf{pp}) \end{bmatrix}$$

We say that the scheme is weakly binding if it is weakly  $\rho$ -binding and  $\rho(\lambda, s, w)$  is negligible in  $\lambda$  for any  $s, w \in poly(\lambda)$ .

The scheme is said to be  $\rho$ -binding if for any PPT adversary  $\mathcal{A}$  and  $w, t = poly(\lambda)$  it holds that the following expression is upper-bounded by  $\rho(\lambda)$ :

$$\Pr \begin{bmatrix} \forall i \in I, \\ \mathsf{Verify}(\mathsf{pp}, f_i, \mathbf{z}_i, \mathbf{y}_i, c, \pi_i) = 1, \\ \land \neg (\exists x \in \mathcal{K}^w, \\ \forall i \in I, f_i(\mathbf{z}_i, x) = \mathbf{y}_i) \end{bmatrix} \xrightarrow{\mathsf{pp}} \leftarrow \mathsf{Setup}(1^\lambda, 1^v, 1^w, 1^o) \\ \leftarrow \mathcal{A}(\mathsf{pp}) \\ \leftarrow \mathcal{A}(\mathsf{pp}) \end{bmatrix}$$

We say that the scheme is binding if it is  $\rho$ -binding and  $\rho(\lambda, s, w)$  is negligible in  $\lambda$  for any  $s, w \in poly(\lambda)$ .

Note that in the binding definition the existence of  $\mathbf{x}$  is checked over the base field  $\mathcal{K}$  rather than the ring  $\mathcal{R}$ . The reason for this choice will become clear when we discuss the binding property of our construction.

We discuss potential approaches to modify the VC construction to achieve hiding and functional hiding.

**Definition A.12 ((Functional) Hiding)).** : A VC scheme for  $(\mathcal{F}, \mathcal{X}, \mathcal{Y})$  is said to be statistically/computationally hiding if for any  $\lambda, w, o \in \mathbb{N}$ , any  $pp \in$ Setup $(1^{\lambda}, 1^{w}, 1^{o})$ , and any  $\mathbf{x}, \mathbf{x}' \in \mathcal{X}^{w}$ , the distributions

 $\{c: (c, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{pp}, \mathbf{x})\}\ and\ \{c: (c, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{pp}, \mathbf{x}')\}\$ 

 $are\ statistically/computationally\ indistinguishable.$ 

A VC scheme for  $(\mathcal{F}, \mathcal{X}, \mathcal{Y})$  is said to be statistically/computationally functional hiding if there exists a tuple of PPT simulators  $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$  such that, for any  $\lambda, w, o \in \mathbb{N}$  and any  $(f, \mathbf{x}, \mathbf{y}) \in \mathcal{F}_{w,o} \times \mathcal{X}^w \times \mathcal{Y}^o$  satisfying  $f(\mathbf{x}) = \mathbf{y}$ , the distributions

$$\left\{\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda}, 1^w, 1^o), \\ (c, \pi) : (c, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{pp}, \mathbf{x}) \\ \pi \leftarrow \mathsf{Open}(\mathsf{pp}, f, \mathsf{aux}) \end{array}\right\}$$

and

$$\left\{ \begin{array}{c} (\mathsf{pp},\mathsf{td}) \leftarrow \mathcal{S}_0(1^\lambda,1^w,1^o) \\ (c,\pi) : \\ (c,\pi) \leftarrow \mathcal{S}_1(\mathsf{pp},\mathsf{td},f,\mathbf{y}) \end{array} \right\}$$

are statistically/computationally indistinguishable.

# A.4 Base VC of Albrecht et al. [2]

The goal of the VC scheme is to allow a dealer to commit to a vector  $\mathbf{x}$  and later provide succinct, non-interactive proofs that linear relations over  $\mathbf{x}$  hold, without revealing  $\mathbf{x}$  itself. We recall the VC scheme [2] in Algorithm 7. Setup: In Setup, the public parameters are generated as follows:

- Sample  $\mathbf{v} \in (R_q^{\times})^w$  to act as the commitment key.
- Sample a vector  $h \in R_p^o$  for random linear combinations in proofs.
- Generate a trapdoor matrix A and auxiliary data td.
- Precompute helper vectors  $u_g = \mathsf{SampPre}(td, g(\mathbf{v})t, \beta)$  for all  $g = \frac{v_i}{v_j}$  in the set G.

The vectors  $u_g$  enable efficient proofs of linear relations by acting as compressed witnesses for relations between components of **x**.

<u>Commit</u>: In Com, to commit to vector  $\mathbf{x} \in R_q^w$ , compute:

$$c = \langle \mathbf{v}, \mathbf{x} \rangle \mod q$$

Additionally, compute auxiliary vectors  $u_i = \sum_{j \neq i} x_j \cdot u_{\frac{v_j}{v_i}}$ . These are stored in aux for future proof generation.

Open: Given a linear function f and the vector  $\mathbf{x}$ , the dealer computes the opening proof:

$$\pi = \sum_{i=1}^{o} \sum_{j=1}^{w} h_i f_{ij}(\mathbf{x}) u_j$$

This proves that  $f(\mathbf{x})$  evaluates to the claimed value without revealing  $\mathbf{x}$ . Verify: The verifier checks two properties:

1. Correctness of the proof  $\pi$ :

$$A \cdot \mathbf{u} \stackrel{?}{=} \left( \sum_{k=1}^{o} h_k \left( \sum_{j=1}^{w} f_{kj} \frac{c}{v_j} - y_k \right) \right) \cdot t \mod q$$

2. Smallness of auxiliary vector  $\mathbf{u}$ :  $\|\mathbf{u}\| \leq \delta_0$ .

Algorithm 7 Vector Commitment [2]

```
procedure SETUP(1^{\lambda}, 1^{v}, 1^{w}, 1^{o})
       \mathbf{v} \leftarrow (R_q^{\times})^w
       h \leftarrow R_p^o
       (A, td) = \mathsf{TrapGen}(1^{\eta}, 1^{l}, q, R, \beta)
       t \leftarrow T
       u_{i,j} = \mathsf{SampPre}(td, \frac{v_i}{v_j}t, \beta), \forall i, j \in \mathbb{Z}_w | i \neq j
       return (A, t, \{u_{i,j}\}, \mathbf{v}, h)
procedure COM(pp, x)
       c = \langle \mathbf{v}, \mathbf{x} \rangle \mod q
       for i = 1 to w do
              u_i = \sum_{j \neq i} x_j \cdot u_{j,i}
       \operatorname{aux} = (u_i)_{i \in \mathbb{Z}_w}
       return (c, aux)
procedure OPEN(pp, f, z, aux)
       \pi = \sum_{i \in \mathbb{Z}_o} \sum_{j \in \mathbb{Z}_w} h_i f_{i,j}(\mathbf{z}) u_j
       return \pi
procedure VERIFY(pp, f, \mathbf{z}, \mathbf{y}, c, \pi)
      a_1 = \left(\mathbf{A} \cdot \mathbf{u} \stackrel{?}{=} \left(\sum_{k \in \mathbb{Z}_o} h_k \left(\sum_{j \in \mathbb{Z}_w} f_{k,j}(\mathbf{z}) \frac{c}{v_j} - y_k\right)\right) \cdot t \mod q\right)
      a_2 = \left( \|\mathbf{u}\| \stackrel{?}{\leq} \delta_0 \right)
       return a_1 \wedge a_2
```

# **B** Omitted Proofs

# B.1 Security Proof of LVC-PoS

**Theorem B.1 (Security Properties of LVC-PoS).** Assuming the correctness, functional hiding, and binding properties of the underlying vector commitment scheme (Algorithm 7), the extended construction Algorithm 3 achieves the following properties for LVC-PoS:

- Correctness: Algorithm 3 correctly verifies both the evaluation of linear functions and the smallness of specified indices in the committed vector.
- Functional Hiding: Algorithm 3 preserves functional hiding for linear function openings, meaning that the commitment and proof reveal no information about the committed vector beyond the output of the opened function and smallness verification.
- Weak Binding: Algorithm 3 satisfies weak binding, ensuring that after committing to a vector, it is computationally infeasible to produce two distinct valid openings.

We prove each property in turn:

*Proof. Correctness.* The proof consists of three components:  $(\pi, \pi_{eq}, \pi_{ip})$ .

- $\pi$ : This is the opening proof from Algorithm 7 for standard linear function evaluation. Since the correctness of Algorithm 7 is established in [2], correctness of this component follows directly.
- $\pi_{eq}$ : This component verifies the auxiliary commitment consistency. Specifically, we use a commitment with parameter vector  $\mathbf{v} = \frac{1}{\mathbf{v}}$ . Since we possess the trapdoor for all ratios  $\frac{\frac{1}{v_i}}{\frac{1}{v_j}}$ , and this commitment operates over the same algebraic structure as the underlying VC, the correctness of  $\pi_{eq}$  follows from the correctness of Algorithm 7.
- $\pi_{ip}$ : This component proves the smallness (binary property) of the committed vector. The validity is demonstrated by the following relation:

$$c_{\mathbf{x}} \cdot \overline{c}_{\mathbf{h} \circ (\mathbf{x} - \mathbf{1})} = \left(\sum_{i \in w} x_i v_i\right) \cdot \left(\sum_{i \in w} h_i (x_i - 1) \frac{1}{v_i}\right)$$
$$= \sum_{i, j \in \mathbb{Z}_w : i \neq j} x_i (h_j (x_j - 1)) \frac{v_i}{v_j} + \langle \mathbf{h} \circ (\mathbf{x} - \mathbf{1}), \mathbf{x} \rangle$$
$$= \langle \mathbf{a}, \pi_{ip} \rangle + 0$$

Therefore, the correctness of  $\pi_{ip}$  is established.

*Functional Hiding.* The functional hiding of Algorithm 3 follows directly from the functional hiding property of Algorithm 7 as shown in [2]. In particular, since our commitments and proofs are constructed entirely from homomorphic evaluations over the original VC structure, and since the auxiliary commitments and openings correspond to linear functions of the original committed vector, functional hiding is preserved.

Moreover, by applying a Gaussian variant of the Leftover Hash Lemma (cf. [2]), we argue that the distributions of the commitments and associated proofs remain statistically close to uniform modulo the leakage of the opened function value and smallness verification. This ensures that no additional information about the committed vector  $\mathbf{x}$  is revealed.

Weak Binding. The binding property of Algorithm 3 similarly follows from the binding of the underlying VC (Algorithm 7) as established in [2]. Specifically, the commitments in Algorithm 3 are homomorphic transformations of the base commitments in Algorithm 7.

The binding property in [2] guarantees that once a commitment is fixed, no adversary can open it to two different vectors that satisfy different outputs of the function or smallness proofs. Since  $\pi_{eq}$  and  $\pi_{ip}$  are linear functions over the same commitment, and since the openings rely on the trapdoor structure inherited from the original VC, the weak binding of Algorithm 3 follows.

# B.2 Security Proofs of Our Generic Compiler for PVSS

In this section, we prove the security properties of our generic PVSS framework. Specifically, we prove that our construction satisfies correctness, verifiability of key generation, verifiability of distribution, verifiability of share decryption, and privacy (IND2-privacy) under standard assumptions on the underlying encryption scheme and vector commitment scheme. Each proof relies on the modular structure of our compiler, allowing security to be inherited directly from the underlying primitives.

**Theorem B.2 (Correctness).** The PVSS framework described in Algorithm 1 satisfies correctness with t-reconstruction, provided that the vector commitment scheme is complete and the encryption scheme is correct.

*Proof.* Assuming all participants honestly generate their keys, each public key  $\mathsf{pk}_i$  is associated with a valid secret key  $\mathsf{sk}_i$ , and the corresponding proof  $\mathsf{pf}_{Key,i}$  validates successfully.

If the dealer is honest, the shares are computed as  $s_i = \langle \mathbf{b}_i, \mathbf{a} \rangle = a'(i) \mod p$ , where a'(x) is the dealer's polynomial of degree t with constant term a'(0) = s, the secret. Each ciphertext is computed as  $C_i = \mathcal{E}.\mathsf{Encrypt}(\mathcal{E}.\mathsf{pp},\mathsf{pk}_i, s_i, r_i)$  for freshly sampled randomness  $r_i$ . The dealer commits to the vector  $\mathbf{x}$  containing  $(\mathbf{a}, s_1, \ldots, s_n, r_1, \ldots, r_n)$ , and the opening proof  $\mathsf{pf}_D$  with commitment c guarantees that  $M(\mathbf{x}) = \mathbf{y}$  holds as defined in the algorithm.

Given valid decryption keys  $\mathsf{sk}_i$ , participants can correctly decrypt their ciphertexts to obtain  $s_i = \mathcal{E}.\mathsf{Decrypt}(\mathcal{E}.\mathsf{pp},\mathsf{sk}_i,C_i)$ . Finally, any subset of at least t correctly decrypted shares suffices to reconstruct the secret s = a'(0) via Lagrange interpolation. Therefore, correctness holds.

# B.3 Security Proof of PVSS Instantiation with Linear Encryption Scheme I

**Theorem B.3 (Security of PVSS Instantiation with Linear Encryption Scheme I).** Assuming the hardness of the LWE and SIS problems, and the security properties of the extended vector commitment and Linear Encryption Scheme I, our instantiated PVSS construction achieves correctness, verifiability, and t-IND2 privacy.

*Proof (Proof Sketch).* Follows from the security of the generic framework established in Section 3, together with the properties of Linear Encryption Scheme I from Section 4.2.

# B.4 Secuity Proof of PVSS Instantiation with Linear Encryption Scheme II

**Theorem B.4 (Security of PVSS Instantiation with Linear Encryption Scheme II).** Assuming the hardness of the LWE and SIS problems, and the security properties of the extended vector commitment and Linear Encryption Scheme II, our instantiated PVSS construction achieves correctness, verifiability, and t-IND2 privacy.

*Proof (Proof Sketch).* Follows from the security of the generic framework established in Section 3, together with the properties of Linear Encryption Scheme II from Section 4.3.