Resilience-Optimal Lightweight High-threshold Asynchronous Verifiable Secret Sharing

Hao Cheng^{1,5}, Jiliang Li², Yizhong Liu³, Yuan Lu¹, Weizhi Meng⁴, and Zhenfeng Zhang¹

¹ Institute of Software, Chinese Academy of Sciences, China {chenghao2020,luyuan,zhenfeng}@iscas.ac.cn

² School of Cyber Science and Engineering, Xi'an Jiaotong University, China jiliang.li@xjtu.edu.cn

³ School of Cyber Science and Technology, Beihang University, China liuyizhong@buaa.edu.cn

 $^4\,$ School of Computing and Communications, Lancaster University, United Kingdom

w.meng3@lancaster.ac.uk

⁵ University of Chinese Academy of Sciences, China

Abstract. Shoup and Smart (SS24) recently introduced a lightweight asynchronous verifiable secret sharing (AVSS) protocol with optimal resilience directly from cryptographic hash functions (JoC 2024), offering plausible quantum resilience and computational efficiency. However, SS24 AVSS only achieves standard secrecy to keep the secret confidential against n/3 corrupted parties if no honest party publishes its share. In contrast, from "heavyweight" public-key cryptography, one can realize so-called high-threshold asynchronous verifiable secret sharing (HAVSS), with a stronger high-threshold secrecy to tolerate n/3 corrupted parties and additional leaked shares from n/3 honest parties. This raises the following question: can we bridge the remaining gap to design an efficient HAVSS using only lightweight cryptography?

We answer the question in the affirmative by presenting a lightweight HAVSS with optimal resilience. When executing across n parties to share a secret, it attains a worst-case communication complexity of $\tilde{\mathcal{O}}(\lambda n^3)$ (where λ is the cryptographic security parameter) and realizes high-threshold secrecy to tolerate a fully asynchronous adversary that can control $t = \lfloor \frac{n-1}{3} \rfloor$ malicious parties and also learn t additional secret shares from some honest parties. The (worst-case) communication complexity of our lightweight HAVSS protocol matches that of SS24 AVSS—the state-of-the-art lightweight AVSS without high-threshold secrecy. Notably, our design is a direct and concretely efficient reduction to hash functions in the random oracle model, without extra setup assumptions like CRS/PKI or heavy intermediate steps like hash-based zk-STARK.

Keywords: Asynchronous Verifiable Secret Sharing · Lightweight Cryptography · High-threshold Secrecy · Asynchronous Multi-party Protocol

1 Introduction

Asynchronous Verifiable Secret Sharing (AVSS) is a two-phase multi-party protocol: During its sharing phase, a designated dealer can confidentially distribute a secret across an asynchronous network of n participants; Following that, another reconstruction phase can be initiated, allowing the honest parties to collectively recover the secret, unanimously. ⁶ AVSS is a fundamental primitive with numerous applications in asynchronous distributed protocols, including consensus [14, 19, 1, 28], distributed random beacon [8, 24], distributed key generation [3, 35, 2, 23, 20, 31], and multi-party computation [36, 41, 40, 18, 29, 30].

AVSS is known to tolerate at most t < n/3 malicious parties, analogy to that either liveness or safety could be violated in asynchronous broadcast protocols when $t \ge n/3$. ⁷ Nevertheless, like many high-threshold cryptosystems [13, 39], AVSS has also been studied for enhanced high-threshold secrecy, aiming to preserve the confidentiality of secret, even if $\ell - t$ honest parties have already published their secret shares [22, 2, 6, 35, 11]. Here ℓ represents the reconstruction threshold, typically corresponding to the degree of secret sharing, and ℓ can be at most 2t in AVSS with an optimal resilience of t < n/3, meaning an optimally resilient AVSS can tolerate a few additional leaked shares from ($\ell - t$) honest parties alongside t malicious parties. Such high-threshold AVSS (HAVSS) provides stronger secrecy than normal AVSS that merely provides a lower reconstruction threshold $\ell = t$, making it essential in certain applications. One example is to realize high-threshold asynchronous coin-tossing [13], making the adversary learn the flipped coins as late as possible and thus rendering simpler design of asynchronous Byzantine agreement protocols [12, 25, 4, 32, 37].

Protocols	Resilience $(t <)$	Comm.	Setup	Assumption
CKLS02 [11]	n/4	$\mathcal{O}(\lambda n^3)$	secure channel	DL
KMS20 [35] HAVEN [6]	${n/3} {n/3}$	$\mathcal{O}(\lambda n^4) \ ilde{\mathcal{O}}(\lambda n^2)$	PKI+secure channel secure channel	$_{ m DL}$
Bingo [2]	n/3	$\mathcal{O}(\lambda n^2)$	CRS+secure channel	$q ext{-SDH} + ext{AGM}$
DXR21 [21] DXT+23 [22]	$\frac{n}{3}$	${ {\cal O}(\lambda n^2) \over {\cal O}(\lambda n^2) }$	PKI+auth channel	DDH+ROM DDH+ROM
	n/3	()	PKI+auth channel	
This work	n/3	$ ilde{\mathcal{O}}(\lambda n^3)$	secure channel	ROM

Table 1. Comparison of high-threshold ACSS (HAVSS) protocols †

[†] We consider the case $n \leq \mathcal{O}(\lambda)$, as following the standard cryptographic practice [12, 19]. Here λ is the cryptographic security parameter and usually is several hundred.

⁶ Note that the reconstruction phase of AVSS can be defined as: for ℓ -degree secret sharing, any (ℓ +1)-sized subset of honest parties can recover the unique secret. Such AVSS has an alias Asynchronous Complete Secret Sharing (ACSS). Looking ahead, resilience-optimal high-threshold AVSS inherently is ACSS, so we ignore the difference in the paper unless otherwise specified.

⁷ Note that AVSS implies asynchronous reliable broadcast (RBC) when executing its two phases sequentially, so AVSS has a resilience of t < n/3, same to that of RBC.

Limitations of existing HAVSS protocols. As illustrated in Table 1, all existing HAVSS protocols are constructed from public-key cryptography (PKC). Cachin et al. [12] gave the first HAVSS construction adapted from the classic Pedersen VSS [38], but the HAVSS design only tolerates n/4 malicious parties. Kokoris-Kogias et al. [35] designed the first optimal-resilience HAVSS protocol, but with $\mathcal{O}(\lambda n^4)$ communication complexity. The study also introduces the idea of using an asymmetric bivariate polynomial (where one dimension has degree tand the other dimension has degree 2t) for implementing HAVSS, in combination with *polynomial commitment scheme* (PCS). Following the idea of using bivariate polynomial, Alhaddad et al. [6] and Abraham et al. [2] proposed a couple of communication-efficient HAVSS protocols HAVEN and Bingo, respectively. Bingo [2] reduces the communication overhead of HAVSS to $\mathcal{O}(\lambda n^2)$, by lifting the succinct and homomorphic KZG polynomial commitment [34] to support bivariate polynomial. HAVEN [6] uses a particularly tailored bivariate polynomial to hide the secret, but still relies on the homomorphism of PKC-based PCS to reduce communication complexity.

Another straightforward idea for implementing HAVSS in the PKC setting is letting the dealer reliably broadcast a (non-interactive) public verifiable secret sharing (PVSS) transcript. Here PVSS enables a dealer to generate a single transcript encrypting each participant's secret share using the party's public key, along with a zk-proof attesting that any $\ell + 1$ subset of those shares can reconstruct a unique secret. Das et al. [21, 23] demonstrated a couple of instantiations of the idea using SCRAPE PVSS [15] and Paillier cryptosystem based PVSS [27]. Very recently, Das et al. [22] improved the approach to obtain a new way to implement HAVSS: the dealer, instead of directly broadcasting PVSS (which might have high computing cost), first distributes secret shares along with a PCS committing these shares as evaluations of a 2t-degree polynomial, then it solicits signatures on the PCS from sufficient parties. The final transcript to broadcast becomes the PCS, the set of valid signatures, and the encryption of secret shares (with proofs attesting the encrypted shares are committed to PCS). This novel method only avoids costly PVSS but still requires other PKC primitives like PCS and verifiable public key encryption.

If our goal is simply building HAVSS in the "minicrypt" world, without emphasizing concrete efficiency, an immediate "approach" seems to adapt the existing HAVSS protocols into the hash-only setting, by employing hash-based zk-STARK [9] to replace their current PKC components. To imagine the theoretic feasibility, consider the next implementation of hash-based bivariate PCS: (i) use n Merkle trees to commit n univariate polynomials of the first variable, and use another n Merkle trees to commit n univariate polynomials of the second variable, (ii) leverage hash-based STARK to prove that all committed n^2 points are consistent evaluations of the same bivariate polynomial with specified degrees for each variable, then (iii) use 2n Merkle tree roots and the above STARK proof as bivariate polynomial commitment, and for a certain point, use the corresponding Merkle paths of two associated Merkle trees as the evaluation proof. One can straightly replace the bivariate PCS in Bingo with the above scheme,

resulting in a literally hash-based HAVSS. However, different from SS24 AVSS with a direct reduction to hash functions, the above "solution" uses an expensive intermediate step of zk-STARK, introducing significant proving overhead (for complicated statements) and compromising the critical concrete efficiency.

Given the fact that there still lacks concretely efficient construction of HAVSS using only "lightweight" cryptography, we are asking the following questions:

Can we design an efficient high-threshold AVSS with optimal resilience, directly from lightweight cryptography like hash functions?

Our contribution. We answer the above question in the affirmative, by presenting a lightweight HAVSS protocol with a direct reduction to the cryptographic hash function. Particularly, our contribution is three-fold:

- As Table 1 illustrates, we construct a lightweight protocol for HAVSS with quasi-cubic word communication, directly using only cryptographic hash functions in the random oracle model, in the presence of a pair-wise secure communication channel between each two honest parties without other setup assumptions like CRS or PKI. The communication complexity of HAVSS protocol matches that of the state-of-the-art lightweight AVSS protocol without high-threshold secrecy i.e. SS24 AVSS [40] (when sharing a single secret). In addition, the sharing phase of our design can also attain quasi-quadratic word communication complexity in the normal case when the dealer is honest. Our HAVSS protocol also achieves optimal resilience, tolerating t < n/3 malicious parties, and has a best-possible reconstruction threshold of $\ell = 2t$.
- As Fig.1 illustrates, at the heart of our design, it is a new technique called distributed high-degree check (DHDC), enabling the distributed parties to check that the dealer has shared a 2t-degree polynomial across the network. Looking ahead, DHDC lets the dealer encode the 2t-degree polynomial into a bivariate polynomial with asymmetric t and 2t degrees for each variable. Then, every party verifies: (i) the dealer uses the probabilistic low-degree checking technique from [40] to commit univariate polynomials with degree up to t to Merkle trees,⁸ and (ii) an exclusive 2t-degree univariate polynomial (sent from the dealer) has a correct degree, and all its points are consistent with Merkle trees committing the t-degree univariate polynomials. As long as t+1 honest parties complete the above checks, a unique 2t-degree polynomial can be fixed, cf. the next Section for a brief overview of our technique.
- To showcase the usefulness of DHDC method, we further simplify the idea to construct a generic HAVSS compiler that can be instantiated from (almost)

⁴ Cheng et al.

⁸ Note that the probabilistic low-degree checking (PLDC) [40] does not generate a cryptographic zk-proof ensuring that a Merkle tree committing the evaluations of a polynomial with degree t, it rather lets every party perform some local check of its exclusive local share, such that a unique t-degree polynomial can be fixed by t + 1 honest parties who hold shares passing PLDC, see § 3 for a brief review of PLDC.

any t-degree AVSS. ⁹ This framework lets the dealer execute a batch of n AVSS instances, where each AVSS distributes a t-degree univariate polynomial derived from the bivariate polynomial with asymmetric degrees. As such, each party subsequently can obtain n evaluations associated with its exclusive 2t-degree univariate polynomial (from n AVSS instances), and locally check whether the degree is not larger than 2t. Once a distributed voting process can assert that t + 1 honest parties complete the local degree check, the bivariate polynomial shared by the dealer is thus fixed. When using some state-of-the-art batch t-degree AVSS [41, 2, 22] to instantiate our framework, it can realize (batch) HAVSS protocols with (amortized) quadratic word communication complexity per shared secret.



Fig. 1. Overview of our DHDC method. Note that evaluations in the red regions are sufficient to reconstruct the whole bivariate polynomial.

2 Technical Overview

Here we brief our key technique to overcome the challenge of designing a concretely efficient HAVSS from only lightweight cryptography.

Challenge of lightweight HAVSS: degree check of shared polynomials. Following many successful HAVSS designs using PKC primitives [35, 6, 2], we also would like to borrow the idea of performing 2t-degree secret sharing in aid of a bivariate polynomial $\phi(X, Y)$. More specifically, the two variables play different roles. The part of the first variable has a higher 2t-degree, and can be used to encode the secret to enforce a high threshold of reconstructing the secret. The part of the second variable has a lower t-degree, which further shares

⁹ To be more precise, we require an ACSS with a slight enhancement that enables a participant to verifiably transfer a reconstructed secret to the whole network—a requirement can be met by all ACSS protocols, possibly with minor modifications.

the 2t-degree shares, enabling the participants to recover the desired high-degree secret shares more easily. Previous works heavily rely on *polynomial commitment* scheme to securely implement the idea of using the bivariate polynomial for HAVSS. In particular, given a commitment of polynomial ϕ , parties can verify both the degree of ϕ and certain evaluations of ϕ . However, it is difficult to realize the functionality using only hash functions; for example, one can use a Merkle tree to commit the evaluations of ϕ , but it wouldn't enable the verification of correct degree, unless we seek heavy intermediate primitives like zk-STARK.

Probabilistic low-degree check (PLDC) and its limitation. Recently, Atapoor et al. [7] and Shoup and Smart [40] independently presented a *probabilistic low-degree check* (PLDC) method to verify the correct degree of a *t*-degree polynomial committed to a Merkle tree. The idea stems from a key observation: if we linearly combine two polynomials of degree *t*, the resulting polynomial should be of the same degree. Shoup and Smart leverage this idea to construct a *t*-degree AVSS protocol. For completeness, we briefly recall the PLDC method of Shoup and Smart hereunder, in the context of SS24 AVSS: ¹⁰

- 1. Dealer \mathcal{D} holds a *t*-degree polynomial ϕ which should be of degree *t* and want to share it among *n* parties. The goal of \mathcal{D} is to convince parties ϕ has the right degree without leaking secret $\phi(\omega_0)$. To this end, \mathcal{D} randomly samples another polynomial $\hat{\phi}$ of the same degree, and uses a Merkle tree to commit ϕ using $\hat{\phi}$ as blinding randomness. \mathcal{D} then derives a challenge θ from the Merkle tree commitment through the Fiat-Shamir heuristic in the random oracle [26]. After that, \mathcal{D} computes polynomial $f = \phi + \theta \hat{\phi}$ and publishes the commitment and f. Besides, each party will also receive their exclusive shares of ϕ and $\hat{\phi}$ along with corresponding Merkle proofs.
- 2. After receiving the above information from \mathcal{D} , each party \mathcal{P}_i will make two verifications: (i) \mathcal{P}_i checks if its shares x_i and y_i are consistent with the commitments by checking Merkle proofs; (ii) it checks if $f(\omega_i) = x_i + \theta y_i$. If both verifications pass, \mathcal{P}_i is a party receiving valid shares. As proven in [40], once there are t + 1 honest parties receiving valid shares, there is only a negligible probability, such that another honest party might receive valid shares that are not consistent with other t + 1 honest parties' valid shares, i.e., interpolating the valid shares of any t + 1 honest parties must return the unique t-degree polynomial with an overwhelming probability.
- 3. Finally, the distributed parties follow the protocol flow of Bracha's reliable broadcast [10] to count if there are t+1 honest parties receiving honest shares, thus deciding an output in an AVSS execution. The remaining subtlety to handle is that only t + 1 honest parties may receive valid shares, and the other honest parties might receive invalid shares, due to the influence of a malicious dealer and asynchronous network. Therefore, a complaint phase is

¹⁰ Note that Shoup and Smart [40] also give a batched version of the above PLDC idea, enabling parties to check the degrees of multiple *t*-degree polynomials at the same time. We re-introduce its non-batch version for presentation simplicity. We also refer interested readers to Section 5 of SS24 [40] for more details.

also designed in SS24 to enable the honest parties receiving invalid shares to generate a proof transcript convincing other honest parties to publish the secret of a corrupted dealer using their valid shares.

Clearly, the above PLDC approach only works for the degree test of t-degree polynomials, and it cannot directly meet our requirement of ensuring that the dealer commits a 2t-degree polynomial. This is because its key assurance that "the valid shares of any t+1 honest parties must fix the unique polynomial" would be breached if the committed polynomial is 2t-degree. If we directly apply LPDC to our setting of 2t-degree verifiable secret sharing, the protocol might never output, as we need to ensure all honest parties receive valid shares to ensure the committed polynomial has the correct 2t-degree. This clearly requires waiting for all vote messages from honest parties, which is impossible in asynchrony, as the adversary can always delay up to t honest parties to let the other honest nodes proceed without the involvement of these delayed parties.

Our distributed high-degree check (DHDC) approach. We propose the distributed high-degree check (DHDC) technique to resolve this problem of checking the correct degree of some committed 2t-degree polynomial. Our approach encodes the to-be-shared 2t-degree polynomial using a bivariate polynomial (as Fig.1 illustrates), and stems from a basic fact that: given $(t+1) \times (2t+1)$ evaluations of a bivariate polynomial (as exemplified by evaluations in the red regions of Fig.1), we can interpolate a unique bivariate polynomial $\phi(X, Y)$ with degrees no more than t in X and no more than 2t in Y. If we can fix such a grid of evaluations received by honest parties, we can bind a bivariate polynomial. Now, the challenge is reduced to how to fix a $(t + 1) \times (2t + 1)$ grid of valid evaluations received by the honest parties.

Notably, a seemingly viable idea of letting the dealer prove that all column polynomials are of degree t and all row polynomials are of degree 2t is infeasible, as PLDC are not applicable to prove 2t-degree polynomials, causing a circular problem of checking another 2t-degree polynomial while sharing a 2t-degree polynomial. Fortunately, we realize a way to break the circular situation: it is unnecessary to make all parties check the degree of all 2t-degree row polynomials, conditioned that if we already leverage PLDC to verify the t-degree of all n column polynomials; and actually, we only need to fix t + 1 row polynomials with the correct degree 2t, after verifying the correct degree of column polynomials. This can be done by letting every party locally check the degree of its exclusive 2t-degree row polynomial (and also verify this polynomial's evaluations are correctly committed to column polynomials' commitments used by PLDC), and then, if both verifications pass, participants perform a distributed voting phase following Bracha's reliable broadcast structure, thus collectively fixing a bivariate polynomial (which further fixes the to-be-shared 2t-degree polynomial).

After adding a *complaint stage* to help the honest parties receiving incorrect row polynomials and a *reconstruction phase* to correct up to t malicious shares, we finally realize a lightweight HAVSS protocol with direct reduction to cryptographic hash functions, cf. Section 4 for details of our construction.

Simplifying DHDC technique towards a generic HACSS compiler. Furthermore, we note that PLDC is not the only option for verifying the degree of all column polynomials in DHDC. As Fig. 2 illustrates, the effect of batched PLDC can also be realized by a batch of t-degree AVSS (particularly ACSS) protocols. Since t-degree AVSS inherently guarantees the correct degree of the shared polynomial, they can effectively be used to verifiably share all t-degree column polynomials in the DHDC approach. Once all t-degree column polynomials are correctly shared, every party can simultaneously obtain all evaluations of its row polynomials, and locally check whether it has a correct degree not larger than 2t, such that the honest parties can follow the structure of Bracha's reliable broadcast to vote whether there are t + 1 honest parties receive valid row polynomials. If the voting phase completes, a bivariate polynomial is thus fixed, ensuring the unique to-be-shared 2t-degree polynomial.

Building on the idea, we give a generic compiler that transforms any t-degree ACSS into 2t-degree HAVSS, cf. Section 5 for the framework's details.



Fig. 2. Overview of our generic HAVSS framework.

3 Preliminaries and Problem Formulation

Notations. We use [n] to denote the set $\{1, ..., n\}$ and let $\{P_i\}_{i \in [n]}$ denote the set of participants. The security parameter is denoted by λ , and $\epsilon(\lambda)$ represents a negligible function in λ . We write $\mathbb{Z}_N := \mathbb{Z}/N\mathbb{Z}$ and $\mathbb{Z}_N[X]_t$ for polynomials of degree t in the variable X and with coefficients in \mathbb{Z}_N . In the same way, $\mathbb{Z}_N[X, Y]_{t,v}$ denotes bivariate polynomials with degree t in X and degree v in Y. Let set $(\omega_1, ..., \omega_n) \in \mathbb{Z}_N^n$ denotes the coordinates where to evaluate univariate polynomials. Thus, for a bivariate polynomial $\phi(X, Y)$ in $\mathbb{Z}_N[X, Y]_{t,v}$, we would consider its $n \times n$ evaluations at these points $\{(\omega_i, \omega_j)\}_{i,j \in [n]}$. For all evaluations in $\phi(X, Y)$ with the same coordinate in X, they can be interpolated to a polynomial in $\mathbb{Z}_N[Y]_v$. We write $\phi(\omega_i, Y)$ for it. We also write $\phi(X, \omega_i)$ similarly for the other variable.

System and threat model. We adopt a widely adopted asynchronous messagepassing model with up to t < n/3 Byzantine corruption, where *n* represents the number of participants. Every pair of parties is connected by a secure communication channel, ensuring both confidentiality and authenticity of messages transmitted between honest parties. Messages can be arbitrarily delayed by the adversary, but would eventually be delivered without tampered. We do not consider other setup assumptions like PKI, CRS, or DKG. Following standard cryptographic practices from the seminal work of Cachin et al. [12], the adversary \mathcal{A} is assumed to be probabilistic polynomial-time (PPT) bounded, and it is considered the number of participants $n \leq \mathcal{O}(\lambda)$ (also assumed by SS24 [40]). Same to SS24 [40], we consider a static adversary who decides t malicious parties to corrupt before the start of protocol execution, throughout the paper.¹¹

Reliable Broadcast (RBC) is an asynchronous protocol with a designed sender and can simulate an ideal broadcast channel in a point-to-point network. A secure RBC protocol satisfies the following properties.

- Agreement: the outputs of any two honest parties are the same.
- *Validity*: if the sender is honest, all honest parties will eventually output the sender's input.
- Totality: if an honest party outputs, all honest parties will eventually output.

High-threshold asynchronous verifiable secret sharing (HAVSS) is a two-phase asynchronous protocol with a designed dealer \mathcal{D} . Syntactically, HAVSS consists of a Share phase and a Reconstruction phase: the dealer shares an input secret s across the participants in the Share phase, and the participants can subsequently recover the shared secret in the Reconstruction phase. A HAVSS protocol shall satisfy the following properties with all but negligible probability:

- Correctness:
 - Conditioned that \mathcal{D} is honest, all honest parties will eventually output in the Share phase.
 - If \mathcal{D} is honest with taking secret s as input, then once all honest parties participate in the Reconstruction phase, they can eventually recover the same secret s.
 - If some honest parties output in the Reconstruction phase, their outputs shall be the same secret.
- *Termination*:
 - If an honest party outputs in the Share phase, all honest parties will eventually output in the Share phase.
 - All honest parties will eventually output a secret if they all terminate in the Share phase and activate the Reconstruction phase.
- Completeness: If some honest party outputs in the Share phase, there must exist a fixed secret s', such that each honest party will eventually output a 2t-degree share of it in the Share phase.
- Secrecy: It is computationally infeasible for the adversary to learn any bit of information about an honest dealer's input secret s, before t + 1 honest parties activate the Reconstruction phase.

¹¹ Note that we speculate that our HAVSS could be resistant to adaptive corruptions in the random oracle model, though we only claim its static security.

4 Our Lightweight HAVSS Protocol

In this section, we present our new lightweight HAVSS protocol that is directly built from hash functions. As aforementioned, our HAVSS can share a 2t-degree secret, thus with a reconstruction threshold $\ell = 2t$. For ease of description, we would divide the sharing phase of our HAVSS protocol into two stages—the normal-case stage and the complaint stage, where the honest parties will skip the later complaint stage if the dealer \mathcal{D} is honest. We also thoroughly analyze our design by presenting detailed security proof in Appendix A.

4.1 Sharing Phase

In the following, we present our the sharing phase of lightweight HAVSS construction. For ease of read, its sharing phase protocol will be presented in three parts: (i) the dealer's code, (ii) the normal-case stage, and (iii) the complaint stage. Here the complaint stage could be skipped if the dealer is honest.

Note that our protocol only uses one cryptographic primitive—hash function \mathcal{H} (modeled as random oracle). For convenience, we use specific notations to denote the interfaces of Merkle tree, which nevertheless is still built from only hash functions. We let MerkleBuild({ $(x_1, r_1), (x_2, r_2), \cdots, (x_n, r_n)$ }) \rightarrow mt to denote the function computes a Merkle tree committing { $x_1 || r_1, x_2 || r_2, \cdots, x_n || r_n$ }, let mt.root to denote the root of Merkle tree mt, and let MerkleVerify denote the function to verify an element is indeed committed to a given Merkle tree root using a Merkle proof (a.k.a. Merkle branch, consisting the minimum number of hash values to enable one recompute the Merkle tree root with a leaf element).

Sharing phase of dealer. The detailed description of the dealer's execution during the sharing phase is shown in Figure 3. After receiving an input secret $s \in \mathbb{F}$ to be shared, the dealer \mathcal{D} follows our DHDC methodology (cf. Section 2 for the technique's overview) to prepare the messages to be sent as follows.

 \mathcal{D} first uniformly samples two bivariate polynomials ϕ and $\hat{\phi}$, where ϕ is from $\mathbb{Z}_F[X,Y]_{2t,t}$ such that $\phi(\omega_0,\omega_0) = s$ and $\hat{\phi}$ is from $\mathbb{Z}_F[X,Y]_{n,t}$ (this can be done by uniformly sampling *n* univariate *t*-degree polynomials) (lines 1-2). Note that, $\hat{\phi}$ is used to Then, \mathcal{D} can compute row polynomials $\alpha_i = \phi(X,\omega_i)$ and $\hat{\alpha}_i = \hat{\phi}(X,\omega_i)$ for every $i \in [n]$, as while as column polynomials $\beta_i = \phi(\omega_i,Y)$ and $\hat{\beta}_i = \hat{\phi}(\omega_i,Y)$ (lines 4-5). For each pairing of column polynomials β_i and $\hat{\beta}_i, \mathcal{D}$ builds a Merkle tree mt_i for $((\beta_i(\omega_1), \hat{\beta}_i(\omega_1), \dots, (\beta_i(\omega_n), \hat{\beta}_i(\omega_n)))$ with root $\mathsf{mt}_i.\mathsf{root}$ (line 6). Here $\mathsf{mt}_i.\mathsf{root}$ is the commitment for all evaluations of β_i and $\hat{\beta}_i$ under coordinates $(\omega_1, \dots, \omega_n)$. \mathcal{D} then builds another tree mt for $(\{\mathsf{mt}_i.\mathsf{root}\}_{i\in n})$ with root $\mathsf{mt}.\mathsf{root}$. The root $\mathsf{mt}.\mathsf{root}$ corresponds to the commitment of all evaluations of all column polynomials at coordinates $(\omega_1, \dots, \omega_n)$, and thus we can let \mathcal{H} return a series of random challenges $(\theta_1^{(i)})_{i\in[n]}$ for PLDC, using the commitment $\mathsf{mt}.\mathsf{root}$ (line 9). \mathcal{D} then computes challenge polynomial g for degree check by linearly combining $\{\beta_i\}_{i\in[n]}$ with $\hat{\beta}_0$ (line 10).

 \mathcal{D} then broadcasts $(g, \{\mathsf{mt}_i.\mathsf{root}\}_{i \in n})$ to all parties via RBC (line 11). After that, for each party $\mathcal{P}_i, \mathcal{D}$ sends the *i*-th row polynomial α_i along with Merkle

Sharing phase of HAVSS for dealer \mathcal{D} // Upon receiving an input secret s to be shared 1: uniformly sample $\phi(X, Y)$ with degree 2t in X and t in Y s.t. $\phi(\omega_0, \omega_0) = s$ 2: uniformly sample $\widehat{\phi}(X, Y)$ with degree n in X and t in Y 3: for all $i \in [n]$ do $\alpha_i \leftarrow \phi(X, \omega_i) , \widehat{\alpha}_i \leftarrow \widehat{\phi}(X, \omega_i)$ 4:
$$\begin{split} & \beta_i \leftarrow \phi(\omega_i, Y) \ , \ \widehat{\beta}_i \leftarrow \widehat{\phi}(\omega_i, Y) \\ & \mathsf{mt}_i \leftarrow \mathsf{MerkleBuild}(\{(\beta_i(\omega_j), \widehat{\beta}_i(\omega_j))\}_{j \in [n]}) \end{split}$$
 $\triangleright \alpha_i(\omega_j) = \beta_j(\omega_i)$ 5: 6: 7: $\widehat{\alpha}_{\theta} \leftarrow \widehat{\phi}(X, \omega_0)$, $\widehat{\beta}_{\theta} \leftarrow \widehat{\phi}(\omega_0, Y)$ 8: $mt \leftarrow MerkleBuild(mt_1.root, ..., mt_n.root)$ 9: compute challenge $(\theta_1^{(i)})_{i \in [n]} \leftarrow \mathcal{H}(\mathsf{mt.root})$ 10: compute $g \leftarrow \widehat{\beta}_0 + \sum_{i \in [n]} \theta_1^{(i)} \beta_i$ 11: reliably broadcast $(g, \{r_i | r_i = \mathsf{mt}_i.\mathsf{root}\}_{i \in [n]})$ 12: for all $i \in [n]$ do 13:send SEND $(\alpha_i, \widehat{\alpha}_i, \{\pi_{col(ki)}\}_{k \in n})$ to \mathcal{P}_i $\pi_{col(ki)}$ is the Merkle proof attesting that $\beta_k(\omega_i) || \hat{\beta}_k(\omega_i)$ is the *i*-th leaf 14:committed to the Merkle tree root r_k

Fig. 3. Our HAVSS protocol: the part of dealer in the sharing phase.

proofs for all its evaluations (lines 12-14). Note that the Merkle proof for an evaluation is simply the path of hash values to attest that it is the i-th leaf of a corresponding Merkle tree committing some column polynomial.

Sharing phase of receiving parties (normal-case stage). Figure 4 shows the normal stage of the receiving parties' execution during the sharing phase. We will divide the description into the following steps based on the message types.

Step 1 (lines 1-12):

Upon receiving $(g, \{r_i\}_{i \in n})$ from \mathcal{D} via RBC, each party \mathcal{P}_j first check whether the degree of g is no greater than t, respectively. After that, \mathcal{P}_j can compute the challenges $\{\theta_1^{(k)}\}_{k \in [n]}$ in the same way with \mathcal{D} (lines 2-3). Then, the party \mathcal{P}_j waits for receiving the SEND $(\alpha_j, \hat{\alpha}_j, \{\pi_{col(kj)}\}_{k \in n})$ message from \mathcal{D} , and performs the following DHDC verifications (lines 4-7):

- The degrees of α_j and $\hat{\alpha}_j$ are both no greater than 2t;
- For each $k \in [n]$, $\pi_{col(kj)}$ is a valid Merkle proof attesting that $\alpha_j(\omega_k) || \hat{\alpha}_j(\omega_k)$ is the k-th leaf committed to Merkle tree root r_k ;
- Conduct the PLDC verification for all committed *t*-degree column polynomials: $g(\omega_j) = \widehat{\alpha}_j(\omega_0) + \sum_{k \in [n]} \theta_1^{(k)} \alpha_j(\omega_k).$

If all the above DHDC checks can pass, \mathcal{P}_j receives a valid SEND message from the dealer. Then, for each $i \in [n]$, \mathcal{P}_j sends ECHO message containing $\alpha_j(\omega_i)$ and $\hat{\alpha}_j(\omega_i)$ to \mathcal{P}_i along with their Merkle proofs (lines 11-12).

Sharing phase (normal-case) of HAVSS for each receiving party \mathcal{P}_i $\mathsf{points}_{\beta,j} \leftarrow \emptyset, \, \mathsf{points}_{\widehat{\beta},j} \leftarrow \emptyset, \, \mathsf{\Pi}_{col(j)} \leftarrow \emptyset, \, \mathsf{\Pi}_{row(j)} \leftarrow \emptyset$ $rowReady \leftarrow false$, $colReady \leftarrow \perp$, $compReady \leftarrow \perp$, $rowSet \leftarrow \emptyset$, $shares \leftarrow \emptyset$ // Handle SEND message 1: wait until receiving $(g, \{r_k\}_{k \in [n]})$ from \mathcal{D} , where $g \in \mathbb{Z}_F[Y]_t$ $r \leftarrow \mathsf{MerkleBuild}(\{r_k\}_{k \in [n]})$ 2: calculate challenge $\{\theta_1^{(k)}\}_{k \in [n]} \leftarrow \mathcal{H}(r)$ 3: 4: upon receiving the first SEND $(\alpha_j, \widehat{\alpha}_j, \{\pi_{col(kj)}\}_{k \in n})$ message from \mathcal{D} , do if the degrees of α_j and $\hat{\alpha}_j$ are both no greater than 2t then 5:if for each $k \in [n]$, MerkleVerify $(\alpha_j(\omega_k), \widehat{\alpha}_j(\omega_k), \pi_{col(kj)}, r_k) =$ true then 6: if $g(\omega_j) = \widehat{\alpha}_j(\omega_0) + \sum_{k \in [n]} \theta_1^{(k)} \alpha_j(\omega_k)$ then 7: \triangleright checks for DHDC $rowReady \leftarrow true$ 8: 9: $\mathsf{points}_{\alpha,j} \leftarrow \{\alpha_j(\omega_i)_{i \in [n]}\}, \mathsf{points}_{\widehat{\alpha},j} \leftarrow \{\widehat{\alpha}_j(\omega_i)_{i \in [n]}\}$ $\Pi_{row(j)} \leftarrow \{\pi_{col(kj)}\}_{k \in n}$ 10: for all $k \in [n]$ do 11: send ECHO $(\alpha_j(\omega_k), \widehat{\alpha}_j(\omega_k), \pi_{col(kj)})$ to \mathcal{P}_k 12:// Handle Есно message 13: **upon** receiving ECHO $(\alpha_k(j), \widehat{\alpha}_k(j), \pi_{col(jk)})$ for the first time from \mathcal{P}_k do if MerkleVerify $(\beta_j(\omega_k), \hat{\beta_j}(\omega_k), \pi_{col(jk)}, r_j) =$ true then 14:15: $\mathsf{points}_{\beta,j} \leftarrow \mathsf{points}_{\beta,j} \cup \beta_j(\omega_k) \ , \ \mathsf{points}_{\widehat{\beta},j} \leftarrow \mathsf{points}_{\widehat{\beta},j} \cup \beta_j(\omega_k)$ $\Pi_{col(j)} \leftarrow \Pi_{col(j)} \cup \pi_{col(jk)}$ 16:17: **upon** $|\mathsf{points}_{\beta,i}| = t + 1$ and $|\mathsf{points}_{\hat{\beta},i}| = t + 1$ do $\beta_j' \leftarrow \mathsf{Interpolate}_t(\mathsf{points}_{\beta,j}), \ \widehat{\beta}'_j \leftarrow \mathsf{Interpolate}_t(\mathsf{points}_{\widehat{\beta},j})$ 18: $r_j' \leftarrow \mathsf{MerkleBuild}(\beta_j', \widehat{\beta}'_i)$ 19: if $r_j = r_j'$ then $colReady \leftarrow$ true else $colReady \leftarrow$ false 20: 21: upon $|\mathsf{points}_{\beta,j}| = 2t + 1$ and $|\mathsf{points}_{\widehat{\beta},j}| = 2t + 1$ do multicast DONE() 22:// Handle DONE message 23: **upon** receiving DONE() from t + 1 distinct parties **do** 24:multicast DONE() if haven't sent 25: **upon** receiving DONE() from n - t distinct parties **do** 26:wait until colReady $\neq \perp$ 27:if colReady = true then $output OUTPUT1(\beta_j, \beta_j, \{r_i\}_{i \in [n]})$ 28:if colReady = false then29:multicast COMPLAIN(points_{β,j}, points_{$\hat{\beta},j$}, $\Pi_{col(j)}$)

Fig. 4. Our HAVSS protocol: the part of the normal-case stage in the sharing phase.

Step 2 (lines 13-22):

Upon receiving an ECHO message, \mathcal{P}_j first checks if the contained evaluations are consistent with r_j (line 14). We call a ECHO message valid if it passes this check. Upon receiving t + 1 valid ECHO messages, \mathcal{P}_j can interpolate its column polynomials β'_j and $\hat{\beta}'_j$ (line 18). \mathcal{P}_j then check if β'_j is consistent with r_j (lines 19-20). We call this column polynomial valid if it passes this check, and whether it's valid is the condition whether \mathcal{P}_j sends a COMPLAIN message in complaint stage. Upon receiving 2t + 1 ECHO messages, \mathcal{P}_j can ensure that at least t + 1honest parties have corresponding valid row polynomials. A DONE message is multicast to notify others of this (lines 21-22).

Step 3 (lines 23-29):

Upon receiving t + 1 DONE messages, \mathcal{P}_j will forward it if it hasn't sent one. Upon receiving 2t + 1 DONE messages, \mathcal{P}_j can ensure that all honest parties will eventually obtain 2t + 1 DONE messages. Here it can output its secret share if β_j and $\hat{\beta}_j$ are valid (line 27). Note that the evaluation of β_j at ω_0 is \mathcal{P}_j 's secret Share. And \mathcal{P}_j will output $\beta_j(\omega_0)$ here.

Note if \mathcal{P}_j interpolates some column polynomials inconsistent with Merkle trees when receiving n-t DONE messages, it will multicast COMPLAIN message to request assistance from others (lines 28-29). The COMPLAIN message contains evaluations received for its column polynomials along with corresponding Merkle proofs. We will continue to explain how to handle COMPLAIN messages in Fig.5.

Complaint stage of the sharing phase. Figure 5 illustrates the complaint stage to handle the bad case that some honest party fails to recover a valid column polynomial (which is supposed to encode the 2t-degree secret share).

Sharing phase (complaint stage) of HAVSS for each receiving party \mathcal{P}_j			
// As continuation of Fig. 4			
1: upon receiving COMPLAIN(points _{β,k} , points _{$\hat{\beta},k$} , $\Pi_{col(k)}$) first time from \mathcal{P}_k do			
2: if $compReady = false then$			
3: if MerkleVerify(points _{β,k} , points _{$\hat{\beta},k$} , $\Pi_{col(k)}, r_k$) = true then			
4: $\beta_k' \leftarrow Interpolate_t(points_{\beta,k}), \ \widehat{\beta}'_k \leftarrow Interpolate_t(points_{\widehat{\beta},k})$			
5: $r' \leftarrow MerkleBuild(\beta_k', \widehat{\beta}'_k)$			
6: if $r' \neq r_k$ then			
7: $compReady = true$			
8: if $rowReady = true then$			
9: reliably broadcast $(\alpha_j, \hat{\alpha}_j, \Pi_{row(j)})$ as sender of ReliableBroadcast _j			
10: upon $compReady = true$ do			
11: upon ReliableBroadcast _k output $(\alpha_k, \widehat{\alpha}_k, \prod_{row(k)})$ do			
12: if MerkleVerify $(\alpha_k, \widehat{\alpha}_k, \prod_{row(k)}, \{r_l\}_{l \in [n]}) = $ true then			
13: if the degree of α_k is no greater than $2t$ then			
14: if $g(\omega_k) = \widehat{\alpha}_k(\omega_0) + \sum_{i \in [n]} \theta_1^{(i)} \alpha_k(\omega_i)$ then			
15: $\operatorname{rowSet} \leftarrow \operatorname{rowSet} \cup \alpha_k$			
16: upon $ rowSet = t + 1$ do			
17: $\phi' \leftarrow Interpolate_{2t,t}(rowSet)$			
18: output OUTPUT2 $(\phi'(\omega_j, \omega_0), \phi'(\omega_0, \omega_0))$			

Fig. 5. Our HAVSS protocol: the part of complaint stage of the sharing phase.

A COMPLAIN message is valid, if it carries a sufficient number of polynomial evaluations that are committed to Merkle trees but their interpolations cannot recover the same Merkle tree roots (lines 3-6). Upon receiving the first valid COMPLAIN message, \mathcal{P}_j checks if it has received valid row polynomials from \mathcal{D} . If so, it will invoke an RBC to reliably broadcast row polynomials along with their Merkle proofs (line 9). We call such a reliable broadcast instance assistant reliable broadcast (assisRBC). Note that if honest parties participate in all possible assisRBCs without a sanity check, the adversary can waste their communication through assisRBCs. To address this problem (and preserve a lower communication cost during the normal case with an honest dealer), \mathcal{P}_{i} will respond for assisRBC only after receiving a valid COMPLAIN message (line 10). Clearly, an assisRBC instance in the complaint stage is similar to the SEND message in the normal-case stage, and \mathcal{P}_{i} conducts the same checks on the outputs of assisRBCs as to the SEND message (lines 11-15). An assisRBC is said valid if its output passes all checks specified by lines 11-15. \mathcal{P}_{j} will store all valid outputs of assis RBC instances. Once receiving t + 1 valid assis RBCs, \mathcal{P}_i can interpolate a bivariate polynomial ϕ' (line 17). The final output share of \mathcal{P}_j therefore becomes $\phi'(\omega_j, \omega_0)$ (line 18). Note that the secret $\phi'(\omega_0, \omega_0)$ is also returned as part of the output, which later will be used by the reconstruction phase.



Fig. 6. Our HAVSS protocol: the part of reconstruction phase.

4.2 Reconstruction Phase

Figure 6 shows the details of the reconstruction phase. After activating reconstruction, \mathcal{P}_j will multicast the column polynomials β_j and $\hat{\beta}_j$ through SHARE message if it has obtained valid ones during the earlier sharing phase (lines 1-2). A SHARE message is valid if its column polynomials are consistent with the corresponding commitment and of the correct degree (t) (lines 7-11). There are two paths to reconstruct the final secret. Firstly, if \mathcal{P}_j can gather 2t + 1 valid shares, it then interpolates a 2t-degree polynomial α_0' . And the secret s is $\alpha_0(\omega_0)$ (lines 12-14). But, if some honest parties obtain invalid polynomials, \mathcal{P}_j may never be able to collect enough valid SHARE messages. So we need a second output path in the malicious case. As earlier described in Fig. 5, all parties can recover the secret in the COMPLAIN stage in case of a malicious dealer, thus ensuring that they can directly output the secret here (lines 5-6).

4.3 Complexity Analysis

We now analyze the communication complexity of our HAVSS protocol. The communication complexity is defined as the expected number of bits sent by honest parties. We classify the possible executions of the sharing phase into two categories: "normal case" and "bad case". The classification is based on whether any honest party receives a valid COMPLAIN message. We also separately analyzed the communication complexity of the reconstruction phase. We use m to denote the bit length of a field element, which essentially corresponds to a statistical security parameter. Thus, the size of a polynomial is $\mathcal{O}(nm)$. We use the reliable broadcast protocol in [21], which achieves a communication complexity of $\mathcal{O}(n|M| + \lambda n^2)$, where |M| is the size of the input.

Normal case of sharing: In "normal case", honest parties only participate in reliable broadcast protocol led by the dealer in the normal case. The input contains two polynomials and n hashes with a total complexity of $\mathcal{O}(nm + n\lambda)$. Thus, the communication complexity of this reliable broadcast is $\mathcal{O}(n^2m + \lambda n^2)$. The SEND message contains two polynomials and n Merkle proofs. Thus, the cost on it is $\mathcal{O}(n^2m + \lambda n^2 \log n)$. The ECHO message contains two evaluations and one Merkle proof and the cost is $\mathcal{O}(n^2m + \lambda n^2)$. The DONE message can be asymptotically ignored as it contains only a single bit. The total communication complexity of the normal case is thus $\mathcal{O}(n^2m + \lambda n^2 \log n)$.

Recall that the statistical security parameter m approximates $\mathcal{O}(\lambda)$ to ensure the security of PLDC (if $n \leq \mathcal{O}(\lambda)$, cf. SS24 for the parameter choice [40]), this finally rendering a normal-case communication complexity of $\mathcal{O}(\lambda n^2 \log n)$.

Bad case of sharing: The additional cost of "bad case" comes from the complaint stage. There might be at most n parties broadcasting their row polynomials by invoking assisRBC. The input contains two polynomials and n Merkle proofs. Thus, the cost of n assisRBC instances is $\mathcal{O}(n^3m + \lambda n^3 \log n)$. Adding this to the cost of normal case, the total communication cost of the "bad case" becomes $\mathcal{O}(n^3m + \lambda n^3 \log n)$, which is $\mathcal{O}(\lambda n^3 \log n)$ for properly chosen m.

Reconstruction: As aforementioned, all honest parties may multicast their column polynomials here, bringing a cost of transmitting $\mathcal{O}(n^3m)$ bits (essentially $\mathcal{O}(\lambda n^3)$), which still corresponds to cubic word complexity.

4.4 Security Analysis

Here we brief the security intuition of our lightweight HAVSS protocol, while deferring the detailed proof to Appendix A:

- For Correctness: If when \mathcal{D} is honestly executing, it is trivial to see that all evaluations will be correctly computed and committed to some broadcasted Merkle trees. Thus, all honest parties can successfully finish protocol and reconstruct the same s as \mathcal{D} , unless the binding of Merkle tree is breached. In addition, if two honest parties reconstruct conflicting secrets, DHDC thus fails, which can only be caused by two bad events: (i) the breach of commitment's binding or (ii) the statistical error of PLDC. Both of them have negligible probability of happening.
- For Termination: When \mathcal{D} is honest, it is trivial to see that all honest parties would output during the normal-case stage. Otherwise, in the presence of some probably malicious dealer, if some honest parties output in the complaint stage, the rest honest parties will also do so, as a result of using RBC to broadcast messages triggering the complaint-stage output.
- For Completeness: The DHPC mechanism ensures that a bivariate polynomial ϕ' from $\mathbb{Z}_F[X, Y]_{2t,t}$ is fixed once an honest party finishes it. Honest parties can detect whether their shares are consistent with ϕ' . Thus, completeness property can be held.
- For Secrecy: As we model the hash function as a random oracle, the commitment for each evaluation is hiding, because we use a randomly sampled polynomial to mask the main bivariate polynomial (encoding s). Thus, leaks no bit of information to the adversary even if s has low entropy.

5 A Generic Framework for Resilience-Optimal HAVSS

In this section, we will introduce our generic compiler for instantiating 2t-degree HAVSS from any t-degree AVSS (or simply referred to as AVSS throughout the section, as long as there is no confusion with HAVSS). As Fig 2 illustrates, the HAVSS framework is inspired by the DHDC method of our lightweight HAVSS construction, which leverages each party's local degree check and conducts a distributed voting process to verify the correct 2t-degree of at least t + 1 rows of a bivariate polynomial, such that fixing the bivariate polynomial (if all its columns have also been verified to have correct t-degree).

An inspiration of simplifying distributed high-degree checking (DHDC). When applying the DHDC technique to implement our lightweight HAVSS, we need an additional complaint stage to enable all honest parties to recover their

shares if needed. This is because we leverage LPDC for verifying the *t*-degree of column polynomials, which cannot prevent the dealer from committing an incorrect column with a higher degree, thus requiring some honest parties to ask for recovering the correct columns with *t*-degree in the bad case. The complaint stage not only complicates the protocol, making it harder to parse and understand, but also introduces a significant cubic communication overhead.

A key simplification realized by us is that: we can simply let the dealer invoke an AVSS protocol (with completeness property) to distribute each column polynomial, thus avoiding the extra complaint stage and realizing a much simpler protocol structure for HAVSS. Moreover, to avoid the communication blow-up of executing n concurrent AVSS protocols, the dealer can also invoke a batch AVSS protocol [2, 22, 5, 40, 41] for efficiency, such that it can distribute n column polynomials simultaneously while preserving an asymptotic communication complexity same to that of sharing a single secret.

Preparing AVSS with minor modification. Our HAVSS framework requires an extra interface and property to the *t*-degree AVSS — *verifiable secret transferability*, to facilitate the reconstruction of the high-threshold secret. This property can be informally stated as follows:

- Verifiable secret transferability: A desired AVSS protocol should allow parties to verify the validity of the secret s' recovered through individual reconstruction. Specifically, all parties can extract some information called Context after sharing phase of this AVSS protocol, and the designated party who reconstructs s' can extract some additional information called Proof after individual reconstruction. There also exists a function SecretVerify to verify whether s' is the valid secret or not, using Proof and Context. Assuming s be the secret reconstructed from the original reconstruction phase of this AVSS protocol, there should be:

 $\Pr[s \neq s' | \text{SecretVerify}(s', \text{Context}, \text{Proof}) = \text{true}] \leq \epsilon(\lambda).$

The above small change can be accommodated by all AVSS protocols with at most minor modifications. If we use PCS-based AVSS, the polynomial commitment and its evaluation proofs can meet the requirement. For AVSS protocols that don't rely on PCS [40, 33], a generic approach to amendment them is introducing signatures on secret shares during reconstruction, assuming extra PKI setup, such that a sufficient number of secret shares with valid signatures can be the proof for secret's correctness.

Construction of the generic HAVSS framework. The resulting HAVSS framework is described in Fig. 7, as a compiler executing the following steps to translate any AVSS (possibly with our minor modifications) into HAVSS.

- Sharing phase (t-degree sharing stage, lines 1-10): The dealer \mathcal{D} first samples a bivariate polynomial $\phi(X, Y)$ with $\phi(\omega_0, \omega_0) = s$ to encode the input secret. Then *n* row and *n* column polynomials can be derived. For each column polynomial, \mathcal{D} invokes an AVSS instance to share it. After all AVSS instances

are terminated, each party \mathcal{P}_i can get n evaluations of $\phi(X, \omega_i)$ for X chosen among $\{\omega_j\}_{j\in[n]}$, and \mathcal{P}_i can extract $\{\text{Context}_j\}_{j\in[n]}$ helping to verify the validity of $\phi(\omega_j, \omega_0)$ for each $j \in [n]$ (here $\phi(\omega_j, \omega_0)$ is the secret shared by each *t*-degree AVSS instance). Then, each party \mathcal{P}_i checks if n evaluations received from n AVSS instances lie on some polynomial $\phi(X, \omega_i)$ of degree no more than 2t.

- Sharing phase (2t-degree share output stage, lines 11-23): After that, a tworound vote is executed by following the structure of Bracha's broadcast. Besides, for each $i \in [n]$, each honest party \mathcal{P}_j will send $share_{ij} := \phi(\omega_i, \omega_j)$ (its share received from *i*-th AVSS instance) to \mathcal{P}_i . The completeness property enables \mathcal{P}_i to interpolate $share_i := \phi(\omega_i, \omega_0)$ through online error correction (OEC) [17]. It is worth noting that if the underlying AVSS provides an interface to verify shares, OEC can be avoided. \mathcal{P}_i then outputs its 2t-degree share $share_i$ along with ContextSet := {Context_j}_{j \in [n]}, and a Proof for the correctness of its share.
- Reconstruction phase (lines 101-108): Upon initiating the reconstruction phase with (share_i, ContextSet, Proof_i)—the output from the sharing phase, \mathcal{P}_i will multicast share_i along with corresponding proof Proof_i (line 102). Upon receiving the pair (share_j, Proof_j) from \mathcal{P}_j , \mathcal{P}_i verifies its validity using the SecretVerify function. If SecretVerify(share_j, ContextSet.Context_j, Proof_j) returns true, \mathcal{P}_i stores share_j (lines 103-105). Once \mathcal{P}_i has collected 2t + 1valid shares, it can interpolate to reconstruct a polynomial $\phi(X, \omega_0)$ of degree 2t, and subsequently retrieve the secret $\phi(\omega_0, \omega_0)$.

Exemplary instantiation of a (batch) HAVSS protocol. Here we present an exemplary instantiation using our generic framework to compile a state-ofthe-art batch AVSS protocol hbACSS [41] into a batch HAVSS protocol.

We begin with hbACSS1 using KZG PCS, and apply a small modification of replacing Merkle tree in its asynchronous verifiable information dispersal component with vector commitment scheme [16], such that the resulting batch AVSS can share n^2 secrets using $\mathcal{O}(\lambda n^3)$ bits, thus realizing an amortized $\mathcal{O}(\lambda n)$ communication overhead per secret. ¹² We then prepare the interface of verifiable secret transferability in hbACSS as follows:

- For Context, every party can simply use KZG polynomial commitment, which is reliably broadcasted by the dealer and commits the *t*-degree polynomial encoding the secret *s* as zero-point evaluation.
- For Proof, once some party \mathcal{P}_i interpolates the *t*-degree polynomial encoding the secret *s*, it then computes it as an evaluation proof attesting that *s* is the zero-point evaluation of the polynomial committed to KZG commitment.

¹² Note that the batch size of n^2 used in hbACSS is caused by a specific communication wasting attack. If the particular attack is not admissible by adversaries, we can use a smaller batch $\mathcal{O}(n)$ to achieve an amortized $\mathcal{O}(\lambda n)$ communication per secret.

Compris HAVSS Compiler				
Generic HAVSS Compiler				
Sharing Phase:				
// For dealer \mathcal{D} who has an input secret s to be shared				
1: uniformly sample $\phi(X, Y)$ with degree 2t in X and t in Y s.t. $\phi(\omega_0, \omega_0) = s$ 2: for all $i \in [n]$ do				
3: $\beta_i \leftarrow \phi(\omega_i, Y)$				
4: invoke an AVSS instance to share polynomial β_i				
// For party \mathcal{P}_i				
5: upon all AVSS instances terminated, and get share set RShares \leftarrow				
$\{share_{ji}\}_{j\in[n]}$ and context set ContextSet \leftarrow $\{Context_j\}_{j\in[n]}$ do				
6: $\alpha_i \leftarrow \text{Interpolate}(\text{RShares})$				
7: if the degree of α_i doesn't exceed 2t then with the degree of α_i doesn't exceed 2t then				
8: multicast VOTE1()				
9: for all $j \in [n]$ do 10: send $share_{ji}$ to \mathcal{P}_j				
11: $ready \leftarrow false$, CShares $\leftarrow \emptyset$				
12: upon receiving $share_{ij}$ from \mathcal{P}_j do				
13: CShares \leftarrow CShares \cup share i_j				
14: if $ CShares \ge 2t + 1$ and $ready = false then$				
15: try to interpolate $\phi(\omega_i, Y)$ from CShares using OEC				
16: if OEC succeeds then				
17: let share _i := $\phi(\omega_i, \omega_0)$, ready \leftarrow true, extract Proof _i for share _i \triangleright The				
specific method for extracting the proof depends on AVSS.				
18: upon receiving VOTE1() from $n - t$ distinct parties do 19: multicast VOTE2() if haven't sent				
20: upon receiving VOTE2() from $t + 1$ distinct parties do 21: multicast VOTE2() if haven't sent				
22: upon receiving VOTE2() from $n - t$ distinct parties and get share; do				
22: upon receiving $\sqrt{OTE2}()$ from $n - i$ distinct parties and get share; do 23: output (share _i , ContextSet, Proof _i)				
Reconstruction Phase:				
// For party \mathcal{P}_i with (share _i , ContextSet, Proof _i) obtained from sharing phase				
101: shares $\leftarrow \emptyset$				
102: multicast SHARE(share _i , Proof _i)				
103: upon receiving SHARE(share _j , Proof _j) from \mathcal{P}_j do 104: if true \leftarrow SecretVerify(share _j , ContextSet.Context _j , Proof _j) then				
105: shares \leftarrow shares \cup share _i				
106: upon $ \text{shares} = 2t + 1$ do				
107: $\phi(X, \omega_0) \leftarrow \text{Interpolate}_{2t}(\text{shares})$				
108: output $\phi(\omega_0, \omega_0)$				

Fig. 7. Our generic framework of constructing 2t-degree HAVSS from t-degree AVSS.

Given the above modified hbACSS variant, we can instantiate a batch HAVSS protocol using it. When the dealer \mathcal{D} receives a batch of n secrets, it samples n random bivariate polynomials with asymmetric t and 2t degrees, and lets each

secret be encoded as one bivariate polynomial's zero-point evaluation. Since a bivariate polynomial can derive n t-degree univariate polynomials, the dealer thus can invoke a hbACSS1 instance to verifiably share all n^2 t-degree univariate polynomials derived from n bivariate polynomials. The participants can subsequently follow our HAVSS framework to share n input secrets, by explicitly dividing the n^2 t-degree univariate polynomials into n groups (where each group consists of nunivariate polynomials related to one bivariate polynomial and encodes an input secret to be shared).

Complexity analysis. The communication cost of our framework mainly comes from three parts: a batch of AVSS to share *t*-degree polynomials, OEC to obtain the 2*t*-degree secret shares, and a Bracha-style voting. The cost of the last two parts is at most $\mathcal{O}(\lambda n^2)$ per HAVSS input secret. When using some stateof-the-art batch AVSS [2, 22, 5, 41] to instantiate the sharing stage of *t*-degree polynomials, the communication complexity of sharing *n t*-degree polynomials becomes $\mathcal{O}(\lambda n^2)$, amortizedly. Further considering that *n t*-degree polynomials are required to be shared for each HAVSS input secret, the amortized communication overhead thus becomes $\mathcal{O}(\lambda n^2)$ per HAVSS input secret.¹³

Security analysis. The security of our HAVSS framework mostly inherits the security of the underlying *t*-degree AVSS, as our framework itself only adds a few degree checks and voting steps atop it (which are cryptographic-free). We provide the security intuition of this simple framework below.

- For Termination: As termination and completeness held by underlying AVSS, if some honest party outputs, it must have terminated in all AVSS instances. Thus, all honest parties will eventually output in all AVSS instances, further ensuring all honest parties have the correct 2t-degree secret share for s. The voting phase itself also ensures that once an honest party outputs, all others will do so. Thus, all honest parties will eventually output.
- For Correctness: With correctness satisfied by the underlying AVSS protocol, secrets of all underlying AVSS instances are consistent with $\phi(X, \omega_0)$. Thus, they will reconstruct the secret $\phi(\omega_0, \omega_0)$ identical to the dealer's input.
- For Completeness: Let $\phi'(X, Y)$ be the bivariate polynomial interpolated from all column polynomials shared by *n* underlying AVSS. The completeness of the underlying AVSS ensures that all column polynomials $\{\phi'(\omega_i, Y)\}_{i \in [n]}$ have degrees no more than *t*. Besides, the voting phase of our framework ensures that at least t + 1 row polynomials $(\phi'(X, \omega_i))$ have degrees no more than 2*t*. Thus, $\phi'(X, Y)$ has degree no more than 2*t* on *X*. Since all shares of *s* are evaluations of $\phi'(X, \omega_0)$, the completeness property therefore holds.
- For Secrecy: In our framework, the adversary will learn t row polynomials and t column polynomials from $\phi(X, Y)$. Upon t honest leaked their shares of s to the adversary, it can learn another t column polynomials. The secrecy

¹³ For example, if we use hbACSS [41] to instantiate our HAVSS framework, the resulting HAVSS can share n secrets by $\mathcal{O}(\lambda n^3)$ bits; if we use Bingo [2] as the underlying AVSS instantiation, our HAVSS framework can share 1 secret by $\mathcal{O}(\lambda n^2)$ bits.

of the underlying AVSS ensures that the adversary learns nothing about the rest n - 2t column polynomials except their own evaluations. In conclusion, for the bivariate polynomial $\phi(X, Y)$ with 2t degree on X and t degree on Y, giving t row polynomials and 2t column polynomials on it leaks nothing about the secret. Thus, the secrecy property holds.

References

- 1. Ittai Abraham, Gilad Asharov, Arpita Patra, and Gilad Stern. Asynchronous agreement on a core set in constant expected time and more efficient asynchronous vss and mpc. *Cryptology ePrint Archive*, 2023.
- Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In Annual International Cryptology Conference, pages 39–70. Springer, 2023.
- Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. In Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing, pages 363–373, 2021.
- Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, pages 337–346, 2019.
- Nicolas Alhaddad, Mayank Varia, and Ziling Yang. Haven++: Batched and packed dual-threshold asynchronous complete secret sharing with applications. Cryptology ePrint Archive, Paper 2024/326, 2024. URL: https://eprint.iacr.org/2024/326.
- Nicolas Alhaddad, Mayank Varia, and Haibin Zhang. High-threshold avss with optimal communication complexity. In *Financial Cryptography and Data Security:* 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II 25, pages 479–498. Springer, 2021.
- Shahla Atapoor, Karim Baghery, Daniele Cozzo, and Robi Pedersen. Vss from distributed zk proofs and applications. In *International Conference on the Theory* and Application of Cryptology and Information Security, pages 405–440. Springer, 2023.
- Akhil Bandarupalli, Adithya Bhat, Saurabh Bagchi, Aniket Kate, and Michael Reiter. Hashrand: Efficient asynchronous random beacon without threshold cryptographic setup. Cryptology ePrint Archive, 2023.
- Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In Advances in Cryptology-CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III 39, pages 701-732, 2019.
- 10. Gabriel Bracha. Asynchronous byzantine agreement protocols. Information and Computation, 75(2):130–143, 1987.
- Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of* the 9th ACM Conference on Computer and Communications Security, pages 88–97, 2002.
- 12. Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*, pages 524–541, 2001.
- Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography. In Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing, pages 123–132, 2000.
- Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In Proceedings of the twenty-fifth annual ACM symposium on Theory of computing, pages 42–51, 1993.

- Ignacio Cascudo and Bernardo David. Scrape: Scalable randomness attested by public entities. In International Conference on Applied Cryptography and Network Security, pages 537–556, 2017.
- Dario Catalano and Dario Fiore. Vector commitments and their applications. In Public-Key Cryptography-PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26-March 1, 2013. Proceedings 16, pages 55-72, 2013.
- Ashish Choudhury and Arpita Patra. An efficient framework for unconditionally secure multiparty computation. *IEEE Transactions on Information Theory*, 63(1):428–468, 2016.
- Ashish Choudhury and Arpita Patra. On the communication efficiency of statistically secure asynchronous mpc with optimal resilience. *Journal of Cryptology*, 36(2):13, 2023.
- Sourav Das, Sisi Duan, Shengqi Liu, Atsuki Momose, Ling Ren, and Victor Shoup. Asynchronous consensus without trusted setup or public-key cryptography. In Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security, 2024.
- Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. In 32nd USENIX Security Symposium (USENIX Security 23), pages 5359–5376, 2023.
- Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pages 2705–2721, 2021.
- 22. Sourav Das, Zhuolun Xiang, Alin Tomescu, Alexander Spiegelman, Benny Pinkas, and Ling Ren. Verifiable secret sharing simplified. *Cryptology ePrint Archive*, 2023.
- 23. Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In 2022 IEEE Symposium on Security and Privacy (SP), pages 2518–2534, 2022.
- Shlomi Dolev, Bingyong Guo, Jianyu Niu, and Ziyu Wang. Sodsbc: a post-quantum by design asynchronous blockchain framework. *IEEE Transactions on Dependable* and Secure Computing, 21(1):47–62, 2023.
- Sisi Duan, Xin Wang, and Haibin Zhang. Fin: practical signature-free asynchronous common subset in constant time. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 815–829, 2023.
- 26. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.
- 27. Pierre-Alain Fouque and Jacques Stern. One round threshold discrete-log key generation without private channels. In Public Key Cryptography: 4th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2001 Cheju Island, Korea, February 13–15, 2001 Proceedings 4, pages 300–316, 2001.
- Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Efficient asynchronous byzantine agreement without private setups. In 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), pages 246–257, 2022.
- Vipul Goyal, Chen-Da Liu-Zhang, and Yifan Song. Towards achieving asynchronous mpc with linear communication and optimal resilience. In Annual International Cryptology Conference, pages 170–206, 2024.
- 30. Jens Groth and Victor Shoup. Design and analysis of a distributed ecdsa signing service. *Cryptology ePrint Archive*, 2022.

- 24 Cheng et al.
- Jens Groth and Victor Shoup. Fast batched asynchronous distributed key generation. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 370–400, 2024.
- 32. Bingyong Guo, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Speeding dumbo: Pushing asynchronous bft closer to practice. *Cryptology ePrint Archive*, 2022.
- Xiaoyu Ji, Junru Li, and Yifan Song. Linear-communication asynchronous complete secret sharing with optimal resilience. In Annual International Cryptology Conference, pages 418–453. Springer, 2024.
- 34. Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16, pages 177–194, 2010.
- 35. Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pages 1751–1767, 2020.
- 36. Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In *Proceedings of the 2019* ACM SIGSAC Conference on Computer and Communications Security, pages 887– 903, 2019.
- 37. Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *Proceedings* of the 39th symposium on principles of distributed computing, pages 129–138, 2020.
- Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Annual international cryptology conference, pages 129–140. Springer, 1991.
- Victor Shoup. Practical threshold signatures. In Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19, pages 207–220, 2000.
- 40. Victor Shoup and Nigel P Smart. Lightweight asynchronous verifiable secret sharing with optimal resilience. *Journal of Cryptology*, 37(3):27, 2024.
- Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew Miller. hbacss: How to robustly share many secrets. In Proceedings of the Network and Distributed System Security Symposium (NDSS) 2022, 2022.

A Security Proof of Our Lightweight HAVSS

The security of our HAVSS protocol is captured by the following main theorem.

Theorem 1. Our HAVSS protocol satisfies correctness, secrecy, completeness, and termination properties against any PPT adversary corrupting up to t < n/3parties as aforementioned, assuming the random oracle model and the existence of pair-wise asynchronous secure channel between any two honest parties.

We validate the statement by proving the following lemmas.

Lemma 1. In our HAVSS protocol, once t + 1 parties received valid SEND messages, let $\phi(X, Y)$ denote the bivariate polynomial interpolated by their row polynomials. The probability that there exists another party, whose row polynomial passes the check but doesn't lie on ϕ , is negligible, under the assumption that $2^n/|\mathbb{Z}_N|$ is negligible¹⁴.

Proof. The formal proof of this lemma can be seen in [40]. We only show intuition here. It's clear that this lemma holds when the dealer is honest. So, we only consider the case with a corrupted dealer. Assuming the corrupted dealer can break this lemma. It means that there exists some column polynomials whose degrees are higher than t. We consider the probability of obtaining a polynomial of degree less than t from randomly linearly combing n t-degree polynomials. If all the coefficients for liner combine are uniformly and independently randomly sampled from \mathbb{Z}_N , the probability is $1/\mathbb{Z}_N$. The factor 2^n is because the corrupted dealer can decide which subset (with size no more than t+1) will have valid row polynomials.

Lemma 2. Our HAVSS protocol satisfies the termination property.

Proof. We first discuss the case that an honest party outputs in Share stage. As shown in Algorithm 4 and 5, there are two ways to output in Share stage. If it outputs OUTPUT1, at least t+1 honest party must have sent VOTE messages. Since honest parties will forward VOTE messages upon receiving t + 1 ones, all honest parties will eventually receive n-t VOTE messages. We then show that all honest parties will receive enough valid ECHO messages in this case. Since honest parties can receive n-t VOTE messages eventually, there must be at least one honest party who has received at least 2t + 1 valid ECHO messages in which at least t+1 from honest parties. Thus, all honest parties will eventually receive at least t + 1 valid ECHO messages. Depending on whether the column polynomials they interpolate are valid, all honest parties will either output OUTPUT1 or multicast a COMPLAIN message. For those honest parties who multicast a COMPLAIN message, it is clear that their COMPLAIN messages are valid for all honest parties. It has been reasoned that there are at least t + 1 honest parties who have valid row polynomials. Thus, there will be at least assisRBC instances lead by these t+1 honest parties, which will ensure that all honest parties eventually obtain at least t + 1 valid row polynomials. All honest parties thus can interpolate a bivariate polynomial through them. The evaluation on (ω_0, ω_0) of it is the secret. They can then output OUTPUT2. Thus, once an honest party outputs in Share stage, all honest parties will eventually output either OUTPUT1, OUTPUT2, or both in the Share stage.

We then consider the case that all honest parties participate in Reconstruction stage. They must have output in Share stage. Because the honest parties may have two possible outputs during Share stage, we further divide the potential scenarios into two categories. Firstly, if all honest parties output OUTPUT1,

¹⁴ To let $2^n/|\mathbb{Z}_N|$ negligible, $|\mathbb{Z}_N|$ typically could be 2^{2n} , indicating that a statistically security parameter *m* (i.e. the bit length of \mathbb{Z}_N elements) approximates 2n.

they will all multicast their column polynomials, and they can thus interpolate a bivariate polynomial through them. They then output the secret which is the evaluation on (ω_0, ω_0) of this polynomial. Secondly, some honest parties only output OUTPUT2. As we reasoned in the last case, all honest parties will output OUTPUT2 in Share stage which contains the final secret, and thus they will all output in Reconstruction stage.

Lemma 3. Our HAVSS protocol satisfies the correctness property.

Proof. We first consider the case that the dealer \mathcal{D} is honest, who samples a bivariate polynomial ϕ in which $s = \phi(\omega_0, \omega_0)$. Following the protocol, all honest parties will obtain valid SEND messages, and thus will multicast valid ECHO messages. They can gather enough ECHO messages from honest parties to interpolate their valid column polynomials. If not, there must be an ECHO message that contains wrong evaluations but passes the check. This implies that the adversary finds a collision for the hash function. More clearly, we use the root of a Merkle tree to commit the evaluations. If the adversary can forge a valid ECHO message, it implies that it finds a collision during the Merkle path, which violates the assumption of collision resistance property of hash function \mathcal{H} . All honest parties thus can be guaranteed to gather 2t + 1 valid ECHO messages to interpolate their column polynomials, and thus multicast VOTE messages, which also means that they can gather at least 2t + 1 VOTE messages then. At this point, all honest parties meet the condition to output OUTPUT1.

Immediately following the above analysis, honest parties are guaranteed to output OUTPUT1. And they thus will transmit valid SHARE messages, which ensures all honest parties gather enough shares to reconstruct the secret once they all participate in Reconstruction stage. Similar to the above, we shall prove that the adversary can't forge a valid SHARE message. The proof is also similar, as the forge of a SHARE message also breaks the assumption of collision resistance property of hash function \mathcal{H} . Thus, they will all output the same secret to the dealer.

Finally, we prove that honest parties who output in Reconstruction stage always output equally. As the above lemma showed, once an honest party outputs the secret, so will the others. Since our protocol has two methods to deliver the secret, we categorize the possible scenarios into three types: these two honest parties output all by gathering enough SHARE messages, all by complaint phase, or both. We proof that separately.

- For the first case, if they output differently, the polynomial $\phi \mathcal{D}$ committed must have a degree higher than 2t on X, as the adversary can't forge a SHARE message. Recalling the Share stage, t + 1 honest parties must have received valid row polynomials if any honest party output in Reconstruction stage. Let ϕ' be the bivariate polynomial we interpolate from these t + 1row polynomials (we actually can obtain two bivariate polynomials, but we here only consider the one with the secret on it). It's fairly easy to see that ϕ' has the right degrees on X and Y. Thus, there must be an index k, such that $\phi(\omega_k, Y)$ don't match with $\phi'(\omega_k, Y)$. There are two possible scenarios here: first, $\phi(\omega_k, Y)$ and $\phi(\omega_k, Y)$ are different but with the same degree; second, $\phi(\omega_k, Y)$ has a higher degree. For the first scenario, it claims that the adversary finds some collisions and opens the commitment to two different polynomials, which is against our assumptions. For the second, it can't be accepted by honest parties as they will check its degree first. Thus, there can't be a column polynomial mismatching with ϕ' , but accepted by honest parties who output in this way will always output the same.

- For the second case, they both output by complaint phase. Assuming that in this case, two honest parties \mathcal{P}_i and \mathcal{P}_j have different outputs. As shown in Algorithm 5, they compute the secret by interpolating a bivariate polynomial from at least t + 1 valid outputs of assisRBCs (similarly to the above, we only consider the polynomial with the secret on it). Using ϕ' to denote the bivariate polynomial of \mathcal{P}_i , and ϕ'' for \mathcal{P}_j . As the reliable broadcast protocol ensures that all honest parties output equally, once ϕ' and ϕ'' are different, there must be an assisRBC instance whose output lies on ϕ'' , but not ϕ' . It means that the row polynomials contained in this output can pass *degree check* but are inconsistent with other t + 1 ones, which are against lemma 1. Thus, honest parties who output in this way will always output the same.
- For the third case, they output in different ways. Following our protocol, once some honest parties are outputting the secret, there must be at least t + 1 honest parties who have received valid SEND messages. Let ϕ' be the bivariate polynomial interpolated from row polynomials contained in these SEND messages. As we analyzed in the first two cases, regardless of how honest nodes output, their outputs are consistent with ϕ' .

Lemma 4. Our HAVSS protocol satisfies the completeness property.

Proof. The termination property of our protocol guarantees that once an honest party outputs in Share stage, so will all other honest parties. Meanwhile, there must be at least t+1 honest parties who have received valid SEND messages. Let ϕ' be the bivariate polynomial interpolated from row polynomials contained in these SEND messages. We have proved in lemma 3 that any column polynomial contained in a valid SHARE message is consistent with ϕ' . Thus, for honest parties who output OUTPUT1, their shares are consistent with ϕ' . We have also proved in lemma 3 that all honest parties who output OUTPUT2 will interpolate the same polynomial as ϕ' . So, once an honest party outputs in Share stage, so will others. And all honest parties' shares are consistent with ϕ' .

Lemma 5. Our HAVSS protocol satisfies the secrecy property.

We prove the secrecy of our HAVSS protocol by showing that the adversary's views are indistinguishable in the real-world execution and the ideal-world execution (where the ideal-world simulator illustrated as Fig. 8 takes as input only admissible leakage of the bivariate polynomial and thus no bit of secret can be learned by the adversary there). The indistinguishability of the two worlds can be seen through the following sequence of games.

Input: n, t, $\mathcal{C} \subset [n]$ with $|\mathcal{C}| = t$ to denote the set of corrupted parties, $\mathcal{H}_{\mathcal{C}} \subset \mathcal{H}_{\mathcal{C}}$ $[n] \setminus \mathcal{C}$ with $|\mathcal{H}_C| = t$ to denote the set of honest parties leaking their column polynomials to the adversary, $\{(\phi(X,\omega_i)\}_{i\in\mathcal{C}}, \text{ and } \{(\phi(\omega_i,Y)\}_{i\in\mathcal{C}_0} \text{ where } \mathcal{C}_0 =$ $\mathcal{C} \cup \mathcal{H}_C$.

- 1. Uniformly sample a new secret s' and calculate polynomial $\phi'(X,Y) \in$ $\mathbb{Z}_F[X,Y]_{2t,t}$, s.t.:
 - For each $i \in \mathcal{C}$, $\alpha_i' = \phi'(X, \omega_i) = \phi(X, \omega_i) = \alpha_i;$ For each $i \in \mathcal{C}_0$, $\beta_i' = \phi'(\omega_i, Y) = \phi(\omega_i, Y) = \beta_i.$

 - $-\phi'(\omega_0,\omega_0)=s'$
- 2. Uniformly sample $\widehat{\phi}'(X,Y)$ from $\mathbb{Z}_F[X,Y]_{n,t}$.
- 3. For each $i \in [n]$, $\mathsf{mt}_i \leftarrow \mathsf{MerkleBuild}(\{(\beta_i(\omega_j), \widehat{\beta}_i(\omega_j))\}_{j \in [n]})$. Compute $\mathsf{mt} \leftarrow \mathsf{MerkleBuild}(\mathsf{mt}_1.\mathsf{root},...,\mathsf{mt}_n.\mathsf{root}).$ Compute challenge $(\theta_1'^{(i)})_{i \in [n]} \leftarrow$ $\mathcal{H}(\mathsf{mt.root}).$ Compute $g' \leftarrow \widehat{\beta}'_{0} + \sum_{i \in [n]} \theta'^{(i)}_{1} \beta_{i}'.$
- 4. Simulate the dealer \mathcal{D} by (i) broadcasting $\{\mathsf{mt}_i : \mathsf{root}\}_{i \in [n]}$ and g', and (ii) sending $\phi'(X, \omega_i)$, $\hat{\phi}'(X, \omega_i)$ and corresponding Merkle proofs to each \mathcal{P}_i .
- 5. Simulate the rest transcripts of honest parties sent in HAVSS by following the protocol.

Fig. 8. Simulator for the secrecy of our HAVSS protocol.

- Game 0. Real-world execution.
- Game 1. Same as Game 0, except that all honest parties are simulated and honestly follow the protocol specification. Let $\phi(X, Y)$ and $\phi(X, Y)$ denote the bivariate polynomials sampled by the simulated dealer during the execution, where $\phi(\omega_0, \omega_0)$ encodes the input secret. Game 0 and Game 1 are indistinguishable, as the simulated honest parties in Game 1 would send messages having a distribution identical to those messages sent by the honest parties in Game 0.
- Game 2. Same as Game 1, except that the simulator executes the steps illustrated in Fig.8 to simulate the behaviors of honest parties to interact with the adversary. Essentially, it replaces ϕ with ϕ' to encode another randomness (likely) different from the dealer's genuine input secret. Game 2 and Game 1 are computationally indistinguishable because:
 - Since ϕ' and $\hat{\phi}'$ all have correct degrees and the simulator honestly follows the protocol to simulate the execution of the dealer and honest parties in the rest, all messages sent by the simulator can pass the adversary's verification.
 - Let $\mathcal{H}_R = [n]/\mathcal{C}_0$ denote the set of honest parties who do not leak their secret shares to the adversary. For each $i \in \mathcal{H}_R$ and $j \in [n]/\mathcal{C}$, $\mathcal{H}(\beta_i(\omega_j), \widehat{\beta}_i(\omega_j) \text{ and } \mathcal{H}(\beta_i'(\omega_j), \widehat{\beta}_i'(\omega_j) \text{ are indistinguishable for the hid-}$ ing of commitment scheme (used for compute Merkle tree leaves), if we model the hash function as a random oracle and $\hat{\beta}_i(\omega_i)$ and $\hat{\beta}'_i(\omega_i)$ are randomly sampled.

• For each $i \in \mathcal{H}_R$, the adversary only holds t evaluations of both $\widehat{\beta}_i$ and $\widehat{\beta}'_i$ both with a degree of t. Thus, the adversary learns nothing about $\widehat{\beta}_i(\omega_j)$ and $\widehat{\beta}'_i(\omega_j)$, for each $i \in \mathcal{H}_R$ and $j \in [n]/\mathcal{C}$. For each $i \in [n]/\mathcal{C}$, the adversary only holds 2t evaluations of both $\widehat{\alpha}_i$

For each $i \in [n]/\mathcal{C}$, the adversary only holds 2t evaluations of both $\hat{\alpha}_i$ and $\hat{\alpha}'_i$. Because the dealer has public challenge polynomials g in Game 1 and g' in Game 2, for each $\hat{\alpha}_i(\omega_0)$ and $\hat{\alpha}'_i(\omega_0)$, we have:

*
$$\widehat{\alpha}_i(\omega_0) = g(\omega_i) - \sum_{k \in [n]} \theta_1^{(k)} \alpha_i(\omega_k)$$

*
$$\widehat{\alpha}'_i(\omega_0) = g'(\omega_i) - \sum_{k \in [n]} \theta'^{(k)}_1 \alpha_i'(\omega_k)$$

So, if the secret of HAVSS has low entropy, the distribution for $\widehat{\alpha}_i(\omega_0)$ and $\widehat{\alpha}'_i(\omega_0)$ are concentrated, indicating low entropy. However, even if we let the adversary learn the exact values of $\widehat{\alpha}_i(\omega_0)$ and $\widehat{\alpha}'_i(\omega_0)$, nothing leaked for $\widehat{\alpha}_i(\omega_j)$ and $\widehat{\alpha}'_i(\omega_j)$ for each $j \in \mathcal{H}_R$, because $\widehat{\alpha}_i$ and $\widehat{\alpha}'_i$ are all *n*-degree polynomials.

Thus, for both ϕ and ϕ' , the adversary doesn't have enough evaluations to reconstruct the secrets s and s'.