Asynchronous Verifiable Secret Sharing with Elastic Thresholds and Distributed Key Generation

Junming Li
Huazhong University of Science and Technology

Zhi Lu
Huazhong University of Science and Technology

Renfei Shen
Huazhong University of Science and Technology

Yuanqing Feng
Huazhong University of Science and Technology

Songfeng Lu
Huazhong University of Science and Technology

Abstract

Distributed Key Generation (DKG) is a technique that enables the generation of threshold cryptography keys among a set of mutually untrusting nodes. DKG generates keys for a range of decentralized applications such as threshold signatures, multiparty computation, and Byzantine consensus. Over the past five years, research on DKG has focused on optimizing network communication protocols to improve overall system efficiency by reducing communication complexity. However, SOTA asynchronous distributed key generation (ADKG) schemes (e.g., Kokoris-Kogias ADKG, CCS 2020 and Das ADKG, S&P 2022, and others) only support recovery thresholds of either f or $2f$, where f is the maximum number of malicious nodes. This paper proposes an asynchronous verifiable secret sharing protocol featuring an elastic threshold, where $t \in [f, n-f-1]$ and $n \geq 3f+1$ is the total number of parties. Our protocol enables a dealer to share up to $t+1$ secrets with a total communication cost of $O(\lambda n^3)$, where λ is the security parameter, and the protocol relies on the hardness of the q-SDH problem. We further modified the Schnorr protocol to enable simultaneous commitments to multiple secrets, which we refer to m-Schnorr.

1 Introduction

Distributed Key Generation (DKG) is a technique for generating public/private key pairs among a group of mutually untrusting nodes, where each parties node ultimately holds a share of the private key and the public key. The private key generated by DKG can be used in threshold cryptographic applications, including threshold encryption[14, 26, 8, 23], threshold signatures[5, 19] and multiparty computation[31].

The core of DKG lies in the secret sharing scheme, a fundamental cryptographic primitive. Secret sharing is a technique in which a party distributes a secret among a group of multiple nodes. The steps involved include secret division, sharing, verification, and reconstruction. According to the most basic definition, secret sharing is performed by an honest dealer who shares a secret among a group of parties. When the number of secret shares exceeds a certain threshold, the original secret can be reconstructed. This is known as the reconstruction threshold. When the reconstruction threshold is equal to f, it is defined as a low threshold. In contrast, when the reconstruction threshold is significantly higher than $f+1$, it is defined as a high threshold.

Research on DKG can be classified based on the network model and the threshold level. In synchronous network environments, DKG has been the subject of extensive study[32, 19, 9, 29, 23, 38, 27, 2, 23, 15, 22, 13, 11, 1]. In 2019, the first ADKG scheme was proposed[27]. Since then, research on DKG has predominantly focused on asynchronous environments[27, 2, 23, 15, 22, 13, 11, 1]. Unlike synchronous networks, asynchronous environments do not require additional assumptions about message delay and reception order between nodes. To enhance robustness, ADKG schemes are designed to be more complex. Compared to low-threshold schemes, high-threshold require more secret shares to reconstruct the original secret, offering better security properties. However, high-threshold ADKG consumes more computational and communication resources. These factors result in high-threshold ADKG being substantially less efficient than synchronous DKG and low-threshold ADKG. Therefore, this paper primarily studies high-threshold *asynchronous* DKG.

1.1 Our results

Elastic thresholds. We design an efficient asynchronous verifiable secret sharing (AVSS) protocol with an elastic threshold. In an asynchronous network of $n \geq 3f+1$ nodes, our protocol allows a dealer to share $m \in [1, t+1]$ secrets with a total communication complexity of $O(\lambda n^2)$, where n is the total number of parties, f is the maximum number of malicious parties, and reconstruction threshold $t \in [f, n-f-1]$. In Table 1 and 2, we compare our scheme with other AVSS schemes. Our scheme supports a greater range of thresholds while maintaining good efficiency. Upon successful completion of our protocol, each party receives a set of threshold secret shares

Table 1: Comparison of AVSS schemes. Common Reference String (CRS) is a set of pre-generated public parameters that all parties trust for ensuring the security and correctness of constructing and verifying zero-knowledge proofs.

Scheme	Word complexity	CRS setup	Threshold	Assumptions
Cachin[8]	$O(\lambda n^3)$	-	$2f$	DL
Backes[4]	$O(\lambda n^2)$	✓	f	$q-$SDH, $q-polyDH$
Haven[3]	$O(\lambda n^2 \log n)$	-	$2f$	DL, ROM
hbACSS[40]	$O(\lambda n^3)$	✓	f	$q-$SDH
Bingo[1]	$O(\lambda n^2)$	✓	$2f$	$q-$SDH, AGM[18]
This work	$O(\lambda n^2)$	✓	$[f, n-f-1]$	$q-$SDH, ROM

Table 2: A comparison of ADKG schemes. All of these schemes can toleratre $f < n/3$ malicious nodes. Abbreviations used are, Discrete Logarithm (DL), Symmetric External Diffie-Hellman (SXDH), Bilinear Diffie-Hellman (BDH), Decisional Diffie-Hellman (DDH), q-Strong Diffie-Hellman (q-SDH), and Random Oracle Model (ROM).

Scheme	Word complexity	Rounds	CRS setup	Threshold	Key as field element	Assumptions
Kokoris-Kogias[27]	$O(\lambda n^4)$	$O(n)$	-	$2f$	✓	DL
Abraham[2]	$O(\lambda n^3)$	$O(1)$	-	f	×	SXDH, BDH, ROM
Das[13]	$O(\lambda n^3)$	$O(\log n)$	-	$2f$	✓	DDH, ROM
Groth[22]	$O(\lambda n^3)$	$O(1)$	-	f	×	DL, ROM
Das[11]	$O(\lambda n^3)$	$O(\log n)$	-	$2f$	✓	DL ROM
Bingo[1]	$O(\lambda n^3)$	$O(1)$	✓	$2f$	✓	$q-$SDH
This work	$O(\lambda n^3)$	$O(1)$	✓	$[f, n-f-1]$	✓	$q-$SDH, ROM

derived from these secrets. When applied to ADKG, each node can receive a set of threshold secret shares of randomly chosen secrets $z \in Z_q$, where Z_q is a field of order q. Therefore, our ADKG scheme can be integrated with off-the-shelf threshold cryptosystems[5, 14, 19, 23, 39, 27].

Faster and more secure secret sharing. We select two asymmetric bivariate polynomials, Φ and $\hat{\Phi}$, as the secret-sharing polynomials. Φ is used to hide the secret, while $\hat{\Phi}$ is a completely randomly generated polynomial. $\hat{\Phi}$ is introduced to enhance polynomial binding, as discussed in 3.3. The use of bivariate polynomials accelerates secret sharing and enhances its robustness in asynchronous networks. In the verifiable part of the protocol, we built upon the work of Abraham et al.[1] and incorporated *HashToCurve*[24] to modify the KZG commitment[25].

Secure and space-efficient m-Schnorr protocol. In order to generate m public key in a single ADKG, we design a non-interactive Schnorr protocol[34, 30, 28, 36] to simultaneously commit to m public key shares. The size of the commitment is independent of m, and our Schnorr commitment incurs a space overhead superior to that of invoking the Schnorr protocol m times separately, amounting to $1/m$ of the latter. This protocol is based on Discrete Logarithm Assumption and Random Oracle Model. Additionally, we reuse the proof generated by the KZG commitment to append further security properties to the public key shares.

Implementation andevaluation. We implemented our AVSS and m-Schnorr protocols using Rust and bls12381 elliptic curve. We evaluated our scheme with up to 256 nodes. The experimental results can be found in Tables 4, 5, and 6.

2 Related Work

AVSS. Our work builds upon asynchronous verifiable secret sharing (AVSS). Cachin et al.[8] proposed a AVSS protocol with a communication complexity of $O(\lambda n^3)$, based on DL assumption. Backes et al.[4] improved upon this by introducing an AVSS protocol with $O(\lambda n^2)$ communication complexity, though its recovery threshold is limited to f. Al-Haddad et al.[3] introduced Haven protocol, which achieves $O(\lambda n^2 \log n)$ communication complexity. Notably, Haven is the first high-threshold secret sharing scheme that does not rely on a PKI or trusted setup. Yurek et al.[40] proposed three variants of hbACSS, which achieve $O(\lambda n^3)$ communication complexity without needing a trusted setup, yet these protocols remain classified as low-threshold secret sharing schemes. Abraham et al.[1] introduced Bingo protocol, which has a total communication complexity of $O(\lambda n^2)$. Bingo is the first high-threshold AVSS protocol that is adaptively secure and can simultaneously share up to $f+1$ secrets, but it requires both a PKI and a trusted setup.

Numerous works have examined th eproblem of Distributed Key Generation with various cryptographic assump-

tions, network conditions, secret sharing and with other properties[32, 19, 9, 29, 23, 38, 27, 2, 23, 15, 22, 13, 11, 1]. We categorize prior work into two classes based on network assumptions: *Synchronous* and *Asynchronous*.

Synchronous DKG. The synchronous network assumes relatively ideal conditions where all nodes have equal performance and resources, and messages are successfully delivered within an ideal time frame. Synchronous DKG has been studied for decades. Pedersen[32] proposed the first DKG scheme based on verifiable secret sharing (VSS)[16]. Gennaro et al.[19] demonstrated that an attacker could bias the public key generated by this protocol and proposed a solution, though computationally expensive. Canetti et al.[9] and Neji et al.[29] improved Gennaro's method by introducing adaptive security and increasing computational efficiency. Gurkan et al.[23] designed a DKG protocol based on publicly verifiable secret sharing (PVSS), with a communication complexity of $O(n^3 \log n)$. However, the generated keys belong to group elements, which is a limitation. Shrestha et al.[38] proposed a protocol with $O(n^3)$ communication complexity that does not rely on broadcast channels.

Asynchronous DKG. The asynchronous network is closer to real-world network conditions, where issues such as varying computational power across nodes and potential message loss or delivery failures due to various factors must be considered. Kokoris et al.[27] designed the first ADKG protocol, which has a communication complexity of $O(\lambda n^4)$ and a round complexity of $O(n)$. Abraham et al.[2] devised a consensus protocol and combined it with Gurkan et al.'s PVSS protocol[23] to construct a high-threshold ADKG with a communication complexity of $O(n^3 \log n)$. However, this protocol inherits the drawback of generating keys that belong to group elements, making it incompatible with widely-used threshold cryptography schemes[15] based on field elements. Groth and Shoup[22] proposed a DKG protocol that does not require a trusted setup and has a communication complexity of $O(n^3)$. Das et al.[13] developed a high-threshold DKG with a communication complexity of $O(\lambda n^3)$, and the generated keys belong to field elements, based on the DDH assumption. More recently, Das et al.[11] introduced an improved scheme based on DL assumption, which offers higher security than the DDH assumption. They also enhanced the efficiency of the high-threshold ADKG, making it nearly as efficient as low-threshold ADKG. Recently, Abraham et al.[1] proposed an ADKG protocol that achieves adaptive security, with a communication complexity of $O(\lambda n^3)$ and a round complexity of $O(1)$. Their AVSS protocol is also capable of simultaneously sharing $f+1$ secrets.

3 Preliminaries

In this section, we first define the basic notation, followed by the definitions of Reliable Broadcast (RBC)[12, 7], Polynomial Commitment Scheme (PCS)[25, 33], Asynchronous Verifiable Secret Sharing[8, 4, 3, 40, 1], Validated Asynchronous Byzantine Agreement[1], Schnorr Protocol[36, 37] and Fiat-Shamir Heuristic[17].

3.1 Notations

We summarize the notations in Table 3. To simplify the notation, if a **variable** is written in boldface, it indicates that the variable is actually a set. λ is the security parameter. For example, λ can denote the output size of a hash function. For a finite set $\mathbb{S}$, $|\mathbb{S}|$ denotes its size. The notation $x \xleftarrow{\$} \mathbb{S}$ denotes sampling a random element from the set $\mathbb{S}$ and assigning it to x. For any $n \in \mathbb{N}$, we define $[n] = \{1, \ldots, n\}$ and $[\![n]\!] = \{0, \ldots, n\}$. Specifically, for two integers $i < j$, we define the ordered set $[i, j] = \{i, \ldots, j\}$. Probabilistic Polynomial Time denoted by PPT. In constructing a polynomial, the randomly selected coefficients are typically chosen from $\mathbb{Z}_q$ by default.

Table 3: Notations used in the paper

Notation	Description
n	Total number of nodes, where $n \geq 3f+1$
f	Maximum number of malicious nodes
t	Reconstruction threhold of secret sharing
$\mathbb{Z}_q$	Field of order q where q is prime
$\mathbb{G}$	Cyclic group of order q
$g, \hat{g}, h$	Random and independent generators of $\mathbb{G}$
z, g^z	ADKG secret and public key
m	Total number of secret, $m \in [1, t+1]$
$\Phi(X, Y)$	Bivariate asymmetric polynomial
$z_k(i)$	k-th secret shared by the i-th node
$z(i)$	Set of secrets shared by the i-th node
$g^{z_k(i)}, g^{z(i)}$	i-th node's threshold public keys
pk_i, sk_i	Communication key pair of the i-th node
Com	NIZK commitment
$\mathcal{A}$	PPT adversaries

3.2 Reliable Broadcast

Reliable Broadcast is a communication primitive guaranteeing, intuitively, that all nodes in a distributed system deliver the same set of messages. The RBC[12, 7] ensures reliable data transmission if more than two-thirds of the nodes are honest. A reliable broadcast has the following properties:

- **Validity.** If the sender is nonfaulty, then every nonfaulty party that completes the protocol outputs the sender's input value v.
- **Agreement.** If any honest node receives a message m, then all honest nodes will eventually receive m.
- **Termination.** If the dealer is nonfaulty, then all nonfaulty parties complete the protocol and output a value. Further-

more, if some nonfaulty party completes the protocol, every nonfaulty party completes the protocol.

3.3 Polynomial Commitment Scheme

A PCS[25, 33] is a cryptographic protocol that allows a party to commit to a polynomial while keeping it hidden and later reveal and prove evaluations of the polynomial at specific points without revealing the polynomial itself. This is particularly useful in various cryptographic applications[21, 20], including zero-knowledge proofs, verifiable computation, and blockchain systems.

Common polynomial commitments include Pedersen commitments[33], KZG commitments[25], and others. A PCS consist of the following algorithms:

- CRS $\leftarrow$ Setup(1^λ) takes a security parameter λ as input and outputs a set of global parameters CRS.
- Com $\leftarrow$ Commit(CRS,$\alpha(x)$) takes CRS and a polynomial α as input and computes the commitment Com.
- $p,\pi \leftarrow$ Eval(CRS,$\alpha(X),x$) takes CRS, a polynomial α, and a variable x as input. The output is $p = \alpha(x)$ along with a proof at $(x,\alpha(x))$.
- True/False $\leftarrow$ Verify(CRS,Com,(x,p),π) takes CRS, Com, (x,p), and a proof π as input. If (x,p) is a point on the polynomial, the verification passes and returns True; otherwise, it returns False.

A polynomial commitment scheme satisfies the following security definitions:

Definition 1 (Binding)[25]. The Binding property encompasses both *polynomial binding* and *evaluation binding*. Polynomial binding means that for a polynomial commitment Com, a PPT adversary can only use a specific polynomial to succeed in verification. Evaluation binding means that for two different points (a,p) and (a,p'), consider a game $G^{Binding}(\lambda)$ in which an adversary $\mathcal{A}$ takes λ as inputs and outputs the (Φ,Φ'). The probability that their proofs π and π' both satisfy the Verify function is negligible.

Definition 2 (Hiding). An adversary $\mathcal{A}$ cannot obtain any information about the original polynomial solely from Com.

Definition 3 (Correctness). The output of Verify is True if and only if all inputs are computed honestly.

3.4 Asynchronous Verifiable Secret Sharing

Asynchronous secret sharing (ASS) protocol consists of two phases: Sharing and Reconstruction. During the sharing phase, the dealer L shares the secret s using Share. During the reconstruction phase, parties use Reconstruct to recover the original secret. ASS is a f-resilient if the following properties hold with probability 1-$\mathcal{V}(\lambda)$ against any $\mathcal{A}$ that corrupts up to f nodes, $\mathcal{V}(\cdot)$ is a negligible function takes λ as input. AVSS has the following properties:

- **Correctness.** If the dealer is honest, then after the sharing phase, all honest nodes will obtain the correct secret shares.
- **Secrecy.** If the dealer is honest, then for any $\mathcal{A}$ controlling at most f nodes, there exists a PPT simulator $\mathcal{S}$ such that the output of $\mathcal{S}$ and the view of $\mathcal{A}$ in the actual protocol are indistinguishable.
- **Agreement.** In the reconstruction phase, if all shares held by the parties are valid, then all parties will be able to reconstruct the same original secret.

AVSS is built upon ASS by adding verifiability, ensuring that all parties can verify the authenticity of the secret shares. To achieve agreement property, we apply KZG commitments[25] to verify the shares, preventing erroneous shares from affecting the correct recovery of the secret. By leveraging binding and hiding properties mentioned in Section 3.3, we ensure that the polynomial is verified without revealing the original values of the shares.

3.5 Validated Asynchronous Byzantine Agreement

Validated asynchronous Byzantine agreement (VABA)[1] is an asynchronous agreement protocol that ensures honest nodes in the system can still reach consensus even if some nodes are malicious. By running this protocol, ADKG can elect $f+1$ parties to use their secret shares to compute the public and private keys. The VABA protocol, which has been proven to be *adaptively secure*[1].

- **Correctness.** All nonfaulty parties that complete the protocol output the same value.
- **Validity.** If a nonfaulty party outputs a value, then it is externally valid.
- **p-Quality.** With probability p or greater, all nonfaulty parties output the value x_i for some nonfaulty i.
- **Termination.** All nonfaulty parties output a value and complete the protocol.

3.6 Schnorr Protocol and Fiat-Shamir

The Schnorr protocol[36, 37] is a specialized zero-knowledge proof system that allows a prover to demonstrate knowledge of a discrete logarithm without revealing the secret itself. It is commonly used in identification schemes where the prover convinces the verifier that they know a secret x such that $h = g^x$, where g is a generator of a cyclic group of prime order q, and h is the public value.

The original Schnorr protocol operates interactively. It consists of three main steps: the prover first commits to a random value, the verifier generates a random challenge, and the prover responds by combining the challenge with the original

secret. This structure ensures that the prover cannot cheat and convince the verifier without knowing the secret. However, this interaction can be communication-heavy, especially in distributed or asynchronous networks.

The Fiat-Shamir Heuristic[17] provides a transformation that converts such an interactive protocol into a Non-Interactive Zero-Knowledge Proof (NIZK) by replacing the verifier's challenge with the output of a cryptographic hash function applied to the prover's commitment. This transformation significantly reduces communication complexity, enabling the prover to generate a proof independently, which the verifier can check later. The prover computes the challenge as $c = H(g,h,u)$, where u is the prover's commitment. This heuristic preserves the zero-knowledge properties of the original protocol under the ROM.

4 Design

In this section, we will describe our secret sharing polynomial, adaptation of KZG commitments, secret sharing, secret reconstruction, and secret share recovery.

Our secret sharing polynomial uses bivariate asymmetric polynomials, while KZG commitments only support univariate polynomials. Therefore, we have made adjustments based on Bingo[1] to address this limitation. During the sharing phase, we leveraged our polynomial to additionally accelerate the process, and once the sharing phase is completed, the reconstruction phases can be executed.

4.1 Secret Sharing Polynomial

Inspired by the work of Bingo[1], our AVSS scheme uses a bivariate asymmetric polynomial, where one dimension is used to hide secret and the other to accelerate sharing process. The main advantages are as follows: (1) Enhanced Security: Assuming $\Phi(X,Y)$ has degree d_1 in X and d_2 in Y respectively, recovering the original polynomial requires $(d_1+1)\times(d_2+1)$ valid secret shares. This significantly increases the difficulty of the attack compared to a univariate polynomial. (2) Accelerated Sharing Process: In asynchronous network environments, the correct delivery of shared secrets by the dealer is not guaranteed. Therefore, by utilizing the dimension Y of the bivariate polynomial, nodes that have successfully obtained their shares can participate in the secret sharing phase. They can use their own share polynomials to compute the shares for other nodes. Once a node collects enough correct shares, it can independently complete the interpolation of the share polynomial. Nodes use KZG commitments to generate evaluation proofs during the sharing process. This approach mitigates the impact of packet loss or a malicious dealer, thereby increasing the scheme's robustness in asynchronous network environments. (3) Mutual Verification: Regardless of whether a node has obtained the share polynomial, it can still verify the shares transmitted by other nodes.

$$M_\Phi = \begin{Bmatrix} \underline{s_0}(r_{0,0}) & \underline{s_1}(r_{1,0}) & \cdots & r_{t-1,0} & r_{t,0} \\ r_{0,1} & r_{1,1} & \cdots & r_{t-1,1} & r_{t,1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ r_{0,f-1} & r_{1,f-1} & \cdots & r_{t-1,f-1} & r_{t,f-1} \\ r_{0,f} & r_{1,f} & \cdots & r_{t-1,f} & r_{t,f} \end{Bmatrix} \quad (1)$$

The coefficients can be viewed as a two-dimensional matrix M_Φ, as shown in Equation (1). The first row of the matrix is responsible for hiding secret, such as s_0 and s_1. The vertical dimension of the matrix is used to accelerate sharing, with a fixed dimension of f. Additionally, to enhance *polynomial binding* property in the scheme, a completely random $\hat{\Phi}$ will be constructed for sharing and verification. The structure of $\hat{\Phi}$ is identical to that of Φ.

4.2 Adaptation of KZG Commitments

KZG commitments are implemented based on elliptic curve bilinear pairings. Therefore, before calculating the CRS, it is necessary to determine the basic parameters. We need to decide on $g \in \mathbb{G}_1, h \in \mathbb{G}_2, \mathbb{G}_T$, and the bilinear pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We use crs' to denote $(g \in \mathbb{G}_1, h \in \mathbb{G}_2, \mathbb{G}_T, e)$. Next, we will sequentially describe all the functions in Figure 1.

- **CRS** $\leftarrow \mathsf{Setup}(crs', d_1, d_2)$. Based on *polynomial binding*, our scheme additionally constructs a polynomial $\hat{\Phi}$ with completely random coefficients. In the Bingo[1], they select an additional generator $\hat{g}$ to compute the commitment to $\hat{\Phi}$. Specifically, they randomly select a value $a \xleftarrow{\$} \mathbb{Z}_q$ and set $\hat{g} = g^a$. Since the dealer knows the linear relationship between g and $\hat{g}$, particularly in schemes based on bilinear pairings, they can forge incorrect proofs[10] that can still pass verification. Generally, the computation of the CRS can be handled by a PKI. However, to further enhance the security of the scheme, we use *HashToCurve*[24] to compute the generator $\hat{g} \in \mathbb{G}_1$. The publicly computed input enables other nodes to verify the generation of $\hat{g}$. This design ensures that our scheme is secure even in an asynchronous network environment without a PKI.

- $\mathrm{Com} \leftarrow \mathsf{Commit}_{\mathsf{uni}}(\mathbf{CRS}, \alpha(X), \hat{\alpha}(X))$. KZG commitments support univariate polynomials, but they cannot directly commit to bivariate asymmetric polynomials.

- $\mathbf{Com} \leftarrow \mathsf{Commit}_{\mathsf{bi}}(\mathbf{CRS}, \Phi(X,Y), \hat{\Phi}(X,Y))$ treats each row of the matrix M_Φ as an independent polynomial for commitment, ultimately resulting in a set of polynomial commitments with d_2+1 terms.

- $\mathbf{CM} \leftarrow \mathsf{DivideCom}(\mathbf{Com}, n)$. To save bandwidth in an asynchronous network, we send commitment CM of size $f+1$. Given that $n \geq 3f+1$, this approach allows us to save

$\underline{\mathbf{CRS} \leftarrow \mathsf{Setup}(crs', d_1, d_2)}$
$\tau \xleftarrow{\$} \mathbb{F}_q$
$\hat{g} \leftarrow HashToCurve$
$\mathbf{CRS} \leftarrow ((g, \hat{g}) \in \mathbb{G}_1, \{g^{\tau^i}, \hat{g}^{\tau^i}\}_{i \in [\![d_1]\!]}, (h, h^{\tau}) \in \mathbb{G}_2, \mathbb{G}_T, e, (d_1, d_2))$

$\underline{\mathrm{Com} \leftarrow \mathsf{Commit}_{\mathsf{uni}}(\mathbf{CRS}, \alpha(X), \hat{\alpha}(X))}$
$\mathrm{Com} \leftarrow g^{\alpha(\tau)} \hat{g}^{\hat{\alpha}(\tau)}$

$\underline{\mathbf{Com} \leftarrow \mathsf{Commit}_{\mathsf{bi}}(\mathbf{CRS}, \Phi(X,Y), \hat{\Phi}(X,Y))}$
$\phi_i(X) = r_{0,i}X^0 + r_{1,i}X^1 + \cdots + r_{d_1,i}X^{d_1}$
$\hat{\phi}_i(X) = \hat{r}_{0,i}X^0 + \hat{r}_{1,i}X^1 + \cdots + \hat{r}_{d_1,i}X^{d_1}$
$\mathbf{Com} \leftarrow \{g^{\phi_i(\tau)} \hat{g}^{\hat{\phi}_i(\tau)}\}_{i=0}^{d_2}$

$\underline{\mathbf{CM} \leftarrow \mathsf{DivideCom}(\mathrm{Com}, \mathsf{n})}$
$\mathrm{CM}_i = g^{\sum_{w=0}^{d_2}[\phi_w(\tau)]i^w} \cdot \hat{g}^{\sum_{w=0}^{d_2}[\hat{\phi}_w(\tau)]i^w}$
$\mathbf{CM}$=$\{\mathrm{CM}_i\}, i \in [n]$

$\underline{p, \hat{p}, \pi \leftarrow \mathsf{Eval}(\mathbf{CRS}, \alpha(X), \hat{\alpha}(X), x)}$
$p \leftarrow \alpha(x), \hat{p} \leftarrow \alpha(x)$
$\theta(\tau) \leftarrow (\alpha(\tau) - p)/(\tau - x)$
$\hat{\theta}(\tau) \leftarrow (\hat{\alpha}(\tau) - \hat{p})/(\tau - x)$
$\pi \leftarrow g^{\theta(\tau)} \hat{g}^{\hat{\theta}(\tau)}$

$\underline{\text{True/False} \leftarrow \mathsf{Verify}(\mathbf{CRS}, \mathbf{CM}, (i,j), (p,\hat{p}), \pi)}$
if $e(\mathbf{CM}_j \cdot g^{-p} \cdot \hat{g}^{-\hat{p}}, h) = e(\pi, h^{\tau - i})$ then return True else return False

Figure 1: Adaptation of Bivariate Asymmetric Polynomials to KZG Commitments.

approximately $2/3$ of the bandwidth consumption in this step. After receiving **Com**, parties can use DivideCom to decompose it into n sub-commitments, which are then used to verify the shares of each of the n nodes. By utilizing the additive homomorphic property of the elliptic curve, DivideCom will compute CM_i to verify the share of parties $i \in [n]$:

$$\sum_{w=0}^{f} [g^{\phi_w(\tau)} \hat{g}^{\hat{\phi}_w(\tau)}] i^w = g^{\sum_{w=0}^{f} \phi_w(\tau) i^w} + \hat{g}^{\sum_{w=0}^{f} \hat{\phi}_w(\tau) i^w}, i \in [n]$$

By substituting $X = \tau$ and $Y = i$ into the matrix M_Φ, $\sum_{w=0}^{f} \phi_w(\tau) i^w$ can be computed as follows:

$$= \begin{bmatrix} r_{0,0}i^0 & r_{1,0}\tau^1 i^0 & \cdots & r_{t,0}\tau^t i^0 \\ \vdots & \vdots & \ddots & \vdots \\ r_{0,f-1}i^{f-1} & r_{1,f-1}\tau^1 i^{f-1} & \cdots & r_{t,f-1}\tau^t i^{f-1} \\ r_{0,f}i^f & r_{1,f}\tau^1 i^f & \cdots & r_{t,f}\tau^t i^f \end{bmatrix}$$

Thus, we obtain $\sum_{w=0}^{f}[\phi_w(\tau)]i^w = \Phi(\tau, i) = \alpha_i(\tau)$ and $g^{\sum_{w=0}^{f}[\phi_w(\tau)]i^w} = g^{\alpha_i(\tau)}$. Similarly, $\hat{g}^{\sum_{w=0}^{f}[\hat{\phi}_w(\tau)]i^w} = \hat{g}^{\hat{\alpha}_i(\tau)}$. Based on the additive homomorphism, verifier can compute Uni_Commit$(\alpha'(X), \hat{\alpha}'(X))$ after receiving α' and $\hat{\alpha}'$. Due to polynomial binding, the verification will succeed if and only if $\alpha = \alpha'$ and $\hat{\alpha} = \hat{\alpha}'$.

- $p, \hat{p}, \pi \leftarrow \mathsf{Eval}(\mathbf{CRS}, \alpha(X), \hat{\alpha}(X), x)$ can generate proofs for the shares provided to other parties. When computing $\theta(\tau)$, τ is required to calculate $\alpha(\tau)$ and $(\tau - x)$, but τ must not be revealed to external nodes. Therefore, we need to use interpolation to compute θ. For node i, **points** contains f points computed by $\alpha_i(X)$. Specifically, $(i, p = \alpha_i(i))$ must not be included in **points**:

$$\mathbf{points} = \{(j, \alpha(j))\}_{j \neq i}^{j \in [n]}, |\mathbf{points}| = f$$

Subtract $(i, \alpha_i(i))$ from each entry in **points** to construct $\mathbf{points}_\theta$:

$$\mathbf{points}_\theta = \{(j - i, \alpha(j) - p)\}_{j \neq i}^{j \in [n]}, |\mathbf{points}_\theta| = f$$

By interpolating $\mathbf{points}_\theta$, $\theta(X)$ can be computed:

$$\mathsf{Interpolate}(\mathbf{points}_\theta) = \theta(X) = \frac{\alpha(X) - p}{X - i}$$

The final π can be directly computed using $\mathsf{commit}_{\mathsf{uni}}$:

$$\mathsf{Commit}_{\mathsf{uni}}(\mathbf{CRS}, \alpha(X), \hat{\alpha}(X) = g^{(\frac{\alpha(\tau) - p}{\tau - i})} \hat{g}^{(\frac{\hat{\alpha}(\tau) - p}{\tau - i})} = \pi$$

- $\mathsf{Verify}(\mathbf{CRS}, \mathbf{CM}, (i,j), (p,\hat{p}), \pi)$. Since p and $\hat{p}$ belong to $\mathbb{Z}_q$, we need to find their additive inverses within $\mathbb{Z}_q$, denoted as $-p$ and $-\hat{p}$, in order to compute g^{-p} and $\hat{g}^{-\hat{p}}$. Similarly, based on the additive homomorphism, we can decompose $CM_j \cdot g^{-p} \cdot \hat{g}^{-\hat{p}}$ into two parts. In the proof process of DivideCom, we know that $CM_j = g^{\alpha_j(\tau)} \hat{g}^{\hat{\alpha}_j(\tau)}$. Thus, CM_j and π satisfy the following algebraic relationship:

$$CM_j \cdot g^{-p} \cdot \hat{g}^{-\hat{p}} = g^{\alpha_j(\tau) - p} \cdot \hat{g}^{\hat{\alpha}_j(\tau) - \hat{p}}$$
$$\left(g^{\frac{\alpha_j(\tau) - p}{\tau - i}} \cdot \hat{g}^{\frac{\hat{\alpha}_j(\tau) - \hat{p}}{\tau - i}} \right)^{\tau - i} = \pi^{\tau - i}$$

According to *evaluation binding*, the bilinear pairing equation in Verify will only pass the verification if and only if p and $\hat{p}$ satisfy α and $\hat{\alpha}$, respectively.

$$e(CM_j \cdot g^{-p} \cdot \hat{g}^{-\hat{p}}, h) = e(\pi^{\tau - i}, h) = e(\pi, h)^{\tau - i} = e(\pi, h^{\tau - i})$$

4.3 Sharing Phase

During the sharing phase, dealer constructs a polynomial based on the recovery threshold $t \in [f, 2f]$, with the secret hidden within the polynomial. Our secret sharing scheme can share $m \in [1, t+1]$ secrets, and can be divided into the following three cases:

1. **Single Secret:** Sharing a single secret s_0 is similar to most secret sharing schemes. The secret is used as the constant term of the polynomial, and then t coefficients are randomly sampled to construct a polynomial of degree t.

$$a(x) = s_0 + r_1 x^1 + r_2 x^2 + \cdots + r_{t-1} x^{t-1} + r_t x^t$$

2. **[2, t] Secrets:** We construct the polynomial by interpolation or by directly setting the secret as one of the coefficients. Subsequently, we randomly select additional coefficients to fill out the polynomial.

$$a(x) = \underline{s_0} + \underline{s_1} x^1 + \underline{s_2} x^2 + \cdots + r_{t-1} x^{t-1} + r_t x^t$$

3. **t+1 Secrets:** This situation should be considered as data sharing. If the polynomial is constructed using interpolation, these secrets will be fully visible to the parties. Conversely, if $t+1$ secrets are directly set as coefficients, then sufficient shares are still required to reconstruct the secrets.

$$a(x) = \underline{s_0} + \underline{s_1} x^1 + \underline{s_2} x^2 + \cdots + \underline{s_{t-1}} x^{t-1} + \underline{s_t} x^t$$

Our secret sharing phase primarily composed of Algorithm 1 and 2. GenPolyandRbcCom outlines the construction of the polynomial and the computation of polynomial commitments. Sharing Phase (for node i) mainly describes the processes of sharing, verifying, and interpolating the secret shares.

Algorithm 1: GenPolyandRbcCom

Input: $n, t, f, k, [s_0, \ldots, s_k]$

1 $\alpha(X) \leftarrow \mathsf{Interpolate}([s_0, \ldots, s_k])$
2 uniformly sample coefficients from $\mathbb{Z}_q$ to fill $\Phi(X,Y)$
3 ▷ degree t in X, f in Y
4 uniformly sample $\hat{\Phi}(X,Y)$ with degree t in X, f in Y
5 **Com** $\leftarrow \mathsf{Commit}_{\mathsf{bi}}(\mathbf{CRS}, \Phi(X,Y), \hat{\Phi}(X,Y))$
6 RBC (<'commits', **Com**>) ▷ reliably broadcast **Com**
7 **for each** $i \in [n]$
8 $\alpha_i(X) \leftarrow \Phi(X,i), \hat{\alpha}_i \leftarrow \hat{\Phi}(X,i)$
9 send <'poly',$\alpha_i(X), \hat{\alpha}_i(X)$> to party i

Algorithm 2: Sharing Phase (for parties i)

Input: n, t, f

1 $\alpha_i \leftarrow \perp, \hat{\alpha}_i \leftarrow \perp, \mathbf{accept}_{\alpha_i} \leftarrow \emptyset, \mathbf{accept}_{\hat{\alpha}_i} \leftarrow \emptyset$
2 $\beta_i \leftarrow \perp, \hat{\beta}_i \leftarrow \perp, \mathbf{accept}_{\beta_i} \leftarrow \emptyset, \mathbf{accept}_{\hat{\beta}_i} \leftarrow \emptyset$
3 **CM** $\leftarrow \emptyset$
4 **upon receiving** <'commits',**Com**> from RBC, **do**
5 **CM** $\leftarrow$ DivideCom(**Com**, n)
6 **upon receiving** <'poly', α_i', $\hat{\alpha}_i'$> from dealer, **do**
7 **upon CM** $\neq \emptyset$, **do**
8 **if** $\alpha_i = \perp$ and $\mathsf{Commit}_{\mathsf{uni}}(\alpha_i', \hat{\alpha}_i') =$ **CM**$_i$ **then**
9 $\alpha_i \leftarrow \alpha_i', \hat{\alpha}_i \leftarrow \hat{\alpha}_i'$
10 **upon** $\alpha_i \neq \perp$, **do**
11 **for each** $j \in [n]$ do
12 $\alpha_i(j), \hat{\alpha}_i(j), \pi_{\alpha_{i,j}} \leftarrow \mathrm{Eval}(\alpha_i, \hat{\alpha}_i, j)$
13 send <'row', $\alpha_i(j), \hat{\alpha}_i(j), \pi_{\alpha_{i,j}}$> to parties j
14 **upon receiving** <'row', $\alpha_j(i), \hat{\alpha}_j(i), \pi_{\alpha_{j,i}}$> from j, **do**
15 **upon CM** $\neq \emptyset$, **do**
16 **if** $|accept_{\beta_i}| \leq f$ **then**
17 **if** Verify(**CM**, $(i,j), \alpha_j(i), \hat{\alpha}_j(i), \pi_{\alpha_{j,i}}$) = True
18 ▷ $\alpha_j(i) = \beta_i(j), \hat{\alpha}_j(i) = \hat{\beta}_i(j)$
19 $\mathbf{accept}_{\beta_i} \leftarrow \mathbf{accept}_{\beta_i} \cup \{(j, \beta_i(j))\}$
20 $\mathbf{accept}_{\hat{\beta}_i} \leftarrow \mathbf{accept}_{\hat{\beta}_i} \cup \{(j, \hat{\beta}_i(j))\}$
21 **else if** $|\mathbf{accept}_{\beta_i}| = |\mathbf{accept}_{\hat{\beta}_i}| = f+1$ **then**
22 $\beta_i \leftarrow \mathsf{Interpolate}(\mathbf{accept}_{\beta_i})$
23 $\hat{\beta}_i \leftarrow \mathsf{Interpolate}(\mathbf{accept}_{\hat{\beta}_i})$
24 **for all** $j \in [n]$ **do**
25 $\beta_i(j), \hat{\beta}_i(j), \pi_{\beta_{i,j}} \leftarrow \mathrm{Eval}(\beta_i, \hat{\beta}_i, j)$
26 send <'col', $\beta_i(j), \hat{\beta}_i(j), \pi_{\beta_{i,j}}$> to parties j
27 **upon receiving** <'col', $\beta_j(i), \hat{\beta}_j(i), \pi_{\beta_{j,i}}$> from j, **do**
28 **upon CM** $\neq \emptyset$, **do**
29 **if** $\alpha_i = \perp$ **and** $\hat{\alpha}_i = \perp$ **then**
30 **if** Verify(**CM**, $(j,i), \beta_j(i), \hat{\beta}_j(i), \pi_{\beta_{j,i}}$) = True
31 ▷ $\beta_j(i) = \alpha_i(j), \hat{\beta}_j(i) = \hat{\alpha}_i(j)$
32 $\mathbf{accept}_{\alpha_i} \leftarrow \mathbf{accept}_{\alpha_i} \cup \{(j, \alpha_i(j))\}$
33 $\mathbf{accept}_{\hat{\alpha}_i} \leftarrow \mathbf{accept}_{\hat{\alpha}_i} \cup \{(j, \hat{\alpha}_i(j))\}$
34 **if** $|\mathbf{accept}_{\alpha_i}| = |\mathbf{accept}_{\hat{\alpha}_i}| = t+1$ **then**
35 $\alpha_i \leftarrow \mathsf{Interpolate}(\mathbf{accept}_{\alpha_i})$
36 $\hat{\alpha}_i \leftarrow \mathsf{Interpolate}(\mathbf{accept}_{\hat{\alpha}_i})$
37 **upon** $\alpha_i \neq \perp, \hat{\alpha}_i \neq \perp$ and $\beta_i \neq \perp, \hat{\beta}_i \neq \perp$, **do**
38 send <'done'> to all parties
39 **upon receiving** <'done'> from $n-f$ parties, **do**
40 **upon** $\alpha_i \neq \perp, \hat{\alpha}_i \neq \perp$ and $\beta_i \neq \perp, \hat{\beta}_i \neq \perp$, **do**
41 **Terminate**

Algorithm 1. In lines 7-9, we compute and encrypt the share polynomials for other nodes and transmit them. The polynomial Φ has a degree of t in X and a degree of f in Y. We set $y = [n]$, which allows us to compute n polynomials α of degree t. Each $\alpha_i(x)$ is used as the secret sharing polynomial for patry i. By setting $x = [n]$, we obtain n polynomials β of degree f. Each $\beta_i(y)$ is used as the accelerated sharing polynomial for party i.

Algorithm 2. In lines 4-5, we calculate n sub-commitments for verification based on the total number of nodes n. In lines 10-13, this part is used to compute the shares for other nodes and send them to the corresponding nodes. Then, in line 17, verification is performed; if it passes, the share is saved for interpolation. Finally, in line 21, when the number of shares

reaches $f+1$, β and $\hat{\beta}$ can be reconstructed. Based on the condition in line 21, we can determine that when a node completes the recovery of β and $\hat{\beta}$, at least $f+1$ honest nodes among all parties have successfully saved α and $\hat{\alpha}$. When another set of $t-f$ honest nodes recover their β and $\hat{\beta}$, these $t+1$ honest nodes can provide $t+1$ shares of α and $\hat{\alpha}$ for the entire network. With these $t+1$ honest nodes, the entire network's nodes can be helped to recover their polynomials. We can simply summarize it as: if $f+1$ nodes in the entire network obtain the correct α and $\hat{\alpha}$, the entire network can recover their polynomials.

In lines 37-41, this is the phase where sharing is concluded. When the dealer is sharing, we do not share β and $\hat{\beta}$. However, our condition for sending the done signal ensures that β and $\hat{\beta}$ are not empty. The reason for this is the same as in the accelerated sharing section. We need to ensure that at least $f+1$ honest nodes in the entire network have completed sharing.

4.4 Reconstruction Phase

Algorithm 3: Reconstruction$_i$ (for node i)

Input: t, **CM**

1 **shares**$_i \leftarrow \emptyset$
2 $\beta_i(0), \hat{\beta}_i(0), \pi_{\beta_{i,0}} \leftarrow \mathsf{Eval}(\beta_i, \hat{\beta}_i, 0)$
3 send <'rec', $\beta_i(0), \hat{\beta}_i(0), \pi_{\beta_{i,0}}$>
4 **upon receiving** <'rec', $\beta_j(0), \hat{\beta}_j(0), \pi_{\beta_{j,0}}$> from j, **do**
5 **if** Verify(**CM**, $(j,0), \beta_j(0), \hat{\beta}_j(0), \pi_{\beta_{j,0}}$) = True **then**
6 **share**$_i \leftarrow$ **share**$_i \cup \{(j, \alpha_0(j))\}$
7 $\triangleright\ \beta_j(0) = \Phi(0,j) = \alpha_0(j)$
8 **if** $|$**share**$_i| = t+1$ **then**
9 $\alpha_0(X) \leftarrow \mathsf{Interpolate}$(**share**$_i$)
10 **Terminate**

We describe how to recover our shared secret in Algorithm 3. After all nodes have completed sharing, the secret reconstruction phase can begin. Our secret sharing scheme cannot recover a specific k-th secret. This is designed to prevent the leakage of secrets during the sharing phase.

The set of secrets is hidden in the polynomial $\alpha_0(X)$. Parties use their β and $\hat{\beta}$ polynomials to compute $\beta_i(0)$, $\hat{\beta}_i(0)$ and proof $\pi_{\beta_{i,0}}$. Once the received secret shares reach $t+1$, reconstruction can begin. Using Lagrange interpolation, the polynomial $\alpha_0(X)$ is recovered. Subsequently, the original secret can be obtained from the coefficients.

5 From AVSS to ADKG

In this section, we use AVSS to construct the ADKG protocol. In the key agreement phase, we utilized the *validated asynchronous Byzantine agreement* (VABA)[1] protocol proposed in the Bingo scheme. All generated keys are field elements, $sk \in \mathbb{Z}_q$. By adjusting the threshold of AVSS, we can construct threshold schemes that cater to different requirements, such as encryption and signature schemes with varying threshold demands[14, 26, 8, 23, 5, 19]. This allows the system to find the optimal balance between security and performance according to different needs.

5.1 Non-interactive m-Schnorr Protocol

In the subsequent key derivation phase, parties need to disclose their public key shares to other nodes. We implemented additional commitments to the public key shares using the m-Schnorr protocol. The traditional Schnorr protocol is only used to commit to a single value[36], but our scheme may involve sharing multiple keys. Our shares follow the following Relation $\mathcal{R}$:

$$\mathcal{R} = \{(\{pk_0, \cdots, pk_m\}, \{z_0, \cdots, z_m\}) : \prod_{i=0}^{m} pk_i = g^{\sum_{i=0}^{m} z_i}\}$$

$g^{\mathbf{z}}, \hat{g}^{\hat{\mathbf{z}}}, \pi, \hat{\pi} \leftarrow \mathsf{Sc.Prove}(\mathbf{CRS}, \mathbf{z}, \hat{\mathbf{z}})$
$\hat{g} \leftarrow HashToCurve$
$g^{\mathbf{z}} \leftarrow \{g^{z(k)}\}_{k=0}^{m}, \hat{g}^{\hat{\mathbf{z}}} \leftarrow \{\hat{g}^{\hat{z}(k)}\}_{k=0}^{m}$
$r_1, r_2 \xleftarrow{\$} \mathbb{Z}_q$
$U_1 \leftarrow g^{r_1}, U_2 \leftarrow \hat{g}^{r_2}$
$C_1 \leftarrow Hash(g|g^{z(0)}|g^{z(1)}|g^{z(2)}|\cdots|g^{z(m)}, U_1)$
$C_2 \leftarrow Hash(\hat{g}|\hat{g}^{\hat{z}(0)}|\hat{g}^{\hat{z}(1)}|\hat{g}^{\hat{z}(2)}|\cdots|\hat{g}^{\hat{z}(m)}, U_2)$
$R_1 \leftarrow r_1 + C_1 \cdot \sum_{k=0}^{m} z_k$
$R_2 \leftarrow r_2 + C_2 \cdot \sum_{k=0}^{m} \hat{z}_k$
$\pi \leftarrow (U_1, C_1, R_1)$
$\hat{\pi} \leftarrow (U_2, C_2, R_2)$

True/False $\leftarrow \mathsf{Sc.Verify}(\mathbf{CRS}, g^{\mathbf{z}}, \hat{g}^{\hat{\mathbf{z}}}, \pi, \hat{\pi})$
if $C_1 \leftarrow Hash(g|g^{z(0)}|g^{z(1)}|g^{z(2)}|\cdots|g^{z(m)}, U_1)$ **and**
$C_2 \leftarrow Hash(\hat{g}|\hat{g}^{\hat{z}(0)}|\hat{g}^{\hat{z}(1)}|\hat{g}^{\hat{z}(2)}|\cdots|\hat{g}^{\hat{z}(m)}, U_2)$ **and**
$g^{R_1} = U_1 + C_1 \cdot g^{\sum_{k=0}^{m} z_k}$ **and**
$\hat{g}^{R_2} = U_2 + C_2 \cdot \hat{g}^{\sum_{k=0}^{m} \hat{z}_k}$ **then** return **True**
else return **False**

Figure 2: m-Schnorr protocol.

In Figure 2, the overall process of the protocol follows the general structure of the Schnorr protocol.

- $g^{\mathbf{z}}, \hat{g}^{\hat{\mathbf{z}}}, \pi, \hat{\pi} \leftarrow \mathsf{Sc.Prove}(\mathbf{CRS}, \mathbf{z}, \hat{\mathbf{z}})$. We computed the corresponding proofs π and $\hat{\pi}$ based on the two sets of keys. (1) **Commitment** a**:** We continue to use *HashToCurve*[24] to compute a new $\hat{g}$, which can be different from the $\hat{g}$ used in AVSS. For two independent generators $(g, \hat{g})$, we randomly sample (r_1, r_2) and compute (U_1, U_2). (2) **Challenge** e**:** We concatenate the public key shares and use them as

the input to the Hash function to construct a non-interactive challenge. (3) **Response** z**:** We compute the final responses using the random values, private keys, and the hash values, respectively. Finally, use (a, e, z) as the proof π for our public key share.

- True/False $\leftarrow$ Sc.Verify($\mathbf{CRS}, g^{\mathbf{z}}, \hat{g}^{\hat{\mathbf{z}}}, \pi, \hat{\pi}$). The traditional Schnorr protocol can only commit to a single public-private key pair. However, our AVSS scheme enhances polynomial binding by using two polynomials. Suppose we plan to negotiate m public keys in one session. The most straightforward approach would be to reuse the non-interactive Schnorr protocol m times, generating $2m$ proofs. Instead, we aggregate all the key shares and commit to them, with the size of the resulting proof determined by the hash function used. We estimate that the space overhead of our proof π is approximately $1/m$ of the space consumed by reusing the Schnorr protocol m times. The m-Schnorr protocol is based on DL and ROM assumptions.

5.2 Key Derivation Phase

We describe how to verify public key shares and derive the final public key in Algorithm 4. During the key agreement phase, we derive the public and private keys using the dealers' set. For the sake of simplicity, we assume that $m+1$ keys need to be negotiated. First, each node computes its own private keys $\mathbf{z}_i$ from the secrets shared by the **dealers**. Next, we use the generator g and $\mathbf{z}_i$ to compute the public key $\mathbf{g}^{\mathbf{z}_i}$ and commit it using a non-interactive Schnorr protocol. We use the **CM** from the **dealers** to compute a new commitment that verifies whether the private key was correctly computed using the appropriate shares.

Algorithm 4. In lines 4-7, we compute the key shares and commitments at $X = k$ based on **dealers**. In lines 8-11, we calculate the total private key along with the corresponding public key. The $\bar{\pi}$ calculated in line 13 is used later to verify whether the shares from other nodes are correct, as detailed in line 19. In lines 21-23, when the number of elements in $\mathbf{share}_i(k)$ reaches $t+1$, the k-th public key is derived through InterpolationExp.

InterpolateExp($\mathbf{share}_i(k)$) is essentially a Lagrange interpolation, with the key difference that the interpolation produces a polynomial where each term has been multiplied by the generator g through scalar multiplication:

$$\text{InterpolateExp}(\mathbf{share}_i) = g^{z_i(x)} = g^{z_{i,0}+z_{i,1}x+\cdots+z_{i,t}x^t}$$

Algorithm 4: Key Derivation Phase (for node i)

Input: dealers, m

1 $\mathbf{share}_i \leftarrow \emptyset$, $\mathbf{CM}_{\mathbf{dealers}} \leftarrow \emptyset$
2 ▷ **share** and **CM** each contain $m+1$ sets.
3 **for** $k \in [\![m]\!]$ **do**
4 **for** $a \in$ **dealers do**
5 $z_{i,a}(k), \hat{z}_{i,a}(k), \pi_{i,a}(k) \leftarrow \text{Eval}(\alpha_{i,a}, \hat{\alpha}_{i,a}, k)$
6 $\mathbf{CM}_a \leftarrow \text{DivideCom}(\mathbf{Com}_a, k)$
7 ▷ $\alpha_{i,a}, \hat{\alpha}_{i,a}$ are shared by a
8 $z_i(k) \leftarrow \sum_{a \in \mathbf{dealers}} z_{i,a}(k)$
9 $\hat{z}_i(k) \leftarrow \sum_{a \in \mathbf{dealers}} \hat{z}_{i,a}(k)$
10 $\bar{\pi}_i(k) \leftarrow \prod_{a \in \mathbf{dealers}} \pi_{i,a}(k)$
11 $\mathbf{CM}_{\mathbf{delaers}}(k) \leftarrow \prod_{a \in \mathbf{dealers}} \mathbf{CM}_a(k)$
12 $\mathbf{z}_i \leftarrow \{z_{i,a}(k)\}_{a \in \mathbf{dealers}}^{k \in [\![m]\!]}, \hat{\mathbf{z}}_i \leftarrow \{\hat{z}_{i,a}(k)\}_{a \in \mathbf{dealers}}^{k \in [\![m]\!]}$
13 $\bar{\pi}_i \leftarrow \{\bar{\pi}_i(k)\}_{a \in \mathbf{dealers}}^{k \in [\![m]\!]}$
14 $\mathbf{g}^{\mathbf{z}_i}, \hat{\mathbf{g}}^{\hat{\mathbf{z}}_i}, \pi_i, \hat{\pi}_i \leftarrow \text{Sc.Prove}(g, \mathbf{z}_i, \hat{\mathbf{z}}_i)$
15 send <'pk share', $\mathbf{g}^{\mathbf{z}_i}, \mathbf{g}^{\hat{\mathbf{z}}_i}, \pi_i, \hat{\pi}_i, \bar{\pi}_i$> to all parties
16 **upon receiving** <'pk share', $\mathbf{g}^{\mathbf{z}_j}, \hat{\mathbf{g}}^{\hat{\mathbf{z}}_j}, \pi_j, \hat{\pi}_j, \bar{\pi}_j$> from j
17 **if** Sc.Verify($\mathbf{g}^{\mathbf{z}_j}, \hat{\mathbf{g}}^{\hat{\mathbf{z}}_j}, \pi_j, \hat{\pi}_j$) = True **then**
18 **for** $k \in [\![m]\!]$ **do**
19 **if** Verify($\mathbf{CM}_{\mathbf{dealers}}(k), (k, j), g^{z_j(k)}, g^{\hat{z}_j(k)}, \bar{\pi}_j(k)$) = True **then**
20 $\mathbf{share}_i(k) \leftarrow \mathbf{share}_i(k) \cup \{(j, z_j(k))\}$
21 **if** $|\mathbf{share}_i(k)| = t+1$ **then**
22 $g^{z_i(x)} \leftarrow \text{InterpolateExp}(\mathbf{share}_i(k))$
23 **output** $(z_i(k), g^{z(k)})$
24 **output** $(\mathbf{z}_i, \mathbf{g}^{\mathbf{z}})$ and **Terminate**
25 ▷ $\mathbf{z}_i = \{z_i(0), \ldots, z_i(m)\}, \mathbf{g}^{\mathbf{z}} = \{g^{z_i(0)}, \ldots, g^{z_i(m)}\}$

6 Analysis

6.1 AVSS Security

The security of our AVSS scheme is established in the following main theorem.

Theorem 1. If the underlying KZG commitment scheme is secure, then our AVSS is an f-resilient AVSS for m secrets, for any $m \leq t \leq \frac{2n}{3}$.

To prove this, we use the q-Strong Diffie-Hellman (q-SDH) assumption and the ROM to demonstrate the security of our PCS, particularly its *binding* and *hiding* properties. Next, we demonstrate the correctness, secrecy, and termination properties of the AVSS protocol.

The q-SDH posits that it is computationally infeasible to produce a valid pair $(c, g^{1/(x+c)})$ given a set of elements $(g, g^x, g^{x^1}, \cdots, g^{x^q}, h, h^x)$ from groups $\mathbb{G}_1^{q+1} \times \mathbb{G}_2^2$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are cyclic groups of prime order p, generated by g and h, respectively, and form a bilinear group. Here, q is an integer, and x is a randomly chosen from $\mathbb{F}_p$.

Lemma 1. The $q-SDH$ ensures that PCS is both binding and secure. $\mathcal{A}$ cannot forge different polynomials that produce the same commitment, nor can they extract information about the committed polynomial from the commitment.

Lemma 2. The ROM guarantees that the generators g and $\hat{g}$, derived via *HashToCurve*, are *uniformly distributed* and

Table 4: Evaluation of our AVSS schemes. The experimental environments consist of both local area networks (LAN) and wide area networks (WAN). The total communication in the LAN matches that in the WAN. Therefore, '—' is used in the table to indicate this.

Network	Threshold t	Avg. runtime (in seconds)				Total communication per node (MB)			
		$n=16$	$n=64$	$n=128$	$n=256$	$n=16$	$n=64$	$n=128$	$n=256$
LAN	f	0.50	2.88	9.91	44.45	—	—	—	—
	$n/2$	0.58	3.75	13.88	77.89	—	—	—	—
	$2n/3$	0.63	4.61	19.62	130.44	—	—	—	—
WAN	f	1.66	7.01	21.02	80.16	0.21	2.13	8.02	38.31
	$n/2$	1.73	7.88	25.47	113.61	0.29	2.98	12.37	70.98
	$2n/3$	1.78	8.74	30.72	166.15	0.34	3.81	17.50	122.29

Table 5: The table presents the time required to generate 1 public key and m public keys, including both the m-Schnorr protocol and key agreement time.

Network	Threshold t	Avg. runtime with 1 public key (in seconds)				Avg. runtime with $m=\frac{f}{2}$ public keys (in seconds)			
		$n=16$	$n=64$	$n=128$	$n=256$	$n=16$	$n=64$	$n=128$	$n=256$
LAN	f	0.44	0.86	1.76	4.62	0.48	2.14	7.48	29.37
	$n/2$	0.45	0.96	2.13	6.10	0.49	2.24	8.11	30.16
	$2n/3$	0.45	1.04	2.46	7.91	0.49	2.34	8.85	43.27
WAN	f	1.39	3.62	6.91	14.60	1.67	13.76	52.02	200.78
	$n/2$	1.40	3.71	7.29	16.07	1.68	13.86	52.65	201.58
	$2n/3$	1.40	3.80	7.62	17.88	1.68	13.97	53.38	214.68

collision-resistant.

Lemma 3. The underlying KZG commitment scheme for univariate polynomials is *binding* and *hiding*. ***Binding*** implies that for any polynomial $\alpha(x)$, the commitment C_α=Commit($\alpha(x)$) uniquely determines $\alpha(x)$, meaning no adversary $\mathcal{A}$ can produce a different polynomial $\alpha'(x)$ such that Commit($\alpha(x)$)=Commit($\alpha'(x)$) unless $\alpha(x)=\alpha'(x)$. ***Hiding*** means that given a commitment C_α, no $\mathcal{A}$ can gain any information about polynomial $\alpha(x)$ other than what is explicitly revealed through the protocol.

Corollary 1. Given that the underlying KZG commitment scheme is both binding and hiding for univariate polynomials, these properties not only hold but are enhanced for bivariate asymmetric polynomials when two different generators are used to commit to two different bivariate asymmetric polynomials.

Proof. (1) By **Lemma 2**, g and $\hat{g}$ are uniformly distributed over the groups $\mathbb{G}_1$. This uniform distribution ensures that the generators do not exhibit any predictable patterns or relationships that could be exploited by an $\mathcal{A}$. And collision resistance prevents $\mathcal{A}$ from creating related generators that could undermine the security of the commitment scheme. (2) **Binding Enhancement:** Let $\Phi(x,y)$ be a bivariate asymmetric polynomial, and consider the commitments $C_{x_i}=g^{\Phi(x_i,\tau)}$ and $C_{y_i}=g^{\Phi(\tau,y_i)}$. By **Lemma 1** and **3**, each commitment C_{x_i} and C_{y_i} uniquely determines the corresponding univariate polynomial $\Phi(x_i,y)$ and $\Phi(x,y_i)$. Since this uniqueness holds for any x_i and y_i, the entire bivariate polynomial $\Phi(x,y)$ is uniquely determined, preserving the binding property. By using different generators g and $\hat{g}$, $\mathcal{A}$ would have to solve two independent DL problems, which is computationally infeasible. (3) **Hiding Enhancement:** By **Lemma 3**, for each fixed x_i and y_i, the commitments C_{x_i} and C_{y_i} reveal no information about $\Phi(x_i,y)$ and $\Phi(x,y_i)$ beyond what is explicitly revealed, due to the hiding property in **Lemma 2**. Thus, the confidentiality of bivariate asymmetric polynomial is preserved. Independent generator further obscures the relationship between two polynomials, thereby enhancing the overall confidentiality of the committed data. □

Theorem 2. If q-SDH, ROM and *binding* hold, then our AVSS satisfies *correctness*.

Proof. The *correctness* property of AVSS means that all honest parties reconstruct the correct secret if dealer is honest. Given that the underlying KZG commitment scheme is *binding* for univariate polynomials, and this property are further strengthened for bivariate asymmetric polynomials, as established in **Corollary 1**, these enhanced properties can be employed to establish the correctness of the AVSS protocol.

Binding ensures that polynomial commitment generated by dealer uniquely corresponds to shared secret. During the sharing and reconstruction phase, all honest parties will be reconstructing the correct polynomials because commitment is binding, meaning no $\mathcal{A}$ can alter the polynomial or forge valid shares that lead to a different result. □

Theorem 3. If q-SDH and ROM hold, then our AVSS satisfies *secrecy*.

Proof. ***Secrecy*** in the AVSS protocol ensures that, during both the sharing and reconstruction phases, an $\mathcal{A}$ with access to fewer than $t+1$ shares cannot learn any information about the secret. This follows from ***Hiding*** property of the PCS, as established in **Corollary 1**, which ensures that the committed polynomial remains concealed until sufficient shares are gathered for reconstruction. Since the $\mathcal{A}$ can only access fewer than $t+1$ shares, and due to *hiding* property of the commitment, no information about the secret can be inferred during the sharing phase. □

Theorem 4. If q-SDH, ROM and *binding* hold, then our AVSS satisfies *termination*.

Proof. ***Termination*** ensures that all honest parties will complete the protocol in finite time, regardless of the actions of malicious parties, as long as the network eventually delivers all messages. (1) **Sharing Phase:** If the dealer is honest, they broadcast valid shares to all parties. Once at least $f+1$ honest parties have received their $\alpha(x)$ and $\hat{\alpha}(x)$ correctly, they will broadcast 'row' messages to all other parties. Each honest party will receive enough 'row' evaluations (at least $f+1$), allowing them to interpolate $\beta(y)$ and $\hat{\beta}(y)$, then send their 'column' messages. (2) **Reconstruction Phase:** Once $t+1$ valid shares are collected, parties can reconstruct the original secret. Since the network eventually delivers the required shares, all honest parties will be able to reconstruct $\alpha_0(x)$ and complete the protocol. □

From the results established in **Theorems 2**, **3**, and **4**, we can now conclude the proof of **Theorem 1**.

Theorem 5. Our AVSS protocol incurs a communication cost of $O(\lambda n^2)$ among all honest parties. Based on the properties established in **Corollary 1**, the protocol ensures that all parties complete the sharing and reconstruction phases in $O(1)$ asynchronous rounds.

6.2 m-Schnorr Protocol Security

The security of m-Schnorr protocol is established in the following main theorem. We use P and V to represent the Prover and Verifier, respectively.

Lemma 4. There exists a simulator $\mathcal{S}$ that, without knowledge of the secret z, can generate a proof transcript indistinguishable from one generated by a real prover.

Table 6: The data in the table represent the space usage comparison between the m-Schnorr algorithm and running the Schnorr protocol m times independently for negotiating m public keys.

Scheme	Space usage ($m = \frac{f}{2}$)			
	$n = 16$	$n = 64$	$n = 128$	$n = 256$
Our work	0.96	3.21	6.31	12.21
m times	1.38	6.88	14.44	28.87
	70.45 %	46.81 %	43.72 %	42.32 %

Theorem 6. (Zero-knowledge) If DL and ROM hold, the V cannot extract any information about the secret z from a valid proof.

Proof. **Theorem 6** can be proved by **Lemma 2** and **4**. (1) The $\mathcal{S}$ picks random challenges $r_1, r_2 \in \mathbb{Z}_q$ and responses $R_1, R_2 \in \mathbb{Z}_q$. C_1 and C_2 are assumed to be the output of a cryptographic hash function, mapping the commitment values and other relevant inputs to elements in $\mathbb{Z}_q$. (2) $\mathcal{S}$ computes the commitments U_1 and U_2. (3) $\mathcal{S}$ outputs the simulated proof $(\pi, \hat{\pi}) = ((U_1, C_1, R_1), (U_2, C_2, R_2))$. Since the challenges C_1 and C_2 are generated through ROM, V cannot distinguish between the simulated proof and a real proof generated by a P who knows the secret z. This holds under the DLP, where computing z from g^z is computationally infeasible. □

Lemma 5. If P knows the secret z and follows the m-Schnorr protocol correctly, the verification equation holds.

Theorem 7. (Completeness) If DL and ROM hold, the V will always accept a proof generated by an honest prover who knows the secret z.

Proof. **Theorem 7** can be proved by **Lemma 2** and **5**. The responses R_1 and R_2 are computed as functions of the secret z. Given that the verifier's challenge values are unpredictable under the ROM assumption, the prover's responses satisfy the verification equations. This holds under the DL assumption, where computing z from g^z is infeasible. □

Theorem 8. (Unforgeability) If DLP and ROM hold, an adversary $\mathcal{A}$ cannot forge a valid proof in the described non-interactive protocol without knowing the secret z.

Proof. (1) Based on **Lemma 2**, $\mathcal{A}$ cannot predict or manipulate the values of C_1 without querying the oracle with specific inputs. Once C_1 are generated, it is independent and unpredictable. (2) To generate a valid response R_1, $\mathcal{A}$ must know secret z, as R_1 is dependent on z through the equation $g^{R_1} = U_1 + C_1 \cdot g^{\sum_{k=0}^{m} z_k}$. According to the DLP, computing z from g^z is computationally infeasible. Therefore, $\mathcal{A}$ cannot compute without knowing z. □

7 Implementation and Evaluation

Implementation. We implemented the cryptographic modules in Rust, including the generation of bivariate asymmetric polynomials, KZG commitments, and the non-interactive m-Schnorr protocol. In our implementation, we used the bls_12_381_plus[6] library for elliptic curve operations. This library provides a robust implementation of the BLS12-381 pairing-friendly elliptic curve, so we did not modify the source code, ensuring the accuracy and reliability of the results. For the *HashToCurve* functionality, we chose SHA-256[35] as the hash function. Our implementation includes some optimizations, such as precomputing necessary data during the system initialization phase to reduce redundant computations during protocol execution, for example, constructing the n-dimensional Vandermonde matrix.

Tables 4 and 5 present the evaluation results of our AVSS and key agreement steps, respectively. Table 6 demonstrates the space optimization achieved by our m-Schnorr protocol. Our evaluation demonstrates the effectiveness of our approach in improving communication performance. By precomputing necessary data, we significantly reduced the impact of redundant computations on efficiency, although this comes at the cost of increased storage space. Our results demonstrate that our protocol can maintain high performance even under high-threshold configurations, particularly in terms of communication efficiency.

8 Conclusion

In this paper, we propose and thoroughly analyze a novel AVSS protocol that features flexible thresholds and the ability to share multiple secrets in a single round. In constructing the secret-sharing polynomial, we use interpolation to convert $m+1$ secrets into a degree-m polynomial, and then randomly select sufficient coefficients from a finite field to expand the polynomial into a bivariate asymmetric polynomial. Additionally, we extend the traditional Schnorr protocol into a non-interactive m-Schnorr protocol, with the aim of generating proofs for multiple public key shares simultaneously.

Building upon the VABA protocol proposed in Bingo[1], we ultimately construct an asynchronous distributed key generation protocol that supports flexible thresholds and can negotiate multiple public keys in a single round. In a network of n nodes, the communication complexity of our AVSS protocol is $O(\lambda n^2)$. When extended to ADKG, our protocol achieves a communication complexity of $O(\lambda n^3)$ even in the worst case. Overall, our protocol is based on the q-SDH and ROM assumptions and remains secure even in networks without a PKI. We implemented the AVSS and non-interactive m-Schnorr protocols using Rust.

Our AVSS protocol currently does not support the recovery of a single secret, which we plan to explore in future research.

References

[1] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 39–70, Cham, 2023. Springer Nature Switzerland.

[2] 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*, PODC'21, page 363–373, New York, NY, USA, 2021. Association for Computing Machinery.

[3] Nicolas AlHaddad, Mayank Varia, and Haibin Zhang. High-threshold avss with optimal communication complexity. In Nikita Borisov and Claudia Diaz, editors, *Financial Cryptography and Data Security*, pages 479–498, Berlin, Heidelberg, 2021. Springer Berlin Heidelberg.

[4] Michael Backes, Amit Datta, and Aniket Kate. Asynchronous computational vss with reduced communication complexity. In Ed Dawson, editor, *Topics in Cryptology – CT-RSA 2013*, pages 259–276, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[5] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In Yvo G. Desmedt, editor, *Public Key Cryptography — PKC 2003*, pages 31–46, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[6] Sean Bowe, Jack Grigg, and Mike Lodder. bls12_381_plus: Implementation of the bls12-381 pairing-friendly elliptic curve construction. https://github.com/mikelodder7/bls12_381_plus, 2024.

[7] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[8] 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*, CCS '02, page 88–97, New York, NY, USA, 2002. Association for Computing Machinery.

[9] Ran Canetti, Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In Michael Wiener, editor, *Advances*

in Cryptology — CRYPTO' 99, pages 98–116, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[10] Sanjit Chatterjee, Darrel Hankerson, Edward Knapp, and Alfred Menezes. Comparing two pairing-based aggregate signature schemes. *Designs, Codes and Cryptography*, 55:141–167, 2010.

[11] 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, Anaheim, CA, August 2023. USENIX Association.

[12] 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*, CCS '21, page 2705–2721, New York, NY, USA, 2021. Association for Computing Machinery.

[13] 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.

[14] Yvo G. Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–458, 1994.

[15] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ecdsa from ecdsa assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1051–1066, 2019.

[16] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438, 1987.

[17] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings on Advances in Cryptology—CRYPTO '86*, page 186–194, Berlin, Heidelberg, 1987. Springer-Verlag.

[18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 33–62, Cham, 2018. Springer International Publishing.

[19] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20:51–83, 2007.

[20] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4), sep 2015.

[21] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[22] Jens Groth and Victor Shoup. Design and analysis of a distributed ECDSA signing service. Cryptology ePrint Archive, Paper 2022/506, 2022.

[23] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Aggregatable distributed key generation. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 147–176, Cham, 2021. Springer International Publishing.

[24] Thomas Icart. How to hash into elliptic curves. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, pages 303–316, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[25] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 177–194, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[26] Eleftherios Kokoris Kogias, Enis Ceyhun Alp, Linus Gasser, Philipp Svetolik Jovanovic, Ewa Syta, and Bryan Alexander Ford. Calypso: Private data management for decentralized ledgers. volume 14, page 586–599, 2021.

[27] 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*, CCS '20, page 1751–1767, New York, NY, USA, 2020. Association for Computing Machinery.

[28] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Designs, Codes and Cryptography*, 87(9):2139–2164, 2019.

[29] Wafa Neji, Kaouther Blibech, and Narjes Ben Rajeb. Distributed key generation protocol with a new complaint management strategy. *Security and Communication Networks*, 9(17):4585–4595, 2016.

[30] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. Musig-dn: Schnorr multi-signatures with verifiably deterministic nonces. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1717–1731, New York, NY, USA, 2020. Association for Computing Machinery.

[31] Arpita Patra, Ashish Choudhury, and C Pandu Rangan. Efficient asynchronous verifiable secret sharing and multiparty computation. *Journal of Cryptology*, 28:49–109, 2015.

[32] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In Donald W. Davies, editor, *Advances in Cryptology — EUROCRYPT '91*, pages 522–526, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[33] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[34] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. Roast: Robust asynchronous schnorr threshold signatures. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 2551–2564, New York, NY, USA, 2022. Association for Computing Machinery.

[35] RustCrypto Developers. Rustcrypto/hashes: Sha-2 implementation in rust. https://github.com/RustCrypto/hashes/tree/master/sha2, 2023.

[36] C. P. Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 239–252, New York, NY, 1990. Springer New York.

[37] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4:161–174, 1991.

[38] Nibesh Shrestha, Adithya Bhat, Aniket Kate, and Kartik Nayak. Synchronous distributed key generation without broadcasts. Cryptology ePrint Archive, Paper 2021/1635, 2021.

[39] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 444–460, 2017.

[40] Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew Miller. hbACSS: How to robustly share many secrets. Cryptology ePrint Archive, Paper 2021/159, 2021.