

Bingo: Adaptively Secure Packed Asynchronous Verifiable Secret Sharing and Asynchronous Distributed Key Generation

Ittai Abraham¹, Philipp Jovanovic², Mary Maller³, Sarah Meiklejohn^{2,4}, and Gilad Stern⁵

¹ VMware Research

² University College London

³ Ethereum Foundation

⁴ Google

⁵ The Hebrew University of Jerusalem

Abstract. In this work we present **Bingo**, an adaptively secure and optimally resilient packed asynchronous verifiable secret sharing (PAVSS) protocol that allows a dealer to share $f + 1$ secrets or one high threshold secret with a total communication complexity of just $O(\lambda n^2)$ words. **Bingo** requires a public key infrastructure and a powers-of-tau setup. Using **Bingo**'s packed secret sharing, we obtain an adaptively secure validated asynchronous Byzantine agreement (VABA) protocol that uses $O(\lambda n^3)$ expected words and constant expected time. Using this agreement protocol in combination with **Bingo**, we obtain an adaptively secure high threshold asynchronous distributed key generation (ADKG) of standard field element secrets that uses $O(\lambda n^3)$ expected words and constant expected time. To the best of our knowledge, **Bingo** is the first ADKG to have an adaptive security proof and have the same asymptotic complexity of the best known ADKG's that only have non-adaptive security proofs.

1 Introduction

The ability of a party to distribute a secret among a set of other parties (i.e., *secret sharing*) is a fundamental cryptographic primitive, with applications such as Byzantine agreement, threshold cryptography, and secure multiparty computation [17, 18, 37, 42, 44]. At its most basic level, secret sharing involves one honest dealer, sharing one secret among a set of n parties, so that if at least t parties coordinate they can reconstruct the secret (where notably an adversary is assumed to control strictly fewer than t parties).

There are many functional enhancements of secret sharing, including *verifiable* secret sharing (VSS) [24], where parties can verify the validity of their shares even in the face of a malicious dealer, and *packed* secret sharing [32], where a dealer can deal m secrets in a way that is more efficient than just running m iterations of the protocol. In terms of enhancements to the network model, *asynchronous* secret sharing [9, 20] requires no assumptions about the

delay on messages between parties or the order in which they are received. These protocols are thus more robust in the face of denial-of-service attacks. Finally, and crucially for systems that are expected to run for long periods of time, *adaptively secure* secret sharing protocols [25] allow the adversary to corrupt parties over time rather than starting with a static set of parties that it controls.

Our main construction, **Bingo**, combines all of these enhancements: specifically, it is an adaptively secure packed asynchronous verifiable secret sharing (PAVSS) protocol that allows a dealer to share $f + 1$ secrets with a total communication complexity of just $O(\lambda n^2)$ words, where n is the total number of parties and f is the number of malicious parties. Additionally, **Bingo** is *optimally resilient* in assuming that $n = 3f + 1$.

Theorem 1 (Adaptively Secure Packed AVSS). *For any n , $f < \frac{n}{3}$, and field \mathbb{F}_p for $p > n$ there exists a packed asynchronous verifiable secret sharing scheme that requires a PKI setup and a univariate powers-of-tau setup [15] (of size $O(\lambda n)$ words) that is adaptively secure, with optimal resilience against an algebraic adversary that can corrupt f parties.*

Sharing $f + 1$ secrets incurs a total communication complexity of $\Theta(\lambda n^2)$ words and constant number of rounds.

Three types of reconstruction are supported, each has a complexity of $O(\lambda n^2)$ words and constant number of rounds.

- *Reconstruction of any single secret, which does not reveal any information about any non-reconstructed secrets.*
- *Given different instances of **BingoShare** sharings and an index k , reconstruction of the sum of the k -th secret shared across several different instances of **BingoShare**, which does not reveal any information about any non-reconstructed secrets.*
- *Reconstruction of all secrets at once, which can be viewed as reconstructing a degree $2f$ sharing.*

The fact that **Bingo** is a packed AVSS scheme is proven in [Theorems 5 to 7](#), and the two additional modes of reconstruction discussed in [Section 4.3](#).

In addition to the core **Bingo** construction, our paper also considers two applications. First, we use **Bingo** to construct an adaptively secure *validated asynchronous Byzantine agreement* (VABA) protocol that reaches agreement on messages of size $O(n)$ and requires just $O(\lambda n^3)$ words.

Theorem 2. [*Adaptively Secure VABA*] *For any n , $f < \frac{n}{3}$, and field \mathbb{Z}_p for $p > n$ there exists a Validated Asynchronous Byzantine Agreement protocol that requires a PKI setup and a univariate powers-of-tau-setup (of size $O(\lambda n)$ words) that has optimal resilience against an adversary that can corrupt f parties. With inputs of size $O(n)$ the protocol runs in $O(1)$ expected time and uses $O(\lambda n^3)$ expected word complexity.*

The proof of this theorem closely resembles the proof of [3], and the adaptations needed are discussed in [Appendix C](#).

Second, we use **Bingo** and the above protocol to construct an adaptively secure high-threshold *asynchronous distributed key generation* (ADKG) for BLS signatures. As compared with previous ADKG protocols, ours achieves adaptive security, requires just $O(1)$ expected rounds and $O(\lambda n^3)$ expected words, and has a secret key that is a field element (which in particular makes it compatible with standard threshold signature schemes like BLS [13]).

Theorem 3. [*Adaptively Secure Asynchronous DKG for threshold BLS signatures*] For any n , $f < \frac{n}{3}$, $f \leq t \leq 2f$, and field \mathbb{Z}_p for $p > n$ there exists an Asynchronous Distributed Key Generation protocol that creates a $(t + 1, n)$ threshold BLS signature scheme, requires a PKI setup and a univariate powers-of-tau-setup (of size $O(\lambda n)$ words) that has optimal resilience against an adversary that can corrupt f parties.

The protocol is proven adaptively secure under the $2f + 1$ one-more discrete log (OMDL) assumption in the algebraic group model (AGM).

The protocol runs in a constant expected number of rounds, and uses $O(\lambda n^3)$ expected word complexity.

This theorem is proven in [Theorems 9](#) and [10](#).

Additional applications. While we limit ourselves to these two applications in this work, **Bingo** might be used to generally improve the efficiency of adaptively secure asynchronous MPC protocols [23]. As a more specific example, we consider the well studied problem of *MEV* (*maximal extractable value*) in blockchains [26, 43], wherein parties controlling the distribution of transactions (e.g., miners) can increase their rewards beyond the (expected) block reward and transaction fees by carefully choosing which transactions to include in a block and in which order. This issue is often addressed via the use of *front-running* protection mechanisms [27], which prevent these types of manipulations. Using **Bingo**, parties can share their transactions among a set of validators rather than sending them in the clear. Once a block gets finalized, the validators can batch-reconstruct all the transactions in the block, at a communication cost of $O(\lambda n^2)$ words. By using **Bingo**, there is thus only a constant overhead for adding front-running protection with adaptive security relative to sending the block via reliable broadcast at the same asymptotic cost of $O(\lambda n^2)$ words. Since VSS implies reliable broadcast [6], at least $\Omega(n^2)$ words are required [30].

Using this approach results in an *adaptively secure* MEV protection scheme. In contrast, alternative approaches that use encryption (or threshold encryption) seem to either have a static security proof or require rather complex encryption primitives.

1.1 Technical overview

The conceptual decomposition of distributed protocols to a distributed computing part against a weaker adversary and a cryptographic commitment and zero-knowledge part goes back to the foundational result of Goldreich, Micali, and

Wigderson [36]. Here we present a high-level overview of Bingo by decomposing it into two parts: an efficient distributed protocol that is resilient to *omission failures* (i.e., failures that are accidental) and an efficient polynomial commitment scheme that essentially forces the malicious adversary to behave as an omission adversary. We start in Section 3 with our polynomial commitment scheme, then show in Section 4 how to use it to get an AVSS, Bingo, that tolerates adaptive malicious adversaries. Our construction builds on the KZG polynomial commitment scheme [39], which means relying on a powers-of-tau setup [15]. Our public parameters are backwards compatible with prior universal setups [40].

Step one: Bingo for omission failures. In this setting, the goal is to share a degree- $2f$ polynomial among $3f + 1$ parties, f of which may suffer omission failures. Due to asynchrony, the dealer can interact with only $2f + 1$ parties, and since f of them may have omission failures, the remaining $f + 1$ honest parties need to enable all honest parties to eventually receive their share of the secret. The approach we take is for the dealer to share a bivariate polynomial $\phi(X, Y)$ of degree at most $2f$ in X and degree f in Y . Visually, we think of a matrix of size $n \times n$ of the evaluations of $\phi(X, Y)$ at roots of unity $\{\omega_1, \dots, \omega_n\}$, as shown in Figure 1. As such, we think of the polynomial $\phi(X, \omega_i)$ as the i -th *row* of the polynomial, which we denote by α_i , and the polynomial $\phi(\omega_i, Y)$ as the i -th *column* of the polynomial, which we denote by β_i . The dealer then sends each party i the i -th row. Each party can then wait for $2f + 1$ parties to acknowledge receiving their rows before knowing that they will be able to complete the protocol. This works because once $f + 1$ honest parties have their row we are guaranteed that all honest parties will eventually be able to recover their share in the following way: First, each honest party i that received a row from the dealer sends each party j the value $\phi(\omega_j, \omega_i)$. Hence each honest party j receives at least $f + 1$ points on its j -th column and is able to reconstruct it. Second, once party j reconstructs its column, it sends each party i the value $\phi(\omega_j, \omega_i)$. In this way, all honest parties eventually reconstruct their columns, so each honest party i hears at least $2f + 1$ values for row i and can reconstruct it.

As described, each party needs to send just $O(n)$ words and the protocol takes a constant number of rounds.

Step two: Bingo for malicious failures. In order to move from omission failures to malicious failures with adaptive security, we use a perfectly hiding and adaptively secure bivariate *polynomial commitment scheme* (PCS) that essentially forces the malicious parties to act as if they can only have omission failures.

Our bivariate PCS has five desirable properties: (1) it requires a standard $O(\lambda n)$ univariate powers-of-tau setup; (2) a commitment has size $O(\lambda n)$; (3) given a commitment, one can generate commitments to all rows; (4) given $f + 1$ evaluations on column j , one can generate evaluation proofs for all points of column j ; and (5) given $2f + 1$ evaluations on row i , one can generate evaluation

| | β_1 | β_2 | β_3 | β_4 | β_5 | β_6 | β_7 | | β_1 | β_2 | β_3 | β_4 | β_5 | β_6 | β_7 |
|------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| α_1 | v_{11} | v_{12} | v_{13} | v_{14} | v_{15} | v_{16} | v_{17} | α_1 | v_{11} | v_{12} | v_{13} | v_{14} | v_{15} | v_{16} | v_{17} |
| α_2 | v_{21} | v_{22} | v_{23} | v_{24} | v_{25} | v_{26} | v_{27} | α_2 | v_{21} | v_{22} | v_{23} | v_{24} | v_{25} | v_{26} | v_{27} |
| α_3 | v_{31} | v_{32} | v_{33} | v_{34} | v_{35} | v_{36} | v_{37} | α_3 | v_{31} | v_{32} | v_{33} | v_{34} | v_{35} | v_{36} | v_{37} |
| α_4 | v_{41} | v_{42} | v_{43} | v_{44} | v_{45} | v_{46} | v_{47} | α_4 | v_{41} | v_{42} | v_{43} | v_{44} | v_{45} | v_{46} | v_{47} |
| α_5 | v_{51} | v_{52} | v_{53} | v_{54} | v_{55} | v_{56} | v_{57} | α_5 | v_{51} | v_{52} | v_{53} | v_{54} | v_{55} | v_{56} | v_{57} |
| α_6 | v_{61} | v_{62} | v_{63} | v_{64} | v_{65} | v_{66} | v_{67} | α_6 | v_{61} | v_{62} | v_{63} | v_{64} | v_{65} | v_{66} | v_{67} |
| α_7 | v_{71} | v_{72} | v_{73} | v_{74} | v_{75} | v_{76} | v_{77} | α_7 | v_{71} | v_{72} | v_{73} | v_{74} | v_{75} | v_{76} | v_{77} |

Fig. 1. A graphical representation of Bingo’s sharing process showing the two ways in which party i can obtain their secret polynomial. The row polynomials are denoted by $\alpha_i = \phi(X, \omega_i)$ whereas the column polynomials are denoted by $\beta_j = (\omega_j, Y)$. On the left-hand side, party 2 receives α_2 directly from the (honest) dealer. On the right-hand side, party 2 did not receive their polynomial from the dealer. Instead i receives evaluations of the column polynomials β_j from at least $2f + 1$ other parties. Because $\beta_j(\omega_2) = \alpha_2(\omega_j)$, this is equivalent to obtaining $2f + 1$ evaluations of α_2 , meaning party 2 can obtain α_2 by interpolation.

proofs for all points in row i . Perhaps surprisingly, our PCS commits to a bivariate polynomial $\phi(X, Y)$ of degree f in each column and degree $2f$ in each row by simply committing to $f + 1$ specific rows, where a commitment to row i is just a standard KZG univariate polynomial commitment for $\phi(X, \omega_i)$ of degree $2f$. It is easy to see that this fulfills the first two properties. For the third property, we prove that interpolation in the exponent of any $f + 1$ row commitments generates commitments to all rows. In order to reduce computation costs, it is also possible to compute the interpolated coefficients and send them instead of sending the commitments. Every party can then only evaluate commitments in the exponent instead of interpolating $f + 1$ commitments and then evaluating the rest. This is the approach taken in the construction of our bivariate PCS.

From Bingo to VABA and ADKG Using Bingo’s $O(n^2)$ word complexity for packing $O(n)$ secrets allows giving each party a *random rank* based on secrets from $f + 1$ parties at a total cost of just $O(n^3)$ words. This allows implementing n adaptively secure *leader election* and *proposal election* protocols which in turn allows implementing a *VABA* protocol with $O(n^3)$ expected word complexity for $O(n)$ sized inputs. This construction uses the ability to individually reconstruct sums of secrets shared by different dealers.

To obtain an ADKG, each party uses Bingo’s $O(n^2)$ word complexity for a *high threshold* secret. Using the VABA protocol above on inputs formed from $f + 1$ completed high threshold sharings allows reaching agreement on a common BLS secret key formed from the sum of $f + 1$ high threshold secret sharings. Once agreement is reached, we reveal the BLS public key by using the standard “recovering in the exponent” technique. We prove that the resulting BLS sig-

natures scheme is adaptively secure using the framework of [5] relying on the $2f + 1$ one-more discrete log assumption and the algebraic group model.

We detail how to use Bingo to obtain a VABA protocol and then an ADKG in Section 5.

1.2 Related work

Cachin et al. [17] study asynchronous verifiable secret sharing (AVSS) in the computational setting. The earlier works of Feldman and Micali [31] and Canetti and Rabin [20] study AVSS in the private channel setting. Backes, Datta, and Kate [6] provide the first construction with asymptotically optimal $\mathcal{O}(\lambda n^2)$ word complexity for AVSS. They use the seminal pairing-based polynomial commitment scheme due to Kate, Zaverucha, and Goldberg (KZG) [39]. Compared to Backes et al., we provide the same asymptotically optimal $\mathcal{O}(\lambda n^2)$ word complexity with $\mathcal{O}(n)$ improvement in the size of the secret and a scheme that is proven to be adaptively secure.

AlHaddad, Varia, and Zhang [4] obtain a high-threshold AVSS for random secrets with $\mathcal{O}(n^2)$ word complexity. Moreover, their scheme can be instantiated with a setup-free polynomial commitment scheme [10, 11, 14, 16] at a $\mathcal{O}(n^2 \log n)$ word complexity. Our construction enables packed secret sharing, meaning that we can share n arbitrary secrets with $\mathcal{O}(n^2)$ word complexity whereas AlHaddad et al. can share only one random secret with the same complexity.

There has been considerable recent interest in practical ADKG and the secret sharing, reliable broadcast, and validated asynchronous Byzantine agreement (VABA) building blocks needed to obtain them. Kokoris-Kogias, Malkhi, and Spiegelman [41] obtain a high threshold ADKG with $\mathcal{O}(n^4)$ communication complexity and $\mathcal{O}(n)$ time. Gurkan et al. suggest an aggregatable publicly verifiable secret sharing (PVSS) scheme [37] that builds upon the SCRAPE PVSS of Cascudo and David [22] and can be used to create a DKG with a secret group element. Having a secret group element makes it incompatible with commonly used threshold cryptography schemes as they usually require field elements as secrets. Moreover, the only known security proofs for SCRAPE based schemes is for static adversaries. When combined with the consensus protocol of Abraham et al. [3], this yields a high-threshold ADKG with $\mathcal{O}(n^3 \log n)$ communication complexity and $\mathcal{O}(1)$ expected time that is secure against static adversaries. Cascudo and David [21] introduce Albatross, which uses packed secret sharing to build a randomness beacon that shares $\mathcal{O}(n^2)$ random values. Albatross also uses the SCRAPE PVSS as a backend and thus cannot be used to share a field element (and has a static security proof). Das, Xiang, and Ren [28] improve the reliable broadcast primitive and among other improvements, this remove the log factor from [3] to get $\mathcal{O}(n^3)$ communication complexity and $\mathcal{O}(1)$ expected time. Das et al. [29] provide a high-threshold DKG that works with a field element secret and $\mathcal{O}(n^3)$ word complexity. Even though experiments report good performance in some cases, under attack, the scheme has a larger $\log(n)$ expected time complexity. Again the security proof is only against a static adversary. Yurek et al. [45] obtain AVSS protocols that are (quasi)linear in both computation and

communication overhead in an amortized sense. Again the security proof is only against a static adversary.

In order to pack many secrets (or a high threshold polynomial), Bingo uses the by-now-standard technique of a bivariate secret sharing with degree f in one dimension and degree $2f$ in the other [1, 2, 38].

Yurek et al. [45] also provide a variant protocol called hbACSS1 that. Like Bingo, hbACSS1 uses the KZG polynomial commitment scheme, and uses bivariate polynomials. Unlike Bingo, which uses both rows and columns during the sharing, in hbACSS1 each party only receives a row during a fault-free sharing, and parties only learn their columns (along with the full bivariate polynomial) when the dealer is proven to be faulty in order to make sure parties receive their shares. In addition, hbACSS1 uses an f -by- f polynomial, whereas Bingo uses a $2f$ -by- f polynomial.

Obtaining an ADKG with $O(n^3)$ words complexity and $O(1)$ expected time that is secure against an adaptive adversary and generates a secret field element remained a major open question in this sub-domain. To the best of our knowledge, only synchronous DKG protocols have been proven to be adaptively secure, and they all have a complexity of $\Omega(n^4)$ sent words [5, 19].

2 Definitions

In this section we start by defining basic notation, and then defining polynomial commitment schemes and reliable broadcast as basic building blocks to be used in our constructions. Following that, we discuss the way we model interactive protocols in order to finally define packed asynchronous verifiable secret sharing.

2.1 Preliminaries

For a finite set S , we denote by $|S|$ its size and by $x \xleftarrow{\$} S$ the process of sampling a member uniformly from S and assigning it to x . Further, $\lambda \in \mathbb{N}$ denotes the security parameter and 1^λ denotes its unary representation. For two integers $i \leq j$, we define $[i, j] = \{i, \dots, j\}$. PPT stands for probabilistic polynomial time. By $y \leftarrow A(x_1, \dots, x_n)$ we denote running algorithm A on inputs x_1, \dots, x_n and assigning its output to y , and by $y \xleftarrow{\$} A(x_1, \dots, x_n)$ we denote running $A(x_1, \dots, x_n; R)$ for a uniformly random tape R . Adversaries are modeled as randomized algorithms. We use code-based games in our security definitions [8]. A game $G_{\mathcal{A}}^{\text{sec}}(\lambda)$, played with respect to a security notion sec and adversary \mathcal{A} , has a MAIN procedure whose output is the output of the game. $\Pr[G_{\mathcal{A}}^{\text{sec}}(\lambda)]$ denotes the probability that this output is equal to 1. As is standard, for every $n \in \mathbb{N}$, we define $[n] = \{1, \dots, n\}$.

Our constructions rely on the discrete logarithm assumption (dlog) which says that it is hard to output x given g^x , where g is a generator of a group \mathbb{G} of prime order p and $x \xleftarrow{\$} \mathbb{F}_p$. We also rely on the q -strong Diffie-Hellman assumption (q-sdh) [12], which says that it is hard to output a pair $(c, g^{1/(x+c)})$ given

$(g, g^x, g^{x^2}, \dots, g^{x^q}, \hat{g}, \hat{g}^x) \in \mathbb{G}_1^{q+1} \times \mathbb{G}_2^2$, where \mathbb{G}_1 and \mathbb{G}_2 are groups of prime order p , generated by g and \hat{g} , and form a bilinear group, q is an integer, and $x \xleftarrow{\$} \mathbb{F}_p$. Finally, our DKG application relies on the k -one-more discrete logarithm (omdl) assumption [7], which says that it is hard to output $(x_1, \dots, x_k) \in \mathbb{F}_p^k$ given $(g, g^{x_1}, \dots, g^{x_k}) \in \mathbb{G}^{k+1}$, where g is a generator of a group \mathbb{G} of prime order p and $x_1, \dots, x_k \xleftarrow{\$} \mathbb{F}_p$, and at most $k - 1$ queries to a discrete log oracle DL that on input X outputs $\log_g(X)$.

Our constructions are proved secure in the algebraic group model (AGM) [33]. In the AGM, whenever an adversary outputs a group element it must output the accompanying algebraic representation of that element relative to all the group elements it has seen thus far; i.e., if it has seen X_1, \dots, X_m then upon outputting a new element Y it must output a_1, \dots, a_m such that $Y = \prod_i X_i^{a_i}$.

We define $\omega_1, \dots, \omega_n$ to be n different roots of unity of order $n + f$. In a slight abuse of notation, we define ω_0 to be 0 and $\omega_{-f}, \dots, \omega_{-1}$ to be the remaining f roots of unity of order $n + f$.

2.2 Polynomial commitments

We define a *polynomial commitment scheme* (PCS) as consisting of the following algorithms:

- $\text{srs} \xleftarrow{\$} \text{Setup}(1^\lambda)$ takes as input a security parameter and outputs a commitment key srs .
- $\mathbf{C} \xleftarrow{\$} \text{Commit}(\text{srs}, \phi)$ takes as input the commitment key and a polynomial ϕ and outputs a commitment \mathbf{C} . We often specify the randomness explicitly using the notation $\mathbf{C} \leftarrow \text{Commit}(\text{srs}, \phi, \hat{\phi})$.
- $m, \hat{m}, \pi \leftarrow \text{Eval}(\text{srs}, \phi, \hat{\phi}, \omega)$ takes as input a commitment key, a pair of polynomials, and a point on which to evaluate. It returns $m = \phi(\omega)$, $\hat{m} = \hat{\phi}(\omega)$ and a proof π that m, \hat{m} are consistent with ω .
- $0/1 \leftarrow \text{Verify}(\text{srs}, \mathbf{C}, \omega, m, \hat{m}, \pi)$ takes as input a commitment key, a commitment, an opening point, a pair of openings, and a proof π . It returns 1 if it is convinced that (m, \hat{m}) is a valid opening of \mathbf{C} at ω and 0 otherwise.

In what follows, we often omit the commitment key srs as an explicit input to the other algorithms. Following Kate et al. [39], we require that a PCS satisfies *correctness*, meaning that $\text{Verify}(\text{Commit}(\phi, \hat{\phi}), \omega, \text{Eval}(\phi, \hat{\phi}, \omega)) = 1$ and both *polynomial binding* and *evaluation binding*. These say, respectively, that an adversary cannot open a single commitment to two different values and that an adversary cannot output two valid but incompatible evaluations of the same pair of polynomials, as represented by a single commitment.

Definition 1 (Polynomial binding). [39] *Consider a game $\mathbb{G}_A^{\text{poly-binding}}(\lambda)$ in which an adversary \mathcal{A} takes 1^λ as input and outputs the tuple $(\phi_1, \hat{\phi}_1, \phi_2, \hat{\phi}_2)$, and wins if (1) $\text{Commit}(\phi_1, \hat{\phi}_1) = \text{Commit}(\phi_2, \hat{\phi}_2)$ and (2) $(\phi_1, \hat{\phi}_1) \neq (\phi_2, \hat{\phi}_2)$. We say the PCS satisfies polynomial binding if for all PPT adversaries \mathcal{A} there exists a negligible function $\nu(\cdot)$ such that $\Pr[\mathbb{G}_A^{\text{poly-binding}}(\lambda)] < \nu(\lambda)$.*

Definition 2 (Evaluation binding). [39] Consider a game $G_{\mathcal{A}}^{eval-binding}(\lambda)$ in which an adversary \mathcal{A} takes 1^λ as input and outputs $(\mathbf{C}, \omega, m_1, \hat{m}_1, \pi_1, m_2, \hat{m}_2, \pi_2)$, and wins if (1) $\text{Verify}(\mathbf{C}, \omega, m_i, \hat{m}_i, \pi_i) = 1$ for $i \in \{1, 2\}$ and (2) $(m_1, \hat{m}_1) \neq (m_2, \hat{m}_2)$. We say the PCS satisfies evaluation binding if for all PPT adversaries \mathcal{A} there exists a negligible function $\nu(\cdot)$ such that $\Pr[G_{\mathcal{A}}^{eval-binding}(\lambda)] < \nu(\lambda)$.

We define another important property for a PCS, *interpolation binding*, which says that given enough evaluations of a committed pair of polynomials, the interpolated polynomials obtained from these evaluations must be the ones contained inside the commitment. For this we use the notation $p \leftarrow \text{Interpolate}(\{\omega_i, y_i\}_i)$ to denote using Lagrange interpolation to obtain a degree- d polynomial given $d + 1$ evaluation points and their corresponding evaluations.

Definition 3 (Interpolation binding). Consider the game $G_{\mathcal{A}}^{int-binding}(1^\lambda)$ defined as follows

```

MAIN( $1^\lambda, d$ )
srs  $\xleftarrow{\$}$  Setup( $1^\lambda, d$ )
 $(\mathbf{C}, \{\omega_i, m_i, \hat{m}_i, \pi_i\}_{i \in [d+1]}) \xleftarrow{\$} \mathcal{A}(\text{srs})$ 
 $p(X) \leftarrow \text{Interpolate}(\{\omega_i, m_i\}_{i \in [d+1]})$ 
 $\hat{p}(X) \leftarrow \text{Interpolate}(\{\omega_i, \hat{m}_i\}_{i \in [d+1]})$ 
check  $\omega_i \neq \omega_j$  for all  $i \neq j$ 
check  $\text{Verify}(\text{srs}, \mathbf{C}, \omega_i, m_i, \hat{m}_i, \pi_i) = 1$  for all  $i \in [d + 1]$ 
check  $\mathbf{C} \neq \text{Commit}(\text{srs}, p(X); \hat{p}(X))$ 
if all checks pass return 1, else return 0

```

We say the PCS satisfies interpolation binding if for all PPT adversaries \mathcal{A} there exists a negligible function $\nu(\cdot)$ such that $\Pr[G_{\mathcal{A}}^{int-binding}(1^\lambda)] < \nu(\lambda)$.

In our construction of Bingo, we do not use the hiding property defined by Kate et al. as it did not fit our use case. We instead provide a new hiding definition, capturing the ability of a simulator to both open commitments and provide evaluations without knowledge of the underlying polynomials. As this definition is somewhat specific to our usage of KZG within Bingo, and in particular to the way in which it is embedded in a bivariate polynomial commitment (as described below), it can be found in Appendix A.3.

In a *bivariate* PCS, ϕ is a polynomial in indeterminates X and Y . This means we consider an additional algorithm:

- $\mathbf{A} \leftarrow \text{PartialEval}(\text{srs}, \mathbf{C}, V_n)$ takes as input the commitment key, the bivariate commitment, and a set of partial evaluation points V_n of size n . It outputs n partial evaluations, consisting of commitments to univariate polynomials $\alpha(X) \leftarrow \phi(X, v)$ and $\hat{\alpha}(X) \leftarrow \hat{\phi}(X, v)$ for each $v \in V_n$.

To prove evaluations of ϕ and $\hat{\phi}$, we can use these univariate polynomials as input to Eval, and their commitments as input to Verify (which must now take in two evaluation points ω and ω_v rather than a single one). In terms of

correctness, we define an algorithm $\mathbf{A} \leftarrow \text{CPE}(\phi, \hat{\phi}, V_n)$ that first runs **Commit** on ϕ and $\hat{\phi}$ and then runs **PartialEval** on its output \mathbf{C} and V_n . We then require that $\text{Verify}(\text{CPE}(\phi, \hat{\phi}, V_n), (\omega, v), \text{Eval}(\phi(X, v), \hat{\phi}(X, v), \omega)) = 1$ for all $v \in V_n$.

2.3 Reliable broadcast

A *reliable broadcast* is an asynchronous protocol with a designated *sender*. The sender has some input value m from a known domain \mathcal{M} and each party may output a value in \mathcal{M} . A reliable broadcast has the following properties assuming all *nonfaulty* (i.e., uncorrupted) parties participate in the protocol:

- **Validity.** If the sender is nonfaulty, then every nonfaulty party that completes the protocol outputs the sender’s input value m .
- **Agreement.** The values output by any two nonfaulty parties are the same.
- **Termination.** If the dealer is nonfaulty, then all nonfaulty parties complete the protocol and output a value. Furthermore, if some nonfaulty party completes the protocol, every nonfaulty party completes the protocol.

2.4 Interactive protocols with adaptive corruptions

At its heart, a verifiable secret sharing (VSS) scheme is an interactive protocol between multiple parties. In this protocol, we assume that all messages sent between parties are encrypted and authenticated, and identify (at least) both their sender and recipient. We leave it as optional for the message to specify other metadata, such as the index of a concurrent session.

For each party i , we define their local state as state_i , which initially consists of their own private key, the public keys of all other parties, and their random tape. We denote by trans_i the transcript of i , which is an ordered list of their sent and received messages, and define their local *view* view_i as the pair $(\text{trans}_i, \text{state}_i)$. To capture how this view evolves, we denote by $\text{view}_{i,t}$ the view of the i -th party at the t -th step of the protocol; the party does not know themselves at which step they are but this is defined in a global sense. We define view_t to be the *global* view of the protocol at step t , which contains $\text{view}_{i,t}$ for all nonfaulty parties i .

To model an adversary participating in an interactive protocol, we consider a game that allows them to send messages to nonfaulty parties and, indirectly, cause nonfaulty parties to send messages to each other. To capture the strongest adversary, we allow them to both (1) have complete control over the scheduling of messages and (2) adaptively corrupt nonfaulty parties. To capture this first ability, we provide the adversary with an oracle $\mathcal{O}_{\text{buffer}}$ that acts as a message buffer. The adversary can query this oracle in three different ways: (1) $\mathcal{O}_{\text{buffer}}(\text{add}, x)$ adds x to the message buffer, and thus allows an adversary to send a message to a nonfaulty party from a party it controls; (2) $\mathcal{O}_{\text{buffer}}(\text{deliver}, j)$ delivers the j -th message in the buffer; and (3) $\mathcal{O}_{\text{buffer}}(\text{see})$ shows the adversary the contents of the message buffer. When messages are delivered to a nonfaulty party, the game also runs the code for that party on that message. This may in turn result in updates to their internal state and in new messages being added to the

buffer and thus allows nonfaulty parties to send messages to each other (with the adversary controlling the delay with which they are delivered).

To capture the ability to corrupt nonfaulty parties, we have the game start with an initial set \mathcal{NF} of nonfaulty parties. We then provide the adversary with access to an oracle $\mathcal{O}_{\text{corr}}$ that, when queried on a given index i at step t , provides the adversary with $\text{view}_{i,t}$ and removes i from \mathcal{NF} .

2.5 Packed asynchronous verifiable secret sharing (PAVSS)

In a verifiable secret sharing (VSS) scheme, a *dealer* can *share* a secret s with n parties such that if some threshold t of the parties interact they can *reconstruct* the original secret s . In an *asynchronous* VSS (AVSS), both of these interactions can happen even if parties are online at different times and messages are thus sent and delivered on an arbitrary schedule. Finally, in a *packed* AVSS (PAVSS), a dealer can share multiple secrets in a way that is more cost-efficient than just running multiple concurrent sessions of the original protocol.

More formally, we define a packed AVSS using two interactive protocols that take place between n parties: **Share** and **Reconstruct**. In **Share**, the designated dealer receives as input a set of secrets s_0, \dots, s_m from a finite field \mathbb{F} and all other parties receive no input. None of the parties have any output at the end of **Share**, but they do update their local state. Because the AVSS is packed, there are $m + 1$ possible invocations of **Reconstruct**, one for each index k . Each party thus provides k as input to the protocol, and has as output a field element $v_k \in \mathbb{F}$, which represents their local view of the k -th secret shared by the dealer.

To capture security definitions for a packed AVSS, we follow the outline described in Section 2.4, starting with a global view view . We augment the view of each nonfaulty party i with $2m + 4$ *flags* designed to capture how far this party has made it in the protocol: first, startedS_i and finishedS_i keep track of whether or not they have, respectively, started and finished the **Share** protocol. We then have $m + 1$ flags $\text{readyR}_{i,k}$ representing whether or not the i -th party is ready to start **Reconstruct**(k), and another $m + 1$ flags $\text{finishedR}_{i,k}$ representing whether or not they have finished **Reconstruct**(k).

A party’s readiness to invoke **Reconstruct**(k) depends on the context in which the VSS is used; we can think of an external predicate $P(k)$ that tells the party if the k -th secret is ready to be reconstructed or not. The i -th party then invokes **Reconstruct**(k) once $\text{finishedS}_i = \text{readyR}_{i,k} = \text{true}$. To continue giving an adversary as much control as possible, we allow it to also determine when these external predicates evaluate to true, as well as when parties start the **Share** protocol. This means that in addition to the $\mathcal{O}_{\text{buffer}}$ and $\mathcal{O}_{\text{corr}}$ oracles described above we give the adversary access to two oracles: \mathcal{O}_S and \mathcal{O}_R . On input $i \in \mathcal{NF}$, \mathcal{O}_S sets $\text{startedS}_i \leftarrow \text{true}$ and has the party run the first step of **Share** according to their current view. On input $i \in \mathcal{NF}$ and $k \in [m]$, \mathcal{O}_R sets $\text{readyR}_{i,k} \leftarrow \text{true}$. It then also runs the first step of **Reconstruct**(k) if $\text{finishedS}_i = \text{true}$; if not, this gets run once a call to $\mathcal{O}_{\text{buffer}}(\text{deliver}, j)$ causes the i -th party to finish **Share**.

All our security definitions below thus follow the same implicit game structure, in a “meta” game that we call $\mathbb{G}_{\mathcal{A}}^{\text{PAVSS}}(\lambda)$: a PPT adversary \mathcal{A} plays a game

with access to four oracles: the message buffer $\mathcal{O}_{\text{buffer}}$, the corruption oracle $\mathcal{O}_{\text{corr}}$, and the two oracles \mathcal{O}_S and \mathcal{O}_R that dictate when nonfaulty parties are ready to start participating in, respectively, the Share and Reconstruct protocols. In every game, security holds only if an adversary can corrupt at most f parties.

Our first definition, *termination*, sets the conditions under which nonfaulty parties can be guaranteed to complete Share and Reconstruct.

Definition 4 (Termination). *We say that the scheduling in $G_{\mathcal{A}}^{\text{PAVSS}}(\lambda)$ is valid if the message buffer is empty at the point at which \mathcal{A} signals that the game is over; i.e., all messages were eventually delivered. In every iteration of the game with valid scheduling, we then require the following three properties to hold with all but negligible probability:*

1. *If the dealer is nonfaulty and all nonfaulty parties start Share, then all non-faulty parties eventually complete Share.*
2. *If some nonfaulty party completes Share, and all nonfaulty parties start Share then they all eventually complete Share as well.*
3. *If all nonfaulty parties complete protocol Share and invoke Reconstruct(k) for some $k \in [0, m]$, they all eventually complete Reconstruct(k).*

Our next definition, *correctness*, captures the requirement that all nonfaulty parties who complete Reconstruct(k) should agree on the same secret, which in turn should be the same as the one used by the dealer (if it was also nonfaulty).

Definition 5 (Correctness). *Define an extractor Ext that is given the state of the nonfaulty parties at the point at which the first nonfaulty party has completed Share, and outputs $m + 1$ values $r_0, \dots, r_m \in \mathbb{F}$. In other words, define t as the first step at which view_t contains $\text{finishedS}_i = \text{true}$ for some index i , and define Ext such that $r_0, \dots, r_m \stackrel{\$}{\leftarrow} \text{Ext}(\text{view}_t)$. There exists a polynomial-time Ext for all adversaries \mathcal{A} such that the following holds for every $k \in [0, m]$ with all but negligible probability in the security parameter:*

1. *If the dealer is nonfaulty, $r_k = s_k$.*
2. *Every nonfaulty party that completes protocol Reconstruct(k) outputs the value r_k .*

Our final definition, *secrecy*, captures the requirement that an adversary should not be able to learn anything about the k -th secret until the point at which some nonfaulty party invokes Reconstruct(k). Our definition is simulation-based, which means that the adversary first picks the secrets and must then provide them to an oracle $\mathcal{O}_{\text{init}}$. During Share the adversary either takes part in the honest protocol, with a dealer who knows its provided set of secrets, or interacts with a simulator who does not. During Reconstruct(k), however, the adversary would be able to trivially distinguish between parties that have some or no information about the secret. Our game thus provides the k -th secret to the simulator only at the point at which the first nonfaulty party invokes Reconstruct(k); i.e., at the latest possible step before it would become trivial for the adversary to determine if it were interacting with the simulator or not. Equally, we also need to provide all the secrets to the simulator if the adversary corrupts the dealer.

Definition 6 (Secrecy). Define a simulator Sim that interacts with \mathcal{A} on behalf of all nonfaulty parties, including the dealer. Then define $\text{Adv}_{\mathcal{A}}^{\text{secrecy}}(\lambda) = 2\Pr[\mathbf{G}_{\mathcal{A}}^{\text{secrecy}}(\lambda)] - 1$, where $\mathbf{G}_{\mathcal{A}}^{\text{secrecy}}(\lambda)$ is defined as follows, assuming the dealer is party 1 who deals only after receiving a “start” message from the adversary and omitting the code for the oracles in which the game does not deviate from their earlier descriptions:

| | |
|---|--|
| $\frac{\text{MAIN } \mathbf{G}_{\text{secrecy}}^{\mathcal{A}}(\lambda)}{b \xleftarrow{\$} \{0, 1\}; \vec{S} \leftarrow \vec{\varepsilon}}$ $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\text{buffer}}(\text{add}, \cdot), \mathcal{O}_{\text{buffer}}(\text{see}), \mathcal{O}^*, \mathcal{O}_{\text{corr}}, \mathcal{O}_{\text{init}}, \mathcal{O}_S, \mathcal{O}_R}(1^\lambda)$ $\text{return } (b' = b)$ | $\frac{\mathcal{O}^*(m)}{\text{if } (b = 0) \text{ run } \text{Sim}(\tau_s, m)}$ $\text{else run } \mathcal{O}_{\text{buffer}}(\text{deliver}, m)$ |
| $\frac{\mathcal{O}_{\text{init}}(s_0, \dots, s_m)}{S_i \leftarrow s_i \ \forall i \in [m]}$ | $\frac{\mathcal{O}_R(i, k)}{\text{readyR}_{i,k} \leftarrow \text{true}}$ |
| $\frac{\mathcal{O}_{\text{corr}}(i)}{\text{if } (i = 1) \text{ add } S_1, \dots, S_m \text{ to } \text{state}_{\text{Sim}}}$ $\text{remove } i \text{ from } \mathcal{NF}$ $\text{return } \text{state}_i$ | $\text{if } (\text{finishedS}_i = \text{true})$ $\text{add } S_k \text{ to } \text{state}_{\text{Sim}}$ $\text{start } \text{Reconstruct}(k) \text{ for party } i$ |

Secrecy holds if for all PPT adversaries \mathcal{A} there exists a negligible function $\nu(\cdot)$ such that $\text{Adv}_{\mathcal{A}}^{\text{secrecy}}(\lambda) < \nu(\lambda)$.

3 A Bivariate Polynomial Commitment Scheme

3.1 Construction

Our construction for a bivariate polynomial commitment scheme, given in Figure 3, builds heavily on top of the univariate PCS due to Kate et al. [39]. As such, we first present this construction in Figure 2.

In both commitment schemes, the setup outputs universal powers-of-tau parameters [15], meaning they are backwards compatible with prior trusted setups [40]. Let $\phi(X, Y)$ be a bivariate polynomial with degree d_1 in X and degree d_2 in Y . A commitment to $\phi(X, Y)$ first decomposes $\phi(X, Y)$ into $d_2 + 1$ univariate polynomials $\phi_i(X)$ such that $\phi(X, Y) = \sum_{i=0}^{d_2} \phi_i(X)Y^i$. The randomness $\hat{\phi}(X, Y)$ is decomposed in the same manner. Then each of the $\phi_i(X)$ are committed to using KZG.Commit with randomness $\hat{\phi}_i(X)$. A commitment \mathcal{C} such that $|\mathcal{C}| = d_2$ has maximum degree d_2 in Y and d_1 in X .

The partial evaluation algorithm takes as input a commitment \mathcal{C} and a set of distinct points V_n of size n . It then runs a discrete Fourier transform (DFT) that maps a polynomial to a set of evaluations. Because the DFT/iDFT algorithm is a linear transformation, it can be applied to (homomorphic) group exponents in the exact same way as it is run for field elements, without having to know the discrete logarithms. To avoid confusion, we nevertheless denote the algorithms

| | |
|---|---|
| <p>KZG.Setup(bp, d_1)</p> $\tau, x \xleftarrow{\$} \mathbb{F}$ $\hat{g} \leftarrow g^x$ $\text{srs} \leftarrow (\text{bp}, \{g^{\tau^i}, \hat{g}^{\tau^i}, h^{\tau^i}\}_{i=0}^{d_1})$ return srs | <p>KZG.Eval(srs, $\alpha(X), \hat{\alpha}(X), \omega_i$)</p> $m \leftarrow \alpha(\omega_i)$ $\hat{m} \leftarrow \hat{\alpha}(\omega_i)$ $q(X) \leftarrow (\alpha(X) - m)/(X - \omega_i)$ $\hat{q}(X) \leftarrow (\hat{\alpha}(X) - \hat{m})/(X - \omega_i)$ $\pi \leftarrow g^{q(\tau)} \hat{g}^{\hat{q}(\tau)}$ return (m, \hat{m}, π) |
| <p>KZG.Commit(srs, $\alpha(X); \hat{\alpha}(X)$)</p> $C \leftarrow g^{\alpha(\tau)} \hat{g}^{\alpha(\tau)}$ return C | <p>KZG.Verify(srs, $C, \omega_i, m, \hat{m}, \pi$)</p> if $e(Cg^{-m} \hat{g}^{-\hat{m}}, h) = e(\pi, h^{\tau-\omega_i})$ return 1 else return 0 |

Fig. 2. The hiding univariate KZG polynomial commitment scheme.

acting on field elements as DFT and iDFT and the algorithms acting on group elements as DFTExp and iDFTExp. PartialEval thus runs and outputs

$$\text{DFTExp} : ((g^{a_0}, \dots, g^{a_{d_1}}), \mathbb{F}^n) \mapsto \{g^{\sum_{j=0}^{d_1} a_j \omega_j^i}\}_{i=0}^{n-1}.$$

If V_n is a multiplicative subgroup of \mathbb{F} containing roots of unity, then DFT and DFTExp run in time $n \log(n)$. If $W \subset V_n$ is a subset of roots of unity, then interpolation over W runs in time $n \log^2(n)$ [34]. In addition to the interpolation algorithm $p \leftarrow \text{Interpolate}(\{(\omega_i, y_i)\})$, we denote by $P \leftarrow \text{InterpolateExp}(\{(\omega_i, Y_i)\})$ the algorithm that performs these operations in the exponent (i.e., by acting on group elements). We also denote by $Y \leftarrow \text{EvalExp}(\omega, P)$ the algorithm that performs polynomial evaluation in the exponent.

To verify that a commitment opens at (ω_i, ω_j) to (m, \hat{m}) , we take as input the partial evaluation \mathbf{A} over the set V_n where $\omega_j \in V_n$. Then $\mathbf{A}_j = g^{\phi(\tau, \omega_j)} \hat{g}^{\hat{\phi}(\tau, \omega_j)}$ is a KZG commitment to $\phi(X, \omega_j)$ under randomness $\hat{\phi}(X, \omega_j)$. The prover can thus provide a KZG opening proof that \mathbf{A}_j opens at ω_i to m under randomness \hat{m} (i.e., the output of KZG.Eval), which the verifier can check using KZG.Verify.

The security of our bivariate PCS follows directly from the security of the KZG univariate PCS, in terms of polynomial binding, evaluation binding, and hiding, which follow in turn from the dlog and q-sdh assumptions. We next prove, in the algebraic group model [33], that the underlying univariate PCS also satisfies interpolation binding. A proof of this lemma can be found in Appendix A.1.

Lemma 1. *If the dlog and q-sdh assumptions hold, then interpolation binding (Definition 3) holds for the KZG PCS.*

3.2 Commitment and proof interpolation

For any bivariate polynomial $\phi(X, Y)$ of degree d_1 in X we have that the points

$$\phi(\omega_{v_1}, \omega_j), \dots, \phi(\omega_{v_{d_1+1}}, \omega_j)$$

| | |
|---|---|
| $\text{Setup}(\text{bp}, d_1)$ return $\text{KZG.Setup}(\text{bp}, d_1)$ | $\text{Commit}(\text{srs}, \phi(X, Y); \hat{\phi}(X, Y))$ $\sum_{i=0}^{d_2} \phi_i(X) Y^i \leftarrow \text{parse}(\phi(X, Y))$ $\sum_{i=0}^{d_2} \hat{\phi}_i(X) Y^i \leftarrow \text{parse}(\hat{\phi}(X, Y))$ $\mathbf{C} \leftarrow \{g^{\phi_i(\tau)} \hat{g}^{\hat{\phi}_i(\tau)}\}_{i=0}^{d_2}$ return \mathbf{C} |
| $\text{PartialEval}(\text{srs}, \mathbf{C}, V_n)$ $\mathbf{A} \leftarrow \text{DFTExp}(\mathbf{C}, V_n)$ return \mathbf{A} | $\text{Verify}(\text{srs}, \mathbf{A}, (i, j), m, \hat{m}, \pi)$ return $\text{KZG.Verify}(\text{srs}, \mathbf{A}_j, \omega_i, m, \hat{m}, \pi)$ |
| $\text{Eval}(\text{srs}, \alpha(X), \hat{\alpha}(X), \omega_i)$ return $\text{KZG.Eval}(\text{srs}, \alpha(X), \hat{\alpha}(X), \omega_i)$ | $\text{GetProofs}(\{(w_i, y_i, \hat{y}_i, \pi_i)\}_{i \in [f+1]}, V_n)$ $\beta(X) \leftarrow \text{Interpolate}(\{(w_i, y_i)\}_{i \in [d_1+1]})$ $\hat{\beta}(X) \leftarrow \text{Interpolate}(\{(w_i, \hat{y}_i)\}_{i \in [d_1+1]})$ $\mathbf{P} \leftarrow \text{InterpolateExp}(\{(w_i, \pi_i)\}_{i \in [d_1+1]})$ $z_1, \dots, z_n \leftarrow \text{DFT}(\beta(X), V_n)$ $\hat{z}_1, \dots, \hat{z}_n \leftarrow \text{DFT}(\hat{\beta}(X), V_n)$ $\bar{\pi}_1, \dots, \bar{\pi}_n \leftarrow \text{DFTExp}(\mathbf{P}, V_n)$ return $\{(z_i, \hat{z}_i, \bar{\pi}_i)\}_{i \in [n]}$ |

Fig. 3. Our bivariate PCS, built on top of the KZG univariate PCS. The set V_n consists of n roots of unity, i.e., values ω_i such that $\omega_i^n = 1$.

suffice to interpolate the partial evaluation $\phi(X, \omega_j)$. A special property about our bivariate PCS is that, given a commitment \mathbf{C} and $d_1 + 1$ openings (with respect to the same ω_j), parties can also compute the opening proofs for \mathbf{C} at (x, ω_j) for any $x \in \mathbb{F}$. This will be useful in Bingo when the dealer is dishonest.

In Figure 3 we describe an additional algorithm $\{(z_i, \hat{z}_i, \bar{\pi}_i)\} \leftarrow \text{GetProofs}(\{(v_i, y_i, \hat{y}_i, \pi_i)\}, V_n)$ that takes as input $d_1 + 1$ opening points, their evaluations and associated proofs, and a set V_n , and outputs n evaluations and their associated proofs over the bigger set V_n . In Lemma 2 we prove the correctness of this algorithm, namely that if every opening (y_i, \hat{y}_i, π_i) verifies with respect to the commitment \mathbf{C} and the indices (j, w_i) , then every output $(z_k, \hat{z}_k, \bar{\pi}_k)$ also verifies with respect to (\mathbf{C}, k, j) . A proof of this lemma can be found in Appendix A.2.

Lemma 2. *Let \mathbf{C} be a bivariate polynomial commitment, let \mathbf{A} be such that $\mathbf{A} \leftarrow \text{PartialEval}(\mathbf{C}, V_n)$, let v_i be indices such that $w_i = \omega_{v_i}$ for every $i \in [d_1 + 1]$, and let $\{(v_i, y_i, \hat{y}_i, \pi_i)\}_{i \in [d_1+1]}$ be values such that*

$$\forall i \in [d_1 + 1] \text{VerifyEval}(\mathbf{A}, (j, v_i), y_i, \hat{y}_i, \pi_i) = 1.$$

If

$$\{(z_i, \hat{z}_i, \bar{\pi}_i)\}_{i \in [n]} \leftarrow \text{GetProofs}(\{(w_i, y_i, \hat{y}_i, \pi_i)\}_{i \in [d_1+1]}, V_n)$$

then $\forall k \in [n], \text{VerifyEval}(\mathbf{A}, (j, k), \beta_j(\omega_k), \hat{\beta}_j(\omega_k), \bar{\pi}_k) = 1$.

Below we prove an additional useful property of our bivariate PCS, namely that by performing interpolation in the exponent on the partial (univariate) commitments we can recover the bivariate commitment.

Lemma 3. *Let $v_1, \dots, v_{d_2+1} \in [n]$ be distinct values, and let $\alpha_{v_1}(X), \dots, \alpha_{v_{f+1}}(X)$ and $\hat{\alpha}_{v_1}(X), \dots, \hat{\alpha}_{v_{f+1}}(X)$ be polynomials of degree no greater than d_1 . Define $\phi(X, Y), \hat{\phi}(X, Y)$ to be the unique bivariate polynomials of degree d_1 in X and d_2 in Y such that $\forall i \in [d_1+1] \alpha_{v_i}(X) = \phi(X, \omega_{v_i}), \hat{\alpha}_{v_i}(X) = \hat{\phi}(X, \omega_{v_i})$. If $\forall i \in [d_2+1] D_i = \text{Commit}(\alpha_{v_i}(X); \hat{\alpha}_{v_i}(X))$ and $\mathbf{C} = \text{InterpolateExp}(\{(\omega_{v_i}, D_i)\}_{i \in [f+1]})$, then $\mathbf{C} = \text{Commit}(\text{srs}, \phi(X, Y); \hat{\phi}(X, Y))$.*

Proof. First note that

$$\phi(\tau, Y) = \text{Interpolate}(\{(\omega_{v_i}, \phi(\tau, \omega_{v_i}))\}_{i \in [f+1]}).$$

By construction

$$\begin{aligned} D_i &= \text{Commit}(\text{srs}, \alpha_{v_i}; \hat{\alpha}_{v_i}) = g^{\alpha_{v_i}(\tau)} \hat{g}^{\hat{\alpha}_{v_i}(\tau)} \\ &= g^{\phi(\tau, \omega_{v_i}) + x \hat{\phi}(\tau, \omega_{v_i})} \end{aligned}$$

where $\hat{g} = g^x$. Thus

$$(g^{\phi_0(\tau) + x \hat{\phi}_0(\tau)}, \dots, g^{\phi_f(\tau) + x \hat{\phi}_f(\tau)}) = \text{InterpolateExp}(\{(\omega_{v_i}, D_i)\}_{i \in [f+1]}).$$

This shows the lemma because

$$(g^{\phi_0(\tau) + x \hat{\phi}_0(\tau)}, \dots, g^{\phi_f(\tau) + x \hat{\phi}_f(\tau)}) = \text{Commit}(\text{srs}, \phi(X, Y); \hat{\phi}(X, Y)).$$

4 Bingo: Packed Asynchronous Verifiable Secret Sharing

In this section we present Bingo, our packed AVSS scheme. We discuss its design in Section 4.1 and its security in Section 4.2.

4.1 Design

Bingo consists of a sharing protocol `BingoShare` (Algorithm 2), and a reconstruction protocol `BingoReconstruct` (Algorithm 3). Additional reconstruction protocols for reconstructing sums of secrets and batch reconstructing are presented in Algorithm 4 and Algorithm 5 respectively. Moreover, `BingoShare` uses a sub-protocol `BingoDeal` (Algorithm 1), that describes the steps performed by the dealer. In more detail:

BingoDeal. The dealer receives secrets $s_k \in \mathbb{F}$ for $k \in [0, m]$ as inputs. It then uniformly samples two bivariate polynomials $\phi, \hat{\phi}$ over \mathbb{F} of degrees $2f$ in X and f in Y such that $\phi(\omega_{-k}, \omega_0) = s_k$ for $k \in [0, m]$. This can be done by uniformly sampling values for $\phi(\omega_i, \omega_0)$ for $i \in [f]$ and interpolating the resulting $\phi(X, \omega_0)$. Following that, the dealer simply uniformly samples $\phi(X, \omega_i)$ for $i \in [f]$ by directly sampling their coefficients, and interpolating the resulting $f + 1$ polynomials into a bivariate polynomial ϕ . The dealer then computes the *row projections* $\alpha_i(X) = \phi(X, \omega_i)$ and $\hat{\alpha}_i(X) = \hat{\phi}(X, \omega_i)$, and the *column projections* $\beta_i(Y) = \phi(\omega_i, Y)$, and $\hat{\beta}_i(Y) = \hat{\phi}(\omega_i, Y)$ for all $i \in [n]$. Looking ahead, the asymmetric degrees of the polynomials (α of degree $2f$ and β of degree f) help parties know that if they complete the **BingoShare** protocol, every other party will eventually do so as well. By definition, $\alpha_i(\omega_j) = \beta_j(\omega_i)$ and $\hat{\alpha}_i(\omega_j) = \hat{\beta}_j(\omega_i)$ for any $i, j \in [n]$. The dealer then broadcasts a commitment to this polynomial (formed using our bivariate PCS) and privately sends every party $i \in [n]$ its pair of row polynomials α_i and $\hat{\alpha}_i$.

Algorithm 1 $\text{BingoDeal}(s_0, \dots, s_m)$

- 1: uniformly sample $\phi(X, Y)$ with degree $2f$ in X and f in Y s.t. $\phi(\omega_{-k}, \omega_0) = s_k \forall k \in [0, m]$
 - 2: uniformly sample $\hat{\phi}(X, Y)$ with degree $2f$ in X and f in Y
 - 3: $\text{CM} \leftarrow \text{Commit}(\phi; \hat{\phi})$
 - 4: **for all** $i \in [n]$ **do**
 - 5: $\alpha_i(X) \leftarrow \phi(X, \omega_i), \hat{\alpha}_i(X) \leftarrow \hat{\phi}(X, \omega_i)$
 - 6: broadcast $\langle \text{“commits”}, \text{CM} \rangle$
 - 7: send $\langle \text{“polynomials”}, \alpha_i, \hat{\alpha}_i \rangle$ to every $i \in [n]$
-

BingoShare. The goal of **BingoShare** (Algorithm 2) is for each party i to learn their row polynomials α_i and $\hat{\alpha}_i$. As depicted in Figure 1, there are two ways this can happen. First, if the dealer is honest, they send the polynomials in **BingoDeal** and party i learns them directly (lines 7-10).

If the dealer is corrupt, however, party i may never receive a “polynomials” message. In this case other nonfaulty parties can help i as follows. First, these parties use their α polynomials to help other parties learn their β column polynomials (lines 15-23), taking advantage of the fact that $\alpha_j(\omega_\ell) = \beta_\ell(\omega_j)$ (we omit the $\hat{\alpha}$ and $\hat{\beta}$ polynomials in this description, but the process for them is identical). In other words, if party ℓ is given $\alpha_j(\omega_\ell)$ by enough other parties j then it can use **GetProofs** to compute evaluations and proofs for all other parties, as shown in line 21. Importantly, while party ℓ could interpolate β_ℓ and compute the evaluations directly, it would be unable to form the proofs using **Eval** as the proof for each party j needs to verify against cm_j (i.e., a commitment to α_j and not β_ℓ).

In the previous step, each party ℓ thus sends evaluations $\beta_\ell(\omega_i)$ to each party i . After receiving enough of these polynomials, party i can then interpolate α_i (in

line 31). Before completing the protocol, parties make sure that enough parties have received their row and column polynomials and are helping everybody reach the end of the protocol. This is done by parties sending “done” messages after having received their row and column polynomial, and terminating only after $n - f$ such messages have been received, guaranteeing that at least $f + 1$ nonfaulty parties shared their information.

BingoReconstruct. Once parties have finished the sharing phase, they can start recovering the shared secrets for all $k \in [0, m]$. The execution of **BingoReconstruct** may not be required in all cases, however, as it depends on the concrete application in which **Bingo** is used. To start recovery of the secret at index k , each party i evaluates its polynomials α_i and $\hat{\alpha}_i$ at position ω_{-k} and creates a proof $\pi_{\alpha,i,-k}$ showing that the evaluations are correct with respect to the commitment cm_i . Afterwards, party i sends a “rec” message with the evaluations $\alpha_i(\omega_{-k})$, $\hat{\alpha}_i(\omega_{-k})$ and the proof $\pi_{\alpha,i,-k}$ to all other parties j . Once party i receives its first “rec” message from party j , it verifies that the included shares are correct and, if so, stores the tuple $(j, \alpha_j(\omega_{-k}))$ in a set $\text{shares}_{i,k}$. Finally, once party i has received $f + 1$ different correct shares for the shared secret at index k , it interpolates $\text{shares}_{i,k}$ to a polynomial β_{-k} , outputs $\beta_{-k}(\omega_0)$ as the secret, and terminates. Note that the points $\alpha_j(\omega_{-k})$ should equal $\phi(\omega_{-k}, \omega_j)$. Interpolating $f + 1$ such points (with different values for j) yields the polynomial $\beta_{-k}(Y) = \phi(\omega_{-k}, Y)$, so $\beta_{-k}(\omega_0) = \phi(\omega_{-k}, \omega_0) = s_k$ as required.

4.2 Security

The security of **Bingo** scheme is captured in the following main theorem.

Theorem 4. *If the underlying commitment scheme is secure, then the pair (**BingoShare**, **BingoReconstruct**), as specified in Algorithms 2 and 3, is an f -resilient packed AVSS for $m + 1$ secrets, for any $m \leq f < \frac{n}{3}$.*

To prove this, we argue for correctness, termination, and secrecy in turn. To prove correctness and termination, we first prove a series of lemmas that consider the relationship between the committed polynomials represented by **CM** and the polynomials $\alpha_i, \hat{\alpha}_i, \beta_i, \hat{\beta}_i$ held by a nonfaulty party i at the point at which they complete **Share**. In all of the following lemmas we consider many instances of the **BingoShare** and **BingoReconstruct** protocols running simultaneously with both faulty and nonfaulty dealers. Each of the lemmas focuses on one of those instances and argues that certain values are consistent within that one instance. We first show that the existence of an extractor that can, for both faulty and nonfaulty dealers, output polynomials ϕ and $\hat{\phi}$ such that $\text{CM} = \text{Commit}(\phi; \hat{\phi})$.

The following lemma demonstrates the existence of an extractor that outputs polynomials consistent with the dealers broadcast commitment **CM** whenever a single nonfaulty party completes **BingoShare**. Where the polynomial commitment scheme is binding, this ensures that the output of **BingoReconstruct** is fully determined once an honest party completes.

Algorithm 2 BingoShare_i()

```
1: if  $i$  is the dealer with input  $s_0, \dots, s_m$  then
2:   BingoDeal( $s_0, \dots, s_m$ )
3:  $\alpha_i \leftarrow \perp, \hat{\alpha}_i \leftarrow \perp, \text{cm} \leftarrow \emptyset$ 
4:  $\text{points}_{\alpha_i} \leftarrow \emptyset, \text{points}_{\hat{\alpha}_i} \leftarrow \emptyset, \text{proofs}_{\beta_i} \leftarrow \emptyset$ 
5: upon receiving a  $\langle$ “commits”, CM $\rangle$  broadcast from the dealer, do
6:    $\text{cm} \leftarrow \text{PartialEval}(\text{CM}, \{\omega_1, \dots, \omega_n\}) \triangleright \text{cm} = (\text{cm}_1, \dots, \text{cm}_n)$ 
7: upon receiving the first  $\langle$ “polynomials”,  $\alpha'_i, \hat{\alpha}'_i$  $\rangle$  message from the dealer, do
8:   upon  $\text{cm} \neq \emptyset$ , do
9:     if  $\alpha_i \neq \perp$  and  $\text{KZG.Commit}(\alpha'_i, \hat{\alpha}'_i) = \text{cm}_i$  then
10:       $\alpha_i \leftarrow \alpha'_i, \hat{\alpha}_i \leftarrow \hat{\alpha}'_i$ 
11: upon  $\alpha_i \neq \perp$  and  $\text{cm}_i \neq \perp$ , do
12:   for all  $j \in [n]$  do
13:      $\alpha_i(\omega_j), \hat{\alpha}_i(\omega_j), \pi_{\alpha_i, j} \leftarrow \text{Eval}(\alpha_i, \hat{\alpha}_i, \omega_j)$ 
14:     send  $\langle$ “row”,  $\alpha_i(\omega_j), \hat{\alpha}_i(\omega_j), \pi_{\alpha_i, j}$  $\rangle$  to party  $j$ 
15: upon receiving the first  $\langle$ “row”,  $\alpha_j(\omega_i), \hat{\alpha}_j(\omega_i), \pi_{\alpha_j, i}$  $\rangle$  message from  $j$ , do
16:   upon  $\text{cm}_j \neq \perp$ , do
17:     if  $|\text{proofs}_{\beta_i}| < f + 1$  then  $\triangleright$  No need to collect points if interpolated
18:       if  $\text{Verify}(\text{cm}, (i, j), \alpha_j(\omega_i), \hat{\alpha}_j(\omega_i), \pi_{\alpha_j, i}) = 1$  then
19:          $\text{proofs}_{\beta_i} \leftarrow \text{proofs}_{\beta_i} \cup \{(\omega_j, \alpha_j(\omega_i), \hat{\alpha}_j(\omega_i), \pi_{\alpha_j, i})\}$ 
20:       if  $|\text{proofs}_{\beta_i}| = f + 1$  then
21:          $(y_1, \hat{y}_1, \pi_1), \dots, (y_n, \hat{y}_n, \pi_n) \leftarrow \text{GetProofs}(\text{proofs}_{\beta_i}, \{\omega_1, \dots, \omega_n\})$ 
22:         for all  $j \in [n]$  do
23:           send  $\langle$ “column”,  $y_j, \hat{y}_j, \pi_n$  $\rangle$  to party  $j$ 
24: upon receiving the first  $\langle$ “column”,  $\beta_j(\omega_i), \hat{\beta}_j(\omega_i), \pi_{\beta_j, i}$  $\rangle$  message from  $j$ , do
25:   upon  $\text{cm} \neq \emptyset$ , do
26:     if  $\alpha_i = \perp$  then  $\triangleright$  No need to collect points if already have  $\alpha_i$ 
27:       if  $\text{Verify}(\text{cm}, (j, i), \beta_j(\omega_i), \hat{\beta}_j(\omega_i), \pi_{\beta_j, i}) = 1$  then
28:          $\text{points}_{\alpha_i} \leftarrow \text{points}_{\alpha_i} \cup \{(\omega_j, \beta_j(\omega_i))\}$ 
29:          $\text{points}_{\hat{\alpha}_i} \leftarrow \text{points}_{\hat{\alpha}_i} \cup \{(\omega_j, \hat{\beta}_j(\omega_i))\}$ 
30:       if  $|\text{points}_{\alpha_i}| = 2f + 1$  then
31:          $\alpha_i \leftarrow \text{Interpolate}(\text{points}_{\alpha_i}), \hat{\alpha}_i \leftarrow \text{Interpolate}(\text{points}_{\hat{\alpha}_i})$ 
32: upon  $\alpha_i \neq \perp, \hat{\alpha}_i \neq \perp$  and  $|\text{proofs}_{\beta_i}| = f + 1$ , do
33:   send  $\langle$ “done” $\rangle$  to all parties
34: upon receiving  $\langle$ “done” $\rangle$  messages from  $n - f$  parties, do
35:   upon  $\alpha_i \neq \perp, \hat{\alpha}_i \neq \perp$ , and  $|\text{proofs}_{\beta_i}| = f + 1$ , do
36:     terminate
```

Algorithm 3 BingoReconstruct_{*i*}(*k*) for *k* ∈ [0, *m*]

```

1: sharesi,k = ∅
2: αi(ω-k), α̂i(ω-k), πα,i,-k ← Eval(αi, α̂i, ω-k)
3: send ⟨“rec”, k, αi(ω-k), α̂i(ω-k), πα,i,-k⟩ to all parties
4: upon receiving the first ⟨“rec”, k, αj(ω-k), α̂j(ω-k), πα,j,-k⟩ message from j, do
5:   if Verify(cm, (-k, j), αj(ω-k), α̂j(ω-k), πα,j,-k) = 1 then
6:     sharesi,k ← sharesi,k ∪ {(ωj, αj(ω-k))}
7:     if |sharesi,k| = f + 1 then
8:       β-k ← Interpolate(sharesi,k)
9:     output β-k(ω0) and terminate

```

Lemma 4. Assume some nonfaulty party completed the BingoShare protocol with respect to the commitment CM broadcast from the dealer. Suppose the (univariate) polynomial commitment scheme satisfies open binding. There exists an efficient extractor Ext that receives the views of the nonfaulty parties and outputs a pair of bivariate polynomials $\phi(X, Y)$ and $\hat{\phi}(X, Y)$ of degree $2f$ in X and f in Y such that

$$\text{CM} = \text{Commit}(\phi(X, Y); \hat{\phi}(X, Y))$$

Furthermore, if the dealer is nonfaulty, then $\forall k \in [0, m]$ $s_k = \phi(\omega_{-k}, \omega_0)$.

Proof. If the dealer is nonfaulty, then Ext can just output the polynomials $\phi(X, Y)$ and $\hat{\phi}(X, Y)$ stored in its local state (or recompute them from the inputs s_k and the dealer’s random tape). By the way the dealer samples $\phi(X, Y)$, we know that $\forall k \in [0, m]$ $s_k = \phi(\omega_{-k}, \omega_0)$.

If the dealer is faulty, observe the time the first non-faulty party completes the BingoShare protocol. It received a “done” message from $n - f$ parties, and out of those parties, at least $f + 1$ are nonfaulty. Let I be a set of $f + 1$ such nonfaulty parties. Before sending a “done” message, each $i \in I$ checks that $\alpha_i(X) \neq \perp, \hat{\alpha}_i(X) \neq \perp$. Define $\phi(X, Y)$ and $\hat{\phi}(X, Y)$ to be the unique bivariate polynomials of degree $2f$ in X and f in Y such that $\forall i \in I$ $\phi(X, \omega_i) = \alpha_i(X), \hat{\phi}(X, \omega_i) = \hat{\alpha}_i(X)$.

Before updating $\alpha_i(X), \hat{\alpha}_i(X)$, each $i \in I$ checks that $\text{cm} \neq \emptyset$. If i found that this is the case, it received a ⟨“commits”, CM⟩ broadcast from the dealer and updated $\text{cm} \leftarrow \text{PartialEval}(\text{CM}, \{\omega_1, \dots, \omega_n\})$. Party i updates $\alpha_i(X), \hat{\alpha}_i(X)$ in either line 10 or in line 31. If i does so in line 10, it first checks that $\text{cm}_i = \text{Commit}(\alpha_i(X), \hat{\alpha}_i(X))$.

If i updates $\alpha_i, \hat{\alpha}_i$ in line 31, then it received a ⟨“column”, y_j, \hat{y}_j, π_j ⟩ from $2f + 1$ different parties j such that $\text{Verify}(\text{cm}, (i, j), y_j, \hat{y}_j, \pi_j) = 1$ and added corresponding tuples to points_{α_i} and $\text{points}_{\hat{\alpha}_i}$. In that case, we have that $\text{Verify}(\text{cm}, (i, j), y_j, \hat{y}_j, \pi_j) = 1$ if and only if $\text{KZGVerify}(\text{cm}_i, \omega_j, y_j, \hat{y}_j, \pi_j) = 1$. Hence we can build a reduction \mathcal{B} such that

$$\Pr[\text{cm}_i \neq \text{Commit}(\alpha_i(X); \hat{\alpha}_i(X))] \leq \text{Adv}_{\mathcal{B}}^{\text{int-binding}}(1^\lambda)$$

that simply returns $(\text{cm}_i, \{\omega_j, \beta_j(i), \hat{\beta}_j(i), \pi_j\}_{j \in J})$, where J is the set of parties from which i received the aforementioned “column” messages. If the KZG poly-

nomial commitment scheme satisfies interpolation binding (see [Lemma 1](#)) then $\text{Adv}_{\mathcal{B}}^{\text{int-binding}}(1^\lambda) \leq \text{negl}(\lambda)$.

Finally observe that since $\text{cm} = \text{DFTExp}(\text{CM}, \{\omega_1, \dots, \omega_n\})$ we also have that $\text{CM} = \text{InterpolateExp}(\{(\omega_i, \text{cm}_i)\}_{i \in I})$. Thus from [Lemma 3](#) we have that $\text{CM} = \text{Commit}(\phi(X, Y); \hat{\phi}(X, Y))$. \square

Corollary 1. *Assume some nonfaulty party completed the BingoShare protocol, that the extractor from [Lemma 4](#) returns $\phi(X, Y), \hat{\phi}(X, Y)$, and the PCS satisfies polynomial binding. If some nonfaulty party i updates $\alpha_i(X), \hat{\alpha}_i(X)$ to values other than \perp , then $\alpha_i(X) = \phi(X, \omega_i)$ and $\hat{\alpha}_i(X) = \hat{\phi}(X, \omega_i)$.*

Proof. Suppose a nonfaulty party updates $\alpha_i(X), \hat{\alpha}_i(X)$ and an extractor outputs $\phi(X, Y)$ and $\hat{\phi}(X, Y)$ such that

$$\text{CM} = \text{Commit}(\phi(X, Y); \hat{\phi}(X, Y))$$

By the correctness of `PartialEval` we have that $\text{cm}_i = \text{Commit}(\phi(X, \omega_i); \hat{\phi}(X, \omega_i))$. If $(\alpha_i(X), \hat{\alpha}_i(X)) \neq (\phi(X, \omega_i), \hat{\phi}(X, \omega_i))$, then the adversary could simulate all of the nonfaulty parties, find two openings of cm_i and thus break polynomial binding.

Now assume that some nonfaulty party completes protocol and define $\phi(X, Y), \hat{\phi}(X, Y)$ to be extracted polynomials. The next lemma demonstrates that any point accepted by any nonfaulty party is consistent with $\phi(X, Y), \hat{\phi}(X, Y)$.

Lemma 5. *If (1) the dealer broadcasts a $\langle \text{“commits”}, \text{CM} \rangle$ message and it gets received by a nonfaulty party, and (2) the underlying PCS satisfies evaluation binding and open binding, and (3) some nonfaulty party completes the BingoShare protocol at time t , then define $\phi(X, Y), \hat{\phi}(X, Y) \leftarrow \text{Ext}(\text{view}_t)$ for `Ext` as in [Lemma 4](#). Then the following properties hold:*

- if a nonfaulty party i adds (j, y_j) and (j, \hat{y}_j) to points $_{\alpha, i}$ and points $_{\hat{\alpha}, i}$ respectively in lines [28](#) and [29](#), then $y_j = \phi(\omega_j, \omega_i)$ and $\hat{y}_j = \hat{\phi}(\omega_j, \omega_i)$, and
- if a nonfaulty party i adds $(j, y_j, \hat{y}_j, \pi_j)$ to proofs $_{\beta, i}$ in line [19](#), then $y_j = \phi(\omega_i, \omega_j)$ and $\hat{y}_j = \hat{\phi}(\omega_i, \omega_j)$.

Proof. From [Lemma 4](#), we have that $\text{CM} = \text{Commit}(\phi(X, Y); \hat{\phi}(X, Y))$. Consequently, for $\text{cm} = \text{PartialEval}(\text{CM}, \{\omega_1, \dots, \omega_n\})$ we have that for every $i \in [n]$, $\text{cm}_i = \text{Commit}(\phi(X, \omega_i); \hat{\phi}(X, \omega_i))$.

Consider an adversary \mathcal{A} that is attempting to break the lemma statement i.e. suppose there exists an adversary \mathcal{A} such that either:

1. party i receives a $\langle \text{“row”}, y_j, \hat{y}_j, \pi_{\alpha, j, i} \rangle$ message from some party j such that $\text{Verify}(\text{cm}, (i, j), (y_j, \hat{y}_j), \pi_{\alpha, j, i})$ and $(y_j, \hat{y}_j) \neq (\phi(\omega_i, \omega_j), \hat{\phi}(\omega_i, \omega_j))$;
2. or party i receives a $\langle \text{“column”}, y_j, \hat{y}_j, \pi_{\beta, j, i} \rangle$ message from some party j such that $\text{Verify}(\text{cm}, (j, i), (y_j, \hat{y}_j), \pi_{\beta, j, i}) = 1$ and $(y_j, \hat{y}_j) \neq (\phi(\omega_j, \omega_i), \hat{\phi}(\omega_j, \omega_i))$.

We first transition to an identical game G_1 where either Ext returns $\phi(X, Y)$, $\hat{\phi}(X, Y) \leftarrow \text{Ext}(\text{view}_t)$ such that $\text{CM} = \text{Commit}(\phi(X, Y); \hat{\phi}(X, Y))$ or the game aborts. By [Lemma 4](#) we have that

$$\mathbf{Adv}_{\mathcal{A}}^0(\lambda) \leq \mathbf{Adv}_{\mathcal{A}}^1(\lambda) + \mathbf{Adv}_{\text{Ext}, \mathcal{B}_1}^{\text{int-binding}}(\lambda)$$

We design an adversary \mathcal{B} that succeeds against evaluation binding. Let $\mathcal{B}(\text{srs})$ be an adversary against evaluation binding that runs \mathcal{A} and Ext as a subroutine. Note that \mathcal{B} runs all of the nonfaulty parties and so has access to their view of the protocol. At time t the reduction \mathcal{B} runs $\phi(X, Y)$, $\hat{\phi}(X, Y) \leftarrow \text{Ext}(\text{view}_t)$, $\text{cm} \leftarrow \text{PartialEval}(\text{CM}, \{\omega_1, \dots, \omega_n\})$ and either

1. if $\text{Verify}(\text{cm}, (j, i), (y_j, \hat{y}_j), \pi_{\beta, j, i})$ and $(y_j, \hat{y}_j) \neq (\phi(\omega_j, \omega_i), \hat{\phi}(\omega_j, \omega_i))$; then \mathcal{B} runs

$$(m, \hat{m}, \pi) \leftarrow \text{Eval}(\phi(X, \omega_i), \hat{\phi}(X, \omega_i), \omega_j)$$

and returns $((j, i), (y_j, \hat{y}_j), \pi_{\beta, j, i}, (m, \hat{m}, \pi))$.

2. if $\text{Verify}((i, j), (y_j, \hat{y}_j), \pi_{\alpha, i, j})$ and $(y_j, \hat{y}_j) \neq (\phi(\omega_i, \omega_j), \hat{\phi}(\omega_i, \omega_j))$; then \mathcal{B} runs

$$(m, \hat{m}, \pi) \leftarrow \text{Eval}(\phi(X, \omega_j), \hat{\phi}(X, \omega_j), \omega_i)$$

and returns $((i, j), (y_j, \hat{y}_j), \pi_{\alpha, i, j}, (m, \hat{m}, \pi))$.

Where (m, \hat{m}, π) are honest evaluations of CM they will verify and thus \mathcal{B} breaks the evaluation binding of the polynomial commitment scheme. \square

Proofs of the following three theorems can all be found in [Appendix B](#). For the correctness property, we start by extracting $\phi, \hat{\phi}$ at the time the first nonfaulty party completes `BingoShare` and define $r_k = \phi(\omega_{-k}, \omega_0)$ for every $k \in [0, m]$. Parties reconstruct by sending the values $\phi(\omega_{-k}, \omega_i)$, interpolating the polynomial $\phi(\omega_{-k}, Y)$ and evaluating it at ω_0 . Therefore, as long as [Lemma 5](#) holds, reconstruction is successful.

Theorem 5. *If q-sdh and interpolation binding ([Definition 3](#)) hold, then Bingo satisfies correctness.*

For the termination property, showing that if the dealer is nonfaulty then all nonfaulty parties complete the `BingoShare` protocol and that all nonfaulty parties complete the `BingoReconstruct` protocol is straightforward and is done by following the messages the dealer and nonfaulty parties are guaranteed to send. Proving that all nonfaulty parties complete the `BingoShare` protocol if one does, on the other hand, is more subtle and requires leveraging the asymmetric degrees of $\phi, \hat{\phi}$. We start by noting that if some nonfaulty party completed the protocol, at least $f + 1$ nonfaulty parties updated their row polynomials $\alpha_i, \hat{\alpha}_i$. These parties send “row” messages to all parties, allowing all nonfaulty parties to receive at least $f + 1$ evaluation on their columns $\beta_i, \hat{\beta}_i$. Since those polynomials are of degree f , this is enough to interpolate the polynomials and proofs and send “column” messages. After receiving such a message from all $n - f \geq 2f + 1$ nonfaulty parties, every party will be able to interpolate their rows, which are of degree no greater than $2f$, and complete the `BingoShare` protocol.

Theorem 6. *If q-sdh and interpolation binding (Definition 3) hold, then Bingo satisfies termination.*

To argue for secrecy, we need to rely on one additional property of the polynomial commitment scheme: that there exist algorithms `SimCommit`, `SimPartialEval` and `SimOpen` that allow for the simulation of bivariate commitments, partial evaluations, and openings of commitments respectively. We highlight that `SimOpen` works as follows: $\psi, \hat{\psi} \stackrel{\$}{\leftarrow} \text{SimOpen}(\tau_s, \text{cm}_\psi, \{y_i, \hat{y}_i\}_i)$ takes in a trapdoor τ_s , a commitment cm_ψ , and a set of evaluations of y_i, \hat{y}_i , and outputs a pair of polynomials ψ and $\hat{\psi}$ such that $\text{cm}_\psi = \text{Commit}(\psi, \hat{\psi})$, $\psi(v_i) = y_i, \hat{\psi}(v_i) = \hat{y}_i$ for all i , and the distribution over $(\psi, \hat{\psi})$ is uniform, given the above restriction. Importantly, this must hold even for adversarially chosen evaluation points v_i and evaluations y_i , (representing the adversary’s ability to see points from this party before corrupting it). For completeness, we provide a formal definition of this property in Appendix A.3.

Theorem 7. *If q-sdh holds then Bingo satisfies secrecy (Definition 6).*

Finally, we also prove the message, word, and round complexity of our protocol. In the following discussion we define asynchronous rounds as defined by Canetti and Rabin [20].

Theorem 8. *The BingoShare protocol requires $O(\lambda n^2)$ words and messages to be sent overall by all nonfaulty parties. Furthermore, if the bivariate and univariate PCSs satisfy correctness, non-malleability, partial evaluation binding and evaluation binding, then every nonfaulty completes the protocol in $O(1)$ rounds after the first nonfaulty party does so, and if the dealer is nonfaulty, all parties complete the protocol in $O(1)$ rounds. In addition, for every k , the BingoReconstruct(k) protocol requires $O(\lambda n^2)$ words and $O(n^2)$ messages to be sent overall by all nonfaulty parties, and takes $O(1)$ asynchronous rounds to complete.*

Proof. The BingoShare protocol starts with a single broadcast of $O(n)$ group elements by the dealer, each consisting of $O(\lambda)$ words. Using the reliable broadcast protocol of [28], the total number of words and messages sent by nonfaulty parties throughout the protocol is $O(\lambda n^2)$. The BingoShare protocol then proceeds with the dealer sending 2 polynomials to every party. Each polynomial can be described in $f + 1$ field elements, totalling in $O(\lambda n^2)$ words and $O(n)$ messages. Throughout the protocol, every party i might send “row” and “column” messages to every party j . Each such message contains $O(1)$ elements, each consisting of $O(\lambda)$ words, meaning that all of those messages total in $O(\lambda n^2)$ messages and words sent by all nonfaulty parties. Finally, every party can send a “done” message to every other party, totalling in $O(n^2)$ words and messages. Overall, we get that the BingoShare protocol requires $O(\lambda n^2)$ words and $O(n^2)$ messages to be sent by nonfaulty parties.

If the dealer is nonfaulty, then it starts by broadcasting a “commits” message and sending every party a “polynomials” message. Every nonfaulty party i receives those messages in $O(1)$ rounds and updates $\alpha_i, \hat{\alpha}_i$ to values other than \perp ,

if they haven't done so earlier. They then send "row" message to all nonfaulty parties. As shown in [Theorem 6](#), after receiving those "row" messages, every nonfaulty party i has $|\text{proofs}_{\beta,i}| = f + 1$. Every nonfaulty party then sends a "done" messages. Every nonfaulty i then receives those "done" messages in 1 more round, sees that it received "done" messages from $n - f$ parties and has $\alpha_i, \hat{\alpha}_i \neq \perp, |\text{proofs}_{\beta,i}| = f + 1$ and terminates. In addition, if some nonfaulty party completes the protocol, it received "done" messages from $n - f$ parties, with $f + 1$ of those parties being nonfaulty. Those nonfaulty parties only send "done" messages if they see that $\alpha_i, \hat{\alpha}_i \neq \perp$. Note that every nonfaulty party completes the reliable broadcast protocol of [\[28\]](#) $O(1)$ rounds after the first nonfaulty does. Following the same arguments as the ones in [Theorem 6](#), every nonfaulty party receives a "row" message from those $f + 1$ nonfaulty parties in a round at most, adds a tuple to $\text{proofs}_{\beta,i}$ after each such message, and sends a "column" message. A round after that, every nonfaulty i receives a "column" message from every nonfaulty party, and interpolates α and $\hat{\alpha}$ in [line 31](#). Every nonfaulty party then sees that $\alpha_i \neq \perp, \hat{\alpha}_i \neq \perp$ and $|\text{proofs}_{\beta,i}| = f + 1$ and sends a "done" message. Finally, one round later, every party receives at least $n - f$ done messages from the nonfaulty parties and has $\alpha_i \neq \perp, \beta_i \neq \perp$, and completes the protocol. In total, every nonfaulty party terminates $O(1)$ rounds after the first nonfaulty party terminates.

The only messages sent in each invocation of $\text{BingoReconstruct}(k)$ are "rec" messages. Each nonfaulty party sends one such message to all parties, containing $O(1)$ elements, each consisting of $O(\lambda)$ words. This totals in $O(\lambda n^2)$ words and messages sent by all nonfaulty parties in $\text{BingoReconstruct}(k)$. Sending this message also requires only a single round.

Corollary 2. *For any $m = \Omega(n)$, there exists a packed AVSS protocol sharing m secrets requiring $O(\lambda n^2 \cdot \frac{m}{n})$ words to be sent by nonfaulty parties in the sharing algorithm and $O(\lambda n^2)$ words to be sent while reconstructing any secret.*

Proof. Assume without loss of generality that $f = \frac{n}{3} - 1$. The dealer can take the m secrets and partition them into $\lceil \frac{m}{f+1} \rceil = \Theta(\frac{m}{n})$ batches of no more than $f + 1$ secrets. The i 'th secret s_i can be identified as the $(i \bmod f + 1)$ 'th secret in the $\lfloor \frac{m}{f+1} \rfloor$ 'th batch. The dealer then shares each batch using BingoShare , yielding a communication complexity of $\Theta(\lambda n^2 \cdot \frac{m}{n})$. Reconstructing the secret entails calling BingoReconstruct once, yielding a word complexity of $O(\lambda n^2)$.

Remark 1. It is possible to share $m + 1$ secrets with a polynomial of degree $f + m$ in X and f in Y , without changing the proofs. This yields rows of degree $f + m$ instead of degree $2f$.

4.3 Efficient reconstruction

In this section, we highlight two ways to efficiently reconstruct secrets shared using BingoShare , namely how to reconstruct sums of secrets and how to batch-reconstruct multiple secrets.

Algorithm 4 BingoReconstructSum_{*i*}(dealers, *k*) for $k \in [0, m]$

```

1: sharesi,k ← ∅
2: ∀ i ∈ [n] cm'i ← ∏j ∈ dealers cmi,j                                ▷ cm' = (cm'1, ..., cm'n)
3: cm' ← (cm'1, ..., cm'n)
4: vi,k, ŵi,k, πi,k ← Eval(∑j ∈ dealers αi,j, ∑j ∈ dealers âi,j, ω-k)
5: send ⟨“rec”, k, vi,k, ŵi,k, πi,k⟩ to all parties
6: upon receiving the first ⟨“rec”, k, vj,k, ŵj,k, πj,k⟩ message from j, do
7:   if Verify(cm', (-k, j), vj,k, ŵj,k, πj,k) = 1 then
8:     sharesi,k ← sharesi,k ∪ {(ωj, vj,k)}
9:     if |sharesi,k| = f + 1 then
10:      β-k ← Interpolate(sharesi,k)
11:      output β-k(ω0) and terminate

```

First, we observe that sharing $O(n)$ secrets requires sending $O(\lambda n^2)$ words and reconstructing each secret requires $O(\lambda n^2)$ words. One way to leverage the efficient sharing protocol is by reconstructing significantly fewer secrets than the number of secrets shared. This can be done by using the fact that the KZG PCS is additively homomorphic, meaning if $\text{cm}_1, \dots, \text{cm}_\ell$ are commitments to $(\phi_1, \hat{\phi}_1), \dots, (\phi_\ell, \hat{\phi}_\ell)$ respectively, then $\prod_{i=1}^\ell \text{cm}_i$ is a commitment to the polynomials $(\sum_{i=1}^\ell \phi_i, \sum_{i=1}^\ell \hat{\phi}_i)$. Therefore, let *dealers* be a set of dealers for which party *i* completed BingoShare, and set some $k \in [0, m]$. Then, if we define $r_{k,j}$ to be the *k*-th secret in the BingoShare invocation with *j* as dealer, parties can reconstruct $\sum_{j \in \text{dealers}} r_{k,j}$. We provide the code for reconstructing the sum of several shared secrets in Algorithm 4 and highlight that $\alpha_{i,j}, \hat{\alpha}_{i,j}$ are the polynomials $\alpha_i, \hat{\alpha}_i$ set by party *i* when running BingoShare with *j* as dealer. Similarly, $\text{cm}_{i,j}$ is the commitment cm_i in the BingoShare invocation with *j* as dealer.

It is also possible to batch-reconstruct all *m* secrets at once while sending only $O(\lambda n^2)$ words, as demonstrated in Algorithm 5. Observe that all secrets are values of the form $\phi(\omega_{-k}, \omega_0)$ for $k \in [0, m]$. This means that instead of reconstructing each secret by interpolating the polynomials $\phi(\omega_{-k}, Y)$ and evaluating them at ω_0 , it is possible to interpolate the degree- $2f$ polynomial $\phi(X, \omega_0)$ in order to reconstruct all *m* secrets. This requires parties to send points on their β polynomials, and to provide adequate proofs. Seeing as those proofs need to be interpolated and verified with respect to a commitment to $\phi(X, \omega_0), \hat{\phi}(X, \omega_0)$, we use PartialEval and GetProofs to compute those commitments and proofs.

In both BingoReconstructSum and BingoReconstructBatch, parties send a single message of the exact same size as the one sent in BingoReconstruct, resulting in identical complexity. The proofs that the BingoReconstructSum protocol and the BingoReconstructBatch protocol satisfy the required properties is identical to the proof of BingoReconstruct, using the commitments cm' and cm_0 respectively instead of cm , and is thus omitted. See the proofs of correctness and termination of BingoReconstruct for details.

Algorithm 5 BingoReconstructBatch_i()

```
1: sharesi ← ∅
2: cm0 ← PartialEval(CM, {ω0})           ▷ only compute partial for ω0
3: (y0, ŷ0, π0) ← GetProofs(proofsβ,i, {ω0})   ▷ only compute proof for ω0
4: send ⟨“rec”, y0, ŷ0, π0⟩ to all parties
5: upon receiving the first ⟨“rec”, yj, ŷj, πj⟩ message from j, do
6:   if Verify((cm0), (0, j), yj, ŷj, πj) = 1 then
7:     sharesi ← sharesi ∪ {(ωj, yj)}
8:     if |sharesi| = n − f then
9:       α0 ← Interpolate(sharesi)
10:    output (α0(ω0), α0(ω−1), . . . , α0(ω−m)) and terminate
```

5 From Bingo to ADKG

In this section we show how to use Bingo to achieve an adaptively secure asynchronous distributed key generation (ADKG) protocol that has communication complexity of $O(\lambda n^3)$ and produces a field element as a secret key. This latter advantage is particularly useful in ensuring our DKG is compatible with established threshold cryptography schemes such as BLS [13].

In order to get to a DKG we use Bingo at two layers:

1. We use Bingo to get an adaptively secure *validated asynchronous Byzantine agreement* (VABA) protocol. The protocol, presented in Appendix C, allows proposals (inputs) of size $O(n)$ and requires $O(n^3)$ expected words.
2. Each party then uses Bingo to share a potential contribution to the DKG. Once the VABA protocol reaches agreement on a proposal, we use the ability of Bingo to reconstruct the sum of secrets. This sum is the secret key, however, whereas the goal of the DKG is to generate the public key. We thus perform this reconstruction only in the exponent.

In more detail, we start by defining $\text{CM}_j, \text{proofs}_{\beta,i,j}$ as the values $\text{CM}, \text{proofs}_{\beta,i}$ in the invocation of BingoShare with *j* as dealer. Intuitively, our DKG protocol works as follows. First, each party *j* acts as the dealer for $f + 1$ secrets, which we can think of as their 0-th row polynomial $\alpha_{0,j}$. Parties must then agree on a set of dealers whose secrets will contribute to the threshold public key g^s , where the corresponding secret key *s* is the polynomial $\alpha_\Sigma = \sum_{j \in \text{dealers}} \alpha_{0,j}$ evaluated at ω_0 . This agreement requires the use of a VABA protocol. Informally, a VABA protocol allows each party to input a value and output some agreed-upon value in a way that is *correct*, meaning all nonfaulty parties that complete the protocol output the same value, and *valid*, meaning that values output by nonfaulty parties satisfy some external validity function. For a formal definition of a VABA protocol, see Definition 9 in Appendix C.

Once this set is agreed upon using the VABA protocol, parties act to reconstruct the g^s term, as well as their own secret share. For the set of agreed dealers *dealers*, this latter value for party *i* is the sum of the column polynomials $\beta_{i,j}$ evaluated at ω_0 , where $\beta_{i,j}$ is *i*'s column polynomial in the BingoShare

invocation with j as the dealer. Because $\beta_{i,j}(\omega_0) = \alpha_{0,j}(\omega_i)$, this is equivalent to evaluating α_Σ at ω_i . If enough parties share these evaluation points, they can thus interpolate α_Σ and evaluate it at ω_0 to reconstruct the secret key. Note that parties do not directly store their $\beta_{i,j}$ polynomials, so they must interpolate evaluations and proofs from their $\mathbf{proofs}_{\beta,i,j}$ sets. Similarly, parties do not compute a commitment to the 0-th row of the polynomial during **BingoShare**, so they must compute it using \mathbf{CM}_j for each dealer j .

We describe how to construct our VABA protocol in Appendix C, following closely the path of Abraham et al. [3], whose protocol structure is similar to ours but uses an aggregated PVSS transcript instead of **BingoShare**. This means we use their **Gather** protocol and **Bingo** to build a *weak leader election* protocol, relying particularly on the ability in **Bingo** to reconstruct sums of secrets, as described in the previous section. From this weak leader election protocol, in which parties are guaranteed to elect the same nonfaulty party with only constant probability p , we build a *proposal election* protocol, and from that we build an adaptively secure VABA protocol. Our protocol has $O(\lambda n^3)$ word complexity and assumes the existence of a PKI and the setup required for the KZG polynomial commitment scheme [39].

Before describing our DKG based on this VABA protocol, we must first extend the $\mathbf{BingoReconstructSum}_i$ algorithm (Algorithm 4). Essentially, whereas $\mathbf{BingoReconstructSum}_i$ reconstructs the sum s of the k -th secrets across a given set of dealers, we need to be able to compute the public key g^s , which involves computing the sum in the exponent. The algorithm for party i , given in Algorithm 6, is similar to $\mathbf{BingoReconstructSum}_i$ but instead of sending y_i and \hat{y}_i to other parties (the evaluations of $\sum_{j \in \text{dealers}} \alpha_{i,j}$ and $\sum_{j \in \text{dealers}} \hat{\alpha}_{i,j}$ at point 0 respectively) it sends $Y_i \leftarrow g^{y_i}$ and $\hat{Y}_i \leftarrow g^{\hat{y}_i}$ as well as proofs of knowledge of y_i and \hat{y}_i . We denote by $\pi \stackrel{\$}{\leftarrow} \text{PoK.Prove}(Y, y)$ and $0/1 \leftarrow \text{PoK.Verify}(Y, \pi)$ the respective algorithms for proving and verifying knowledge of y , and by Verify' the PCS algorithm that takes in Y, \hat{Y} rather than y, \hat{y} , which is defined as follows.

– $0/1 \leftarrow \text{Verify}'(\text{cm}, \omega, Y, \hat{Y}, \pi)$ Output 1 if $e(\text{cm} \cdot (Y \cdot \hat{Y})^{-1}, h) = e(\pi, h^{\tau-\omega})$, and otherwise output 0.

Finally, we denote by $Y_j \leftarrow \text{IntEvalExp}(\{v_i, Y_i\}_{i=0}^f, \omega_j)$ the algorithm that performs $\text{EvalExp}(\omega_j, \text{InterpolateExp}(\{v_i, Y_i\}_i))$; i.e., that interpolates the degree- f polynomial given $f + 1$ evaluations and then evaluates it at ω_j (all in the exponent).

With this subprotocol and our VABA in place, we construct our full DKG as shown in Algorithm 7. Once a party has completed **BingoShare** for at least $f + 1$ dealers, it asks at least $f + 1$ other parties to verify that those **BingoShare** sessions were indeed completed by sending the set of those dealers in a “proposal” message. After completing the **BingoShare** calls for all of these dealers, those parties reply with a signature on the set of $f + 1$ dealers. All parties then agree on a set of $f + 1$ dealers, dealers , and $f + 1$ signatures, sigs , using the VABA protocol with an external validity function defined as follows:

Algorithm 6 BingoSumExpAndRec_i(dealers)

- 1: $\text{shares}_i \leftarrow \emptyset$
- 2: $\forall j \in \text{dealers } \text{cm}_{0,j} \leftarrow \text{PartialEval}(\text{CM}_j, \{\omega_0\})$
- 3: $\text{cm}_0 \leftarrow \prod_{j \in \text{dealers}} \text{cm}_{0,j}$
- 4: $\forall j \in \text{dealers } y_{i,j}, \hat{y}_{i,j}, \pi_{i,j} \leftarrow \text{GetProofs}(\text{proofs}_{\beta,i,j}, \{\omega_0\})$
- 5: $\text{sk}_i \leftarrow \sum_{j \in \text{dealers}} y_{i,j}, \hat{y}_i \leftarrow \sum_{j \in \text{dealers}} \hat{y}_{i,j}, \pi_i \leftarrow \prod_{j \in \text{dealers}} \pi_{i,j}$
- 6: $Y_i \leftarrow g^{\text{sk}_i}, \pi \xleftarrow{\$} \text{PoK.Prove}(Y_i, \text{sk}_i)$
- 7: $\hat{Y}_i \leftarrow g^{\hat{y}_i}, \hat{\pi} \xleftarrow{\$} \text{PoK.Prove}(Y_i, \hat{y}_i)$
- 8: send $\langle \text{"key share"}, Y_i, \hat{Y}_i, \pi_i, \pi, \hat{\pi} \rangle$ to all parties
- 9: **upon** receiving the first $\langle \text{"key share"}, Y_j, \hat{Y}_j, \pi_j, \pi, \hat{\pi} \rangle$ message from party j , **do**
- 10: **if** $\text{Verify}'(\text{cm}_0, \omega_j, Y_j, \hat{Y}_j, \pi_j) = \text{PoK.Verify}(Y_j, \pi) = \text{PoK.Verify}(\hat{Y}_j, \hat{\pi}) = 1$ **then**
- 11: $\text{shares}_i \leftarrow \text{shares}_i \cup \{(\omega_j, Y_j)\}$
- 12: **if** $|\text{shares}_i| = 2f + 1$ **then**
- 13: $\text{pk} \leftarrow \text{IntEvalExp}(\text{shares}_i, \omega_0)$
- 14: **output** (pk, sk_i) and **terminate**

Algorithm 7 ADKG_i()

- 1: $\text{prop}_i \leftarrow \emptyset, \text{dealers}_i \leftarrow \emptyset, \text{sigs}_i \leftarrow \emptyset$
- 2: $s_0, \dots, s_f \xleftarrow{\$} \mathbb{F}$
- 3: call BingoShare as dealer sharing s_0, \dots, s_f
- 4: participate in BingoShare with j as dealer for every $j \in [n]$
- 5: **upon** completing BingoShare with j as dealer, **do**
- 6: $\text{dealers}_i \leftarrow \text{dealers}_i \cup \{j\}$
- 7: **if** $|\text{dealers}_i| = f + 1$ **then**
- 8: $\text{prop}_i \leftarrow \text{dealers}_i$
- 9: send $\langle \text{"proposal"}, \text{prop}_i \rangle$ to every $j \in [n]$
- 10: **upon** receiving the first $\langle \text{"proposal"}, \text{prop}_j \rangle$ message from party j , **do**
- 11: **upon** completing BingoShare with k as leader for every $k \in \text{prop}_j$, **do**
- 12: send $\langle \text{"signature"}, \text{Sign}(\text{sk}_i, \text{prop}_j) \rangle$ to party j
- 13: **upon** receiving $\langle \text{"signature"}, \sigma_j \rangle$ from j , **do**
- 14: **if** $\text{prop}_i \neq \emptyset$ and $\text{Verify}(\text{pk}_j, \text{prop}_i, \sigma_j) = 1$ **then**
- 15: $\text{sigs}_i \leftarrow \text{sigs}_i \cup \{(j, \sigma_j)\}$
- 16: **if** $|\text{sigs}_i| = f + 1$ **then**
- 17: **invoke** VABA with input $(\text{prop}_i, \text{sigs}_i)$ and external validity function checkValidity
- 18: **upon** VABA terminating with output $(\text{prop}, \text{sigs})$, **do**
- 19: **upon** completing the BingoShare call with j as dealer for every $j \in \text{prop}$, **do**
- 20: **invoke** BingoSumExpAndRec_i with input prop
- 21: **upon** BingoSumExpAndRec_i terminating with output (pk, sk_i) , **do**
- 22: **output** pk and **terminate**

$$\text{checkValidity}(\text{dealers}, \text{sigs}) = (|\text{dealers}| \geq f + 1 \wedge |\text{sigs}| \geq f + 1 \wedge \text{Verify}(\text{pk}_j, \sigma_j, \text{dealers}) \forall (j, \sigma_j) \in \text{sigs}). \quad (1)$$

If this holds, meaning at least $f + 1$ parties provided a signature for the set of dealers, then at least one nonfaulty party provided a signature. This nonfaulty party thus completed `BingoShare`, and by termination every nonfaulty party will eventually do so as well. Parties then wait to complete the $f + 1$ `BingoShare` calls for the agreed set of dealers. Party i can then invoke `BingoSumExpAndReci` to output pk and sk_i .

In terms of the security of our DKG, we follow Gennaro et al. [35] in showing that it satisfies *robustness*, meaning that all honest parties agree on the same public key and that there exists an algorithm to allow parties to reconstruct the corresponding secret key, and *secrecy*, meaning that its output is uniformly random and reveals nothing about the secret key. In particular, this latter property is defined by the existence of a simulator that takes in a value Y as input and produces an output distribution that is indistinguishable from the adversary's view in the honest protocol, as long as the adversary corrupts at most t parties, but such that the output of the DKG is Y .

We first prove that the properties that hold for field elements in `BingoShare` also hold for group elements containing the sum of these elements in the exponent.

Lemma 6. *Let dealers be a set of dealers for which at least one nonfaulty party completed an invocation of `BingoShare` with $m = t - f$ for $f \leq t \leq 2f$, and for every $i \in \text{dealers}$ let $r_{i,0}$ be the value r_0 defined in the correctness property of `Bingo` for the invocation with i as dealer. If all nonfaulty parties invoke `BingoSumExpAndRec`, then each nonfaulty i outputs $p(\omega_i), g^{\sum_{j \in \text{dealers}} r_{j,0}}$ and terminates, with p being a degree- $2f$ polynomial with $p(\omega_0) = \sum_{j \in \text{dealers}} r_{j,0}$.*

Proof. This proof deals only with the maximum possible degree $2f$. As noted in Remark 1, it is possible to run `BingoShare` with degree $f + m$ in X , allowing any degree t as described above.

First we show that all nonfaulty parties will eventually complete the protocol and output some value. For every $i \in \text{dealers}$ define $\phi_i, \hat{\phi}_i$ as in Lemma 4 for the `BingoShare` invocation with i as dealer and let CM_i be the commitment CM broadcast by i . As shown in the proof of correctness, for every such i , $r_{i,0} = \phi_i(\omega_0, \omega_0)$. For every $i \in [n], j \in \text{dealers}$, let $\text{cm}_{i,j}$ be cm_i in the `BingoShare` invocation with j as dealer. From Lemma 4, for every $j \in \text{dealers}$, $\text{CM}_j = \text{Commit}(\phi_j(X, Y); \hat{\phi}_j(X, Y))$. From the construction of `PartialEval`, for every $j \in \text{dealers}$, $\text{cm}_{j,0} \leftarrow \text{PartialEval}(\text{CM}_j, \{\omega_0\})$ satisfies $\text{cm}_{j,0} = \text{KZG.Commit}(\phi(X, \omega_0); \hat{\phi}(X, \omega_0))$. From the homomorphic nature of the KZG commitment scheme, for

every $i \in [n]$,

$$\begin{aligned}
\text{cm}_0 &= \prod_{j \in \text{dealers}} \text{cm}_{j,0} \\
&= \prod_{j \in \text{dealers}} \text{Commit}(\phi_j(X, \omega_0); \hat{\phi}_j(X, \omega_0)) \\
&= \text{Commit}\left(\sum_{j \in \text{dealers}} \phi_j(X, \omega_0); \sum_{j \in \text{dealers}} \hat{\phi}_j(X, \omega_0)\right).
\end{aligned}$$

From [Lemma 2](#), for every nonfaulty i , $\forall j \in \text{dealers}$ $\text{KZG.Verify}(\text{cm}_{0,j}, \omega_0, y_{i,j}, \hat{y}_{i,j}, \pi_{i,j}) = 1$, and from the homomorphic nature of the KZG commitment scheme, $\text{KZG.Verify}(\prod_{j \in \text{dealers}} \text{cm}_{0,j}, \omega_0, \sum_{j \in \text{dealers}} y_{i,j}, \sum_{j \in \text{dealers}} \hat{y}_{i,j}, \prod_{j \in \text{dealers}} \pi_{i,j}) = 1$. Note that for every nonfaulty party i , $\text{cm}_0 = \prod_{j \in \text{dealers}} \text{cm}_{0,j}$, $Y_i = g^{\sum_{j \in \text{dealers}} y_{i,j}}$, $\hat{Y}_i = g^{\sum_{j \in \text{dealers}} \hat{y}_{i,j}}$ and $\pi_i = \prod_{j \in \text{dealers}} \pi_{i,j}$, and thus, from the construction of Verify' , $\text{Verify}'(\text{cm}_0, \omega_0, Y_i, \hat{Y}_i, \pi_i) = 1$. Every nonfaulty party will eventually receive “key share” messages sent by each nonfaulty party, see that its contents pass verification, and add a tuple to shares . After adding such a tuple for all $n - f \geq 2f + 1$ nonfaulty parties, every nonfaulty parties computes pk , outputs a value, and terminates.

Now we show that the output of the nonfaulty parties satisfies the conditions of the lemma. Following the same argument as the one in [Lemma 5](#), from the extractability of the proofs of knowledge and the evaluation binding of KZG, nonfaulty parties only add tuples of the form $(\omega_k, g^{\sum_{j \in \text{dealers}} \phi_j(\omega_0, \omega_k)})$ to shares_i . Therefore, when calling EvalExp on shares_i after adding $t + 1$ such shares, nonfaulty parties output $\text{pk} = g^{\sum_{j \in \text{dealers}} \phi_j(\omega_0, \omega_0)} = g^{\sum_{j \in \text{dealers}} r_{j,0}}$.

Finally, define $p(X) = \sum_{j \in \text{dealers}} \phi_j(X, \omega_0)$. By definition, we know that $p(\omega_0) = \sum_{j \in \text{dealers}} \phi_j(\omega_0, \omega_0) = \sum_{j \in \text{dealers}} r_{j,0}$, and since each of the polynomials ϕ_j is of degree $2f$, so is p . All that is left to show is that every nonfaulty party i outputs $p(\omega_i)$. For every nonfaulty i and $j \in \text{dealers}$, from [Lemma 5](#), $y_k = \phi_j(\omega_i, \omega_k)$ for every $(\omega_k, y_k, \hat{y}_k, \pi_k) \in \text{proofs}_{\beta, i, j}$. Therefore, by construction $\text{GetProofs}(\text{proofs}_{\beta, i, j}, \{\omega_0\})$ outputs $y_{i,j}, \hat{y}_{i,j}, \pi_{i,j}$ such that $y_{i,j} = \phi_j(\omega_i, \omega_0)$. Every nonfaulty party i outputs $sk_i = \sum_{j \in \text{dealers}} y_{i,j} = \sum_{j \in \text{dealers}} \phi_j(\omega_i, \omega_0) = p(\omega_i)$, as required.

Theorem 9. *If Bingo and the VABA protocol both satisfy correctness and termination, and the VABA protocol satisfies validity, then the ADKG in Algorithm 7 satisfies robustness against an adaptive adversary that can control f parties, where the total number of parties is $n > 3f$.*

Proof. Every nonfaulty party i starts the ADKG protocol by uniformly sampling some value r_i , calling BingoShare , and participating in BingoShare with j as dealer for every $j \in [n]$. From the termination property of the BingoShare protocol, all nonfaulty parties eventually complete the BingoShare invocations with nonfaulty dealers. This means that eventually every nonfaulty i will add j to dealers_i for every nonfaulty j . After adding $f + 1$ such indices, party i sets prop_i

and sends a “proposal” message to all parties. Because party i completed the `BingoShare` invocation with j as dealer for every $j \in \text{prop}_i$, by the termination property of `BingoShare` every other nonfaulty party will do so as well.

Every nonfaulty party will eventually receive party i ’s “proposal” message and, as stated above, complete the `BingoShare` invocation with j as dealer for every $j \in \text{prop}_i$. It then replies with a “signature” message and a signature σ_j on prop_i . Party i eventually receives such a “signature” message from every nonfaulty party and adds its valid signature to its sigs_i set. After adding $f + 1$ such signatures to its sigs_i set, every nonfaulty party i calls `VABA`. Every such party i sets prop_i to be a set of $f + 1$ dealers, adds only valid signatures on prop_i to sigs_i , and invokes `VABA` after having $|\text{sigs}_i| = f + 1$. In other words, every nonfaulty party eventually invokes `VABA` with an externally valid input.

By the termination property of the `VABA` protocol, all nonfaulty parties eventually complete the protocol with some output. By the correctness of the `VABA` protocol, all these parties have the same output $(\text{prop}, \text{sigs})$, and by the validity of the `VABA` protocol this output is externally valid, meaning there are signatures from $f + 1$ parties on the set prop , one of which must have been nonfaulty. Nonfaulty parties only send such signatures in line 12 if they completed `BingoShare` for every $j \in \text{prop}$. Since there is at least one nonfaulty party that completed the `BingoShare` invocation with j as dealer for every $j \in \text{prop}$, from the termination property of the `BingoShare` protocol, all nonfaulty parties will eventually complete those `BingoShare` invocations as well and invoke `BingoSumExpAndRec` in line 20. From Lemma 6 they will complete `BingoSumExpAndRec` and output the same value pk (the first requirement of robustness) and output shares that can be reconstructed into the unique sk for pk (the second requirement of robustness).

For secrecy, it is unfortunately not clear how to satisfy the definition of Genaro et al., as the `Bingo` secrecy definition guarantees the ability to simulate interactions in the `BingoShare` protocol but for a DKG we need to be able to continue simulating throughout reconstruction (albeit in the exponent) despite not knowing the underlying secret or polynomial. We thus prove that our protocol satisfies the notion of *oracle-aided algebraic simulatability*, and thus also *oracle-aided algebraic security* as recently defined by Bacho and Loss [5, Definition 3.1]. This definition requires us to construct an algebraic simulator that is given k group elements and allowed to query a discrete logarithm oracle `DL` at most $k - 1$ times, providing on the i -th query the algebraic coefficients \vec{a}_i representing the relationship between its queried group element and its k inputs. The simulator is then successful if it (1) simulates all nonfaulty parties in the DKG in a way that is indistinguishable from the interaction expected by the adversary controlling the corrupted parties, and (2) the matrix containing its algebraic coefficients is invertible.

Theorem 10. *If `Bingo` satisfies correctness and secrecy and the `VABA` satisfies correctness and external validity, the ADKG in Algorithm 7 has $(f, 2f + 1)$ -oracle-aided algebraic security against an adaptive adversary that can control f parties, where the total number of parties is $n > 3f$.*

Proof. We begin by describing the simulator $\text{Sim}_{\mathcal{A}}^{\text{DL}(\cdot)}$, which takes as input a generator g and $2f + 1$ group elements $Z_0, Z_1, \dots, Z_f, \hat{Z}_1, \dots, \hat{Z}_f$. To simulate nonfaulty parties in the DKG protocol, Sim acts as the Bingo simulator during BingoShare interactions (this simulator is guaranteed to exist by secrecy, and is described in the proof of Theorem 7.) During all other parts of the DKG before line 20, the simulator behaves completely honestly; i.e. it honestly computes and sends “proposal” messages, responds with “signature” messages when it receives “proposal” messages, and invokes the VABA protocol once it has enough signatures.

When the first nonfaulty party completes the VABA protocol with output $(\text{dealers}, \text{sigs})$, Sim sets C to be the set of currently corrupted parties. From the correctness of the VABA protocol, all nonfaulty parties will also output $(\text{dealers}, \text{sigs})$. In addition, from the external validity property, sigs contains at least $f + 1$ signatures on the set dealers , which means that it includes at least one signature from a nonfaulty party. Nonfaulty parties only sign dealers if they have completed the BingoShare invocations with j as dealer for every $j \in \text{dealers}$, and thus at least one nonfaulty party completed the protocol for each such dealer. As shown in Lemma 4, for every faulty dealer $j \in \text{dealers}$, it is possible to extract polynomials $\phi_j, \hat{\phi}_j$ from the combined views of the nonfaulty parties, which Sim can do as it has the views of all nonfaulty parties and behaves completely honestly when the dealer is faulty. On the other hand, in the proof of Theorem 7, the simulator defines polynomials $\alpha_{i,j}, \hat{\alpha}_{i,j}, \beta_{i,j}, \hat{\beta}_{i,j}$ for every faulty i in the simulated BingoShare invocation with a nonfaulty j as dealer. Putting this together, Sim thus knows the polynomials $\alpha_{i,j}, \hat{\alpha}_{i,j}, \beta_{i,j}, \hat{\beta}_{i,j}$ for faulty dealers $j \in \text{dealers}$ and all parties i and the polynomials $\alpha_{i,j}, \hat{\alpha}_{i,j}, \beta_{i,j}, \hat{\beta}_{i,j}$ for nonfaulty dealers $j \in \text{dealers}$ and faulty parties i .

Let ℓ be the number of parties corrupted at the time the first nonfaulty party completes the VABA protocol, and let $C = \{i_1, \dots, i_\ell\}$. Sim chooses $I = \{i_{\ell+1}, \dots, i_f\} \subset [n]$ to be some subset of $[n]$ of size $f - k$ such that $C \cap I = \emptyset$ (for example, the $f - k$ minimal indices that aren't in C). Sim chooses an additional set $I' = \{i_{f+1}, \dots, i_{2f}\}$ ⁶ such that $I' \cap C = \emptyset$ and $I' \cap I = \emptyset$. Finally, let i_{2f+1}, \dots, i_n be the indices of the remaining parties, i.e. $\{i_{2f+1}, \dots, i_n\} = [n] \setminus (C \cup I \cup I')$. Sim then defines $Z'_0 \leftarrow Z_0$ and $i_0 = 0$, as well as the following:

- For every $k \in [\ell]$, $Z'_{i_k} \leftarrow g^{\sum_{j \in \text{dealers}} \beta_{i_k, j}^{(0)}}$ and $\hat{Z}'_{i_k} \leftarrow \hat{g}^{\sum_{j \in \text{dealers}} \hat{\beta}_{i_k, j}^{(0)}}$.
- For every $k \in \{\ell + 1, \dots, f\}$, $Z'_{i_k} \leftarrow Z_k$ and $\hat{Z}'_{i_k} \leftarrow \hat{Z}_k$.
- For every $k \in \{f + 1, \dots, 2f\}$, Sim samples $z_{i_k}, \hat{z}_{i_k} \xleftarrow{\mathbb{S}} \mathbb{F}$ and sets $Z'_{i_k} \leftarrow g^{z_{i_k}}$ and $\hat{Z}'_{i_k} \leftarrow \hat{g}^{\hat{z}_{i_k}}$.
- For every $k \in \{2f + 1, \dots, n\}$, $Z'_{i_k} \leftarrow \text{IntEvalExp}(\{(\omega_{i_m}, Z'_{i_m})\}_{m=0}^{2f}, \omega_{i_k})$.

The simulator then computes $Z'_\tau \leftarrow \text{IntEvalExp}(\{(\omega_{i_m}, Z'_{i_m})\}_{m=0}^{2f}, \tau)$, as well as $\text{cm}_0 \leftarrow \prod_{j \in \text{dealers}} \text{cm}_{0,j}$ (as computed in BingoSumExpAndRec), and $\hat{Z}'_\tau \leftarrow (\text{cm}_0 \cdot (Z'_\tau)^{-1})^{\frac{1}{\alpha}}$. It then computes $\hat{Z}'_i = \text{IntEvalExp}(\{(\omega_{i_k}, \hat{Z}'_{i_k})\}_{k \in [2f] \cup \{\tau\}}, \omega_i)$

⁶ For a threshold of $f + m + 1$, define $I' = \{i_{f+1}, \dots, i_{f+m}\}$ instead.

for every $i \in [n]$. Finally, Sim calls its discrete log oracle 2ℓ times on $Z_1, \dots, Z_\ell, \hat{Z}_1, \dots, \hat{Z}_\ell$.

After computing these values, Sim is now ready to simulate nonfaulty parties in Algorithm 6. Whenever a nonfaulty party i should send a “key share” message, Sim computes $\pi_i \leftarrow (\mathbf{cm}_0 \cdot (Z'_i \hat{Z}'_i)^{-1})^{\frac{1}{\tau - \omega_i}}$ as well as simulated proofs of knowledge $\pi, \hat{\pi}$ for Z'_i and \hat{Z}'_i respectively. Sim then adds messages to the buffer as if i sent the message $\langle \text{“key share” } Z'_i, \hat{Z}'_i, \pi_i, \pi, \hat{\pi} \rangle$ to all parties. If the adversary corrupts party i after this point such that $i \notin \{i_{f+1}, \dots, i_{2f}\}$, Sim calls its discrete log oracle twice to get $z_i = \text{DL}(Z'_i), \hat{z}_i = \text{DL}(\hat{Z}'_i)$. On the other hand, if the adversary corrupts party i such that $i = i_k$ for some $k \in \{f+1, \dots, 2f\}$, uses the previously sampled z_{i_k} and \hat{z}_{i_k} instead and does not call its DL oracle. It then generates i 's view following the Bingo simulator (described in the proof of secrecy) in all invocations of Bingo with honest dealers except for one nonfaulty dealer $j \in \text{dealers}$ (including generating appropriate α and β polynomials for i). For this dealer j , it uniformly samples a degree- f polynomial $\beta_{i,j}(Y)$ such that $\beta_{i,j}(0) = z_i - \sum_{k \in \text{dealers} \setminus \{j\}} \beta_{i,k}(0)$ and $\alpha_{k,j}(\omega_i) = \beta_{i,j}(\omega_k)$ for all corrupted k . Similarly, it samples a degree- f polynomial $\hat{\beta}_{i,j}(Y)$ such that $\hat{\beta}_{i,j}(0) = \hat{z}_i - \sum_{k \in \text{dealers} \setminus \{j\}} \hat{\beta}_{i,k}(0)$ and $\hat{\alpha}_{k,j}(\omega_i) = \hat{\beta}_{i,j}(\omega_k)$ for every corrupted k . Again following the Bingo simulator, Sim calls SimOpen to define $\alpha_{i,j}, \hat{\alpha}_{i,j}$ given the sampled $\beta_{i,j}, \hat{\beta}_{i,j}$, i.e. computes $\alpha_{i,j}, \hat{\alpha}_{i,j} \stackrel{\S}{\leftarrow} \text{SimOpen}(\tau_{s,j}, \mathbf{cm}_{i,j}, c_{i,j}, \{\omega_k, \beta_{k,j}(\omega_i), \hat{\beta}_{k,j}(\omega_i)\}_{k \in C})$, with C being the set of currently corrupted parties. Finally, in order to generate i 's view as a dealer, Sim calls SimOpen on $f+1-m$ values $\mathbf{cm}_{j,i}$ for nonfaulty parties j , where m is the current number of corrupted parties, including i . Calling SimOpen on those values yields $f+1-m$ pairs of polynomials $\alpha_{j,i}, \hat{\alpha}_{j,i}$. Sim then generates i 's view of the BingoShare invocation with it as dealer as if it sampled the polynomials $\phi_i, \hat{\phi}_i$ such that $\phi_i(X, \omega_j) = \alpha_{j,i}(X)$ and $\hat{\phi}_i(X, \omega_j) = \hat{\alpha}_{j,i}(X)$ for every j as defined above and for every corrupted j . Following that, Sim adds i to the set of corrupted parties and continues in the simulation. When some nonfaulty party completes the ADKG protocol, let $j_{\ell+1}, \dots, j_{\ell+m}$ be the indices of the parties corrupted after the first nonfaulty party completed the VABA protocol such that for every $k \in \{\ell+1, \dots, \ell+m\}$, $j_k \notin I'$. Sim chooses indices $j_{\ell+m+1}, \dots, j_f \notin I'$ of parties that weren't corrupted by the adversary and calls $\text{DL}(Z'_{j_k})$ and $\text{DL}(\hat{Z}'_{j_k})$ for every $k \in \{\ell+m+1, \dots, f\}$. Finally, Sim outputs Z_0 as pk and terminates.

We now consider the matrix defined by the algebraic coefficients given by Sim when querying its DL oracle, with the goal of proving that it is invertible. For each $i \in [\ell]$, the algebraic representation for the oracle call $\text{DL}(Z_i)$ is simply the indicator vector that equals 1 in the coordinate corresponding to the input element Z_i and 0 elsewhere. Similarly, for each $i \in [\ell]$, the algebraic representation for $\text{DL}(\hat{Z}_i)$ is the indicator vector for \hat{Z}_i and the algebraic representation of $\text{pk} = Z_0$ is the indicator vector for Z_0 . We can thus rearrange the rows and columns of the matrix—which does not affect its invertibility—so that the first 2ℓ rows and columns are the indicator vectors corresponding to the elements $Z_1, \dots, Z_\ell, \hat{Z}_1, \dots, \hat{Z}_\ell$. The remaining algebraic expressions for each $\text{DL}(Z'_i)$ call

result from interpolating Z'_0, Z'_1, \dots, Z'_f and then evaluating at ω_i , both of which are linear functions. The algebraic expressions for \hat{Z}'_i are computed in a similar fashion.

The first set of elements were not used in forming any of the Z'_i, \hat{Z}'_i group elements, and thus the rearranged matrix is a block matrix of the form $L = \begin{pmatrix} I & 0 \\ 0 & A \end{pmatrix}$, where I is the identity matrix of size $2\ell \times 2\ell$ and A is a matrix with the algebraic representation of the $\text{DL}(Z'_i)$ and $\text{DL}(\hat{Z}'_i)$ calls, as well as the algebraic representation of Z_0 .

In order to show that L is invertible, it is enough to show that A is invertible, since I is trivially invertible. We do that by showing that the linear transformation defined by A is invertible. Let $j_{\ell+1}, \dots, j_f$ be defined as above. Then A represents some linear transformation from $Z_0, Z_{\ell+1}, \dots, Z_f, \hat{Z}_{\ell+1}, \dots, \hat{Z}_f$ to $\text{pk}, Z'_{j_{\ell+1}}, \dots, Z'_{j_f}, \hat{Z}'_{j_{\ell+1}}, \dots, \hat{Z}'_{j_f}$. This function has the same size domain and range (since the number of elements is the same), so to prove that it is invertible it suffices to show that it is one-to-one. Assume that two sets of inputs $Z_0, Z_{\ell+1}, \dots, Z_f, \hat{Z}_{\ell+1}, \dots, \hat{Z}_f$ and $X_0, X_{\ell+1}, \dots, X_f, \hat{X}_{\ell+1}, \dots, \hat{X}_f$ yield the same output $\text{pk}, Z'_{j_{\ell+1}}, \dots, Z'_{j_f}, \hat{Z}'_{j_{\ell+1}}, \dots, \hat{Z}'_{j_f}$. The discrete logs of Z'_0, Z'_1, \dots, Z'_n and Z'_τ all lie on the same f -degree polynomial, and thus any $f+1$ such elements define the polynomial fully and the rest of the points. Therefore, the elements $Z'_0, Z'_{i_1}, \dots, Z'_{i_\ell}, Z'_{j_{\ell+1}}, \dots, Z'_{j_f}$ fully define the entire set Z'_0, Z'_1, \dots, Z'_n . In this case, $Z'_0, Z'_{j_{\ell+1}}, \dots, Z'_{j_f}$ are all parts of the output of the function, and $Z'_{i_1}, \dots, Z'_{i_\ell}$ are constants computed directly by Sim. Therefore, the function's output uniquely defines $Z'_0, Z'_1, \dots, Z'_n, Z'_\tau$. Note that by construction $Z'_0 = Z_0 = X_0$ and also $Z'_{i_k} = Z_k = X_k$ for every $k \in \{\ell+1, \dots, f\}$, and thus the first half of the inputs is equal. In addition, \hat{Z}'_τ is uniquely defined given the previous values, and thus $\hat{Z}'_\tau, \hat{Z}'_{i_1}, \dots, \hat{Z}'_{i_\ell}, \hat{Z}'_{j_{\ell+1}}, \dots, \hat{Z}'_{j_f}$ define the group elements $\hat{Z}'_1, \dots, \hat{Z}'_n$. Therefore, for similar reasons, $\hat{Z}'_{i_k} = \hat{Z}'_k = \hat{X}_k$ for every $k \in \{\ell+1, \dots, f\}$. In other words, all elements of the input must be equal, and thus the function is one-to-one.

From [Theorem 7](#), the simulated runs of the BingoShare protocol are computationally indistinguishable from normal runs of the protocol. The simulator then simulates the ADKG protocol entirely honestly up to line 20. In the non-simulated invocation of `BingoSumExpAndRec`, each nonfaulty party i sends a “key share” message with $Y_i = g^{\sum_{j \in \text{dealers}} \beta_{i,j}(0)}$, $\hat{Y}_i = \hat{g}^{\sum_{j \in \text{dealers}} \hat{\beta}_{i,j}(0)}$, π_i being the unique KZG proof for which `Verify'` verifies, and two proofs of knowledge. Importantly, the pair of polynomials $\phi_\Sigma = \sum_{j \in \text{dealers}} \phi_j$ and $\hat{\phi}_\Sigma = \sum_{j \in \text{dealers}} \hat{\phi}_j$ satisfy $\text{cm}_0 = \text{Commit}(\phi_\Sigma; \hat{\phi}_\Sigma)$. Note that $\sum_{j \in \text{dealers}} \beta_{i,j}(0) = \sum_{j \in \text{dealers}} \phi_j(\omega_i, 0)$ and similarly $\sum_{j \in \text{dealers}} \hat{\beta}_{i,j}(0) = \sum_{j \in \text{dealers}} \hat{\phi}_j(\omega_i, 0)$. From [Theorem 7](#), before some nonfaulty party calls `BingoReconstruct(0)` on a value shared by a nonfaulty dealer, the value is entirely independent of the adversary's view. This is because the simulator could complete the run to correctly reconstruct any possible secret from that point on. Therefore, since the one nonfaulty dealer in `dealers` uniformly

sampled its secrets, the sum is uniform and independent of the adversary’s view. In the simulation, the nonfaulty parties also send messages with Z'_i, \hat{Z}'_i such that their discrete logs lie on uniformly sampled polynomials of the same degree which are consistent with cm_0 , and with the points $\sum_{j \in \text{dealers}} \phi_j(\omega_k, 0)$ and $\sum_{j \in \text{dealers}} \hat{\phi}_j(\omega_k, 0)$ of faulty parties. In addition, the proof π is the unique proof for which Verify' verifies, and the proofs of knowledge are perfectly simulated. In addition, whenever a party i is corrupted during BingoSumExpAndRec , its view is made consistent with Z'_i, \hat{Z}'_i and the rest of the BingoShare simulation is identical to the simulation described in [Theorem 7](#), and thus its view is sampled identically as well.

6 Acknowledgements

We would like to thank Alin Tomescu, Kobi Gurkan, Julian Loss, and Renas Bacho for many insightful discussions.

References

- [1] I. Abraham, G. Asharov, S. Patil, and A. Patra. “Asymptotically Free Broadcast in Constant Expected Time via Packed VSS”. In: *IACR Cryptol. ePrint Arch.* (2022), p. 1266. URL: <https://eprint.iacr.org/2022/1266>.
- [2] I. Abraham, G. Asharov, and A. Yanai. “Efficient Perfectly Secure Computation with Optimal Resilience”. In: *J. Cryptol.* 35.4 (2022), p. 27. DOI: [10.1007/s00145-022-09434-2](https://doi.org/10.1007/s00145-022-09434-2). URL: <https://doi.org/10.1007/s00145-022-09434-2>.
- [3] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. “Reaching Consensus for Asynchronous Distributed Key Generation”. In: *PODC '21: ACM Symposium on Principles of Distributed Computing*. 2021, pp. 363–373.
- [4] N. AlHaddad, M. Varia, and H. Zhang. “High-Threshold AVSS with Optimal Communication Complexity”. In: *Financial Cryptography and Data Security*. 2021, pp. 479–498.
- [5] R. Bacho and J. Loss. “On the Adaptive Security of the Threshold BLS Signature Scheme”. In: *Cryptology ePrint Archive* (2022).
- [6] M. Backes, A. Datta, and A. Kate. “Asynchronous Computational VSS with Reduced Communication Complexity”. In: *Topics in Cryptology – CT-RSA 2013*. 2013, pp. 259–276.
- [7] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. “The One-More-RSA-Inversion Problems and the Security of Chaum’s Blind Signature Scheme.” In: *Journal of Cryptology* 16.3 (2003).
- [8] M. Bellare and P. Rogaway. “The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs”. In: *Advances in Cryptology – EUROCRYPT 2006*. Vol. 4004. Lecture Notes in Computer Science. Springer, 2006, pp. 409–426.

- [9] M. Ben-Or, R. Canetti, and O. Goldreich. “Asynchronous secure computation”. In: *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*. 1993, pp. 52–61.
- [10] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. “Fast Reed-Solomon Interactive Oracle Proofs of Proximity”. In: *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. 2018, 14:1–14:17.
- [11] E. Ben-Sasson, L. Goldberg, S. Kopparty, and S. Saraf. “DEEP-FRI: Sampling Outside the Box Improves Soundness”. In: *11th Innovations in Theoretical Computer Science Conference, ITCS*. 2020, 5:1–5:32.
- [12] D. Boneh and X. Boyen. “Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups”. In: *Journal of Cryptology* 21.2 (2008), pp. 149–177.
- [13] D. Boneh, B. Lynn, and H. Shacham. “Short Signatures from the Weil Pairing”. In: *J. Cryptol.* 17.4 (2004), pp. 297–319.
- [14] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. “Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting”. In: *Advances in Cryptology - EUROCRYPT 2016*. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 327–357.
- [15] S. Bowe, A. Gabizon, and I. Miers. *Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model*. Cryptology ePrint Archive, Paper 2017/1050. 2017.
- [16] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. “Bulletproofs: Short Proofs for Confidential Transactions and More”. In: *2018 IEEE Symposium on Security and Privacy*. 2018, pp. 315–334.
- [17] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Stroh. “Asynchronous verifiable secret sharing and proactive cryptosystems”. In: *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002*. 2002, pp. 88–97.
- [18] C. Cachin, K. Kursawe, and V. Shoup. “Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography”. In: *Journal of Cryptology* 18 (2005), pp. 219–246.
- [19] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. “Adaptive Security for Threshold Cryptosystems”. In: *Advances in Cryptology — CRYPTO’ 99*. Ed. by M. Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 98–116.
- [20] R. Canetti and T. Rabin. “Fast asynchronous Byzantine agreement with optimal resilience”. In: *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*. 1993, pp. 42–51.
- [21] I. Cascudo and B. David. “ALBATROSS: Publicly Attestable Batched Randomness Based On Secret Sharing”. In: *Advances in Cryptology - ASIACRYPT 2020*. Vol. 12493. Lecture Notes in Computer Science. Springer, 2020, pp. 311–341.
- [22] I. Cascudo and B. David. “SCRAPE: Scalable Randomness Attested by Public Entities”. In: *Proceedings of the 15th International Conference on*

- Applied Cryptography and Network Security, ACNS 2017*. Springer International Publishing, 2017, pp. 537–556.
- [23] A. Chopard, M. Hirt, and C. Liu-Zhang. “On Communication-Efficient Asynchronous MPC with Adaptive Security”. In: *Theory of Cryptography - 19th International Conference, TCC 2021*. Vol. 13043. Lecture Notes in Computer Science. Springer, 2021, pp. 35–65.
- [24] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. “Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults (Extended Abstract)”. In: *26th Annual Symposium on Foundations of Computer Science*. 1985, pp. 383–395.
- [25] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. “Efficient Multiparty Computations Secure Against an Adaptive Adversary”. In: *Advances in Cryptology — EUROCRYPT ’99*. Ed. by J. Stern. Springer Berlin Heidelberg, 1999, pp. 311–326.
- [26] P. Daian et al. “Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges”. In: *IEEE Symposium on Security & Privacy*. 2020.
- [27] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman. “Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus”. In: *EuroSys ’22*. ACM, 2022, pp. 34–50.
- [28] S. Das, Z. Xiang, and L. Ren. “Asynchronous Data Dissemination and its Applications”. In: *CCS ’21: 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 2705–2721.
- [29] S. Das, T. Yurek, Z. Xiang, A. Miller, L. Kokoris-Kogias, and L. Ren. “Practical Asynchronous Distributed Key Generation”. In: *2022 IEEE Symposium on Security and Privacy (SP)*. 2022, pp. 2518–2534.
- [30] D. Dolev and R. Reischuk. “Bounds on Information Exchange for Byzantine Agreement”. In: *J. ACM* 32.1 (1985), pp. 191–204.
- [31] P. Feldman and S. Micali. “Optimal Algorithms for Byzantine Agreement”. In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*. ACM, 1988, pp. 148–161.
- [32] M. K. Franklin and M. Yung. “Communication Complexity of Secure Computation (Extended Abstract)”. In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*. ACM, 1992, pp. 699–710.
- [33] G. Fuchsbauer, E. Kiltz, and J. Loss. “The Algebraic Group Model and its Applications”. In: *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference*. Vol. 10992. Lecture Notes in Computer Science. Springer, 2018, pp. 33–62.
- [34] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra (3. ed.)*. Cambridge University Press, 2013. ISBN: 978-1-107-03903-2.
- [35] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. “Secure Distributed Key Generation for Discrete-Log Based Cryptosystems”. In: *J. Cryptol.* 20.1 (2007), pp. 51–83.
- [36] O. Goldreich, S. Micali, and A. Wigderson. “How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority”. In: *Pro-*

- ceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA.* ACM, 1987, pp. 218–229.
- [37] K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. “Aggregatable Distributed Key Generation”. In: *Advances in Cryptology - EUROCRYPT 2021*. Vol. 12696. Lecture Notes in Computer Science. Springer, 2021, pp. 147–176.
 - [38] A. Kate, A. Miller, and T. Yurek. *Brief Note: Asynchronous Verifiable Secret Sharing with Optimal Resilience and Linear Amortized Overhead*. 2019. arXiv: [1902.06095](https://arxiv.org/abs/1902.06095) [cs.CR].
 - [39] A. Kate, G. M. Zaverucha, and I. Goldberg. “Constant-Size Commitments to Polynomials and Their Applications”. In: *Advances in Cryptology - ASIACRYPT 2010*. 2010, pp. 177–194.
 - [40] M. Kohlweiss, M. Maller, J. Siim, and M. Volkhov. “Snarky Ceremonies”. In: *Advances in Cryptology - ASIACRYPT 2021*. Vol. 13092. Lecture Notes in Computer Science. Springer, 2021, pp. 98–127.
 - [41] E. Kokoris Kogias, D. Malkhi, and A. Spiegelman. “Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures”. In: *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020.
 - [42] A. Patra, A. Choudhury, and C. P. Rangan. “Efficient Asynchronous Verifiable Secret Sharing and Multiparty Computation”. In: *Journal of Cryptology* 28 (2015), pp. 49–109.
 - [43] K. Qin, L. Zhou, and A. Gervais. “Quantifying Blockchain Extractable Value: How dark is the forest?” In: *IEEE Symposium on Security & Privacy*. 2022.
 - [44] E. Syta et al. “Scalable Bias-Resistant Distributed Randomness”. In: *38th IEEE Symposium on Security and Privacy*. San Jose, CA, May 2017.
 - [45] T. Yurek, L. Luo, J. Fairoze, A. Kate, and A. K. Miller. “hbACSS: How to Robustly Share Many Secrets”. In: *Proceedings of the Network and Distributed System Security Symposium (NDSS) 2022*. 2022.

A Proofs and Additional Properties for Our PCS

A.1 A proof of interpolation binding (Lemma 1)

Proof. Let \mathcal{A} be an algebraic adversary against $\mathbf{G}_{\mathcal{A}}^{\text{int-binding}}(1^\lambda)$. We demonstrate the existence of adversaries $\mathcal{B}_1, \mathcal{B}_2$ such that

$$\Pr[\mathbf{G}_{\mathcal{A}}^{\text{int-binding}}(1^\lambda)] \leq \mathbf{Adv}_{\mathcal{B}_1}^{\text{dlog}}(1^\lambda) + \mathbf{Adv}_{\mathcal{B}_2}^{\text{qsdh}}(1^\lambda)$$

We proceed by transitioning to a game $\mathbf{G}_{\mathcal{A}}^1(1^\lambda)$ such that

$$\begin{aligned} \Pr[\mathbf{G}_{\mathcal{A}}^{\text{int-binding}}(1^\lambda)] - \Pr[\mathbf{G}_{\mathcal{A}}^1(1^\lambda)] &\leq \mathbf{Adv}_{\mathcal{B}_1}^{\text{dlog}}(1^\lambda) \\ \Pr[\mathbf{G}_{\mathcal{A}}^1(1^\lambda)] &\leq \mathbf{Adv}_{\mathcal{B}_2}^{\text{qsdh}}(1^\lambda). \end{aligned}$$

The two properties combined give us our result.

$\mathbf{G}_{\mathcal{A}}^{\text{int-binding}}(1^\lambda) \rightarrow \mathbf{G}_{\mathcal{A}}^1(1^\lambda)$: Let $\mathbf{G}_{\mathcal{A}}^1(1^\lambda)$ be the game that initially behaves identically to $\mathbf{G}_{\mathcal{A}}^{\text{int-binding}}(1^\lambda)$. When \mathcal{A} returns $(\mathbf{A}, \{(\omega_i, m_i, \hat{m}_i, \pi_i)\}_{i \in [d+1]}) \stackrel{\$}{\leftarrow} \mathcal{A}(\text{srs})$ and the algebraic representations $c(X), \hat{c}(X), q(X), \hat{q}(X)$ such that $\mathbf{A} = g^{c(\tau)} \hat{g}^{\hat{c}(\tau)}$ and $\pi_i = g^{q_i(\tau)} \hat{g}^{\hat{q}_i(\tau)}$, then check whether for all i we have that

$$e(g^{c(\tau)-m_i}, h) = e(g^{q_i(\tau)}, h^{\tau-\omega_i}) \wedge e(\hat{g}^{\hat{c}(\tau)-\hat{m}_i}, h) = e(\hat{g}^{\hat{q}_i(\tau)}, h^{\tau-\omega_i})$$

If this event happens then $\mathbf{G}_{\mathcal{A}}^1(\lambda)$ sets a flag `bad` and returns 1.

Since these two games are identical until this event E , $\Pr[\mathbf{G}_{\mathcal{A}}^{\text{int-binding}}(\lambda)] - \Pr[\mathbf{G}_{\mathcal{A}}^1(\lambda)] = \Pr[E]$. We design an adversary \mathcal{B}_1 against `dlog` such that $\Pr[E] \leq \mathbf{Adv}_{\mathcal{B}_1}^{\text{dlog}}(1^\lambda)$. \mathcal{B}_1 behaves as follows.

$$\begin{aligned} &\mathcal{B}_1(g, \hat{g}) \\ &\tau \stackrel{\$}{\leftarrow} \mathbb{F} \\ &\text{srs} \leftarrow \{g^{\tau^i}, \hat{g}^{\tau^i}, h^{\tau^i}\}_{i=0}^{d_1} \\ &((\mathbf{A} \mid c(X), \hat{c}(X)), \{(\omega_i, m_i, \hat{m}_i, (\pi_i \mid q_i(X), \hat{q}_i(X)))\}_{i \in [d+1]}) \stackrel{\$}{\leftarrow} \mathcal{A}(\text{srs}) \\ &\text{for } 1 \leq i \leq d+1 : \\ &\quad u_i \leftarrow c(\tau) - m_i - q_i(\tau)(\tau - \omega_i) \\ &\quad \hat{u}_i \leftarrow \hat{q}_i(\tau)(\tau - \omega_i) - \hat{c}(\tau) + \hat{m}_i \\ &\quad \text{if } \hat{u}_i \neq 0 \text{ then return } u_i / \hat{u}_i \end{aligned}$$

If $\mathbf{G}_{\mathcal{A}}^{\text{int-binding}}(1^\lambda)$ returns 1 but $\mathbf{G}_{\mathcal{A}}^1(1^\lambda)$ returns 0 then (1) all proofs verify and (2) there exists some i such that $\hat{u}_i \neq 0$. Indeed if

$$e(\hat{g}^{\hat{c}(\tau)-\hat{m}_i}, h) \neq e(\hat{g}^{\hat{q}_i(\tau)}, h^{\tau-\omega_i})$$

then $\hat{c}(\tau) - \hat{m}_i \neq \hat{q}_i(\tau)(\tau - \omega_i)$. Further, since the verification equation passes we have that

$$\begin{aligned} &e(g^{c(\tau)-m_i} \hat{g}^{\hat{c}(\tau)-\hat{m}_i}, h) = e(g^{q_i(\tau)} \hat{g}^{\hat{q}_i(\tau)}, h^{\tau-\omega_i}) \\ &\Rightarrow e(g^{c(\tau)-m_i}, h) e(g^{-q_i(\tau)}, h^{\tau-\omega_i}) = e(\hat{g}^{-(\hat{c}(\tau)-\hat{m}_i)}, h) e(\hat{g}^{\hat{q}_i(\tau)}, h^{\tau-\omega_i}) \\ &\Rightarrow g^{u_i} = \hat{g}^{\hat{u}_i} \end{aligned}$$

and \mathcal{B}_1 returns a valid discrete logarithm.

$\mathbf{G}_{\mathcal{A}}^1(1^\lambda)$: We design an adversary \mathcal{B}_2 against \mathbf{q} -sdh such that

$$\Pr[\mathbf{G}_{\mathcal{A}}^1(1^\lambda)] \leq \mathbf{Adv}_{\mathcal{B}_2}^{\mathbf{q}\text{-sdh}}(1^\lambda)$$

that behaves as follows.

$$\begin{aligned} & \underline{\mathcal{B}_2(g, h, g^\tau, h^\tau \dots, g^{\tau^{d_1}}, h^{\tau^{d_1}})} \\ & x \xleftarrow{\$} \mathbb{F}, \mathbb{G} \\ & \text{srs} \leftarrow \{g^{\tau^i}, g^{x\tau^i}, h^{\tau^i}\}_{i=0}^{d_1} \\ & ((A \mid c(X), \hat{c}(X)), \{(\omega_i, m_i, \hat{m}_i, (\pi_i \mid q_i(X), \hat{q}_i(X)))\}_{i \in [d+1]}) \xleftarrow{\$} \mathcal{A}(\text{srs}) \\ & \text{for } 1 \leq i \leq d+1 : \\ & \quad a(X) = c(X) - m_i - q(X)(X - \omega_i) \\ & \quad \text{if } a(X) \neq 0: \\ & \quad \quad \text{let } j \text{ be smallest such that } a_j \neq 0 \\ & \quad \quad \text{return } (g^{-a_j^{-1} \sum_{k=j+1}^{d_1} a_k}, 0) \\ & \quad \hat{a}(X) = \hat{c}(X) - \hat{m}_i - \hat{q}(X)(X - \omega_i) \\ & \quad \text{if } \hat{a}(X) \neq 0: \\ & \quad \quad \text{let } j \text{ be smallest such that } \hat{a}_j \neq 0 \\ & \quad \quad \text{return } (g^{-\hat{a}_j^{-1} \sum_{k=j+1}^{d_1} \hat{a}_k}, 0) \end{aligned}$$

If $\mathbf{G}_{\mathcal{A}}^1(1^\lambda)$ returns 1 then either

1. $c(X) \neq \text{Interpolate}(\{\omega_i, m_i\}_{i \in [d+1]})$;
2. or $\hat{c}(X) \neq \text{Interpolate}(\{\omega_i, \hat{m}_i\}_{i \in [d+1]})$

Thus there exists some i such that either

1. $c(\omega_i) \neq m_i$;
2. or $\hat{c}(\omega_i) \neq \hat{m}_i$;

But then either

1. $c(X) - m_i - q(X)(X - \omega_i) \neq 0$;
2. or $\hat{c}(X) - \hat{m}_i - \hat{q}(X)(X - \omega_i) \neq 0$;

since they don't evaluate to 0 at ω_i and hence \mathcal{B}_2 returns a correct \mathbf{q} -sdh solution.

A.2 A proof of **GetProofs** correctness (Lemma 2)

Proof. Suppose we have such values $\mathbf{C}, \mathbf{A}, \{(v_i, y_i, \hat{y}_i, \pi_i)\}$. Then the commitment \mathbf{C} is some bivariate commitment and there exists (at least one) $c(X), \hat{c}(X)$ degree- d_1 polynomials such that

$$\mathbf{C} = (g^{c_0} \hat{g}^{\hat{c}_0}, \dots, g^{c_{d_1}} \hat{g}^{\hat{c}_{d_1}}).$$

Where the evaluations verify we have that

$$e(\mathbf{A}_{v_i} g^{-y_i} \hat{g}^{-\hat{y}_i}, h) = e(\pi_i, h^{\tau - \omega_j}).$$

Substituting

$$\mathbf{A}_i = \prod_{\ell=0}^{d_1} C_{\ell}^{\omega_{v_i}^{\ell}} = g^{c(\omega_{v_i})} \hat{g}^{\hat{c}(\omega_{v_i})}$$

yields

$$e(g^{c(\omega_{v_i}) - y_{v_i}} \hat{g}^{\hat{c}(\omega_{v_i}) - \hat{y}_{v_i}}, h) = e(\pi_i, h^{\tau - \omega_j}).$$

Thus when we define

$$\gamma_i = (c(\omega_{v_i}) - y_i) / (\tau - \omega_j) \text{ and } \hat{\gamma}_i = (\hat{c}(\omega_{v_i}) - \hat{y}_i) / (\tau - \omega_j)$$

we have that $\pi_i = g^{\gamma_i} \hat{g}^{\hat{\gamma}_i}$.

Now set $p(X) = \text{Interpolate}(\{(\omega_{v_i}, \gamma_i)\}_{i \in [d_1+1]})$ and $\hat{p}(X) = \text{Interpolate}(\{(\omega_{v_i}, \hat{\gamma}_i)\}_{i \in [d_1+1]})$. Then $\mathbf{P} = \text{InterpolateExp}(\{(\omega_{v_i}, \pi_i)\}_{i \in [d_1+1]})$ as defined in `GetProofs`, and thus we have that

$$\mathbf{P} = (g^{p_0} \hat{g}^{\hat{p}_0}, \dots, g^{p_{d_1}} \hat{g}^{\hat{p}_{d_1}}).$$

Now observe that where

$$c(\omega_{v_i}) - y_i = p(\omega_i)(\tau - \omega_j)$$

and $c(X), p(X)$ are degree d_1 polynomials (by definition), then when `GetProofs` defines the degree d_1 polynomial

$$\beta_j(X) \leftarrow \text{Interpolate}(\{(\omega_{v_i}, y_i)\}_{i \in [d_1+1]})$$

then

$$c(X) - \beta_j(X) = p(X)(\tau - \omega_j).$$

Similarly

$$\hat{c}(X) - \hat{\beta}_j(X) = \hat{p}(X)(\tau - \omega_j),$$

and thus

$$e(g^{c(\omega_i) - \beta_j(\omega_i)} \hat{g}^{\hat{c}(\omega_i) - \hat{\beta}_j(\omega_i)}, h) = e(g^{p(\omega_i)} \hat{g}^{\hat{p}(\omega_i)}, h^{\tau - \omega_j}).$$

This means that

$$\text{VerifyEval}(\mathbf{A}, (j, i), \beta_j(\omega_i), \hat{\beta}_j(\omega_i), \bar{\pi}_i) = 1$$

as required.

A.3 Hiding requirements for the univariate PCS

Definition 7. Consider the game $G_{\text{sim}}^A(1^\lambda)$ that works as follows

$$\begin{array}{l}
\text{MAIN}(1^\lambda, d) \\
\text{srs} \xleftarrow{\$} \text{Setup}(1^\lambda, d; \text{tr}) \\
b \xleftarrow{\$} \{0, 1\}; Q \leftarrow \emptyset \\
\alpha(X) \xleftarrow{\$} \mathcal{A}(\text{srs}), \hat{\alpha}(X) \xleftarrow{\$} \mathbb{F}[X] \\
\text{if } b = 0: \quad C \leftarrow \text{KZGCommit}(\text{srs}, \alpha(X); \hat{\alpha}(X)) \\
\text{if } b = 1: \quad (C, c) \xleftarrow{\$} \text{SimCommit}(\text{srs}) \\
b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_b}(\text{srs}, C) \\
\text{return } b = b' \\
\\
\frac{\mathcal{O}_{\text{Eval}_0}(\omega)}{\text{return KZGEval}(\text{srs}, C, \alpha(X), \hat{\alpha}(X), \omega)} \quad \frac{\mathcal{O}_{\text{Eval}_1}(\omega)}{Q' \leftarrow Q \cup \{(\omega, \alpha(\omega), \hat{\alpha}(\omega))\}} \\
\quad \quad \quad (\hat{y}, \pi) \leftarrow \text{SimEval}(\text{srs}, \text{tr}, C, c, \omega, Q') \\
\quad \quad \quad \text{if } \omega \notin Q: \\
\quad \quad \quad \quad Q \leftarrow Q \cup \{(\omega, \alpha(\omega), \hat{y})\} \\
\quad \quad \quad \text{return } (\alpha(\omega), \hat{y}, \pi) \\
\\
\frac{\mathcal{O}_{\text{Open}_0}()}{\text{return } (\alpha(X), \hat{\alpha}(X))} \quad \frac{\mathcal{O}_{\text{Open}_1}()}{\alpha^*, \hat{\alpha}^* \xleftarrow{\$} \text{SimOpen}(\tau_s, (\text{cm}, c), Q)} \\
\quad \quad \quad \text{while } |Q| \leq d + 1: \\
\quad \quad \quad \quad \text{choose } \omega \notin Q \\
\quad \quad \quad \quad Q \leftarrow Q \cup \{(\omega, \alpha^*(\omega), \hat{\alpha}^*(\omega))\} \\
\quad \quad \quad \text{return } (\alpha^*(X), \hat{\alpha}^*(X))
\end{array}$$

with respect to the simulated commitment algorithm $(C, c) \xleftarrow{\$} \text{SimCommit}(\text{srs})$ and the simulated evaluation algorithm $(\hat{y}, \pi) \xleftarrow{\$} \text{SimEval}(\text{srs}, \text{tr}, y, \omega)$. Let the advantage of an adversary \mathcal{A} against hiding be $\text{Adv}_{\mathcal{A}}^{\text{hiding}}(1^\lambda) = |2 \Pr[G_{\text{hiding}}^A(1^\lambda)] - 1|$. We say that a (univariate) polynomial commitment scheme is hiding if there exists a simulator $(\text{SimCommit}(), \text{SimEval}(), \text{SimOpen}())$ such that for all adversaries \mathcal{A} , $\text{Adv}_{\mathcal{A}}^{\text{hiding}}(1^\lambda) \leq \text{negl}(1^\lambda)$.

In [Lemma 7](#) we prove that our univariate polynomial commitment scheme in [Figure 2](#) is hiding. To do this we show the existence of a simulator that can (using a trapdoor) open a polynomial commitment to up to d evaluations that it only learns upon being queried. The simulator is indistinguishable from a real evaluator that does know the contents of its polynomial commitment.

Lemma 7. The KZG polynomial commitment scheme in [Figure 2](#) is hiding.

Proof. Define the simulation algorithm that works as follows

| | |
|---|---|
| $\frac{\text{SimCommit}(\text{srs})}{\begin{array}{l} c \xleftarrow{\mathbb{S}} \mathbb{F} \\ C \leftarrow g^c \\ \text{return } (C, c) \end{array}}$ | $\frac{\text{SimOpen}(\text{srs}, (\tau, x), (C, c), Q)}{\begin{array}{l} Q' \leftarrow Q \\ \text{while } Q' \leq d: \\ \quad \text{choose } v_i \notin Q \\ \quad y_i, \hat{y}_i \xleftarrow{\mathbb{S}} \mathbb{F} \\ \quad (v_i, y_i, \hat{y}_i) \xleftarrow{\mathbb{S}} \mathbb{F} \\ \quad Q' \leftarrow Q' \cup \{(v_i, y_i, \hat{y}_i)\} \\ \\ \alpha(X) \leftarrow \text{Interpolate}(\{(\omega_{v_i}, y_i)\}_{i=1}^d) \\ \hat{c} \leftarrow (c - \alpha(\tau))/x \\ \hat{\alpha}(X) \leftarrow \text{Interpolate}(\{(\omega_{v_i}, \hat{y}_i)\}_{i \in [1, d]} \cup \{(\tau, \hat{c})\}) \\ \text{return } (\alpha(X), \hat{\alpha}(X)) \end{array}}$ |
| $\frac{\text{SimEval}(\text{srs}, (\tau, x), C, c, \omega, Q)}{\begin{array}{l} \text{set } k \text{ s.t. } \omega_k = \omega \text{ for } (\omega_k, y_k, \hat{y}_k) \in Q \text{ the } k\text{th entry in } Q \\ \text{if } k \leq d: \quad \pi \leftarrow (Cg^{-y_k}\hat{g}^{-\hat{y}_k})^{\frac{1}{\tau-\omega}} \\ \\ \text{if } k > d: \\ \quad \alpha(X) \leftarrow \text{Interpolate}(\{(\omega_i, y_i)\}_{i=1}^{d+1}) \\ \quad \hat{c} \leftarrow (c - \alpha(\tau))/x \\ \quad \hat{\alpha}(X) \leftarrow \text{Interpolate}(\{(\omega_i, \hat{y}_i)\}_{i \in [1, d]} \cup \{(\tau, \hat{c})\}) \\ \quad (y_k, \hat{y}_k, \pi) \leftarrow \text{KZGEval}(\text{srs}, C, \alpha(X), \hat{\alpha}(X), \omega) \\ \\ \text{return } (\hat{y}_k, \pi) \end{array}}$ | |

We must argue that SimCommit , SimOpen , SimEval are indistinguishable from KZGCommit , KZGOpen , KZGEval .

Evaluations given as input in Q are identical to the honest evaluations. The first $d-1$ proofs π_1, \dots, π_{d-1} are the unique values satisfying the verifiers equation given C , y_i , \hat{y}_i and thus are distributed identically in the honest and simulated cases. Subsequent evaluations $\hat{\alpha}(\omega_{d+k})$ are uniquely defined given C and the previous evaluations i.e. they are evaluations of the unique points such that $C = g^{\alpha(\tau)}\hat{g}^{\hat{\alpha}(\tau)}$ and $\hat{\alpha}(\omega_i) = \hat{y}_i$. Subsequent proofs are the unique values satisfying the verifiers equation given C , y_i , \hat{y}_i and thus are distributed identically in the honest and simulated cases. The opening $\hat{\alpha}(X)$ is the unique degree $(d-1)$ polynomial that passes through $d-1$ random points and the point $(c - \alpha(\tau))/x$ at τ and is thus distributed identically in the honest and simulated cases.

Lemma 8. *Consider the game $G_{\text{partialsim}}^A(1^\lambda)$ that works as follows*

$$\begin{aligned}
& \text{MAIN}(1^\lambda, 2f + 1, f + 1) \\
b & \xleftarrow{\$} \{0, 1\} \\
\text{if } b = 0: & \\
& (C, c) \leftarrow \text{SimCommit}() \\
\text{if } b = 1: & \\
& \{(\text{CM}_i, d_i) \xleftarrow{\$} \text{SimCommit}()\}_{i=0}^f \\
& (\text{cm}, \mathbf{c}) \leftarrow \text{SimPartialEval}(\text{CM}, \mathbf{d}, V) \\
& (C, c) = (\text{cm}_j, c_j) \xleftarrow{\$} (\text{cm}, \mathbf{c}) \\
b' & \xleftarrow{\$} \mathcal{A}(C, c) \\
& \text{return } b = b'
\end{aligned}$$

The bivariate polynomial commitment scheme in Fig. 3 is such that there exists an algorithm $\text{SimPartialEval}()$ such that $|2 \Pr[\mathcal{G}_{\text{partialsim}}^{\mathcal{A}}(1^\lambda)] - 1| = 0$.

Proof. The simulated partial evaluation algorithm takes as input \mathbf{d} such that $\text{CM}_i = g^{d_i}$ for each commitment CM_i . It runs $(\text{cm}, \mathbf{c}) \leftarrow (\text{DFTExp}(\text{CM}, V), \text{DFT}(\mathbf{d}, V))$ such that $\text{cm}_i = g^{c_i}$. Hence for a random j we have that (cm_j, c_j) this is perfectly indistinguishable from the output of $\text{SimCommit}()$.

B Security Proofs for Bingo

B.1 A proof of correctness (Theorem 5)

Proof. Assume some nonfaulty party completed the BingoShare protocol. Let t be the time the first nonfaulty party completes BingoShare, and define $\phi, \hat{\phi} \leftarrow \text{Ext}(\text{view}_t)$, as defined in Lemma 4. For every $k \in [0, m]$, define $r_k = \phi(\omega_{-k}, \omega_0)$. Before completing the protocol, that nonfaulty party updated α_i to be non- \perp , which in turn means it updated cm after receiving a “commits” message. For the first part of the property, note that if the dealer is nonfaulty, from Lemma 4, it is indeed the case that $r_k = \phi(\omega_{-k}, \omega_0) = s_k$.

Turning to the second part of the property, let i be some nonfaulty party that completed the call to $\text{BingoReconstruct}(k)$ for some $k \in [0, m]$. Before doing so, it must have completed the call to BingoShare and as shown in the termination proof, at that time it had already updated its cm variable. This could have only happened after receiving a “commits” broadcast from the dealer, and from the agreement property of the broadcast protocol it received the same message as described above, and updated its cm variable to contain the values defined above. Party i completed the call to $\text{BingoReconstruct}(k)$, so it first saw that $|\text{shares}_{i,k}| = f + 1$. Following the exact same logic as in Lemma 5, it must be the case that $y_j = \phi(\omega_{-k}, \omega_j)$ for every $(j, y_j) \in \text{shares}_{i,k}$. Therefore, when i interpolates the points, it gets the polynomial $\beta_{-k}(Y) = \phi(\omega_{-k}, Y)$ and outputs $\beta_{-k}(\omega_0) = \phi(\omega_{-k}, \omega_0) = r_k$ as required.

B.2 A proof of termination (Theorem 6)

Proof. We start with the second property, assuming that some nonfaulty party completed BingoShare and showing that the same will eventually hold for any

other nonfaulty party that starts BingoShare, if the commitment scheme satisfies interpolation binding and evaluation binding. Before terminating, this party must have received “done” message from $n - f$ parties, of which at least $f + 1$ are nonfaulty.

Let i be one of those nonfaulty parties. Before sending the message, i saw that $\alpha_i \neq \perp$. At that time its local cm_i variable is equal to $\text{cm}_i = \text{Commit}(\alpha_i(X); \hat{\alpha}_i(X))$. Indeed if a nonfaulty party i updates $\alpha_i(X), \hat{\alpha}_i(X)$ to values other than \perp , then either the dealer sends valid $\alpha'_i, \hat{\alpha}'_i$ (lines 8 - 11) or i receives $2f + 1$ points $(\mathbf{v}, \mathbf{y}, \hat{\mathbf{y}}, \boldsymbol{\pi})$ such that $\text{Verify}(\text{cm}, (\omega_i, \omega_{v_j}), y_j, \hat{y}_j, \pi_j) = 1$ (lines 25 - 33). In the second case, we have that the correctness of $\alpha_i(X), \hat{\alpha}_i(X)$ follows from the interpolation binding of the KZG commitment scheme, as shown in [Lemma 1](#).

Party i updates cm_i after receiving a ⟨“commits”, CM⟩ broadcast from the dealer, setting $\text{cm} \leftarrow \text{PartialEval}(\text{CM}, \{\omega_1, \dots, \omega_n\})$. From the termination and agreement properties of the broadcast protocol, all nonfaulty parties eventually receive the same message and update their respective cm variables to the same values. Since i updated α_i to a value other than \perp , it sent an “row” message to every nonfaulty party. Every nonfaulty j receives that message, sees that $\text{Verify}(\text{cm}, (i, j), \alpha_i(\omega_j), \hat{\alpha}_i(\omega_j), \pi_{\alpha, i, j}) = 1$ because of the correctness property of the commitment scheme, and adds the tuple $(\omega_j, \alpha_i(\omega_j), \hat{\alpha}_i(\omega_j), \pi_{\alpha, i, j})$ to $\text{proofs}_{\beta, j}$. After receiving such a message from the $f + 1$ nonfaulty parties mentioned above, $|\text{points}_{\beta, j}| = f + 1$, so j computes $(y_1, \hat{y}_1, \pi_1), \dots, (y_n, \hat{y}_n, \pi_n) \leftarrow \text{GetProofs}(\text{proofs}_{\beta, j})$, if it hasn’t done so earlier. Then every nonfaulty j sends a “column” message to every party. Every nonfaulty party k receives a “column” message from every nonfaulty party j , and from [Lemma 2](#), k finds that it satisfies the required conditions and adds a pair $(\omega_j, \beta_j(\omega_k))$ to $\text{points}_{\alpha, k}$ and a pair $(\omega_j, \hat{\beta}_j(\omega_k))$ to $\text{points}_{\hat{\alpha}, k}$. After receiving such a message from every nonfaulty party, it has $|\text{points}_{\alpha, j}| \geq 2f + 1$, interpolates the sets $\text{points}_{\alpha, j}$ and $\text{points}_{\hat{\alpha}, j}$ to polynomials, and updates α_j and $\hat{\alpha}_j$. In summary, every nonfaulty party j eventually has $\alpha_j \neq \perp, \hat{\alpha}_j \neq \perp$ and $|\text{proofs}_{\beta, j}| = f + 1$, so every nonfaulty party sends a “done” message. Finally every nonfaulty party i receives a “done” message from at least $n - f$ nonfaulty parties, and has the same conditions as above, so it completes the protocol. Note that no nonfaulty party completes the protocol before $\alpha_i \neq \perp$ and $|\text{proofs}_{\beta, i}| = f + 1$ so these conditions hold before they terminate, and thus they indeed send the messages described above before terminating.

For the first property, assume the dealer is nonfaulty and that all nonfaulty parties start BingoShare, and show that all nonfaulty parties eventually complete BingoShare if the commitment scheme is correct. In this case, the dealer honestly runs BingoDeal, which means it samples a pair of bivariate polynomials $\phi, \hat{\phi}$ and for every $i \in [n]$ it computes $\alpha_i, \hat{\alpha}_i$, where $\alpha_i(X) = \phi(X, \omega_i)$ and $\hat{\alpha}_i(X) = \hat{\phi}(X, \omega_i)$, and computes $\text{CM} \leftarrow \text{Commit}(\phi, \hat{\phi})$. From the termination and validity properties of the broadcast protocol, every party receives the “commits” broadcast. Afterwards, every nonfaulty party computes $\text{cm} \leftarrow \text{PartialEval}(\text{CM}, \{\omega_1, \dots, \omega_n\})$. From the correctness of [PartialEval](#), for every $i \in [n]$, $\text{cm}_i = \text{Commit}(\alpha_i(X); \hat{\alpha}_i(X))$. The dealer then sends a “polynomials” mes-

sage to every party i with the polynomials $\alpha_i, \hat{\alpha}_i$. Upon receiving that message, every nonfaulty party i updates its local $\alpha_i, \hat{\alpha}_i$ variables, unless it has done so earlier. From this point, at least $f + 1$ nonfaulty parties have $\alpha_i \neq \perp$ and $\hat{\alpha}_i \neq \perp$. Following the exact same proof of the first property, all nonfaulty parties eventually complete the BingoShare protocol.

For the third part of the property, we assume all nonfaulty parties completed BingoShare and invoked BingoReconstruct(k) for some $k \in [0, m]$, and show that they all complete BingoReconstruct(k) if the commitment scheme is correct and satisfies interpolation binding. In that case, every nonfaulty party i computes $\alpha_i(\omega_{-k}), \hat{\alpha}_i(\omega_{-k}), \pi_{\alpha,i,-k} \leftarrow \text{Eval}(\alpha_i, \hat{\alpha}_i, \omega_{-k})$ and sends the message $\langle \text{“rec”}, k, \alpha_i(\omega_{-k}), \hat{\alpha}_i(\omega_{-k}), \pi_{\alpha,i,-k} \rangle$ to all parties. Before completing BingoShare, every nonfaulty i must have $\alpha_i \neq \perp$ and $\hat{\alpha}_i \neq \perp$. As argued for the second termination property, this means that i 's local fields also satisfy $\text{cm}_j \neq \perp$ for every $j \in [n]$. Now, using identical arguments to the ones in the proof of the second part of the termination property, every nonfaulty i receives a “rec” message from every nonfaulty j , sees that $\text{Verify}(\text{cm}, (-k, j), \alpha_j(\omega_{-k}), \hat{\alpha}_j(\omega_{-k}), \pi_{\alpha,j,-k}) = 1$, and adds a pair $(j, \alpha_j(\omega_{-k}))$ to $\text{shares}_{i,k}$. At some point while adding such a pair for every nonfaulty party, i sees that $|\text{shares}_{i,k}| = f + 1$, interpolates $\text{shares}_{i,k}$ to a polynomial β_{-k} , outputs $\beta_{-k}(\omega_0)$ and terminates.

B.3 A proof of secrecy (Theorem 7)

Proof. At a high level, Sim behaves as follows. When it is asked to act as the dealer, Sim generates and broadcasts a simulated commitment CM. Upon party i being corrupted, Sim uses SimOpen to pick polynomials $\alpha_i(X), \hat{\alpha}_i(X)$ that are consistent with the partial commitment cm_i and any other evaluations given out for that party (which are selected uniformly at random). For future messages to party i from some nonfaulty party j , it then simulates “row”, “column”, and “rec” messages using SimEval, according to the fact that $\beta_j(\omega_i) = \alpha_i(\omega_j)$.

In more detail, Sim takes some trapdoor τ_s as input and maintains two sets: a set C of all corrupted parties and a set I of indices for which at least one nonfaulty party invoked BingoReconstruct(i). For convenience, when a nonfaulty party invokes BingoReconstruct(i), we actually add $-i$ to I , since the secret s_i is supposed to be equal to $\phi(\omega_{-i}, \omega_0)$. It also maintains a map M from I to a pair of polynomials $\beta_{-i}(X), \hat{\beta}_{-i}(X)$.

When \mathcal{A} delivers a “start” message to the dealer, Sim computes $(\text{CM}_i, d_i) \xleftarrow{\$} \text{SimCommit}()$ for $i \in [0, f]$ and sets $\text{CM} \leftarrow (\text{CM}_0, \dots, \text{CM}_f)$. Sim also computes $(\text{cm}, \mathbf{c}) \leftarrow \text{SimPartialEval}(\text{CM}, \mathbf{c}, \{\omega_1, \dots, \omega_n\})$ and the corresponding trapdoor openings \mathbf{c} according to the simulated partial evaluation algorithm in Lemma 8. It then adds messages to the buffer as if it were broadcasting $\langle \text{“commits”}, \text{CM} \rangle$ and sending a “polynomials” message for each $i \in [n]$. Because messages are encrypted in transit, we can rely on forward secrecy to observe that the simulator does not need to decide on the actual content of messages sent between nonfaulty parties until the point at which the recipient of those messages gets corrupted.

When \mathcal{A} calls $\mathcal{O}_R(i, k)$ and s_k becomes known to Sim for the first time, Sim samples β_{-k} uniformly at random from the set of degree- f polynomials such

that $\beta_{-k}(\omega_0) = s_k$ and $\beta_{-k}(\omega_j) = \alpha_j(\omega_{-k})$ for all $j \in C$. In choosing $\hat{\beta}_{-k}(X)$, Sim must ensure that it is consistent with CM. Thus Sim chooses \hat{y}_k randomly, sets

$$Q \leftarrow \{\omega_\ell, \beta_\ell(\omega_0), \hat{\beta}_\ell(\omega_0)\}_{\ell \in C \cup I} \cup \{\omega_k, \beta_{-k}(\omega_0), \hat{y}_k\},$$

queries

$$(\beta_{-k}(\omega_0), \hat{s}_k, \pi) \stackrel{\S}{\leftarrow} \text{SimEval}(\tau_s, \text{cm}_0, d_0, \omega_k, Q)$$

and sets $\hat{\beta}_{-k}(X)$ uniformly at random from the set of degree- f polynomials such that $\hat{\beta}_{-k}(\omega_0) = \hat{s}_k$ and $\hat{\beta}_{-k}(\omega_j) = \hat{\alpha}_j(\omega_{-k})$ for all $j \in C$. It then stores $M[k] \leftarrow \beta_{-k}(X), \hat{\beta}_{-k}(X)$, adds $-k$ to I , and adds a “rec” message to the buffer for every other party.

When \mathcal{A} calls $\mathcal{O}_{\text{corr}}$ on a party i that isn't the dealer, and before the dealer is corrupted, Sim defines the evaluation points $\text{eval}_{\alpha,i} \leftarrow \{\omega_j, \beta_j(\omega_i), \hat{\beta}_j(\omega_i)\}_{j \in C \cup I}$ and computes $\alpha_i(X), \hat{\alpha}_i(X) \stackrel{\S}{\leftarrow} \text{SimOpen}(\tau_s, \text{cm}_i, c_i, \text{eval}_{\alpha,i})$. It then defines the evaluation points $\text{eval}_{\beta,i} \leftarrow \{\omega_j, \alpha_j(\omega_i), \hat{\alpha}_j(\omega_i)\}_{j \in C \cup \{i\}}$. Then Sim interpolates to find random $\beta_i(X), \hat{\beta}_i(X)$ such that for every $(\omega_j, y_j, \hat{y}_j) \in \text{eval}_{\beta,i}$ we have that $\beta_i(\omega_j) = y_j$ and $\hat{\beta}_i(\omega_j) = \hat{y}_j$.

Sim then goes over the sequence of messages that were delivered to i in earlier calls to \mathcal{O}^* and updates its transcript and internal state as follows:

- For any message i received from a faulty party,⁷ Sim honestly updates its state according to that message.
- If i received a “polynomials” message from a nonfaulty dealer, Sim updates its state as if it received the message $\langle \text{“polynomials”}, \alpha_i, \hat{\alpha}_i \rangle$.
- If i received a “row” message from a nonfaulty j , Sim updates its state as if it received the message $\langle \text{“row”}, \alpha_j(\omega_i), \hat{\alpha}_j(\omega_i), \pi_{\alpha,j,i} \rangle$, where

$$(\beta_i(\omega_j), \hat{\beta}_i(\omega_j), \pi_{\alpha,j,i}) \stackrel{\S}{\leftarrow} \text{SimEval}(\tau_s, \text{cm}_j, c_j, \omega_i, \{\omega_\ell, \beta_\ell(\omega_j), \hat{\beta}_\ell(\omega_j)\}_{\ell \in C \cup I})$$

and $\beta_i(\omega_j) = \alpha_j(\omega_i)$, $\hat{\beta}_i(\omega_j) = \hat{\alpha}_j(\omega_i)$.

- If i received a “column” message from a nonfaulty j , Sim updates its state as if it received the message $\langle \text{“column”}, \beta_j(\omega_i), \hat{\beta}_j(\omega_i), \pi_{\beta,j,i} \rangle$, where

$$(\beta_j(\omega_i), \hat{\beta}_j(\omega_i), \pi_{\beta,j,i}) \stackrel{\S}{\leftarrow} \text{Eval}(\alpha_i(X), \hat{\alpha}_i(X), \omega_j).$$

- If i received a “rec” message for index k from a nonfaulty party j , Sim retrieves $\beta_{-k}, \hat{\beta}_{-k} \leftarrow M[k]$. It then updates its state as if it received the message $\langle \text{“rec”}, k, \beta_{-k}(\omega_j), \hat{\beta}_{-k}(\omega_j), \pi_{\alpha,j,-k} \rangle$ for

$$(\beta_{-k}(\omega_j), \hat{\beta}_{-k}(\omega_j), \pi_{\alpha,j,-k}) \stackrel{\S}{\leftarrow} \text{SimEval}(\tau_s, \text{cm}_j, c_j, \omega_{-k}, \{\omega_\ell, \beta_\ell(\omega_j), \hat{\beta}_\ell(\omega_j)\}_{\ell \in C \cup I})$$

⁷ We identify the faultiness of the sender according to their corruption status at the step at which their message was added to the buffer, not their status at the step at which it is delivered.

After making these changes, Sim adds i to C and returns view_i to \mathcal{A} .

Going forward, Sim follows these same rules in calls to \mathcal{O}^* for any messages that are being delivered to a party $i \in C$. If instead a call to \mathcal{O}^* results in delivering a message to some nonfaulty party i , Sim acts as follows:

- Organizationally, Sim follows the behavior of a nonfaulty party. This means that if at any point the party should send a “done” message, Sim adds that message to the buffer, and similarly if the party receives an “row” or “column” message before receiving the “commits” message it should ignore them and delay any response until after the “commits” message is delivered.
- If i receives a “polynomials” message, Sim adds an “row” from i to the buffer for every party.
- If i receives an “row” message from a nonfaulty j , Sim assumes the message is valid (i.e., it skips the validity check on the evaluation points in line 18 of `BingoShare`). If instead i receives an “row” message from a faulty j , Sim performs this validity check before proceeding further. If this results in i having accepted $f + 1$ “row” messages, Sim adds a “column” message to the buffer for every party (from party i , if it hasn’t already done so).
- Similarly, if i receives a “column” message from a nonfaulty or faulty j , Sim respectively skips the check in line 27 of `BingoShare` or performs it honestly. If this results in i having accepted $2f + 1$ “column” messages, Sim adds an “row” message to the buffer for every party

To summarize, the messages formed by Sim are as follows:

```

// Acting as the dealer
1 (CM,  $\mathbf{d}$ )  $\stackrel{\$}{\leftarrow}$  SimCommit()
2 ((cm0, cm), (c0, c))  $\leftarrow$  SimPartialEval(CM,  $\mathbf{d}$ , { $\omega_0, \omega_1, \dots, \omega_n$ })

// Forming "row" messages from nonfaulty j
3  $\forall j \in C \beta_i(\omega_j) = \alpha_j(\omega_i), \hat{\beta}_i(\omega_j) = \hat{\alpha}_j(\omega_i)$ 
4 ( $\beta_i(\omega_j), \hat{\beta}_i(\omega_j), \pi_{\alpha,j,i}$ )  $\stackrel{\$}{\leftarrow}$  SimEval( $\tau_s, \mathbf{cm}_j, c_j, \omega_i, \{\omega_\ell, \beta_\ell(\omega_j), \hat{\beta}_\ell(\omega_j)\}_{\ell \in C \cup I}$ )

// Forming "column" messages from nonfaulty j
5 ( $\alpha_i(\omega_j), \hat{\alpha}_i(\omega_j), \pi_{\beta,j,i}$ )  $\stackrel{\$}{\leftarrow}$  Eval( $\alpha_i(X), \hat{\alpha}_i(X), \omega_j$ )

// Forming "rec" messages for k from nonfaulty j
6 sample  $\beta_{-k}(X)$  such that  $\beta_{-k}(\omega_0) = s_k$  and  $\beta_{-k}(\omega_j) = \alpha_j(\omega_{-k}) \forall j \in C$ 
7  $\hat{y}_k \stackrel{\$}{\leftarrow} \mathbb{F}_p; Q \leftarrow \{\omega_\ell, \beta_\ell(\omega_0), \hat{\beta}_\ell(\omega_0)\}_{\ell \in C \cup I} \cup \{\omega_k, \beta_{-k}(\omega_0), \hat{y}_k\}$ 
8 ( $\beta_{-k}(\omega_0), \hat{s}_k, \pi$ )  $\stackrel{\$}{\leftarrow}$  SimEval( $\tau_s, \mathbf{cm}_0, c_0, \omega_k, Q$ )
9 sample  $\hat{\beta}_{-k}(X)$  such that  $\beta_{-k}(\omega_0) = \hat{s}_k$  and  $\hat{\beta}_{-k}(\omega_j) = \hat{\alpha}_j(\omega_{-k}) \forall j \in C$ 
10 ( $\beta_{-k}(\omega_j), \hat{\beta}_{-k}(\omega_j), \pi_{\alpha,j,-k}$ )  $\stackrel{\$}{\leftarrow}$  SimEval( $\tau_s, \mathbf{cm}_j, \omega_{-k}, \{\omega_\ell, \beta_\ell(\omega_j), \hat{\beta}_\ell(\omega_j)\}_{\ell \in C \cup I \cup \{-k\}}$ )

// Forming corr responses for faulty i
11  $\mathbf{eval}_{\alpha,i} \leftarrow \{\omega_j, \beta_j(\omega_i), \hat{\beta}_j(\omega_i)\}_{j \in C \cup I}$ 
12  $\alpha_i(X), \hat{\alpha}_i(X) \stackrel{\$}{\leftarrow}$  SimOpen( $\tau_s, \mathbf{cm}_i, c_i, \mathbf{eval}_{\alpha,i}$ )
13 sample  $\beta_i(X), \hat{\beta}_i(X)$  such that  $\beta_i(\omega_j), \hat{\beta}_i(\omega_j) = \alpha_j(\omega_i), \hat{\alpha}_j(\omega_j) \forall j \in C \cup \{i\}$ 

```

For any set of secrets s_0, \dots, s_m , there exist $|\mathbb{F}|^{(f+1)(2f+1)-m}$ polynomials ϕ such that $\phi(\omega_{-i}, \omega_0) = s_i$ for all $i \in [0, m]$. That is because any set of $2f+1-m$ additional evaluation on $\phi(X, \omega_0)$ uniquely define $\phi(X, \omega_0)$, and there are $f(2f+1)$ additional coefficients to choose independently for the monomials of $\phi(X, Y)$ with Y terms. In addition, there are $|\mathbb{F}|^{(f+1)(2f+1)}$ options for choosing $\hat{\phi}$ because there are $(f+1)(2f+1)$ different coefficients to choose with no restriction. If the adversary runs polynomially many concurrent sessions of BingoShare and BingoReconstruct, the probability that there exist two sessions with the same ϕ and $\hat{\phi}$ polynomials is thus negligible.

If \mathcal{A} calls $\mathcal{O}_{\text{corr}}$ on the dealer, then Sim receives s_0, \dots, s_m and chooses a set $J \subseteq \{1, \dots, n\}$ such that $J \cap C = \emptyset$ and $|J \cup C| = f$. This is always possible since C contains at most f indices. If the dealer did not send any message before being corrupted, Sim updates its state as if it had s_0, \dots, s_m as input and returns from the $\mathcal{O}_{\text{corr}}$ call. Otherwise, Sim would have already acted as the dealer and computed \mathbf{cm}_i, c_i for every $i \in \{0, \dots, n\}$ using SimPartialEval. It then defines $\mathbf{eval}_{\alpha,i} \leftarrow \{\omega_j, \beta_j(\omega_i), \hat{\beta}_j(\omega_i)\}_{j \in C \cup I}$ for every $i \in J$. In addition, for every $j \in \{-m, \dots, 0\} \setminus I$ Sim defines $\beta_j(\omega_0) = s_{-j}$, uniformly samples a value $\hat{\beta}_j(\omega_0) \stackrel{\$}{\leftarrow} \mathbb{F}$ and sets $\mathbf{eval}_{\alpha,0} \leftarrow \{\omega_j, \beta_j(\omega_0), \hat{\beta}_j(\omega_0)\}_{j \in C \cup \{-m, \dots, 0\}}$. After

computing the eval sets, Sim computes $\alpha_i, \hat{\alpha}_i \xleftarrow{\$} \text{SimOpen}(\tau_s, \text{cm}_i, c_i, \text{eval}_{\alpha,i})$ for every $i \in J \cup \{0\}$. Following that, Sim interpolates $\{(\omega_i, \alpha_i)\}_{i \in C \cup J \cup \{0\}}$ to the unique bivariate polynomial ϕ of degree $2f$ in X and f in Y such that $\forall j \in C \cup J \cup \{0\} \phi(X, \omega_j) = \alpha_j(X)$. Similarly, it interpolates $\{(\omega_i, \hat{\alpha}_i)\}_{i \in C \cup J \cup \{0\}}$ to $\hat{\phi}$. Sim then updates the dealer's transcript and internal state as if it sent messages corresponding to having sampled ϕ and $\hat{\phi}$ in BingoDeal . It then updates all nonfaulty parties' transcript and internal state as if they received correct messages from the dealer, and then acted according to BingoShare and BingoDeal throughout the rest of the protocol. From this point on, Sim then honestly follows the Bingo protocol and updates the transcripts and states of nonfaulty parties accordingly. Anytime the adversary calls $\mathcal{O}_{\text{corr}}$ after corrupting the dealer, Sim can then immediately return the associated state.

We now need to argue that interactions with Sim are indistinguishable from interactions with the honest challenger. To do this, we define \mathbf{G}_0 as the game described above; i.e., the game in which \mathcal{A} is interacting with Sim . We then define the following series of games.

\mathbf{G}_1 . We first consider the bad event in which there exists an “row”, “column”, or “rec” message sent from a nonfaulty party to a nonfaulty party such that the Verify checks would fail; in this case the honest party would not proceed but the simulator would (as it skips these checks), thus allowing the adversary to distinguish between them. We define \mathbf{G}_1 as the game in which this bad event does not occur. The only way a nonfaulty party could send a message that didn't verify would be for it to have interpolated a polynomial that is different from the one chosen for it by the dealer, which implies the ability to break evaluation binding. Using “row” messages as an example (the argument is the same for “column” and “rec” messages), the only way for a nonfaulty party j to send a message that does not pass the check in line 18 is for it to have $\alpha_j, \hat{\alpha}_j$ different from the polynomials $\phi(X, \omega_j), \hat{\phi}(X, \omega_j)$. To have interpolated these polynomials, it must have received a “column” message from a party ℓ such that $\text{Verify}(\text{cm}, (\ell, j), y, \hat{y}, \pi_{\beta, \ell, j}) = 1$ but such that $y, \hat{y} \neq \phi(\omega_\ell, \omega_j), \hat{\phi}(\omega_\ell, \omega_j)$. An adversary that outputs

$$\text{cm}, (\ell, j), (y, \hat{y}, \pi_{\beta, \ell, j}), \text{Eval}(\phi(X, \omega_j), \hat{\phi}(X, \omega_j), \omega_\ell)$$

can thus win at evaluation binding, meaning $\Pr[\mathbf{G}_1] - \Pr[\mathbf{G}_0] < \mathbf{Adv}_{\mathcal{B}_1}^{\text{eval-binding}}(\lambda)$.

\mathbf{G}_2 . Case 1, if the dealer is never corrupted: Instead of forming CM using $\text{SimCommit}()$, \mathbf{G}_2 forms $\phi(X, Y)$ and $\hat{\phi}(X, Y)$ as in BingoDeal ; i.e., it samples ϕ so that $\phi(\omega_{-i}, \omega_0) = s_i$ for all $i \in [0, m]$. This means that \mathbf{G}_2 knows the secrets s_0, \dots, s_m in advance where \mathbf{G}_1 does not. This means changing lines 1,2 as

$$\begin{array}{l} 0.5 \quad \boxed{\text{uniformly sample } \phi, \hat{\phi} \text{ s.t. } \phi(\omega_{-i}, \omega_0) = s_i \forall i \in [0, m]} \\ 1 \quad (\text{CM}, \mathbf{d}) \leftarrow \text{SimCommit}() \quad \boxed{\text{CM} \leftarrow \text{Commit}(\phi; \hat{\phi})} \\ 2 \quad ((\text{CM}_0, \text{cm}), (c_0, \mathbf{c})) \leftarrow \text{SimPartialEval}(\text{CM}, \mathbf{d}, \{\omega_0, \omega_1, \dots, \omega_n\}) \\ \quad \boxed{(\text{cm} \leftarrow \text{PartialEval}(\text{CM}, \{\omega_0, \omega_1, \dots, \omega_n\}))} \end{array}$$

Because the game now knows the polynomials for every participant, it also no longer needs to compute `SimOpen` and can instead compute the evaluation points honestly when giving input to `SimEval` or sampling $\beta_i(X), \hat{\beta}_i(X)$. More precisely, it can omit lines 3,6-9,11-13, compute line 5 identically, and can change lines 4 and 10 to be

```
// Forming "row" messages from nonfaulty j
4  $(\beta_i(\omega_j), \hat{\beta}_i(\omega_j), \pi_{\alpha,j,i}) \stackrel{\$}{\leftarrow} \text{SimEval}(\tau_s, \text{cm}_j, c_j, \omega_i, \{\omega_\ell, \beta_\ell(\omega_j), \beta_\ell(\omega_j)\}_{\ell \in \text{CUI}})$ 
    $(\beta_i(\omega_j), \hat{\beta}_i(\omega_j), \pi_{\alpha,j,i}) \stackrel{\$}{\leftarrow} \text{Eval}(\tau_s, \text{cm}_j, \omega_i, \beta_i(\omega_j), \hat{\beta}_i(\omega_j))$ 
// Forming "rec" messages for k from nonfaulty j
10  $\pi_{\alpha,j,-k} \stackrel{\$}{\leftarrow} \text{SimEval}(\tau_s, \text{cm}_j, \omega_{-k}, \{\omega_\ell, \beta_\ell(\omega_j), \hat{\beta}_\ell(\omega_j)\}_{\ell \in \text{CUI} \cup \{-k\}})$ 
     $\pi_{\alpha,j,-k} \stackrel{\$}{\leftarrow} \text{Eval}(\text{cm}_j, \omega_{-k}, \beta_{-k}(\omega_j), \hat{\beta}_{-k}(\omega_j))$ 
```

This game is indistinguishable from G_1 because of the ability to simulate commitments, openings, and evaluations. Note that in G_2 , all opened values are consistent with the degree $2f$ by f polynomials $\phi, \hat{\phi}$, and thus their joint distribution is defined by the distribution over $\phi, \hat{\phi}$. Any such polynomial defines a uniform distribution on each row polynomial, α , or column polynomial, β , given that it is consistent with the previously sampled polynomials, and sampling rows and columns in such a way yields the same distribution over $\phi, \hat{\phi}$. In G_1 , each such α and β are sampled uniformly, under the condition that they are consistent with previously defined α and β polynomials, and overall points are only provided on $f + 1$ polynomials β and $2f + 1$ polynomials α . In other words, sampling α and β polynomials in this way yields the same joint distribution over the evaluations. This means that $\Pr[G_1] - \Pr[G_2] = 0$.

The game G_2 is now identical to the honest game, so we have that

$$\mathbf{Adv}_{\mathcal{A}}^{\text{secretcy}}(\lambda) < \mathbf{Adv}_{B_1}^{\text{eval-binding}}(\lambda) + \mathbf{Adv}_{B_2}^{\text{sim}}(\lambda).$$

Case 2, if the dealer is corrupted at some point: If the dealer is corrupted before ever sending a message, then the simulator perfectly simulates a run of the protocol because all parties send correctly formed messages according to the `Bingo` protocol and update their transcript and state according to those messages.

If the dealer is corrupted after sending a message, then `Sim` computed $\text{cm}_0, \dots, \text{cm}_n$ and then continued the simulation. Before corrupting the dealer, the arguments are identical to the ones above. After corrupting the dealer, `Sim` chooses a set calls `SimOpen` on $f + 1$ different partial commitments cm_i in a way consistent with the view the adversary has seen up until that point. In addition, for the commitment cm_0 , it also does so in a way consistent with the secrets s_0, \dots, s_m . From [Lemma 3](#), the value `CM` computed by `Sim` is a commitment to $\phi, \hat{\phi}$. Following similar arguments to the ones described above, the distribution of $\phi, \hat{\phi}$ defined by `Sim`'s behaviour is the uniform distribution on all bivariate polynomial of the correct degrees, given that they are consistent with the adversary's view and with s_0, \dots, s_m . Therefore, the dealer's transcript and state are

distributed correctly. In addition, the simulated nonfaulty parties send messages identical to the ones they would have sent in the `Bingo` protocol, as instructed by the simulator, and thus their simulation is perfect as well.

C An Adaptively Secure VABA Protocol

To build up to a VABA protocol, we start with weak leader election, following the same outline as Abraham et al. [3]. In a weak leader election protocol, all parties aim to elect a single party. The protocol is “weak” in the sense that there is only a constant probability p that all parties elect the same nonfaulty party. With the remaining $1 - p$ probability, a faulty party might be elected, or different parties might output different leaders. More formally:

Definition 8. *In a weak leader election protocol, there are n parties, and each party i outputs some value $v_i \in [n]$. A weak leader election protocol has the following properties if all nonfaulty parties participate in it:*

- ***p*-Correctness.** *All nonfaulty parties output a value $i \in [n]$ such that party i is nonfaulty with probability p or greater.*
- ***Termination.*** *All nonfaulty parties output some value and terminate.*

A common approach based on secret sharing is as follows: First, every party shares a secret with every other party. Each party waits to complete $f + 1$ invocations of the share protocol, “attaches” those secrets to itself, and informs all other parties about its attached secrets. Afterwards, the parties start reconstructing the secrets. Finally, the leader is the party whose secrets have the largest sum.

Using this protocol blueprint, `Bingo`, and the `Gather` protocol by Abraham et al. [3] with the external validity function `checkValidity` defined in Equation 1, we specify in Algorithm 8 a weak leader election protocol.

Initially, every party samples a random secret for every other party, and shares them all using `BingoShare`. Parties then wait to complete all the `BingoShare` calls for $f + 1$ different dealers before attaching that set of dealers to themselves. Afterwards, each party uses the `Gather` protocol to output a set of attached dealers. This set of attached dealers must be one for which all `BingoShare` calls are guaranteed to terminate. Parties send each other “attach” messages, requesting signatures on the set `attached` of attached dealers. Nonfaulty parties reply with such a signature only after seeing that all `BingoShare` calls have been completed for each party in `attached`, guaranteeing (by termination) that every nonfaulty party eventually completes these calls as well.

After gathering $f + 1$ signatures in `sigs` for their set of attached dealers, `attached`, each party calls `Gather` with input `(attached, sigs)`. By the properties of the `Gather` protocol, all nonfaulty parties eventually complete this call and output a set $\{(j, (\text{attached}_j, \text{sigs}_j))\}$ such that party j input the set `(attachedj, sigsj)` to the protocol. In addition, this set is externally valid, which means (following Equation 1) that there are at least $f + 1$ dealers in `attachedj` and valid signatures

in sig_j , and there exists a *common core* of $n - f$ pairs $(j, (\text{attached}_j, \text{sig}_j))$ that all nonfaulty parties include in their output.

Using these properties, after completing the `Gather` call and outputting some set X , every party broadcasts the indices $I = \{j \mid \exists(j, (\text{attached}_j, \text{sig}_j)) \in X\}$, which allows every other party to call `GatherVerify` and outputs the original set X . Afterwards, for every $(j, (\text{attached}_j, \text{sig}_j)) \in X$, every party waits to complete the `BingoShare` calls for all dealers in attached_j , and then participates in reconstructing the sum of the j -th secrets shared by the dealers in attached_j . The reconstructed value is then associated with party j . Finally, after completing the `Gather` protocol with some set X and having some value associated with every party j such that $(j, (\text{dealers}_j, \text{sig}_j)) \in X$, each party checks which of those parties has the maximal associated value and picks it as leader.

The leader election protocol described in [Algorithm 8](#) resembles the proposal election protocol described by Abraham et al., with two important conceptual differences. First, our leader election protocol uses an interactive packed AVSS protocol instead of a non-interactive PVSS protocol. This means that in order to make sure that a given packed AVSS instance will indeed complete, parties have to check that at least one nonfaulty party completed it. This is done by collecting signatures from at least $f + 1$ parties, as described above. Second, the protocol of Abraham et al. relied on parties being able to aggregate $f + 1$ PVSS transcripts and then forward the aggregated transcript to all parties. Parties then reconstruct the aggregated secret by computing their share of the secret and forwarding it to everybody. In our protocol, there is no transcript that can be forwarded to all parties. Instead, parties inform each other of the dealers they “attached” to themselves. Parties then locally aggregate shares for the appropriate secrets, before reconstructing them using `BingoReconstructSum`.

Due to the similarities of our protocol with the one due to Abraham *et al.*, its security follows naturally from their proof. Intuitively, the properties of the `Gather` protocol guarantee that every nonfaulty party sees the same set dealers_j associated with j . Since no nonfaulty party starts reconstructing the sum of secrets associated with party j before completing `GatherVerify`, no faulty party learns anything about any of the values shared by nonfaulty parties for party j . Party j has to choose at least one nonfaulty dealer to add to dealers_j , so the sum of the values shared by these is uniformly distributed and cannot be biased by j . Therefore, each party has the same probability of having the maximal value overall. If the maximal value ends up being one associated with a nonfaulty party j in the common core of the `Gather` protocol, then all nonfaulty parties see that party in their outputs from the `Gather` protocol. They will then see that j has the maximal value and elect it as leader. The common core contains at least $n - f$ tuples, so at least $n - 2f \geq \frac{n}{3}$ of the parties in it are nonfaulty. All parties thus elect the same nonfaulty leader with at least a $\frac{n}{3} \cdot \frac{1}{n} = \frac{1}{3}$ probability, meaning we achieve $\frac{1}{3}$ -correctness as desired.

The protocol consists of a constant number of all-to-all communication rounds with messages of size $O(\lambda n)$ words, as well as $O(n)$ calls to `BingoShare`, $O(n)$ calls to `BingoReconstructSum`, a call to `Gather` with inputs of size $O(\lambda n)$, and

a single broadcast by each party of messages of size $O(n)$ words. This yields a word complexity of $O(\lambda n^3)$ overall and a round complexity of $O(1)$.

In a proposal election (PE) protocol, every party i has an input x_i . Parties are then required to all output the same x_i such that i is nonfaulty with probability p or greater. In addition, parties need to be able to output proofs that their output is correct, and verify those proofs. The protocol described in [Algorithm 8](#) can be transformed into a proposal election protocol by parties receiving a proposal prop_i as input and calling `Gather` with input $(\text{prop}_i, \text{attached}_i, \text{sigs}_i)$. Then instead of outputting the identifier i of the elected party, they can output its proposal prop_i .

Using this result, it is possible to construct an adaptively secure Validated Byzantine agreement (VABA) protocol.

Definition 9 (VABA). *Let V be a set of possible inputs, and let $\text{validate} : V \rightarrow \{\text{true}, \text{false}\}$ be an external validity function. In a validated asynchronous Byzantine agreement (VABA) protocol, every party i has an externally valid input x_i , and they each output some value $y_i \in V$. A VABA protocol has the following properties if all nonfaulty parties participate in it:*

- **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.*

In particular, using the adaptively secure PE protocol described above in the NWH protocol of Abraham *et al.* (which only assumes the existence of a PKI) yields an adaptively secure VABA protocol with $O(\lambda n^3)$ word complexity.

Algorithm 8 LeaderElection()

```
1:  $\text{dealers}_i \leftarrow \emptyset, \text{attached}_i \leftarrow \emptyset, \text{sigs}_i \leftarrow \emptyset, X_i \leftarrow \emptyset, \text{evals}_i \leftarrow \emptyset$ 
2:  $(r_1, \dots, r_n) \xleftarrow{\$} \mathbb{F}^n$ 
3: call BingoShare as dealer three times: sharing  $r_1, \dots, r_{f+1}$ , sharing  $r_{f+2}, \dots, r_{2f+2}$ ,
   and sharing  $r_{2f+3}, \dots, r_n$ 
4: participate in all three sessions of BingoShare with  $j$  as dealer for every  $j \in [n]$ 
5: upon completing all three BingoShare calls with  $j$  as dealer, do
6:    $\text{dealers}_i \leftarrow \text{dealers}_i \cup \{j\}$ 
7:   if  $|\text{dealers}_i| = f + 1$  then
8:      $\text{attached}_i \leftarrow \text{dealers}_i$ 
9:     send  $\langle \text{"attach"}, \text{attached}_i \rangle$  to all parties
10: upon receiving  $\langle \text{"attach"}, \text{attached}_j \rangle$  from party  $j$ , do
11:   upon  $\text{attached}_j \subseteq \text{dealers}_i$ , do
12:     send  $\langle \text{"signature"}, \text{Sign}(\text{sk}_i, \text{attached}_j) \rangle$  to party  $j$ 
13: upon receiving  $\langle \text{"signature"}, \sigma_j \rangle$  from  $j$ , do
14:   if  $\text{attached}_i \neq \emptyset$  and  $\text{Verify}(\text{pk}_j, \sigma_j, \text{attached}_i) = 1$  then
15:      $\text{sigs}_i \leftarrow \text{sigs}_i \cup \{(j, \sigma_j)\}$ 
16:     if  $|\text{sigs}_i| = f + 1$  then
17:       call  $\text{Gather}_i(\text{attached}_i, \text{sigs}_i)$  with external validity function
         checkValidity
18: upon  $\text{Gather}_i$  outputting the set  $X = \{(j, (\text{attached}_j, \text{sigs}_j))\}$ , do
19:    $X_i \leftarrow X$ 
20:    $I_i \leftarrow \{j \mid \exists (j, (\text{attached}_j, \text{sigs}_j)) \in X_i\}$ 
21:   broadcast  $\langle \text{"indices"}, I_i \rangle$ 
22: upon receiving the first  $\langle \text{"indices"}, I_j \rangle$  message from  $j$ , do
23:   upon  $\text{GatherVerify}_i(I_j)$  outputting  $X_j$  and  $\text{Gather}_i$  outputting some value, do
24:     for all  $(k, (\text{attached}_k, \text{sigs}_k)) \in X_j$  do
25:       upon all BingoShare calls terminating with  $\ell$  as dealer for every  $\ell \in$ 
          $\text{attached}_k$ , do
26:         invoke  $\text{BingoReconstructSum}(\text{attached}_k, k)$ 
27: upon  $\text{BingoReconstructSum}(\text{attached}_k, k)$  terminating with output  $r_k$ , do
28:    $\text{evals}_i \leftarrow \text{evals}_i \cup \{(k, r_k)\}$ 
29: upon  $\forall (k, \text{dealers}_k) \in X_i \exists (k, \text{evaluation}_k) \in \text{evals}_i$  and  $X_i \neq \emptyset$ , do
30:    $\ell \leftarrow \text{argmax}_k \{\text{evaluation}_k \mid (k, (\text{prop}_k, \text{vrf\_dkg}_k)) \in S_i\}$ 
31:   output  $\ell$  and terminate
```
