Lightweight Asynchronous Verifiable Secret Sharing with Optimal Resilience

Victor Shoup[1] and Nigel P. Smart[2,3]

[1] Offchain Labs, New York, USA.
[2] COSIC, KU Leuven, Leuven, Belgium.
[3] Zama Inc., Paris, France.
victor@shoup.net, vshoup@offchainlabs.com
nigel.smart@kuleuven.be, nigel@zama.ai

November 22, 2023

Abstract. We present new protocols for *Asynchronous Verifiable Secret Sharing* for Shamir (i.e., threshold $t < n$) sharing of secrets. Our protocols:

- Use only "lightweight" cryptographic primitives, such as hash functions;
- Can share secrets over rings such as $\mathbb{Z}/(p^k)$ as well as finite fields $\mathbb{F}_q$;
- Provide *optimal resilience*, in the sense that they tolerate up to $t < n/3$ corruptions, where n is the total number of parties;
- Are *complete*, in the sense that they guarantee that if any honest party receives their share then all honest parties receive their shares;
- Employ *batching* techniques, whereby a dealer shares many secrets in parallel, and achieves an amortized communication complexity that is linear in n, at least on the "happy path", where no party *provably* misbehaves.

Table of Contents

1 Introduction

We present new protocols for **asynchronous verifiable secret sharing (AVSS)**. An AVSS protocol allows one party, the dealer, to distribute shares of a secret to parties $P_1, \ldots, P_n$. Important properties of such a protocol are *correctness*, which means that even if the dealer is corrupt, the shares received by the honest parties are valid (i.e., they correspond to points that interpolate a polynomial of correct degree), and *privacy*, which means that if the dealer is honest, an adversary should only learn the shares held by the corrupt parties. A third property that is important in many applications is *completeness*, which means that if the dealer is honest, or if any honest party obtains a share, then eventually all honest parties obtain a share. In this paper, we will only be interested in AVSS protocols that satisfy the completeness property: some authors also call this **asynchronous complete secret sharing (ACSS)**. Our protocols allow the dealer to share secrets that lie in a finite field $\mathbb{F}_q$, or more generally a finite ring, such as $\mathbb{Z}/(p^k)$.

In the asynchronous setting, we assume secure (authenticated and private) point-to-point channels between parties, but we do not assume any bound on how quickly messages are transmitted between parties. In defining completeness, "eventually" means "if and when all messages sent between honest parties are delivered". While there is a vast literature on secret sharing in the *synchronous* communication model, there has been considerably less research in the *asynchronous* model. We feel that this is unfortunate, as the asynchronous model is the only one that corresponds to the practical setting of a wide area network. For this reason, we focus exclusively on the asynchronous model.

It is well known that any AVSS protocol can withstand at most $t < n/3$ corrupt parties. If an AVSS protocol can withstand this many corruptions, we say it provides **optimal resilience**. In this paper, we will focus exclusively on AVSS protocols that provide optimal resilience.

We are mainly focused here in designing AVSS protocols with good communication complexity. We define the communication complexity to be the sum of the length of all messages sent by honest parties (to either honest or corrupt parties) over the point-to-point channels. That said, we are interested protocols with good computational complexity as well.

In many applications, it is possible to run many AVSS protocols together as a "batch". That is, a dealer has many secrets that he wants to share, and can share them all in parallel. Please note that such "batched" secret sharing operations are not to be confused with "packed" secret sharing operations: in a "batched" secret operation (a technique used, for example, in [DN07]), many secrets are shared in parallel, resulting in many ordinary sharings, while in a "packed" secret sharing (a technique introduced in [FY92]), many secrets are packed in a single sharing. With "packed" secret sharing, one must sacrifice optimal resilience, which we are not interested in here. Our focus will be exclusively on "batched" secret sharing. With "batching", it is still possible to achieve optimal resilience, while obtaining very good communication and computational complexity in an *amortized* sense (i.e., per sharing).

We also make a distinction between the "happy path" and the "unhappy path". To enter the "unhappy path", a corrupt party must *provably* misbehave. If this happens, all honest parties will learn of this and can take action: in the short term, the honest parties can safely ignore this party, and in the longer term, the corrupt party can be removed from the network. Also, such provable misbehavior could lead to legal or financial jeopardy for the corrupt party, and this in itself may be enough to discourage such behavior. Note that the "happy path" includes corrupt behavior that cannot be used as reliable evidence to convince other honest parties or an external authority of corrupt behavior — this includes collusion among the corrupt parties, as well as behavior that may

clearly be seen as corrupt by an individual honest party. For these reasons, we believe it makes sense to make a distinction between the complexity of the protocol on the "happy path" versus the "unhappy path".

1.1 Information Theoretic vs Computational Security

Up until now, most research in this area has been focused in two different settings: *information theoretic* and *computational*.

In the information theoretic setting, security is unconditional, while in the computational setting, the protocol may use various cryptographic primitives and the security of the protocol is conditioned on specific cryptographic assumptions. In the information theoretic setting, one can further make a distinction between *statistically secure* protocols, which may be broken with some negligible probability, and *perfectly secure* protocols, which cannot be broken at all. We shall not be particularly interested in the distinction between statistical and perfectly secure information theoretic protocols in this paper.

In the cryptographic setting, the cryptography needed is often quite "heavyweight", being based, for example, on the discrete logarithm problem or even pairings.

- The state of the art in batched, complete AVSS protocols with optimal resilience in the *information theoretic* setting (with statistical security) is the protocol from [CP23], which achieves amortized communication complexity that is *cubic* in n.
- In contrast, the state of the art in batched, complete AVSS protocols with optimal resilience in the *computational* setting is the protocol from [AJM+23], which achieves amortized communication complexity that is *linear* in n. This protocol relies on discrete logarithms and pairings (although as noted in [GS22], pairings are not needed to achieve the same result if we amortize over larger batches).

For both of these protocols, the complexity bounds are *worst case* bounds (making no distinction between a "happy path" and a "unhappy path").

1.2 The space in between: "lightweight" cryptography

In this paper, we explore the space between the information theoretic and computational settings. Specifically, we consider the computational setting, but where we only allow "lightweight" cryptographic primitives, such as collision resistant hash functions and pseudo-random functions. In one of our protocols, we need to make a somewhat nonstandard (but entirely reasonable) assumption about hash functions: a kind of related-key indistinguishability assumption for hash functions, which certainly holds in the random oracle model [BR93]. In another protocol, we fully embrace the random oracle model, which yields an even simpler and more efficient protocol. Both protocols are batched, complete AVSS protocols with optimal resilience that achieve communication complexity that is *linear* in n on the "happy path" and *quadratic* in n on the "unhappy path".

We believe there are several reasons to explore this space of protocols that rely only on "lightweight" cryptography:

- Such protocols are potentially harder to break than protocols that rely on such things as discrete logarithms and pairings. In particular, they provide *post-quantum security*.

– Such protocols will typically exhibit much better *computational complexity* than those that rely on "heavyweight" cryptography. For example, the protocol in [AJM+23] requires that each receiving party perform a constant number of exponentiations and pairings per sharing (in an amortized sense). In contrast, in our protocol, each receiving party only performs a constant number of field operations and hashes per sharing (again, in an amortized sense, at least on the "happy path").

Moreover, using *any form of* cryptography can allow improvements in both communication and computational complexity over protocols using only information theoretic tools.

Our protocols do not require any public-key cryptography. However, as we shall point out, in a practical implementation, it might be advantageous to sparingly use some public-key cryptographic techniques in certain places.

1.3 Fields vs Rings

To our knowledge there has been no work in the asynchronous setting for VSS protocols sharing secrets over rings such as $\mathbb{Z}/(p^k)$, with all prior work in the setting focused on sharing elements in finite fields $\mathbb{F}_q$. In the synchronous setting there has recently been an interest in MPC over rings such as $\mathbb{Z}/(p^k)$, see, for example: [CDE+18,OSV20,CKL21,EXY22] in the dishonest majority (and computational) setting; [ACD+19,ACD+20] in the honest majority setting (and information theoretic) setting; and [JSL22] in the honest majority setting (and computational) setting.[4] The heart of the protocol [ACD+19] is a synchronous VSS protocol for elements in $\mathbb{Z}/(p^k)$, which itself is a natural generalization of the method for fields from [BTH06,BTH08]. The methods from these last two papers are perfectly information theoretically secure.

Another approach, related to [BTH06,BTH08], is that of [DN07]. This is a statistically secure information theoretic MPC protocol that works in the synchronous setting with honest majority, which at its heart performs a highly efficient batched VSS protocol over the finite field $\mathbb{F}_q$. The batch is proved to be correct using a probabilistic checking procedure, which has a negligible probability of being by-passed by an adversary (a similar probabilistic check was used in a different context in [BGR98]). While [DN07] provides only statistical security, its advantage over other techniques is the fact that the batch sizes are larger, resulting in a greater practical efficiency.

In the synchronous setting, generalizing these results from fields to Galois rings appears at first sight to be tricky. The field results are almost all defined for Shamir sharing, which in its standard presentation for n players over $\mathbb{F}_q$, requires $n > q$. When working with rings such as $\mathbb{Z}/(2^k)$ it is not clear, a priori, that the theory for fields will pass over to the ring case. However, by using so-called Galois rings and carefully defining the Shamir evaluation points and other data structures, the entire theory for fields can be carried over to the ring setting with very little change. The original work in this space for rings can be traced back, at least, to [Feh98], with a more complete treatment being provided in [ACD+19]. The last paper generalizes the synchronous protocol from [BTH08] to the case of $\mathbb{Z}/(p^k)$ completely.

In this work we initiate the study of *asynchronous VSS protocols for rings*. As explained above we focus on a middle ground which utilizes lightweight cryptography. Our motivating starting point is the underlying batched synchronous VSS protocol contained in [DN07]. At a high level, this protocol works in the following steps:

[4] For specific access structures, secret sharing over $\mathbb{Z}/(p^k)$ for practical protocols is much older, going back to at least the original Sharemind protocol [BLW08].

1. The sharing party shares a large number of values.
2. After the shares are distributed a random beacon is called in order to generate a random value. In [DN07] this is instantiated with a "standard" traditional VSS protocol.
3. Using the value from the random beacon random linear equations on the originally produced shares are computed and opened. The checking of these random linear combination for correctness implies the original shares are correct, with a negligible probability of success.

We follow the same strategy, but we need to modify this slightly, not only to deal with our asynchronous network situation, but also to deal with the potential uses of rings such as $\mathbb{Z}/(p^k)$. In [DN07], a single linear equation is checked over a field extension, in the case of small q, in order to obtain an appropriate soundness level. In our case we will need to check multiple such equations in parallel, relying on a generalization of the Schwarz-Zippel lemma to rings.

Another technique we employ to move from the synchronous setting of [DN07] to our asynchronous setting is the "encrypt then disperse" technique from [YLF+21]. However, as developed in [YLF+21], this technique relies on "heavyweight" cryptography, including discrete logarithms and pairings. We show how to replace all of this "heavyweight" cryptography by "lightweight" cryptography.

In [DN07] the shared secret is guaranteed to be an element in $\mathbb{F}_q$ if q is large enough to support Shamir secret sharing over $\mathbb{F}_q$, i.e. $n < q$. When sharing secrets in $\mathbb{Z}/(p^k)$ (or a small finite fields $\mathbb{F}_q$ with $n \geq q$), the shares themselves lie in a Galois ring (or field) extension. In fact, a corrupt dealer might share a secret that lies in the extension, rather than in the base ring (or field). In most applications of our AVSS protocol, this will not be an issue (for example, in producing multiplication triples for MPC protocols); however, in some applications, we really need to ensure that the shared elements are indeed in the base ring (or field) and not some extension. In this situation, we require further machinery which we introduce.

1.4 Application to AMPC

Of course, as has already been alluded to, one of the main applications of AVSS over fields is to **asynchronous secure multiparty computation (AMPC)**, especially in the information theoretic setting. The state of art for AMPC with optimal resilience for arithmetic circuits over finite fields in the *information theoretic* setting is the protocol from [CP23], whose communication complexity grows as $n^4 \cdot c_M$, where c_M is the number of multiplication gates in the circuit to be evaluated.

We can use our new "lightweight" cryptographic AVSS protocols as a drop-in replacement for the information-theoretic AVSS protocol in [CP23], which yields an AMPC protocol whose communication complexity grows as $n^2 \cdot c_M$ on the "happy path" and $n^3 \cdot c_M$ on the "unhappy path". One can easily improve the communication on the "happy path" to $n \cdot c_M$ by assuming that $t < (1/3 - \epsilon) \cdot n$ for some constant ϵ. Alternatively, one can achieve the same communication bound with $t < n/3$ at least on a "very happy path" where at least $(2/3 + \epsilon) \cdot n$ parties are actually online and cooperative (which in practice is often reasonable to assume). Indeed, the technique in [CP23], which derives from [CP17], involves a step where we have to wait for $n - t$ parties to each contribute sharings of validated Beaver triples. From this collection of sharings, some number of truly random shared triples may be extracted. Unfortunately, when $n = 3t + 1$, and we only collect $n - t = 2t + 1$ triples, this extraction process yields only one truly random triple. However, in practice, it may make sense to just wait a little while to try to collect more triples. Indeed, if

we can collect $(2/3+\epsilon)\cdot n$ triples, we can extract $\Omega(n)$ truly random triples. Hence, on this "very happy path", the communication complexity grows as $n \cdot c_M$. Note that this pragmatic approach to reducing the communication complexity on this "very happy path" does not require us to assume anything more than $t < n/3$ — so we still obtain optimal resilience, but we also obtain linear communication complexity per multiplication gate on this "very happy path".

As is well known, one can realize AMPC in the computational setting without using AVSS. Indeed, the state of art for AMPC with optimal resilience in the *computational* setting is the protocol from [Coh16], which has a communication complexity that is independent of the circuit size. This protocol relies on very "heavyweight" cryptography: threshold fully homomorphic encryption and threshold signatures. Using somewhat less "heavyweight" cryptography, namely, additively homomorphic threshold encryption, the protocol in [HNP08] has communication complexity that grows as $n^2 \cdot c_M$.

So we see that with our new AVSS protocols, one can achieve secure AMPC in the computational setting with very good communication complexity using only "lightweight" cryptography.

As has already been mentioned, our lightweight AVSS protocol works not only over fields but over rings such as $\mathbb{Z}/(p^k)$. These rings offer many advantages for various forms of MPC computation, especially when the ring is chosen to be $\mathbb{Z}/(2^k)$. It remains an open question as to how the above techniques for AMPC can be extended from fields to rings, given our AVSS protocol as a building block. In future work, we aim to investigate this in our context of utilizing lightweight cryptography.

1.5 The rest of the paper

In Section 2, we review basic concepts such as polynomial interpolation, Reed-Solomon codes, and secret sharing. In particular, in Section 2.3, we give the formal definition of AVSS that we will use throughout the paper. In Section 3, we review the subprotocols we will need to build our new AVSS protocols. Some of these subprotocols are standard, some are slight variations of standard protocols, and some are new. In particular, in Section 3.3, we define a new type of protocol, which we call a *secure message distribution* protocol. In this section, we just state the properties such a protocol should satisfy, and then in Section 4 we show how to build one. In Section 5, we present and analyze our new AVSS protocol. In Section 6, we extend our AVSS protocol to ensure that the secrets shared by a corrupt dealer lie in a restricted domain. The AVSS protocols in Sections 5 and Section 6 rely in a random beacon. In Section 7, we show how to modify both of these protocols so that they do not rely on a random beacon, but instead rely on modeling a hash function as a random oracle. The resulting protocols also have the advantage of requiring fewer rounds of communication (and we speculate that they are resistant to adaptive corruptions, rather than just static corruptions).

2 Polynomial interpolation, Reed-Solomon codes, and secret sharing

We recall some basic facts about polynomial interpolation, Reed-Solomon codes, and secret sharing. As we want to work over both finite fields and Galois rings, we state these facts more generally, working over an arbitrary, finite, commutative ring with identity. For more details see [ACD+19], [Feh98] or [QBC13].

If $\mathbb{A}$ is a commutative ring with identity, we let $\mathbb{A}^*$ denote its group of units. Let $\mathbb{A}[x]$ denote the ring of univariate polynomials over $\mathbb{A}$ in the variable x. For positive integer d, let $\mathbb{A}[x]_{<d}$ denote the $\mathbb{A}$-subalgebra of $\mathbb{A}[x]$ consisting of all polynomials of degree less than d.

2.1 Polynomial interpolation

The key to making polynomial interpolation work over an arbitrary ring $\mathbb{A}$ is to restrict the choice of points at which we evaluate polynomials over $\mathbb{A}$. To this end, we work with the notion of an **exceptional sequence**, which is a sequence $(s_1, \ldots, s_n)$ such that each $s_i \in \mathbb{A}$, and for all $i, j \in [n]$ with $i \neq j$, we have $s_i - s_j \in \mathbb{A}^*$.[5] When ordering does not matter, we use the (somewhat nonstandard but more natural) term **exceptional set** to denote a set $\mathcal{E} \subseteq \mathbb{A}$ such that $s - t \in \mathbb{A}^*$ for all $s, t \in \mathcal{E}$ with $s \neq t$. Clearly, if $\mathcal{E}$ is an exceptional set, then so is any subset of $\mathcal{E}$. The size of the largest exceptional set in a ring $\mathbb{A}$ is called the *Lenstra constant* of the ring.

For example, if $\mathbb{A}$ is a field, then $\mathbb{A}$ is itself an exceptional set. As another example, suppose $\mathbb{A}$ is a Galois ring $\mathbb{Z}[y]/(p^k, F(y))$, where $F(y)$ is a monic polynomial of degree δ whose image in $\mathbb{Z}/(p)[y]$ is irreducible. Then $\mathbb{A}$ contains an exceptional set of size p^δ. Such a set $\mathcal{E}$ may be formed by taking any set of polynomials in $\mathbb{Z}[y]$ whose images in $\mathbb{Z}[y]/(p, F(y))$ are distinct, and setting $\mathcal{E}$ to be the images of these polynomials in $\mathbb{Z}[y]/(p^k, F(y))$.

So now consider an exceptional sequence of *evaluation coordinates* $\boldsymbol{e} = (e_1, \ldots, e_n) \in \mathbb{A}^n$. Because $\boldsymbol{e}$ is an exceptional sequence, polynomial interpolation with respect to these evaluation coordinates works just as expected. That is, for every $\boldsymbol{a} = (a_1, \ldots, a_n) \in \mathbb{A}^n$, there exists a unique polynomial $f \in \mathbb{A}[x]_{<n}$ such that $f(e_j) = a_j$ for all $j \in [n]$. Indeed, the coefficient vector of f is given by $\boldsymbol{a} \cdot V^{-1}$, where $V \in \mathbb{A}^{n \times n}$ is the Vandermonde matrix determined by the vector of evaluation coordinates $\boldsymbol{e}$. Because $\boldsymbol{e}$ is an exception sequence, the determinant of V is a unit, and hence V is invertible.

2.2 Reed-Solomon codes

Let $\mathbb{A}$ be a ring and $\boldsymbol{e} \in \mathbb{A}^n$ be a exceptional sequence. For a positive integer d, we define the (n, d)-**Reed-Solomon code** over $\mathbb{A}$ (with respect to $\boldsymbol{e}$) to be the $\mathbb{A}$-subalgebra of $\mathbb{A}^n$ consisting of the vectors

$$\{(f(e_1), \ldots, f(e_n)) : f \in \mathbb{A}[x]_{<d}\}.$$

Elements of this subalgebra are called *codewords*. Let $C \in \mathbb{A}^{n \times (n-d)}$ be the matrix consisting of the rightmost $n - t$ columns of V^{-1}. Then for each $\boldsymbol{a} \in \mathbb{A}^n$, we see that $\boldsymbol{a}$ is a codeword if and only if $\boldsymbol{a} \cdot C = 0$ (this just expresses the condition that the unique polynomial obtained by interpolation has degree less than d). The matrix C is called as a *check matrix* for the code.

2.3 Asynchronous verifiable secret sharing

We now turn to secret sharing, specifically, *asynchronous verifiable secret sharing (AVSS)*. We have n parties $P_1, \ldots, P_n$, of which at most $t < n/3$ may be corrupt. We assume *static* corruptions (although we claim, without a full proof, that one of our AVSS protocols is secure against *adaptive* corruptions in the random oracle model). Let $\mathcal{H}$ denote the indices of the honest parties, and let $\mathcal{C}$ denote the indices of the corrupt parties.

We assume the parties are connected by secure point-to-point channels, which provide both privacy and authentication. As we are working exclusively in the asynchronous communication model, there is no bound on the time required to deliver messages between honest parties.

Let $\mathbb{A}$ be a ring and $\boldsymbol{e} \in \mathbb{A}^n$ be a exceptional sequence. An (n, d, L)-**AVSS protocol** over $\mathbb{A}$ (with respect to $\boldsymbol{e}$) should allow a dealer $D \in \{P_1, \ldots, P_n\}$ to input polynomials $f_1, \ldots, f_L \in \mathbb{A}[x]_{<d}$ so

[5] $[n]$ denotes the set $\{1, \ldots, n\}$, and $[0..n]$ denotes the set $\{0, \ldots, n\}$.

that these polynomials are disseminated among $P_1, \ldots, P_n$ in such a way that each party P_j outputs the corresponding the *shares* $f_1(e_j), \ldots, f_L(e_j)$. Such a protocol should satisfy the following security properties (informally stated):

Correctness: If any honest parties produce an output, then there must exist polynomials $f_1, \ldots, f_L \in \mathbb{A}[x]_{<d}$ such that each honest P_j outputs $\{f_\ell(e_j)\}_{\ell=1}^{L}$, if it outputs anything at all. Moreover, if the dealer D is honest, these must be the same polynomials input by D.

Privacy: If D is honest, the protocol should reveal no more to the adversary than the values

$$\left\{ f_\ell(e_j) \right\}_{\substack{\ell \in [L] \\ j \in \mathcal{C}}},$$

that is, the shares of the corrupt parties.

Note that in the correctness condition, it is essential that the protocol constrain a corrupt dealer D so ensure that the polynomials $f_1, \ldots, f_L$ have degree less than d.

These security properties can be better captured by working in the *universal composability (UC)* framework [Can00] and defining an *ideal functionality* $\mathcal{F}_{\text{avss}}$, see Fig. 1.

$\mathcal{F}_{\text{avss}}$ captures the *correctness* property by the fact that a corrupt dealer D is constrained in the ideal world to input polynomials of the right degree. In a more detailed definition of $\mathcal{F}_{\text{avss}}$, one might have the dealer D input a bit string to $\mathcal{F}_{\text{avss}}$, and then $\mathcal{F}_{\text{avss}}$ would parse this bit string (according to some standard convention) as a list of L polynomials over $\mathbb{A}$ of degree less than d. If this failed for any reason, $\mathcal{F}_{\text{avss}}$ would not accept this input from D. For the sake of clarity, we omit such details, and throughout this paper assume that ideal functionalities and other protocol machines make such "syntax checks" by default.

$\mathcal{F}_{\text{avss}}$ in fact captures a stronger form of *correctness*, namely, *input extractability*. This property intuitively means that a corrupt dealer D must explicitly commit to all of it polynomials before any honest party outputs its own shares. This follows from the fact that in the UC framework, the ideal-world adversary (or simulator) must somehow extract, from the protocol messages it sees, the polynomials it needs to submit to $\mathcal{F}_{\text{avss}}$ as an input on behalf of D before it can request that outputs are sent to any honest parties. This property is essential in some applications (including a protocol we present later in Section 6).

$\mathcal{F}_{\text{avss}}$ captures the *privacy* property by the fact that when the dealer D is honest, the only information obtained by an adversary in the ideal world are the outputs sent from $\mathcal{F}_{\text{avss}}$ to the corrupt parties, and these outputs consist of just the shares of these parties, as required.

2.3.1 Completeness. A protocol that securely realizes the ideal functionality $\mathcal{F}_{\text{avss}}$ does not necessarily satisfy the *completeness* property mentioned in Section 1. Indeed, as stated, the ideal-world adversary may choose to have $\mathcal{F}_{\text{avss}}$ deliver outputs to some honest parties but not others.

Intuitively speaking, the completeness property for an AVSS protocol says that if an honest dealer D inputs a value or if any honest party output a value, then *eventually*, all honest parties output a value. Here, "eventually" means if and when all messages sent between honest parties have been delivered. Thus, completeness is not an unconditional guarantee: in the asynchronous communication setting, we formally leave the scheduling of message delivery entirely to the adversary, who may decide to deliver messages sent between honest parties in an arbitrary order, or may choose not to deliver some of them at all.

$\mathcal{F}_{\text{avss}}$

$\mathsf{Input}(f_1, \ldots, f_L)$: this operation is invoked once by the dealer D, who inputs polynomials $f_1, \ldots, f_L \in \mathbb{A}[x]_{<d}$ to $\mathcal{F}_{\text{avss}}$. In response, $\mathcal{F}_{\text{avss}}$ sends the message $\mathsf{NotifyInput}()$ to the ideal-world adversary.

$\mathsf{RequestOutput}(j)$: after the input has been received, this operation may be invoked by the ideal-world adversary, who specifies $j \in [n]$. In response, $\mathcal{F}_{\text{avss}}$ sends to P_j the message

$$\mathsf{Output}\Big(\{ f_\ell(e_j) \}_{\ell \in [L]} \Big).$$

Fig. 1. The AVSS Ideal Functionality (parameterized by n, d, L, $\mathbb{A}$, $\boldsymbol{e}$, and D)

Turning the above intuitive definition of completeness into a formal one is fairly straightforward. One simply defines an attack game in which the adversary (who controls the corrupt parties and the scheduling of message delivery) wins the game if he can drive the protocol to a state which violates the stated completeness condition, that is, a state such that:

(i) an honest dealer D has input a value or some honest party has output a value,
(ii) all messages sent between honest parties have been delivered, and
(iii) some honest party has not output a value.

Completeness means that any efficient adversary wins this game with negligible probability.

This simple notion of completeness will be sufficient for our purposes. Note that one technical limitation of this definition is that it is only meaningful if the **message complexity** (that is, the total number of messages sent by any honest party to any other party) is **uniformly bounded** (that is, bounded by a polynomial that is independent of the adversary, at least with overwhelming probability). Indeed, the completeness property is vacuously satisfied by any protocol in which there are always more messages that need to be delivered (so condition (ii) in the previous paragraph can never be attained). Fortunately, all of the protocols we shall consider here have a uniformly bounded message complexity.

Our approach to modeling completeness closely adheres to the approach introduced in [CKPS01]. We note that the related work [CP23] studies AVSS and AMPC protocols in the UC framework, but makes use of a formal notion of time introduced in [KMTZ13] to model completeness. Our view is that this extra (and somewhat complicated) machinery is unnecessary and, moreover (echoing the view of [HS15]), that notions such as *liveness* and related notions such as *completeness* are more simply and quite adequately modeled as properties of concrete protocols (as we have done so here) rather than as security properties captured by ideal functionalities.

2.4 Higher-level secret sharing interfaces

Our ideal functionality $\mathcal{F}_{\text{avss}}$ essentially matches that in [CP23], and models a rather minimalistic, low-level interface. As given, the dealer inputs polynomials over $\mathbb{A}$ and parties receives shares. However, there are no interfaces for encoding a secret value as a polynomial, or for performing various operations on shares, such as opening shares, combining shares to reconstruct a secret, or performing linear operations on sharings.

Our choice of this minimalistic interface is intentional, as it is simple and sufficient for our immediate needs. However, higher-level interfaces can easily be implemented on top of it using standard techniques. For example, the standard way to encode a secret $s \in \mathbb{A}$ as a polynomial is

to make s the constant term of the polynomial and choose the other coefficients at random. Doing this, the secret is essentially encoded as the value of the polynomial at the evaluation coordinate 0. For this to work, we require that $(0, e_1, \ldots, e_n)$ is an exceptional sequence. If this requirement is satisfied, and if $d > t$, then we know that the shares leaked to the adversary reveal no information about the secret s. Moreover, if $n \geq d + 2t$, we know we can reconstruct the polynomial, and hence the secret, using a protocol based on "online error correction" (originating in [BCG93], but see [CP17] for a nice exposition of this and many other related protocols in the asynchronous setting). However, this is not the only mechanism that may be used to encode a secret. For example, one may in fact encode the secret as the leading coefficient, rather than the constant term — while this alleviates the requirement of extending the vector of evaluation coordinates to $n + 1$ elements, it may not be convenient in some applications. As another example, with "packed" secret sharing, several secrets may be encoded in a polynomial, by encoding these secrets at different evaluation coordinates [FY92] — while this can improve the performance of some higher-level protocols, it also reduces the resiliency of such protocols.

Also observe that our minimalistic interface also requires that the ring $\mathbb{A}$ already has appropriate evaluation coordinates. In some applications, the secret may lie in some ring $\mathbb{S}$ that does not contain a large enough exceptional sequence. For example, $\mathbb{S}$ may be a finite field $\mathbb{F}_q$ where q is very small, or a ring such as $\mathbb{Z}/(p^k)$, where p is very small. In this case, the standard technique is to secret share over a larger ring $\mathbb{A} \supseteq \mathbb{S}$ — for example, a field extension in the case $\mathbb{S} = \mathbb{F}_q$ or a Galois ring extension in the case $\mathbb{S} = \mathbb{Z}/(p^k)$. Note that a direct application of this technique allows a corrupt dealer to share a secret that lies in $\mathbb{A} \setminus \mathbb{S}$. In some applications, this may be acceptable, while in others, it may not. In Section 6, we show how our basic AVSS protocol for secret sharing over $\mathbb{A}$ can be extended to enforce the requirement that secrets do in fact lie in the subring $\mathbb{S}$.

2.5 The number of roots of a polynomial

The following result is standard. Since it is typically proved with respect to fields, for completeness, we give a proof here with respect to rings.

Lemma 2.1 (Schwartz-Zippel over rings). *Let $\mathbb{A}$ denote a commutative ring with identity and let $P \in \mathbb{A}[x_1, x_2, \ldots, x_n]$ be a non-zero polynomial of total degree $\mathfrak{d} \geq 0$. Let $\mathcal{E} \subseteq \mathbb{A}$ be an exceptional set, and let $r_1, \ldots, r_n$ be selected uniformly, and independently, from $\mathcal{E}$. Then*

$$\Pr[\ P(r_1, \ldots, r_n) = 0\ ] \leq \frac{\mathfrak{d}}{|\mathcal{E}|}.$$

Proof. We first consider the case of univariate polynomials. Let $f \in \mathbb{A}[x]$ be of degree $\mathfrak{d}$. We show that it can only have at most $\mathfrak{d}$ roots in $\mathcal{E}$. This is done by induction, with the base case of $\mathfrak{d} = 0$ being trivial. Now suppose $f(x)$ is of degree $\mathfrak{d} + 1$, and the result is true for polynomials of degree $\mathfrak{d}$. We work by contradiction and assume that $f(x)$ has $\mathfrak{d} + 2$ distinct roots in $\mathcal{E}$, which we label $r_1, \ldots, r_{\mathfrak{d}+2}$. We can write $f(x) = (x - r_{\mathfrak{d}+2}) \cdot g(x)$ for some polynomial $g(x)$ of degree $\mathfrak{d}$. Since the roots come from an exceptional set we know that $r_i - r_{\mathfrak{d}+2}$ is invertible for every $i = 1, \ldots, \mathfrak{d} + 1$. Hence $r_1, \ldots, r_{\mathfrak{d}+1}$ must be roots of $g(x)$, and so $g(x)$ has $\mathfrak{d} + 1$ distinct roots. This contradicts the inductive hypothesis.

We now prove the main result for multivariate polynomials by induction on n, where the base case of $n = 1$ is the univariate case we just considered. So we assume the statement holds for $n \geq 1$,

and consider the case of multivariate polynomial with $n+1$ variables $f(x_1, \ldots, x_{n+1})$. We can write

$$f(x_1, \ldots, x_n) = \sum_{i \leq \mathfrak{d}} x_{n+1}^i \cdot f_i(x_1, \ldots, x_n)$$

where f_i is a multivariate polynomial in n variables. Since $f(x_1, \ldots, x_{n+1})$ is not identically zero there is at least one $f_i(x_1, \ldots, x_n)$ which is not identically zero. Let $\mathfrak{d}'$ denote the largest such index i. We have $\deg(f_{\mathfrak{d}'}) \leq \mathfrak{d} - \mathfrak{d}'$ since f has degree at most $\mathfrak{d}$.

We know, by the inductive hypothesis, that

$$\Pr[\ f_{\mathfrak{d}'}(r_1, \ldots, r_n) = 0\ ] \leq \frac{\mathfrak{d} - \mathfrak{d}'}{|\mathcal{E}|},$$

for randomly chosen $r_1, \ldots, r_n \in \mathcal{E}$.

Now if $f_{\mathfrak{d}'}(r_1, \ldots, r_n) \neq 0$ then $f(r_1, \ldots, r_n, x_{n+1})$ is a non-zero univariate polynomial of degree $\mathfrak{d}'$. So by the base case we have, for randomly chosen $r_{n+1} \in \mathcal{E}$,

$$\Pr[\ f(r_1, \ldots, r_n, r_{n+1}) = 0 \mid f_{\mathfrak{d}'}(r_1, \ldots, r_n) \neq 0\ ] \leq \frac{\mathfrak{d}'}{|\mathcal{E}|}.$$

By Bayes' Theorem, and some manipulation, we therefore have

$$\Pr[\ f(r_1, \ldots, r_n, r_{n+1}) = 0\ ] \leq \frac{\mathfrak{d} - \mathfrak{d}'}{|\mathcal{E}|} + \frac{\mathfrak{d}'}{|\mathcal{E}|} = \frac{\mathfrak{d}}{|\mathcal{E}|}.$$

□

3 Subprotocols

In this section, we review the subprotocols that our new AVSS protocol will need. Here and throughout the rest of this paper, we assume a network of n parties $P_1, \ldots, P_n$, of which at most $t < n/3$ of them may be statically corrupted, and which are connected by secure point-to-point channels (providing both privacy and authentication). We also assume network communication is asynchronous.

3.1 Random Beacon

A **random bacon** is a protocol in which each party initiates the protocol and outputs a common value ω that is effectively chosen at random from an **output space** Ω. Such a protocol should satisfy the following security properties (informally stated):

Correctness: All honest parties that output a value output the same value ω.

Privacy: The adversary learns nothing about ω until at least one honest party initiates the protocol.

These security properties are best defined in terms of the ideal functionality $\mathcal{F}_{\text{Beacon}}$, which is given in Fig. 2. Note that in $\mathcal{F}_{\text{Beacon}}$, a party P_j initiates the protocol by explicitly supplying the input init.

We also want such a protocol to satisfy the following *completeness* property: if all honest parties initiate the protocol, then *eventually*, all honest parties output a value. Just as in Section 2.3.1, "eventually" means if and when all messages sent between honest parties have been delivered.

$\mathcal{F}_{\text{Beacon}}$

Input(init): This operation may be invoked once by each party P_j. If this is the first time this is invoked by any honest party, $\mathcal{F}_{\text{Beacon}}$ chooses $\omega \in \Omega$ at random and sends NotifyInput(j, ω) to the ideal-world adversary; otherwise, $\mathcal{F}_{\text{Beacon}}$ sends NotifyInput(j) to the ideal-world adversary.

RequestOutput(j): after the value ω has been generated in response to an Input operation, this operation may be invoked by the ideal-world adversary, who specifies $j \in [n]$. In response, $\mathcal{F}_{\text{avss}}$ sends to P_j the message Output(ω).

Fig. 2. The Random Beacon Functionality $\mathcal{F}_{\text{Beacon}}$ (parameterized by output space Ω)

While our main AVSS protocol relies on a random beacon, we will also give a simpler AVSS protocol (in Section 7) that does not need a random beacon at all, but instead is analyzed in the random oracle model. But even in our main AVSS protocol, we only need to run one instance of a random beacon protocol to distribute shares of a large batch polynomials. As such, a random beacon can, in principle, be securely realized with any (not very efficient) protocol without impacting the overall amortized cost of the AVSS protocol (at least for a sufficiently large batch size).

3.1.1 Implementing a random beacon. One could implement a random beacon using a $(t+1)$-out-of-n threshold BLS signature scheme [BLS01,Bol03,SJK+17]. Despite being based on "heavyweight" cryptography, this beacon may be efficient enough for use in our AVSS protocol as well as other applications. However, this beacon requires a hardness assumption that is not post-quantum secure, as well additional set-up assumptions (including some kind of distributed key distribution step).

Very recently, [BBB+23] showed how to implement a random beacon using just "lightweight" cryptography and no set-up assumptions (other than secure channels). Although the communication complexity of their *HashRand* protocol is super-linear, it is likely good enough for use in our AVSS protocol as well as other applications.

Consider a long-running system in which we will need an unlimited supply of random beacons. Suppose we prepare a sufficiently large initial batch I of beacons, using a protocol such as *HashRand*. Then we can in fact prepare an unlimited supply of beacons, with a linear amortized communication cost per beacon, as follows. We use the standard technique of having each party generate a batch of sharings of a random secret, agreeing on a set of such batches using a consensus protocol, and then adding up the batches in the set to obtain a batch of sharings of random secrets that are unknown to any party. In fact, we can obtain a linear number of such batches in one go by using well-known "batch randomness extraction" techniques (see [HN06]). The result of this step is one very large batch J of sharings of secrets that are unknown to any party. Each sharing in the batch J can now be used as a random beacon, by just opening the sharing when it is time to reveal the beacon. The construction of J requires an AVSS protocol and a consensus protocol. We could use our new AVSS protocol, which may or may not require a beacon (depending on the version used), and an efficient consensus protocol such as *FIN* [DWZ23], which definitely requires a beacon. With these protocols, so long as I is sufficiently large to run them, we can make J arbitrarily large in relation to I. So we can arrange that $|J| \gg |I|$, and then partition J into two batches I' and J', where $|I'| = |I|$. Then we can use the batch J' for applications and the batch I' to repeat the process again. The amortized communication complexity per sharing of our AVSS protocol is linear.

Although we have to run it a linear number of times per beacon, by using the batch randomness extraction technique, the amortized communication complexity per beacon is still linear. Thus, except for an initial "bootstrapping phase", we can prepare an unlimited supply of beacons with a linear amortized communication cost per beacon.

Note that while the technique in the previous paragraph allows to *prepare* batches of random beacons with a linear amortized communication cost per beacon, the communication cost to *reveal* one beacon is quadratic. There are batching techniques that allow one to reveal many such beacons at once at lower cost (see, for example, Section III of [CP17]), but it is not clear if this type of batching has many useful applications.

3.1.2 Extending the output space of a random beacon. Suppose we have a protocol Π that securely realizes a random beacon with output space Ω. We can use Π to securely realize a random beacon with a larger output space. One way to do this is to run N instances of Π concurrently and concatenate the outputs. This immediately yields a protocol that securely realizes a random beacon with output space Ω^N.

Of course, the approach in the previous paragraph comes at a significant cost. A more practical approach is to use a cryptographic hash function $G : \Omega \to \Omega'$. If Ω is sufficiently large (so that $1/|\Omega|$ is negligible), and if we model G as a random oracle, then the protocol Π' that runs Π to obtain the output $\omega \in \Omega$ and then outputs $\omega' := G(\omega) \in \Omega'$ securely realizes a random beacon with output space Ω'. Indeed, in the random oracle model, a simulator that is given an output $\omega' \in \Omega'$ of the ideal functionality for the Ω'-beacon can generate $\omega \in \Omega$ at random and program the random oracle representing G so that $G(\omega) = \omega'$.

The above security proof relied heavily on the ability to program the random oracle representing G. If instead of modeling G as a random oracle, we just assume that G is a secure pseudorandom generator, then the above security proof falls apart, and in fact, protocol Π' will not securely realize a random beacon. However, the output of Π' will still have properties (such as unpredictability) that may be useful in certain applications. The typical setting where this works is one where the security analysis requires a certain failure event $\mathcal{E}$ to occur with negligible probability for randomly chosen $\omega' \in \Omega'$, and the occurrence of $\mathcal{E}$ can be detected efficiently as a function of ω' and data that is available to the adversary prior to invoking protocol Π.

3.2 Reliable broadcast

A **reliable broadcast** protocol allows a sender S to broadcast a single message m to $P_1, \ldots, P_n$. Such a protocol should satisfy the following security property (informally stated):

> **Correctness:** All honest parties that output a message must output the same message. Moreover, if the sender S is honest, that message is the one input by S.

This security property is best defined in terms of the ideal functionality $\mathcal{F}_{\text{ReliableBroadcast}}$, which is given in Fig. 3.

We also want such a protocol to satisfy the following *completeness* property: if an honest sender S inputs a message or if any honest party outputs a message, then *eventually*, all honest parties output a message. As usual, "eventually" means if and when all messages sent between honest parties have been delivered.

$\mathcal{F}_{\text{ReliableBroadcast}}$

$\mathsf{Input}(m)$: this operation is invoked once by the sender S, who inputs a message m. In response, $\mathcal{F}_{\text{ReliableBroadcast}}$ sends the message $\mathsf{NotifyInput}(m)$ to the ideal-world adversary.

$\mathsf{RequestOutput}(j)$: after the input has been received, this operation may be invoked by the ideal-world adversary, who specifies $j \in [n]$. In response, $\mathcal{F}_{\text{ReliableBroadcast}}$ sends to P_j the message $\mathsf{Output}(m)$.

Fig. 3. The Reliable Broadcast Ideal Functionality (parameterized by S)

$\Pi_{\text{BrachaBroadcast}}$

```
// Sender S with input m
send (send, m) to P_1, ..., P_n

// Receiving party P_j
acquired ← false, voted ← false, done ← false

while not done do
    wait until either:
        not acquired and for some m: received (send, m) from S ⇒
            acquired ← true
            send (echo, m) to P_1, ..., P_n

        not voted and for some m: received (echo, m) from n − t distinct parties ⇒
            send (vote, m) to P_1, ..., P_n
            voted ← true

        not voted and for some m: received (vote, m) from t + 1 distinct parties ⇒
            send (vote, m) to P_1, ..., P_n
            voted ← true

        voted and for some m: received (vote, m) from n − t distinct parties ⇒
            // output stage
            output m
            done ← true
```

Fig. 4. Bracha's Protocol for Reliable Broadcast

3.2.1 Bracha broadcast. A simple reliable broadcast protocol is called **Bracha Broadcast** [Bra87], and is given in Fig. 4. We express the logic for the sender as a separate process, even though the sending party is also one of the receiving parties. First note, that the communication complexity of Bracha broadcast is clearly $O(n^2 \cdot |m|)$.

Since we will be using variants of Bracha broadcast later, we highlight some important properties of the *echo/vote* logic of this protocol. Suppose that there are $t' \leq t$ corrupt parties.

Bracha Property B0: If an honest party receives $n-t$ *votes* for the same message, then all honest parties will eventually *vote* for some message.

Suppose an honest party receives $n - t$ *votes* for the same message. Then at least $n - 2t \geq t + 1$ honest parties must have *voted* for this message. Upon receipt of these $t+1$ *votes*, each honest party will *vote* if they have not done so already.

Bracha Property B1: If an honest sender S inputs a message, then all honest parties will eventually *vote* for some message.

Eventually, all honest parties will *echo* the sender's message, unless some honest party enters the output stage without doing so. In the former case, upon receipt of these *echoes*, each honest party will *vote* if they have not already done so. In the latter case, some honest party must have received $n - t$ *votes* on the same message, and the property follows from Bracha Property B0.

Bracha Property B2: If an honest party *votes* for a message, then at least $n - t - t'$ honest parties must have *echoed* the same message.

This is because the very first time any honest party *votes* for a given message it must be be the case that this party received $n - t$ *echoes* on that message from distinct parties, of which $n - t - t'$ must be honest.

Bracha Property B3: If two honest parties *vote* for a message, they must *vote* for the same message.

This is because if two honest parties *vote* different messages, then by Bracha Property B2 we have one set of $n-t-t'$ honest parties *echoing* one message, and a second, disjoint set of $n-t-t'$ honest parties *echoing* a different message, which means $2(n-t-t') \leq n-t'$, which implies $n \leq 2t+t' \leq 3t$.

The *correctness* property of Bracha broadcast easily follows from Bracha Properties B2 and B3 (B3 implies all honest parties must output the same message, and B2 implies that if the sender is honest, this message must be the one input by the sender). The *completeness* properties easily follows from Bracha Properties B0, B1, and B3. Since, B0 and B1 imply that if an honest sender inputs a message or an honest party outputs a message, then all honest parties eventually *vote* for a message, and B3 says they all *vote* for the same message, which implies all honest parties eventually output a message.

3.2.2 Compact broadcast. The communication complexity of Bracha broadcast can be improved by the use of *erasure codes*. To this end, we need an $(n, n - 2t)$ **erasure code**, which has the following properties: a message m can be efficiently encoded as a vector of n *fragments* $(f_1, \ldots, f_n)$ in such a way that m can be efficiently reconstructed (decoded) from any subset of

$n - 2t$ fragments. An $(n, n-2t)$-Reed-Solomon code can be used for this purpose, but other constructions are possible as well. In any reasonable construction, the size of each fragment will be about $|m|/(n-2t)$.

We can use such an erasure code to build a reliable broadcast protocol with better communication complexity as follows. Given a long message m, the sender S encodes the message using an $(n, n-2t)$ erasure code, to obtain a vector of n fragments $(f_1, \ldots, f_n)$. Each fragment has size roughly $|m|/(n-2t)$ — so assuming $n > 3t$, the size of each fragment is at most roughly $3|m|/n$. The sender S then sends each fragment f_j to party P_j, who then echoes that fragment to all other parties. Each party can then collect enough fragments to reconstruct m. To deal with dishonest parties, some extra steps must be taken.

This approach was initially considered in [CT05], who give a protocol with communication complexity $O(n \cdot |m| + \lambda \cdot n^2 \cdot \log n)$. Here, λ is the output length of a collision-resistant hash function. The factor $\log n$ arises from the use of *Merkle trees*. If $|m| \gg \lambda \cdot n \cdot \log n$, this is essentially optimal — indeed, any reliable broadcast protocol must have communication complexity $\Omega(n \cdot |m|)$, since every party must receive m.

Recall that that a Merkle tree allows one party, say Charlie, to commit to a vector of values $(v_1, \ldots, v_k)$ using a collision resistant hash function by building a binary tree whose leaves are the hashes of $v_1, \ldots, v_k$, and where each internal node of the tree is the hash of its (at most two) children. The root r of the tree is the commitment. Charlie may "open" the commitment at an index $i \in [k]$ by revealing v_i along with a "validation path" π_i, which consists of the siblings of all nodes along the path in the tree from the hash of v_i to the root r. We call π_i a **validation path for** v_i **under** r **at** i. Such a validation path is checked by recomputing the nodes along the corresponding path in the tree, and testing that the recomputed root is equal to the given commitment r. The collision resistance of the hash function ensures that Charlie cannot open the commitment to two different values at a given index.

We give here the details of a reliable broadcast protocol that is based on erasure codes and Merkle trees, and which achieves the same communication complexity as that in [CT05]. This protocol is similar to that presented in [CT05], but is a bit simpler and also bears some resemblence to a related protocol in the DispersedLedger system [YPA+21]. Our reason for presenting this protocol is twofold: first, for the sake of making this paper more self contained, and second, later in this paper, we will modify this protocol to achieve other goals. We call this reliable broadcast protocol $\Pi_{\text{CompactBroadcast}}$ and it is given in Fig. 5.

The reader may observe that $\Pi_{\text{CompactBroadcast}}$ has essentially the same structure as Bracha's reliable broadcast protocol, where the message being broadcast is the root of a Merkle tree. The *correctness* property of $\Pi_{\text{CompactBroadcast}}$ follows from Bracha Properties B2 and B3, and the collision resistance of the hash function used for building the Merkle trees. The *completeness* property of $\Pi_{\text{CompactBroadcast}}$ follows from Bracha Properties B0, B1, B2 and B3. Specifically, Bracha Property B2 in this context ensures that in $\Pi_{\text{CompactBroadcast}}$, if any honest party *votes* for a root r, then at least $n - 2t$ honest parties must have *echoed* r along with a corresponding validation path and fragment, so that all honest parties will eventually be able to reconstruct a message from these $n - 2t$ fragments in the output stage.

3.2.3 Other reliable broadcast protocols. For somewhat shorter messages, a protocol such as that in [DXR21] may be used, which achieves a communication complexity of $O(n \cdot |m| + \lambda \cdot n^2)$. The protocol $\Pi_{\text{CompactBroadcast}}$ above uses only an erasure code, while the protocol in [DXR21]

$\Pi_{\text{CompactBroadcast}}$

```
// Sender S with input m
encode m as (f_1, ..., f_n) using an (n, n − 2t) erasure code
build the Merkle tree for (f_1, ..., f_n) with root r
for j ∈ [n] : send (send, r, π_j, f_j) to P_j, where π_j is the validation path for f_j under r at j

// Receiving party P_j
acquired ← false, voted ← false, done ← true

while not done do
    wait until either:
        not acquired and for some (r, π_j, f_j): received (send, r, π_j, f_j) from S ⇒
            acquired ← true
            if π_j is a correct validation path for f_j under r at j then
                send (echo, r, π_j, f_i) to P_1, ..., P_n

        not voted and for some r: received (echo, r, ·, ·) from n − t distinct parties ⇒
            // output stage
            send (vote, r) to P_1, ..., P_n
            voted ← true

        not voted and for some r: received (vote, r) from t + 1 distinct parties ⇒
            send (vote, r) to P_1, ..., P_n
            voted ← true

        voted and for some r: received (vote, r) from n − t distinct parties ⇒
            // output stage
            wait until: for some I ⊆ [n], {π_i, f_i}_{i∈I}:
                |I| = n − 2t and for all i ∈ I:
                    received (echo, r, π_i, f_i) from P_i and
                    π_i is a correct validation path for f_i under r at i
            reconstruct the message m' from the fragments {f_i}_{i∈I}
            compute the fragments (f'_1, ..., f'_n) of message m'
            build the Merkle tree for (f'_1, ..., f'_n) with root r'
            if r = r'
                then  output m'
                else  output ⊥
            done ← true
```

Fig. 5. A Reliable Broadcast Protocol Based on Erasure Codes and Merkle Trees

requires an "online" error correcting code, which may be more computationally expensive than erasure codes. Another potential advantage of $\Pi_{\text{CompactBroadcast}}$ over the protocol in [DXR21] is that the former has a very balanced communication pattern, which can be important to prevent a communication bandwidth bottlenecks. The paper [DXR22] improves on [DXR21], obtaining the same communication complexity, but with a balanced communication pattern.

3.2.4 Relation to AVID. The design of $\Pi_{\text{CompactBroadcast}}$ is based on the notion of *Asynchronous Verifiable Information Dispersal*, or *AVID*. In an AVID protocol, a sender S wants to send a message m to some or possibly all of the parties $P_1, \ldots, P_n$. There are two phases to such a protocol: the **dispersal phase**, where S disperses m (or fragments of m) among $P_1, \ldots, P_n$, and the **retrieval phase**, where individual P_j's may retrieve m. The *correctness* property for such a protocol is essentially the same as that of a reliable broadcast protocol:

> All honest parties that output a message in the retrieval phase output the same message. Moreover, if S is honest, that message is m.

The *completeness* property has two parts. First, in the dispersal phase:

> If an honest sender S inputs a message or if one honest party completes the dispersal phase, then every honest party eventually completes the dispersal phase.

Second, in the retrieval phase:

> If the dispersal phase has completed (for some honest parties), and the retrieval phase for an honest party P_j is initiated, then P_j eventually outputs a message.

Note that one can use any AVID protocol to implement reliable broadcast, by first dispersing the message and then having every party retrieve the message.

The point of using an AVID protocol is that in situations where only a small number of parties need to retrieve the message, communication complexity can be much lower than in a reliable broadcast protocol. For example, the dispersal phase of the AVID protocol in the DispersedLedger system [YPA+21] is very similar to our protocol $\Pi_{\text{CompactBroadcast}}$, except that the *echo* messages in the former do not include the validation paths and fragments — rather, this data is only disseminated to those parties that actually need to retrieve the message. The resulting AVID protocol thus has a communication complexity of $O(|m| + \lambda \cdot n^2)$ in the dispersal phase and $O(|m| + \lambda \cdot n \cdot \log n)$ per retrieval.

3.2.5 One-sided voting. A degenerate version of Bracha broadcast can be used as a simple *one-sided voting* prtocol, see Fig. 6. In this protocol, each party may initiate the protocol and may output the value done. The key security property of this protocol (informally stated) is as follows:

> **Correctness:** If any honest party outputs done, then at least $n - t - t'$ honest parties initiated the protocol, where $t' \leq t$ is the number of corrupt parties.

This security property is best defined in terms of the ideal functionality $\mathcal{F}_{\text{OneSidedVote}}$, which is given in Fig. 7. Note that in $\mathcal{F}_{\text{OneSidedVote}}$, a party P_j initiates the protocol by explicitly supplying the input init.

This protocol also satisfies the following *completeness* property: if all honest parties initiate the protocol or some honest party party outputs done, then *eventually*, all honest parties output

$\Pi_{\text{OneSidedVote}}$

```
// Party P_j
acquired ← false, voted ← false, done ← false

while not done do
    wait until either:
        not acquired and received input init ⇒
            acquired ← true
            send (echo) to P_1, ..., P_n

        not voted and received (echo) from n − t distinct parties ⇒
            send (vote) to P_1, ..., P_n
            voted ← true

        not voted and received (vote) from t + 1 distinct parties ⇒
            send (vote) to P_1, ..., P_n
            voted ← true

        voted and received (vote) from n − t distinct parties ⇒
            // output stage
            output done
            done ← true
```

Fig. 6. Degenerate Version of Bracha's Protocol for One-Sided Voting

$\mathcal{F}_{\text{OneSidedVote}}$

Input(init): This operation may be invoked once by each party P_j. In response, $\mathcal{F}_{\text{OneSidedVote}}$ sends NotifyInput(j) to the ideal-world adversary.

RequestOutput(j): after $n - t - t'$ honest parties have received input init (where $t' \leq t$ is the number of corrupt parties), this operation may be invoked by the ideal-world adversary, who specifies $j \in [n]$. In response, $\mathcal{F}_{\text{OneSidedVote}}$ sends to P_j the message Output(done).

Fig. 7. The One-Sided Voting Ideal Functionality $\mathcal{F}_{\text{OneSidedVote}}$

done. As usual, "eventually" means if and when all messages sent between honest parties have been delivered.

The *correctness* property follows from the analog of Bracha Property B2. The *completeness* property follows from the analogs of Bracha Properties B0 and B1.

3.3 Secure Message Distribution

We require a new type of protocol, which we call a **secure message distribution** protocol. Such a protocol enables a sender S to securely distribute a vector $\mathbf{m} = (m_1, \ldots, m_n)$ of messages, so that during an initial *distribution* phase, each party P_j outputs m_j. After receiving its own message m_j, party P_j may optionally *forward* this message to another party. Moreover, after receiving *any* message m_u (either its own or one forwarded to it), party P_j may optionally forward m_u to another party. This forwarding functionality will be needed to deal with the "unhappy" path of our AVSS protocol.

Such a protocol should satisfy the following security properties (informally stated):

Correctness: If any honest parties produce an output in the distribution or forwarding phases, those messages must be consistent with a message vector $\mathbf{m} = (m_1, \ldots, m_n)$ — that is, the message output by honest P_j during the distribution phase must be m_j, and any message output by some honest party during the forwarding phase as ostensibly belonging to to some party P_u must be m_u. Moreover, if the sender S is honest, $\mathbf{m}$ must be the same message vector input by S.

Privacy: If the sender S and party P_u are honest, and no honest party forwards m_u to a corrupt party, then the adversary learns nothing about m_u.

Note that whenever a party outputs a message (in either the distribution or forwarding phase), that message may be $\perp$, which can only happen if the sender is corrupt.

It will also be convenient for us to allow a party to include an identifying *tag* along with the forwarded message.

We may more precisely formulate the security properties for secure message distribution as the ideal functionality $\mathcal{F}_{\text{SecMsgDst}}$, which is given in Fig. 8. We note that $\mathcal{F}_{\text{SecMsgDst}}$ also captures an *input extractability* property that intuitively means that a corrupt sender S must explicitly commit to a vector of all input messages before any honest party outputs its own message in the distribution phase (or any forwarded message for that matter). This is a property that will be essential in the security analysis of our AVSS protocol.

We bring to the reader's attention the logic in the ideal functionality $\mathcal{F}_{\text{SecMsgDst}}$ for processing a request by a party P_j to forward a message m_u to another party. The logic enforces a precondition that requires that either (i) the message to be forwarded is one that P_j has already received (either its own or one forwarded to it), or (ii) P_j and S are both corrupt. In any implementation, an honest party will always simply ignore an input that requests it to forward a message it does not have. If we did not have such a precondition, an ideal-world adversary could trivially circumvent the privacy property by having a corrupt P_j simply ask the ideal functionality to forward a message m_u belonging to an honest party P_u to itself or to some other corrupt party.

In fact, $\mathcal{F}_{\text{SecMsgDst}}$ is a bit stronger than we need, in the sense that we may assume that when the sender S is honest, no honest P_j will forward its message m_j to any other party. As we will see, this constraint will be satisfied by our AVSS protocol. In the UC framework, this can be captured by only considering *restricted environments* that satisfy this constraint. We will give an efficient protocol that securely realizes $\mathcal{F}_{\text{SecMsgDst}}$ with respect to such constrained environments. As we will also see, in the random oracle model, essentially the same protocol is secure even without this constraint.

We also want a secure message distribution protocol to satisfy the following *completeness* property, which has two parts. First, in the distribution phase:

> If an honest sender S inputs a vector of messages, or if one honest party outputs a message in the distribution phase, then *eventually*, all honest parties output a message in the distribution phase.

Second, in the forwarding phase:

> If an honest party P_j forwards a message m_u to an honest party P_i, then *eventually*, P_i receives m_u.

$\mathcal{F}_{\text{SecMsgDst}}$

Input(distribute, $\mathbf{m}$): this operation is invoked once by the sender S, who inputs a message vector $\mathbf{m} = (m_1, \ldots, m_n)$. In response, $\mathcal{F}_{\text{SecMsgDst}}$ sends the message NotifyInput(distribute) to the ideal-world adversary.

RequestOutput(distribute, j): after the input $\mathbf{m} = (m_1, \ldots, m_n)$ has been received, this operation may be invoked by the ideal-world adversary, who specifies $j \in [n]$. In response, $\mathcal{F}_{\text{SecMsgDst}}$ sends to P_j the message Output(distribute, m_j).

Input(forward, $u, i,$ tag): party P_j may invoke this operation once per $(u, i) \in [n] \times [n]$, indicating that the message m_u should be forwarded to party P_i with tag tag, subject to the precondition that either:

(i) the message to be forwarded is one that P_j has already received (either its own or one forwarded to it), or

(ii) P_j and S are both corrupt.

In response, $\mathcal{F}_{\text{SecMsgDst}}$ sends the message NotifyInput(forward, $u, j, i,$ tag) to the ideal-world adversary.

RequestOutput(forward, $u, j, i,$ tag): after P_j has invoked Input(forward, $u, i,$ tag), and after P_i has received its own message m_i from S in the distribution phase, this operation may be invoked by the ideal-word adversary. In response, $\mathcal{F}_{\text{SecMsgDst}}$ sends the message Output(forward, $u, j,$ tag, m_u) to P_i.

Fig. 8. The Ideal Functionality for Secure Message Delivery $\mathcal{F}_{\text{SecMsgDst}}$ (parameterized by S)

As usual, "eventually" means if and when all messages sent between honest parties have been delivered.

In Section 4 we give a secure message distribution protocol that is built from "lightweight" cryptographic primitives, specifically, semantically secure symmetric key encryption and a hash function. The hash function needs to be collision resistant and to also satisfy a kind of related-key indistinguishability assumption (see Section 4.2 for more details). The communication complexity of the distribution phase of our protocol is $O(|\mathbf{m}| + \lambda \cdot n^2 \cdot \log n)$. If an honest party forwards a message m_u to another party, this adds $O(|m_u| + \lambda \cdot n \cdot \log n)$ to the communication complexity.

In our application to AVSS, we will only use the forwarding mechanism on the "unhappy path", in which a corrupt sender provably misbehaves. In particular, unless we are on the "unhappy path", the forwarding mechanism will not contribute to the communication complexity at all.

4 Building Secure Message Distribution

In this section we show how to implement the secure message distribution functionality from Section 3.3. Note that [YLF+21] and [GS22] show how to implement this type of functionality using "heavyweight" cryptographic primitives based on discrete logarithms. In particular, [GS22] rigorously defines a particular multi-encryption primitive with an appropriate notion of chosen ciphertext security and a verifiable decryption protocol, and presents practical constructions that are provably secure in the random oracle model.

While such constructions may well yield acceptable performance in practice, we show here that one can implement this functionality using only "lightweight" cryptographic primitives. The resulting protocols are certainly more efficient than those based on discrete logarithms, and also have the advantage of providing post-quantum security.

4.1 Reliable Message Distribution

We start out by considering the simpler notion of a **reliable message distribution** protocol, which satisfies all the properties of a secure message secure message distribution protocol *except privacy*.

We implement this using a variant of the reliable broadcast protocol $\Pi_{\text{CompactBroadcast}}$ in Section 3.2. In the distribution phase, the sender S starts with a vector of messages $\mathbf{m} = (m_1, \ldots, m_n)$; it encodes each m_i as a vector of fragments $(f_{i1}, \ldots, f_{in})$ and then builds a Merkle tree for $(f_{i1}, \ldots, f_{in})$ with root r_i; it then sends each P_j the collection of values $\{(r_i, \pi_{ij}, f_{ij})\}_{i=1}^{n}$, where each π_{ij} is the validation path for f_{ij} under r_i at j. Thus, each P_j receives the jth fragment of all n messages. An *echo* message from P_j to P_i now includes $r, \pi_i, r_i, \pi_{ij}, f_{ij}$, where r is the root of the Merkle tree built from $(r_1, \ldots, r_n)$ and π_i is the validation path for r_i under r at i. The *vote* messages include just the root r. Once party P_j collects sufficiently many *vote* messages for a root r, it enters the output stage, and waits to collects sufficiently many *echo* messages that contain valid fragments of the jth message from which it can reconstruct that message. Details of the complete distribution phase are in Fig. 9.

When P_j completes the distribution phase of the protocol, it can optionally forward m_j to another party Q by sending to Q the values it obtained in the last step of the distribution phase, specifically, the values π_j, r_j along with the collection of $n - 2t$ values $\{\pi_{ji}, f_{ji}\}_i$. Party Q, who we assume has also completed the distribution phase, can validate this information and compute the message using the same logic used by P_j. This validation consists of two parts:

- In the first part, Q checks that the validation paths are correct with respect to the root r acquired when entering the *output stage*; if this check fails, the forwarding subprotocol fails and no output is delivered.
- In the second part, Q reconstructs the fragments and their Merkle tree, and compares the Merkle tree roots. If this check fails, the forwarding subprotocol outputs $\perp$; otherwise, it outputs the message.

The first part detects if the party forwarding the message is misbehaving, while the second detects if the sender S was misbehaving.

The same logic above can obviously be adapted to allow a party to forward *any* message that it has received, either its own or one that was forwarded to it.

4.1.1 Correctness and completeness.

The *correctness* and *completeness* properties for protocol $\Pi_{\text{RelMsgDst}}$ follow from essentially the same argument used in Section 3.2.2 for the corresponding properties for protocol $\Pi_{\text{CompactBroadcast}}$. Protocol $\Pi_{\text{RelMsgDst}}$ also satisfies the *input extractability* property, which follows from the same argument used in Section 3.2.2 for the *completeness* property for protocol $\Pi_{\text{CompactBroadcast}}$. Specifically, Bracha Property B2 ensures that when one honest party reaches the *output stage* in $\Pi_{\text{RelMsgDst}}$, at least $n - 2t$ honest parties have echoed validation paths and fragments for *all* input messages $m_1, \ldots, m_n$. Therefore, assuming collision resistance for the hash function used for building Merkle trees, all these messages are fully determined at this time.

4.1.2 Communication complexity.

The communication complexity of the distribution phase is $O(|\mathbf{m}| + \lambda \cdot n^2 \cdot \log n)$. If an honest party forwards a message m_u to another party, this adds $O(|m_u| + \lambda \cdot n \cdot \log n)$ to the communication complexity.

$\Pi_{\text{RelMsgDst}}$

// ***Sender*** S ***with input*** $\mathbf{m} = (m_1, \ldots, m_n)$
for $i \in [n]$: encode m_i as $(f_{i1}, \ldots, f_{in})$ using an $(n, n-2t)$ erasure code
    and build the Merkle tree for $(f_{i1}, \ldots, f_{in})$ with root r_i
for $j \in [n]$: send $(\mathsf{send}, \{(r_i, \pi_{ij}, f_{ij})\}_{i=1}^n)$ to P_j, where
    π_{ij} is the validation path for f_{ij} under r_i at j for all $i \in [n]$

// ***Receiving party*** P_j
acquired $\leftarrow$ false, *voted* $\leftarrow$ false, *done* $\leftarrow$ false

while not *done* do
    wait until either:
        not *acquired* and for some $\{(r_i, \pi_{ij}, f_{ij})\}_{i=1}^n$: received $(\mathsf{send}, \{(r_i, \pi_{ij}, f_{ij})\}_{i=1}^n)$ from $S \Rightarrow$
            acquired $\leftarrow$ *true*
            if π_{ij} is a correct validation path for f_{ij} under r_i at j for all $i \in [n]$ then
                build the Merkle tree for $(r_1, \ldots, r_n)$ with root r
                for $i \in [n]$: send $(\mathsf{echo}, r, \pi_i, r_i, \pi_{ij}, f_{ij})$ to P_i,
                    where π_i is the validation path for r_i under r

        not *voted* and for some r: received $(\mathsf{echo}, r, \cdot, \cdot, \cdot, \cdot)$ from $n-t$ distinct parties $\Rightarrow$
            send (vote, r) to $P_1, \ldots, P_n$
            voted $\leftarrow$ *true*

        not *voted* and for some r: received (vote, r) from $t+1$ distinct parties $\Rightarrow$
            send (vote, r) to $P_1, \ldots, P_n$
            voted $\leftarrow$ *true*

        voted and for some r: received (vote, r) from $n-t$ distinct parties $\Rightarrow$
            // *output stage*
            wait until: for some $\pi_j, r_j,\ \mathcal{I} \subseteq [n],\ \{\pi_{ji}, f_{ji}\}_{i \in \mathcal{I}}$:
                π_j is a correct validation path for r_j under r at j, $|\mathcal{I}| = n - 2t$, and for all $i \in \mathcal{I}$:
                    received $(\mathsf{echo}, r, \pi_j, r_j, \pi_{ji}, f_{ji})$ from P_i and
                    π_{ji} is a correct validation path for f_{ji} under r_j at i
            reconstruct the message m'_j from the fragments $\{f_{ji}\}_{i \in \mathcal{I}}$
            compute the fragments $(f'_{j1}, \ldots, f'_{jn})$ of message m'_j
            build the Merkle tree for $(f'_{j1}, \ldots, f'_{jn})$ with root r'_j
            if $r_j = r'_j$
                then output m'_j
                else output $\perp$
            done $\leftarrow$ true

Fig. 9. The Distribution Phase of a Reliable Message Distribution Protocol

Note that when a corrupt party forwards a message to an honest party, this does not contribute anything to the communication complexity. This property will be important when we analyze the communication complexity of our AVSS protocol on the "happy path". The reason this type of forwarding does not contribute anything is that in defining communication complexity, we count the number of bits *sent* by all honest parties to all parties. One might argue that this is a theoretical distinction, and that in practice, one should count all bits sent *and received* by honest parties. However, such a definition of communication complexity is unworkable, as it cannot be upper bounded at all — corrupt parties may "spam" their honest peers with an unbounded amount of data. In practice, honest parties would likely attempt to distribute their download bandwidth equitably among all of its peers, and employ some kind of "spam prevention" strategy to protect itself against peers who try to monopolize its download bandwidth.

4.1.3 Relation to AVID. In Section 3.2.4 we briefly recalled the notion of an AVID protocol. In fact, the design of our protocol $\Pi_{\text{RelMsgDst}}$ is inspired by the AVID protocol in the DispersedLedger system [YPA+21]. In principle, one could build a reliable message distribution protocol simply by running n instances of an AVID protocol concurrently, one for each input message m_j: the retrieval mechanism of the AVID protocol could be used both to deliver m_j to P_j and to optionally forward m_j to other parties. So we could have simply implemented this generic strategy using the AVID protocol in [YPA+21]. There are several reasons we did not do this:

- This generic strategy, instantiated with DispersedLedger's AVID protocol, would result in a reliable message distribution protocol whose communication complexity in the distribution phase is $O(|\mathbf{m}|+\lambda\cdot n^3)$ rather than $O(|\mathbf{m}|+\lambda\cdot n^2\cdot\log n)$. Note that the same communication complexity would result if we instantiated with the AVID protocol in [DXR22].
- This generic strategy, instantiated with DispersedLedger's AVID protocol, would result in a reliable message distribution protocol where the number of rounds of communication in the distribution phase was 4 rather than 3.
- In this generic strategy, where we use the retrieval mechanism of AVID to implement the forwarding mechanism of reliable message distribution, we run into a subtle problem regarding communication complexity. As noted above, in our reliable message distribution protocol, when a corrupt party forwards its message to an honest party, this does not contribute anything to the communication complexity. However, if we use the retrieval mechanism of AVID for message forwarding, when a corrupt party attempts to forward its message to an honest party, all honest parties must participate in the protocol, which contributes to the communication complexity. So in this case, some additional mechanism would be required to prevent or at least detect misusing the forwarding mechanism.

In addition to the above, we wanted to present a concrete reliable message distribution protocol which we could then easily modify to add data privacy and so obtain a secure message distribution protocol.

4.2 Secure Key Distribution

The above reliable message distribution protocol does not provide any data privacy. This can be remedied by augmenting it with a protocol for **secure key distribution**, and then using these secret keys to encrypt the messages and using $\Pi_{\text{RelMsgDst}}$ to distribute the resulting ciphertexts.

We sketch briefly here the properties that a secure key distribution protocol should satisfy and how to build one, and then below in Section 4.3, we show in how to integrate this protocol into our protocol $\Pi_{\text{SecMsgDst}}$ to obtain a secure message distribution protocol.

In a *secure message distribution* protocol, the goal is to have the sender S distribute a vector of keys $\boldsymbol{k} = (k_1, \ldots, k_n)$, so that each party P_j obtains k_j. Here, if the sender S is honest, the keys $k_1, \ldots, k_n$ are not input by the sender, but rather are generated by the protocol itself, and the sender obtains the vector $\boldsymbol{k}$ as an output of the protocol; however, a corrupt sender S may effectively choose and input an arbitrary vector $\boldsymbol{k}$. In addition, just like for reliable message distribution, the protocol should allow a party to forward a key that it has received to another party. Such a protocol should satisfy analogous *correctness* and *completeness* (as well as *input extractability*) properties. In addition, the following property should hold:

Privacy: If the sender S and party P_u are honest, and no honest party forwards k_u to a corrupt party, then the adversary learns nothing about k_u.

To implement such a scheme, which we call $\Pi_{\text{SecKeyDst}}$, we modify the reliable message distribution protocol $\Pi_{\text{RelMsgDst}}$ so that instead of encoding a key using an erasure code, we share it using Shamir secret sharing. In more detail, let $H : [0..n] \times F \to \mathcal{K}$ be a cryptographic hash function, where F is a large finite field and $\mathcal{K}$ is the key space. Let $(\eta_0, \eta_1, \ldots, \eta_n)$ be fixed sequence of distinct elements in F.[6]

The sender proceeds as follows. For each $i \in [n]$, the sender chooses a random polynomial $\theta_i \in F_{<n-2t}$. Let $s_{ij} := \theta_i(\eta_j)$ for $j \in [0..n]$. The key k_i is defined as $k_i := H(0, s_{i0})$. The sender builds a Merkle tree from $((H(1, s_{i1}), \ldots, H(n, s_{in}))$ with root r_i. and sends to each P_j the collection of values $\{(r_i, \pi_{ij}, s_{ij})\}_{i=1}^n$, where π_i is a validation path for $H(j, s_{ij})$ under r_i at j.

Upon receiving such a message from S, and validating it is of the correct form, each party P_j then builds a Merkle tree for $(r_1, \ldots, r_n)$ with root r and *echoes* to each P_i the tuple $(r, \pi_i, r_i, \pi_{ij}, s_{ij})$, where π_i is a validation path for r_i under r. The *voting* logic works just as before. In the output stage, P_j waits for valid *echo* messages from a set of $n-2t$ distinct parties P_i, and then reconstructs a polynomial $\theta'_j \in F[x]_{<n-2t}$ via polynomial interpolation from $\{s_{ij}\}_i$. It then builds a Merkle tree from $(H(1, \theta'_j(\eta_1)), \ldots, H(n, \theta_j(\eta_n)))$ with root r'_j. If $r_j = r$, it outputs the key $H(0, \theta'_j(0))$, and otherwise outputs $\perp$.

When P_j completes the distribution phase of the protocol, it can optionally forward k_j to another party Q by sending to Q the values it obtained in the last step of the distribution phase, specifically, the values π_j, r_j along with the collection of $n - 2t$ values $\{\pi_{ji}, s_{ji}\}_i$. Party Q who can validate this information and compute k_j using the same logic used by P_j. This strategy can obviously be adapted to allow a party to forward *any* key that it has received, either its own or one that was forwarded to it.

We note that $\Pi_{\text{SecKeyDst}}$ has some rough similarities to the *asynchronous weak VSS* protocol in [DW20], but the goals and a number of details are quite different.

4.2.1 Correctness and completeness.

One can easily adapt the analysis of $\Pi_{\text{RelMsgDst}}$ to show that $\Pi_{\text{SecKeyDst}}$ satisfies the *correctness* and *completeness* (as well as *input extractability*) properties, assuming the hash function used to implement the Merkle trees, as well as H, are collision resistant.

[6] Note that the choice of the field F is an implementation detail and need not have any relationship to the ring $\mathbb{A}$ used in the context of AVSS.

4.2.2 Proving privacy under the linear hiding assumption. To prove the *privacy* property for $\Pi_{\text{SecKeyDst}}$, we make the following assumption on the hash function $H : [0..n] \times F \to \mathcal{K}$, which we call the **linear hiding assumption**. This is a kind of indistinguishability assumption under a "related key attack".

This assumption is defined by a game in which the adversary first chooses a collection of pairs $\{(a_i, b_i)\}_{i \in \mathcal{I}}$, where $\mathcal{I} \subseteq [0..n]$ and each a_i is nonzero. The task of the adversary is to distinguish the distribution

$$\big\{H(i, a_i \cdot s + b_i)\big\}_{i \in \mathcal{I}},$$

where $s \in F$ is randomly chosen, from the uniform distribution on $\mathcal{K}^{\mathcal{I}}$. The assumption states that no computationally bounded adversary can effectively distinguish these two distributions.

This assumption is certainly true in the random oracle model, assuming $1/|F|$ is negligible. Indeed, if we model H as a random oracle, the best the adversary can do is evaluate H at many points (i, s^*) for $i \in \mathcal{I}$ and $s^* \in F$, and hope that $s^* = a_i \cdot s + b_i$. So it seems a reasonable assumption.

We can use this assumption to prove the privacy of $\Pi_{\text{SecKeyDst}}$ as follows. Assume the sender S is honest and consider any one honest party. For this party, the sender chooses a random polynomial $\theta \in F[x]_{<n-2t}$ and computes $s_j := \theta(\eta_j)$ for $j \in [0..n]$. Let $\mathcal{C}$ be the set of corrupt parties, which we are assuming is of cardinality $\leq t < n - 2t$, and let $\mathcal{H} := [n] \setminus \mathcal{C}$ be the set of honest parties. During the execution of the protocol, the adversary learns s_j for $j \in \mathcal{C}$. The only other information about the polynomial θ that the adversary learns is derived as a function of $H(i, s_i)$ for $i \in \mathcal{H}$. We want to argue that given this information, the adversary cannot distinguish the actual key $H(0, s_0)$ from a random key, under the linear hiding assumption.

Without loss of generality, we may give the adversary even more information, namely let $\mathcal{C}' \subseteq [n]$ be an arbitrary set of size exactly $n - 2t - 1$ containing $\mathcal{C}$, and let us assume that the adversary is given s_j for $j \in \mathcal{C}'$ and $H(i, s_i)$ for $i \in \mathcal{H}' := [n] \setminus \mathcal{C}'$. By Lagrange interpolation, for each $i \in \mathcal{H}'$, there exist nonzero constants $\{\lambda_{ij}\}_{j \in \mathcal{C}' \cup \{0\}}$ in the field F such that

$$s_i = \sum_{j \in \mathcal{C}' \cup \{0\}} \lambda_{ij} \cdot s_j.$$

The indistinguishability of $H(0, s_0)$ from random follows directly from the linear hiding assumption, where in the attack game for that assumption, we use the adversarially chosen pairs

$$\{(a_i, b_i)\}_{i \in \mathcal{H}' \cup \{0\}},$$

where

$$(a_i, b_i) = \begin{cases} (\,\lambda_{i0},\ \textstyle\sum_{j \in \mathcal{C}'} \lambda_{ij} \cdot s_j\,), & \text{if } i \in \mathcal{H}'; \\ (1, 0), & \text{if } i = 0. \end{cases}$$

That proves the privacy property of a single honest party's key. The proof can easily be extended to cover all honest parties' keys by a standard "hybrid" argument.

We note that the indistinguishability property for keys, and the fact that the key space itself must be large (as we are assuming the key space is the output space of a collision resistant hash), implies that keys are *unpredictable*.

4.2.3 Domain separation strategies for H. Our construction uses the simple "domain separation" strategy for H, where we include both j and s_{ij} in the input to H. The inclusion of j is not strictly necessary, but it yields a simpler and quantitatively better security analysis in the random oracle model. In fact, if we include i as well in the input, we would obtain an even better concrete security bound (avoiding the "hybrid" argument mentioned above). Moreover, as a practical matter, one should include even more contextual information as an input to H that identifies the individual instance of the protocol, including the identity of the sender. This is not only good security practice, but will also yield better concrete security bounds for a system in which many instances of the protocol are run.

4.2.4 Communication complexity. Assuming individual keys are of size $O(\lambda)$, the communication complexity of the distribution phase is $O(\lambda \cdot n^2 \cdot \log n)$. If an honest party forwards a key to another party, this adds $O(\lambda \cdot n \cdot \log n)$ to the communication complexity. Just as in Section 4.1.2, hen a corrupt party forwards its key to an honest party, this does not contribute anything to the communication complexity.

4.3 A Secure Message Distribution Protocol

We can build a protocol that securely realizes $\mathcal{F}_{\text{SecMsgDst}}$ by using $\Pi_{\text{SecKeyDst}}$ to securely distribute secret keys that are used to encrypt messages using any semantically secure symmetric-key encryption scheme, and then distributing the resulting ciphertexts using $\Pi_{\text{RelMsgDst}}$. In fact, we can integrate the logic of $\Pi_{\text{SecKeyDst}}$ directly into $\Pi_{\text{RelMsgDst}}$, so that there is just a single root r that controls both keys and ciphertexts. The distribution phase of the resulting protocol $\Pi_{\text{SecMsgDst}}$ is given in Fig. 10. Here, $c \leftarrow \mathsf{Encrypt}(k, m)$ encrypts the message m under the key k, producing the ciphertext c, and $m \leftarrow \mathsf{Decrypt}(k, c)$ performs the corresponding decryption.

When P_j completes the distribution phase of the protocol, it can optionally forward m_j to another party Q by sending to Q the values it obtained in the last step of the distribution phase, specifically, the values π_j, r_j along with the collection of $n - 2t$ values $\{\pi_{ji}, s_{ji}, f_{ji}\}_i$. Party Q who can validate this information and compute m_j using the same logic used by P_j. This strategy can obviously be adapted to allow a party to forward *any* message that it has received, either its own or one that was forwarded to it.

The same comments on domain separation for H in Section 4.2.3 apply here as well.

4.3.1 Security and completeness. If $\Pi_{\text{SecMsgDst}}$ securely realizes the $\mathcal{F}_{\text{SecMsgDst}}$ functionality, then it will surely satisfy the corresponding *correctness* and *privacy* (as well as *input extractability*) properties. In the security analysis of our AVSS protocol, we will want to work directly with the $\mathcal{F}_{\text{SecMsgDst}}$ functionality, rather than with these specific security properties (which are anyway too informal).

Recall from Section 3.3 the notion of a *restricted environment*, which is an environment in the UC framework that never instructs an honest party to forward its own message when S is honest. Such restricted environments are actually sufficient for our AVSS application.

It is straightforward to show that if

- H and the hash function used for building Merkle trees are collision resistant,
- H satisfies the linear hiding assumption, and
- the encryption scheme used to encrypt messages is semantically secure

$\Pi_{\text{SecMsgDst}}$

*// **Sender** S **with input** $\mathbf{m} = (m_1, \ldots, m_n)$*
for $i \in [n]$: choose a random polynomial $\theta_i \in F_{<n-2t}$, and compute
  $s_{ij} \leftarrow \theta_i(\eta_j)$, $k_{ij} \leftarrow H(j, s_{ij})$ for $j \in [0..n]$ and $c_i \leftarrow \mathsf{Encrypt}(k_{i0}, m_i)$
for $i \in [n]$: encode c_i as $(f_{i1}, \ldots, f_{in})$ using an $(n, n-2t)$ erasure code
  and build the Merkle tree for $((k_{i1}, f_{i1}), \ldots, (k_{in}, f_{in}))$ with root r_i
for $j \in [n]$: send $(\mathsf{send}, \{(r_i, \pi_{ij}, s_{ij}, f_{ij})\}_{i=1}^n)$ to P_j, where
  π_{ij} is the validation path for (k_{ij}, f_{ij}) under r_i at j for all $i \in [n]$

*// **Receiving party** P_j*
acquired $\leftarrow$ false, *voted* $\leftarrow$ false, *done* $\leftarrow$ false

while not *done* do
  wait until either:
    not *acquired* and for some $\{(r_i, \pi_{ij}, s_{ij}, f_{ij})\}_{i=1}^n$: received $(\mathsf{send}, \{(r_i, \pi_{ij}, s_{ij}, f_{ij})\}_{i=1}^n)$ from $S \Rightarrow$
      acquired $\leftarrow$ *true*
      if π_{ij} is a correct validation path for $(H(j, s_{ij}), f_{ij})$ under r_i at j for all $i \in [n]$ then
        build the Merkle tree for $(r_1, \ldots, r_n)$ with root r
        for $i \in [n]$: send $(\mathsf{echo}, r, \pi_i, r_i, \pi_{ij}, s_{ij}, f_{ij})$ to P_i,
          where π_i is the validation path for r_i under r

    not *voted* and for some r: received $(\mathsf{echo}, r, \cdot, \cdot, \cdot, \cdot, \cdot)$ from $n - t$ distinct parties $\Rightarrow$
      send (vote, r) to $P_1, \ldots, P_n$
      voted $\leftarrow$ *true*

    not *voted* and for some r: received (vote, r) from $t + 1$ distinct parties $\Rightarrow$
      send (vote, r) to $P_1, \ldots, P_n$
      voted $\leftarrow$ *true*

    voted and for some r: received (vote, r) from $n - t$ distinct parties $\Rightarrow$
      // output stage
      wait until: for some $\pi_j, r_j,\ \mathcal{I} \subseteq [n],\ \{\pi_{ji}, s_{ji}, f_{ji}\}_{i \in \mathcal{I}}$:
        π_j is a correct validation path for r_j under r at j, $|\mathcal{I}| = n - 2t$, and for all $i \in \mathcal{I}$:
          received $(\mathsf{echo}, r, \pi_j, r_j, \pi_{ji}, s_{ji}, f_{ji})$ from P_i and
          π_{ji} is a correct validation path for $(H(i, s_{ji}), f_{ji})$ under r_j at i
      reconstruct the ciphertext c'_j from the fragments $\{f_{ji}\}_{i \in \mathcal{I}}$
      compute the fragments $(f'_{j1}, \ldots, f'_{jn})$ of ciphertext c'_j
      construct the polynomial $\theta'_j \in F[x]_{<n-2t}$ by interpolation through the points $\{(\eta_i, s_{ji})\}_{i \in \mathcal{I}}$
      compute the values $s'_{ji} \leftarrow \theta'_j(\eta_i)$, $k'_{ji} \leftarrow H(i, s'_{ji})$ for $i \in [0..n]$
      build the Merkle tree for $((k'_{j1}, f'_{j1}), \ldots, (k'_{jn}, f'_{jn}))$ with root r'_j
      if $r_j = r'_j$
        then output $\mathsf{Decrypt}(k'_{j0}, c'_j)$
        else output $\perp$
      done $\leftarrow$ true

Fig. 10. The Distribution Phase of a Secure Message Distribution Protocol

then protocol $\Pi_{\text{SecMsgDst}}$ securely realizes $\mathcal{F}_{\text{SecMsgDst}}$ with respect to restricted environments.

It is also straightforward to show that if H and the hash function used for building Merkle trees are collision resistant, then $\Pi_{\text{SecMsgDst}}$ satisfies the completeness property for secure message distribution.

One can in fact prove that $\Pi_{\text{SecMsgDst}}$ securely realizes $\mathcal{F}_{\text{SecMsgDst}}$ without restriction under the same assumptions on the hash functions, but where the encryption scheme is built using a random oracle to make it "equivocable" — namely, where we define $\mathsf{Encrypt}(k, m) := \mathit{hash}(k) \oplus m$, and model *hash* as a random oracle. We also speculate that under these assumptions, $\Pi_{\text{SecMsgDst}}$ securely realizes $\mathcal{F}_{\text{SecMsgDst}}$ under adaptive corruptions (although we have not worked out the details of this).

4.3.2 Communication complexity. The communication complexity of the distribution phase is $O(|\mathbf{m}| + \lambda \cdot n^2 \cdot \log n)$. If an honest party forwards a message m_u to another party, this adds $O(|m_u| + \lambda \cdot n \cdot \log n)$ to the communication complexity. Just as in Section 4.1.2, when a corrupt party forwards a message to an honest party, this does not contribute anything to the communication complexity.

5 Our AVSS protocol

We now present and analyze our new AVSS protocol. This is a *generic protocol* which works over an arbitrary ring. We will show how to instantiate it over finite fields and Galois rings. As will see, different instantiations lead to different failure bounds in the analysis.

5.1 Description of the protocol

Notation is as in Section 2; specifically, we have a finite commutative ring $\mathbb{A}$ and a vector of evaluation coordinates $\boldsymbol{e} = (e_1, \ldots, e_n) \in \mathbb{A}^n$ that forms an exceptional sequence, that is, we have $e_i - e_j \in \mathbb{A}^*$ for all $i \neq j$.

As usual, we assume a network of n parties $P_1, \ldots, P_n$, of which at most $t < n/3$ of them may be statically corrupted, and which are connected by secure point-to-point channels (providing both privacy and authentication). We also assume network communication is asynchronous.

Our AVSS protocol Π_{avss1} is presented in Fig. 11. It is an (n, d, L)-AVSS protocol over $\mathbb{A}$ (with respect to $\boldsymbol{e}$). The protocol requires $t < d \leq n - 2t$. In addition, it makes use of a variation of the probabilistic degree check from [DN07] (a similar probabilistic check was used in a different context in [BGR98]). This probabilistic check involves several parameters:

- A ring extension $\mathbb{B}$ of $\mathbb{A}$.
- A *repetition parameter* R, which is a positive integer.
- A *challenge space* Θ, which is a subset of $\mathbb{B}^L$.

  For example, for a given subset of $\mathcal{E}$, we might use the challenge space

$$\Theta^{\mathcal{E}}_{\text{pow}} := \{(\theta, \theta^2, \ldots, \theta^L) : \theta \in \mathcal{E}\}$$

  or the challenge space

$$\Theta^{\mathcal{E}}_{\text{lin}} := \mathcal{E}^L.$$

In choosing the above parameters, there are various trade-offs between security, efficiency, and underlying cryptographic assumptions. These will be discussed below.

Our AVSS protocol makes use of several subprotocols. We describe our AVSS protocol as a hybrid protocol that makes use of the following ideal functionalities:

- $\mathcal{F}_{\text{Beacon}}$ for a random beacon, as described in Section 3.1, whose output space is defined to be

$$\Omega := \left\{ \ \{\theta_\ell^{(r)}\}_{r\in[R],\ell\in[L]} \ : \ (\theta_1^{(r)},\ldots,\theta_L^{(r)}) \in \Theta \text{ for } r = 1,\ldots,R \ \right\},$$

  where R and Θ are the parameters discussed above;
- $\mathcal{F}_{\text{ReliableBroadcast}}$ for reliable broadcast, as described in Section 3.2;
- $\mathcal{F}_{\text{OneSidedVote}}$ for one-sided voting, as described in Section 3.2.5;
- $\mathcal{F}_{\text{SecMsgDst}}$ for secure message distribution, as described in Section 3.3.

5.1.1 Additional commentary. Analogously to what we did in the description of various broadcast and broadcast-like protocols in Sections 3 and 4, we express the logic for the dealer as a separate process, even though the dealer is also one of the receiving parties. In particular, the dealer will receive an output from the random beacon, just like the receiving parties. We also define a subroutine Happy, which returns a Boolean value and may be called by any party.

The protocol starts when the dealer D is initiated with inputs $f_1,\ldots,f_L \in \mathbb{A}[x]_{<d}$. The dealer generates random "blinding" polynomials $g^{(r)} \in \mathbb{B}[x]_{<d}$ for $r \in [R]$. The dealer then computes shares of its input and "blinding" polynomials, that is, $v_{\ell,j} := f_\ell(e_j) \in \mathbb{A}$ for $\ell \in [L], j \in [n]$ and $w_j^{(r)} := g^{(r)}(e_j) \in \mathbb{B}$ for $r \in [R], j \in [n]$. Next, the dealer sends each party P_j its share of all of these polynomials as a message m_j via the secure message distribution subprotocol. That is, $m_j = \left(\ \{v_{\ell,j}\}_{\ell\in[L]}, \ \{w_j^{(r)}\}_{r\in[R]} \ \right) \in \mathbb{A}^{[L]} \times \mathbb{B}^{[R]}$ — but if the dealer is corrupt, it may be that $m_j = \perp$.

Upon receiving m_j via the secure message distribution subprotocol, party P_j will initiate the random beacon subprotocol, and then wait for the output $\{\theta_\ell^{(r)}\}_{r\in[R],\ell\in[L]}$ of the random beacon. The random beacon output serves as a random "challenge".

Given the random "challenge" from the beacon, the dealer D computes "response" polynomials $h^{(r)} := g^{(r)} + \sum_{\ell\in[L]} \theta_\ell^{(r)} \cdot f_\ell \in \mathbb{B}[x]_{<d}$ for $r \in [R]$, and then broadcasts these polynomials to all parties using the reliable broadcast subprotocol.

Upon receiving these "response" polynomials via the reliable broadcast subprotocol, each party P_j checks the validity of the data it has received so far (this is done using the Happy subroutine in the figure). Essentially, P_j checks the identities $h^{(r)}(e_j) = w_j^{(r)} + \sum_{\ell\in[L]} \theta_\ell^{(r)} \cdot v_{\ell,j}$ for $r \in [R]$. If the dealer is honest, this check will certainly pass; however, if the dealer is corrupt, it may fail (and will certainly fail if $m_j = \perp$). If this check passes, P_j sets $happy_j$ to true, and we say P_j is "happy"; otherwise, P_j sets $happy_j$ to false, and we say P_j is "unhappy".

Next, each party P_j initiates the one-sided voting subprotocol if $happy_j$ is true. If this subprotocol returns done, the parties enter the *output stage*.

In the *output stage*, a "happy" party P_j may immediately output its shares $\{v_{\ell,j}\}_{\ell\in[L]}$, but waits around in case it receives a valid "complaint" from any "unhappy" party, to which it will respond by broadcasting an "assist" to all parties. Conversely, an "unhappy" party P_j broadcasts a "complaint" against the dealer and then waits for sufficiently many valid "assists", which will allow it to construct its correct shares.

Π_{avss1}

*// **Dealer** D **with input** $f_1, \ldots, f_L \in \mathbb{A}[x]_{<d}$*
for all $r \in [R]$: choose random $g^{(r)} \in \mathbb{B}[x]_{<d}$
for all $\ell \in [L], j \in [n]$: compute $v_{\ell,j} \leftarrow f_\ell(e_j) \in \mathbb{A}$
for all $r \in [R], j \in [n]$: compute $w_j^{(r)} \leftarrow g^{(r)}(e_j) \in \mathbb{B}$
invoke operation $\mathsf{Input}(\mathsf{distribute}, (m_1, \ldots, m_n))$ on $\mathcal{F}_{\text{SecMsgDst}}$, where
$m_j \coloneqq \big(\{v_{\ell,j}\}_{\ell\in[L]},\ \{w_j^{(r)}\}_{r\in[R]} \big) \in \mathbb{A}^{[L]} \times \mathbb{B}^{[R]}$ for all $j \in [n]$
wait for $\mathcal{F}_{\text{Beacon}}$ to deliver a message $\mathsf{Output}(\{\theta_\ell^{(r)}\}_{r\in[R],\ell\in[L]})$ *// with each* $\theta_\ell^{(r)} \in \mathbb{B}$
for all $r \in [R]$: compute $h^{(r)} \leftarrow g^{(r)} + \sum_{\ell\in[L]} \theta_\ell^{(r)} \cdot f_\ell \in \mathbb{B}[x]_{<d}$
invoke operation $\mathsf{Input}(\{h^{(r)}\}_{r\in[R]})$ on $\mathcal{F}_{\text{ReliableBroadcast}}$

*// **Receiving party** P_j*
wait for $\mathcal{F}_{\text{SecMsgDst}}$ to deliver the message $\mathsf{Output}(\mathsf{distribute}, m_j)$ *// m_j may be $\perp$ if D is corrupt*
invoke operation $\mathsf{Input}(\mathsf{init})$ on $\mathcal{F}_{\text{Beacon}}$
wait for $\mathcal{F}_{\text{Beacon}}$ to deliver a message $\mathsf{Output}(\{\theta_\ell^{(r)}\}_{r\in[R],\ell\in[L]})$ *// with each* $\theta_\ell^{(r)} \in \mathbb{B}$
wait for $\mathcal{F}_{\text{ReliableBroadcast}}$ to deliver a message $\mathsf{Output}(\{h^{(r)}\}_{r\in[R]})$
$happy_j \leftarrow \mathsf{Happy}(j, \{\theta_\ell^{(r)}\}_{r\in[R],\ell\in[L]}, m_j, \{h^{(r)}\}_{r\in[R]})$
if $happy_j$ then invoke operation $\mathsf{Input}(\mathsf{init})$ on $\mathcal{F}_{\text{OneSidedVote}}$
wait for $\mathcal{F}_{\text{OneSidedVote}}$ to deliver $\mathsf{Output}(\mathsf{done})$

// output stage
if $happy_j$ then
  output $\{v_{\ell,j}\}_{\ell\in[L]}$
  wait until: for some index $i \in [n]$ and message m_i:
    $\mathcal{F}_{\text{SecMsgDst}}$ delivers $\mathsf{Output}(\mathsf{forward}, i, i, \mathsf{complaint}, m_i)$
    and not $\mathsf{Happy}(i, \{\theta_\ell^{(r)}\}_{r\in[R],\ell\in[L]}, m_i, \{h^{(r)}\}_{r\in[R]})$
  for all $i^* \in [n] \setminus \{j\}$:
    invoke operations $\mathsf{Input}(\mathsf{forward}, j, i^*, \mathsf{assist})$ and $\mathsf{Input}(\mathsf{forward}, i, i^*, \mathsf{report})$ on $\mathcal{F}_{\text{SecMsgDst}}$
else
  for all $i^* \in [n] \setminus \{j\}$: invoke operation $\mathsf{Input}(\mathsf{forward}, j, i^*, \mathsf{complaint})$ on $\mathcal{F}_{\text{SecMsgDst}}$
  wait until: for some $\mathcal{I} \subseteq [n]$, $\{m_i\}_{i\in\mathcal{I}}$:
    $|\mathcal{I}| = d$ and for all $i \in \mathcal{I}$:
      $\mathcal{F}_{\text{SecMsgDst}}$ delivers $\mathsf{Output}(\mathsf{forward}, i, i, \mathsf{assist}, m_i)$
      and $\mathsf{Happy}(i, \{\theta_\ell^{(r)}\}_{r\in[R],\ell\in[L]}, m_i, \{h^{(r)}\}_{r\in[R]})$
  for each $i \in \mathcal{I}$: write m_i as $\big(\{v_{\ell,i}\}_{\ell\in[L]}, \cdot \big)$
  for all $\ell \in [L]$: compute $v^*_{\ell,j} \leftarrow \sum_{i\in\mathcal{I}} \lambda^{\mathcal{I}}_{i,j} v_{\ell,i} \in \mathbb{A}$
  // the values $\lambda^{\mathcal{I}}_{i,j}$ are Lagrange interpolation coefficients
  output $\{v^*_{\ell,j}\}_{\ell\in[L]}$

*// **Auxiliary routine***
$\mathsf{Happy}(\ j, \{\theta_\ell^{(r)}\}_{r\in[R],\ell\in[L]},\ m, \{h^{(r)}\}_{r\in[R]}\ ) \coloneqq$ *// $j \in [n]$, $\{\theta_\ell^{(r)}\}_{r\in[R],\ell\in[L]} \in \mathbb{B}^{[R]\times[L]}$*
  return $\Big($ m is of the form $\big(\{v_\ell\}_{\ell\in[L]},\ \{w^{(r)}\}_{r\in[R]} \big) \in \mathbb{A}^{[L]} \times \mathbb{B}^{[R]}$, and for all $r \in [R]$:
    $h^{(r)} \in \mathbb{B}[x]_{<d}$ and $h^{(r)}(e_j) = w^{(r)} + \sum_{\ell\in[L]} \theta_\ell^{(r)} \cdot v_\ell$ $\Big)$

Fig. 11. An AVSS protocol over $\mathbb{A}$

A "complaint" is a message that is forwarded from a party via the secure message distribution subprotocol (with a complaint tag) that serves as a proof that it is indeed "unhappy" — and that the dealer must be corrupt. An "assist" is also a message that is forwarded from a party via the secure message distribution subprotocol (with an assist tag). These "complaints" and "assists" contain the original message received by a party in the secure message distribution subprotocol, and any other party may validate these messages (from the point of view of the "complaining" and "assisting" party). Of course, if the dealer is honest, there cannot be any "unhappy" parties — honest or corrupt. When an "unhappy" party receives d valid "assists", this will allow it to compute its correct shares by polynomial interpolation. In the figure, for a subset $\mathcal{I} \subseteq [n]$ of size d, we denote by $\{\lambda_{i,j}^{\mathcal{I}}\}_{i\in\mathcal{I},j\in[n]}$ the "Lagrange coefficients", which are elements of $\mathbb{A}$ that satisfy

$$f(j) = \sum_{i\in\mathcal{I}} \lambda_{i,j}^{\mathcal{I}} \cdot f(i)$$

for any polynomial $f \in \mathbb{A}[x]_{<d}$, and which are efficiently computable. An "unhappy" party may use these Lagrange coefficients to compute its shares from the shares of its "happy" peers that provided valid "assists".

Note that in the protocol, when a party broadcasts an "assist" it also broadcasts a "report" — this is a copy of the original "complaint" that triggered the "assist" broadcast, and it is sent using the forwarding mechanism (with a report tag). The purpose of this is to ensure that if *any* honest party broadcasts a "complaint" or an "assist", then *all* honest parties will *eventually* receive evidence that implicates the corrupt dealer. We do not specify here how parties process this evidence. For example, in the short term, once an honest party has identified a corrupt party, the honest party can safely ignore all messages from the corrupt party in the future (or at least until some "proactive refresh" of the system occurs). When sufficiently many honest parties start ignoring a corrupt dealer, that dealer will no longer be able to trigger any more "complaints" or "assists" — at least, assuming we use protocol $\Pi_{\text{SecMsgDst}}$ for secure message distribution (or something similar), which guarantees that if sufficiently many honest parties ignore the dealer, no honest party will even receive any shares. In the longer term, the honest parties can vote to remove the corrupt party from the network. Also, such evidence of corrupt behavior could lead to legal or financial jeopardy for the corrupt party, and this in itself may be enough to discourage such behavior.

Without this extra step of broadcasting a "report", an adversary could force the protocol off the "happy path", causing the communication complexity to blow up, but without ever leading to a situation where all honest parties have obtained evidence that implicates the corrupt dealer. For example, a corrupt dealer could give bad shares to a corrupt party P_j, and then P_j could "complain" to just a subset $\mathcal{S}$ of the honest parties. The members of $\mathcal{S}$ would all see the "complaints" and broadcast "assists", but honest parties outside of $\mathcal{S}$ would never see any "complaints", and the "assists" by themselves do not implicate the corrupt dealer.

If the dealer is honest, it is fairly easy to see that all honest parties will eventually output their shares of the dealer's polynomials. Moreover, the "blinding" polynomials, together with the fact that no corrupt party can lodge a valid complaint against an honest dealer, will ensure that the adversary obtains no information about the dealer's polynomials other than the shares belonging to corrupt parties.

Now suppose the dealer is corrupt. In this case, if any party enters the *output stage*, the one-sided voting protocol ensures that at least $n-2t \geq d$ honest parties are "happy" and that all honest

parties eventually enter the *output stage*. In addition, these d "happy" honest parties will ensure that any "unhappy" honest parties eventually receive the "assists" that they need to obtain their shares. In the security proof, we will argue that with overwhelming probability, the shares of all the "happy" parties (honest or corrupt) must lie on polynomials of degree less than d, as required. For this argument to work, it is crucial that honest parties do not initiate the random beacon subprotocol before they receive their message from the secure message distribution subprotocol — this ensures that the "challenge" is not revealed before the dealer has committed to the entire message vector $(m_1, \ldots, m_n)$.

Although our protocol is a secure AVSS protocol, is has an interesting property. Namely, a corrupt dealer does not really commit to its input polynomials until after it obtains the "challenge" from the beacon and reveals its "response" polynomials. For example, let $n = 3t + 1$ and $d = t + 1$. A corrupt dealer can give the honest parties completely uncorrelated random shares, and after receiving the "challenge", it can choose a random set of $t + 1$ honest parties, interpolate through just these parties' shares to obtain corresponding input polynomials $f_1, \ldots, f_L$, and then compute corresponding "response" polynomials. These $t+1$ honest parties together with the t corrupt parties can force all honest parties to output shares of $f_1, \ldots, f_L$. Of course, in this example, the remaining t honest parties will "complain" against the dealer, proving that the dealer is corrupt.

5.2 Security analysis

In order to analyze the failure probability of this generic protocol, we need the following definition.

Definition 5.1 (Inner product bound). *With $\mathbb{A}$, $\mathbb{B}$, and $\Theta \subseteq \mathbb{B}^L$ as above, we define*

$$\chi(\mathbb{A}, \mathbb{B}, \Theta)$$

to be the maximum, over all $b \in \mathbb{B}$ and nonzero $\boldsymbol{a} \in \mathbb{A}^L$, of the probability that

$$b + \langle \boldsymbol{a}, \boldsymbol{\theta} \rangle = 0,$$

where $\boldsymbol{\theta}$ is chosen uniformly at random from Θ.

In a typical instantiation, one would choose $\mathcal{E}$ to be a maximum sized exceptional set in $\mathbb{B}$ so that we can apply Schwarz-Zippel (Lemma 2.1). In this setting, we have: if $\Theta = \Theta_{\text{pow}}^{\mathcal{E}}$, then $\chi(\mathbb{A}, \mathbb{B}, \Theta) \leq L/|\mathcal{E}|$, and if $\Theta = \Theta_{\text{lin}} = \mathcal{E}^L$, then $\chi(\mathbb{A}, \mathbb{B}, \Theta) = 1/|\mathcal{E}|$. Choosing a larger ring $\mathbb{B}$ will allow one to increase the size of $\mathcal{E}$, however this comes at the expense of increasing the size of the elements which need to be transmitted.

Theorem 5.1 (Security of Π_{avss1}). *Assume $2^n \cdot \chi(\mathbb{A}, \mathbb{B}, \Theta)^R$ is negligible. Then we have:*

(i) Π_{avss1} securely realizes $\mathcal{F}_{\text{avss}}$ in the $(\mathcal{F}_{\text{SecMsgDst}}, \mathcal{F}_{\text{Beacon}}, \mathcal{F}_{\text{ReliableBroadcast}}, \mathcal{F}_{\text{OneSidedVote}})$-hybrid model.

(ii) If Π_{avss1} is instantiated with concrete protocols for $\mathcal{F}_{\text{SecMsgDst}}$, $\mathcal{F}_{\text{Beacon}}$, $\mathcal{F}_{\text{ReliableBroadcast}}$, and $\mathcal{F}_{\text{OneSidedVote}}$ that are secure (i.e., securely realize the corresponding functionality) and complete (i.e., satisfy the corresponding completeness property), then the resulting concrete protocol

(a) securely realizes $\mathcal{F}_{\text{avss}}$, and

(b) satisfies the AVSS completeness property.

Proof. We start with statement (i) of the theorem. To that end, we need to show that there is a simulator that interacts with $\mathcal{F}_{\text{avss}}$ in the ideal world such that no environment can effectively distinguish the ideal world from the hybrid world.

Without loss of generality, we may assume that in the hybrid world, the adversary is a "dummy" adversary that essentially acts as a "router" between the environment and the hybrid functionalities. In addition, in the ideal world, our simulator is actually in charge of implementing the hybrid functionalities. In particular, in the ideal world, any messages sent from (resp., to) the adversary to (resp., from) these hybrid functionalities are actually sent directly to (resp., from) our simulator — this including the inputs (resp., outputs) of corrupt parties.

If the dealer is honest, the proof reduces to showing that the values $\{h^{(r)}\}_{r\in[R]}$ and $\{w_j^{(r)}\}_{r\in[R]}$ for $j \in \mathcal{C}$ do not leak any extra information. This is a standard argument, based on the randomness supplied by the "blinding" polynomials $\{g^{(r)}\}_{r\in[R]}$. In more detail, the ideal functionality $\mathcal{F}_{\text{avss}}$ gives the simulator the values $v_{\ell,j}$ for $\ell \in [L]$ and $j \in \mathcal{C}$. For $r \in [R]$, the simulator then chooses $h^{(r)} \in \mathbb{B}[x]_{<d}$ at random, and then computes

$$w_j^{(r)} \leftarrow h^{(r)}(e_j) - \sum_{\ell\in[L]} \theta_\ell^{(r)} \cdot v_{\ell,j} \quad (\text{for } j \in \mathcal{C}).$$

Note that the simulator can also generate the random beacon values $\theta_\ell^{(r)}$ in advance of this computation.

The more interesting case is that when the dealer is corrupt. The crux of the proof in this case is showing that by the first point in time at which any honest party outputs its shares, the simulator can effectively extract corresponding polynomials $f_1, \ldots, f_L \in F[x]_{<d}$. The proof is similar to the analysis in [DN07]. The main difference is that our protocol may terminate successfully if any subset of $n - 2t$ honest parties is happy, and this subset may be determined *after* the random beacon is revealed. A simple way to deal with this is to apply the union bound to the collection of all subsets of parties, which is where the factor 2^n in the theorem statement comes from.

In more detail, consider the inputs $(m_1, \ldots, m_n)$ to $\mathcal{F}_{\text{SecMesDst}}$, where each m_j is either $\perp$ or of the form

$$m_j = \big(\{v_{\ell,j}\}_{\ell\in[L]},\ \{w_j^{(r)}\}_{r\in[R]} \big), \tag{1}$$

and which must be submitted to $\mathcal{F}_{\text{SecMsgDst}}$ and hence to our simulator, before the random beacon is revealed. (Here, we are essentially using the input extractability property of the secure message distribution protocol.) We will generally ignore indices j such that $m_j = \perp$.

Consider the later point in time at which some any party first enters the *output stage*. At this point in time, we can define $\mathcal{P}^*$ to be the set of indices $j \in [n]$ for which P_j is happy, as determined by the input m_j, the random beacon value, and the "response" polynomials $h^{(r)}$. This set includes *all* parties, including honest parties P_j that have not computed $happy_j$, as well as corrupt parties. By the correctness property of the simple asynchronous agreement protocol, we have $|\mathcal{P}^* \cap \mathcal{H}| \geq n - 2t \geq d$.

For each $\ell \in [L]$, the simulator extracts D's input polynomial f_ℓ as the unique polynomial of degree less than $|\mathcal{P}^*|$ that interpolates through the points $\{(e_j, v_{\ell,j})\}_{j\in\mathcal{P}^*}$. The simulation fails iff any of these polynomials has degree $\geq d$. Indeed, if any of these polynomials has degree $\geq d$, then the simulation obviously fails. Conversely, if all of these polynomials have degree $< d$, then one

sees that the complaint mechanism works correctly: the honest parties hold enough good shares to reconstruct the polynomials by themselves (since $|\mathcal{P}^* \cap \mathcal{H}| \geq d$); moreover, the corrupt parties cannot contribute bad shares during this process (which is why we include corrupt parties in the definition of $\mathcal{P}^*$).

We want to bound the probability that the simulation fails. To this end, let us first make some definitions. Consider any fixed inputs $(m_1, \ldots, m_n)$ to $\mathcal{F}_{\text{SecMsgDst}}$, where each $m_j = \perp$ or is of the form (1). Consider any set $\mathcal{P} \subseteq [n]$ with $n' \coloneqq |\mathcal{P}| \geq d$ and $m_j \neq \perp$ for all $j \in \mathcal{P}$. We say $\mathcal{P}$ is **d-consistent** if for each $\ell \in [L]$, the points $\{(e_j, v_{\ell,j})\}_{j \in \mathcal{P}}$ lie on a polynomial over $\mathbb{A}$ of degree less than d. Consider any fixed element $\{\theta_\ell^{(r)}\}_{r \in [R], \ell \in [L]}$ of the output space Ω of the random beacon. We say $\mathcal{P}$ is **d-consistent modulo** $\{\theta_\ell^{(r)}\}_{r \in [R], \ell \in [L]}$ if for each $r \in [R]$, the points

$$\left\{ \left(e_j,\ w_j^{(r)} + \sum_{\ell \in [L]} \theta_\ell^{(r)} \cdot v_{\ell,j} \right) \right\}_{j \in \mathcal{P}}$$

lie on a polynomial over $\mathbb{B}$ of degree less than d.

Claim: *If $\mathcal{P}$ is not d-consistent, then the probability that it is d-consistent modulo a randomly chosen element of Ω is at most $\chi(\mathbb{A}, \mathbb{B}, \Theta)^R$.*

To prove the claim, we consider the (n', d)-Reed-Solomon code over $\mathbb{A}$ with respect to the evaluation coordinates $\{e_j\}_{j \in \mathcal{P}}$, and the corresponding check matrix $C \in \mathbb{A}^{n' \times (n'-d)}$. We also consider the corresponding "extended" (n', d)-Reed-Solomon code over the extension ring $\mathbb{B}$, which has the same check matrix C as our original code (since the evaluation coordinates lie in $\mathbb{A}$).

Now suppose $\mathcal{P}$ is not d-consistent. For $\ell \in [L]$, define the vector

$$\boldsymbol{v}_\ell \coloneqq \{v_{\ell,j}\}_{j \in \mathcal{P}} \in \mathbb{A}^{n'}.$$

The assumption that $\mathcal{P}$ is not d-consistent means that for some $\ell^* \in [L]$, the vector $\boldsymbol{v}_{\ell^*}$ is not a codeword, which means $\boldsymbol{v}_{\ell^*} \cdot C \neq 0$. So if we define the matrix $Q \in \mathbb{A}^{L \times n'}$ whose ℓ-th row is $\boldsymbol{v}_\ell$ for $\ell \in [L]$, then $Q \cdot C \in \mathbb{A}^{L \times (n'-d)}$ is nonzero matrix.

Now suppose $\mathcal{P}$ is d-consistent modulo $\{\theta_\ell^{(r)}\}_{r \in [R], \ell \in [L]}$. For each $r \in [R]$, define

$$\boldsymbol{w}^{(r)} \coloneqq \{w_j^{(r)}\}_{j \in \mathcal{P}} \in \mathbb{B}^{n'} \quad \text{and} \quad \boldsymbol{\theta}^{(r)} \coloneqq (\theta_1^{(r)}, \ldots, \theta_L^{(r)}) \in \Theta$$

so that

$$\boldsymbol{w}^{(r)} + \boldsymbol{\theta}^{(r)} \cdot Q$$

lies in the extended Reed-Solomon code, which implies

$$\boldsymbol{w}^{(r)} \cdot C + \boldsymbol{\theta}^{(r)} \cdot Q \cdot C = 0. \tag{2}$$

Since $Q \cdot C$ is a nonzero matrix, we can choose one nonzero column of $Q \cdot C$, and (2) implies that for some fixed $b \in \mathbb{B}$ and fixed, nonzero $\boldsymbol{a} \in \mathbb{A}^L$, we have

$$b + \langle \boldsymbol{a}, \boldsymbol{\theta}^{(r)} \rangle = 0. \tag{3}$$

So for each $r \in [R]$, if we choose $\boldsymbol{\theta}^{(r)} \in \Theta$ at random, equation (3) holds with probability at most $\chi(\mathbb{A}, \mathbb{B}, \Theta)$, and repeating this R times gives the desired probability and proves the claim.

We now return to the task of bounding the probability that the simulation fails. If the simulation fails, this implies that $\mathcal{P}^*$ is not d-consistent yet is d-consistent modulo the output of the random beacon. Now, even though the inputs to $\mathcal{F}_{\text{SecMsgDst}}$ are chosen before the random beacon is revealed, the subset $\mathcal{P}^*$ is chosen by the adversary *after* the random beacon is revealed. Nevertheless, for the simulation to fail, the following event must occur: the adversary submits inputs to $\mathcal{F}_{\text{SecMsgDst}}$ such that there exists a subset $\mathcal{P} \subseteq [n]$ of the required form that is not d-consistent but ends up being d-consistent modulo the output of the random beacon. As there are at most 2^n choices for $\mathcal{P}$, by the union bound, the probability that the simulation fails is therefore at most $2^n \cdot \chi(\mathbb{A}, \mathbb{B}, \Theta)^R$.

That proves statement (i) of the theorem. Statement (ii)(a) is a direct consequence of statement (i) and the UC composition theorem. Statement (ii)(b) easily follows from the security and completeness properties of the concrete subprotocols, and the logic of Π_{avss1}, along the lines discussed in Section 5.1.1. □

Note that we obtain a somewhat better failure bound when we use $\Theta = \Theta^{\mathcal{E}}_{\text{lin}}$. However, in this instantiation, our random beacon has to output very long vectors $(\theta_1, \ldots, \theta_L) \in \mathcal{E}^L$. As discussed in Section 3.1.2, this could be avoided by using a random beacon that outputs a short seed that is then stretched using a cryptographic hash G. This implementation will indeed be secure if we model G as a random oracle. However, we cannot justify the security of this implementation if we simply assume that G is a pseudorandom generator. As discussed in Section 3.1.2, to do this, it would suffice that the failure event in Theorem 5.1 can be efficiently detected as a function of the output of G and data that is available to the adversary prior to revealing the beacon. Unfortunately, this event is a union over an exponentially large set of events, and so is not efficiently detectable.

Note that the factor 2^n in the failure bound in Theorem 5.1 does not arise in the analysis of the corresponding protocol in [DN07]. For modest sized n, this should be acceptable, using a larger extension $\mathbb{B}$, or larger value of R, as necessary. Of course, using a larger $\mathbb{B}$ or R can impact communication complexity. This is discussed in detail below in Section 5.3. However, in some important applications, $\mathbb{A}$ is a field of size $\approx 2^{256}$, and so for $n \leq 128$, we already get nearly 128-bit security with $\mathbb{B} = \mathbb{A}$ and $R = 1$.

An example. Here is a simple example that shows why an exponential factor in the failure probability in Theorem 5.1 seems hard to avoid. This example gives a specific, efficient adversary and specific protocol parameters such that the adversary breaks the security of the protocol with probability $2^{\Omega(n)} \cdot \chi(\mathbb{A}, \mathbb{B}, \Theta)^R$.

Let $n = 3t + 1$, $d = t + 1$, $L = 1$, $R = 1$, $\mathbb{B} = \mathbb{A}$, and $\Theta = \Theta_{\text{pow}}$. The corrupt dealer starts the protocol by distributing shares of $g := x^{t+2} + x^{t+1}$ and $f := x^{t+2}$ (which play the roles of $g^{(1)}$ and f_1, respectively, in the protocol). Now consider any set $\mathcal{P} \subseteq [n]$ of size $t + 2$, and define

$$g^{(\mathcal{P})} := g \bmod \prod_{j \in \mathcal{P}} (x - e_j)$$

and

$$f^{(\mathcal{P})} := f \bmod \prod_{j \in \mathcal{P}} (x - e_j).$$

Then we have

$$g^{(\mathcal{P})} = (1 + T^{(\mathcal{P})}) \cdot x^{t+1} + \text{lower order terms}$$

and

$$f^{(\mathcal{P})} = T^{(\mathcal{P})} \cdot x^{t+1} + \text{lower order terms},$$

where

$$T^{(\mathcal{P})} := -\sum_{j \in \mathcal{P}} e_i.$$

For a given challenge θ, the adversary will break the protocol if it can find a subset $\mathcal{P} \subseteq \mathcal{H}$ of size $t+2$ such that

$$(1 + T^{(\mathcal{P})}) + \theta \cdot T^{(\mathcal{P})} = 0,$$

or in other words

$$T^{(\mathcal{P})} = -1/(1+\theta).$$

Indeed, in this case, the adversary can compute a response polynomial that makes the parties in $\mathcal{P}$ happy, even though their shares do not lie on a polynomial of degree at most t. Moreover, the t corrupt parties, together with the parties in $\mathcal{P}$, can force the one-sided voting protocol to succeed, so all honest parties enter the output stage. To obtain an efficient attack that succeeds with probability at least $2^{\Omega(n)}/|\mathbb{B}|$, it suffices that the map $\mathcal{P} \mapsto T^{(\mathcal{P})}$ is (nearly) one-to-one and easy to invert. For example, if $|\mathbb{A}| > 2^n$ and the evaluation coordinates are $1, 2, \ldots, 2^{n-1}$, then this is the case.

5.3 Communication Complexity

We now consider the communication complexity of Π_{avss1}. Here, the communication complexity of a protocol is defined to be the sum of the length of all messages sent by honest parties (to either honest or corrupt parties) over the point-to-point channels.

We make a distinction between the "happy path" and the "unhappy path". To enter the "unhappy path", a corrupt party must *provably* misbehave. In our protocol, this corresponds to the situation where a party complains against a corrupt dealer. If this happens, all honest parties will learn of this and can take action: in the short term, the honest parties can safely ignore this party, and in the longer term, the corrupt party can be removed from the network. Also, such provable misbehavior could lead to legal or financial jeopardy for the corrupt party, and this in itself may be enough to discourage such behavior. Note that the "happy path" includes corrupt behavior, including collusion among the corrupt parties, as well as behavior that is clearly corrupt as observed by an individual honest party, but that cannot be used as reliable evidence to convince other honest parties or an external authority of corrupt behavior.

For these reasons, we believe it makes sense to make a distinction between the complexity of the protocol on the "happy path" versus the "unhappy path".

We also make a couple of simplifying assumptions. Namely, we assume that $\mathbb{B} = \mathbb{A}$ and that $\Theta = \Theta^{\mathcal{E}}_{\text{lin}}$, where $\mathcal{E}$ is an exceptional set of maximal size. In this case, the failure bound in Theorem 5.1 becomes $2^n/|\mathcal{E}|^R$. We will want to set R so that this bound is negligible. This is discussed below.

5.3.1 The Happy Path.

Each message m_j input into the $\mathcal{F}_{\text{SecMsgDst}}$ has size

$$O\left((L+R) \cdot \log|\mathbb{A}|\right).$$

Our protocol $\Pi_{\text{SecMsgDst}}$ for secure message distribution from Section 4 has communication complexity $O(|\mathbf{m}| + \lambda \cdot n^2 \cdot \log n)$, and so its contribution to the total communication complexity is

$$O\left(n \cdot (L+R) \cdot \log|\mathbb{A}| + \lambda \cdot n^2 \cdot \log n\right).$$

The message input to $\mathcal{F}_{\text{ReliableBroadcast}}$ is of size $O(n \cdot R \cdot \log|\mathbb{A}|)$. Using protocol $\Pi_{\text{CompactBroadcast}}$ from Section 3.2.2, this contributes

$$O\left(n^2 \cdot R \cdot \log|\mathbb{A}| + \lambda \cdot n^2 \cdot \log n\right)$$

to the overall communication complexity. We may implement $\mathcal{F}_{\text{OneSidedVote}}$ as in Section 3.2.5, which contributes $O(n^2)$ to the communication complexity. We shall ignore for now the communication complexity of the random beacon. Thus the total communication complexity, ignoring the random beacon, is

$$O\left(n \cdot (L+nR) \cdot \log|\mathbb{A}| + \lambda \cdot n^2 \cdot \log n\right). \tag{4}$$

If we want σ bits of security, we should select R as

$$R = \left\lceil \frac{\sigma + n}{\log_2|\mathcal{E}|} \right\rceil. \tag{5}$$

With this setting of R, our communication complexity bound (4) becomes

$$O\left(n \cdot L \cdot \log|\mathbb{A}| + n^2 \cdot (\sigma+n) \cdot \frac{\log|\mathbb{A}|}{\log|\mathcal{E}|} + \lambda \cdot n^2 \cdot \log n\right). \tag{6}$$

5.3.2 Finite Field Case. Suppose $\mathbb{A}$ is a field extension of degree δ over the finite field $\mathbb{S} = \mathbb{F}_q$. In this case, $\mathcal{E} = \mathbb{A}$ and $|\mathbb{A}| = |\mathbb{S}|^\delta$; therefore, (6) simplifies to

$$O\left(n \cdot L \cdot \delta \cdot \log|\mathbb{S}| + n^3 + n^2 \cdot \sigma + \lambda \cdot n^2 \cdot \log n\right),$$

and if $\max\{n, \sigma\} \leq \lambda$, this simplifies even further to

$$O\left(n \cdot L \cdot \delta \cdot \log|\mathbb{S}| + \lambda \cdot n^2 \cdot \log n\right).$$

While we have ignored the communication complexity of the random beacon, we may assume it is bounded by a polynomial in n and λ (which, in practice, is typically $O(n^2\lambda)$). Therefore, for sufficiently large L (polynomial in n, σ, and λ) the amortized communication complexity per sharing is

$$O(n \cdot \delta \cdot \log|\mathbb{S}|).$$

Suppose that our AVSS protocol is used in an application where the secrets lie in the field $\mathbb{S} = \mathbb{F}_q$, where $q \leq n$. In this case, as discussed in Section 2.4, we will have to run our protocol over a field $\mathbb{A}$ of degree δ over $\mathbb{S}$, where $\delta = \lceil \log_q(n+1) \rceil$. In this case, the amortized communication complexity per sharing is

$$O(n \cdot \log_q n \cdot \log|\mathbb{S}|),$$

which is

$$O(n \cdot \log n).$$

5.3.3 Galois Ring Case. Suppose $\mathbb{A}$ is a Galois ring of degree δ over the ring $\mathbb{S} = \mathbb{Z}/(p^k)$. In this case, $|\mathbb{A}| = |\mathbb{S}|^\delta = p^{k\cdot\delta}$ but $|\mathcal{E}| = p^\delta$, and (6) simplifies to

$$O\left(n \cdot L \cdot \delta \cdot \log|\mathbb{S}| + n^2 \cdot (\sigma + n) \cdot k + \lambda \cdot n^2 \cdot \log n\right).$$

Therefore, for sufficiently large L (polynomial in n, σ, λ, and k) the amortized communication complexity per sharing is

$$O(n \cdot \delta \cdot \log|\mathbb{S}|).$$

Suppose that our AVSS protocol is used in an application where the secrets lie in the ring $\mathbb{S} = \mathbb{Z}/(p^k)$, where $p \leq n$. In this case, as discussed in Section 2.4, we will have to run our protocol over a Galois ring $\mathbb{A}$ of degree δ over $\mathbb{S}$, where $\delta = \lceil \log_p(n+1) \rceil$. In this case, the amortized communication complexity per sharing is

$$O(n \cdot \log_p n \cdot \log|\mathbb{S}|),$$

which is

$$O(n \cdot \log n \cdot k).$$

5.3.4 The Unhappy Path. For the "unhappy path", the communication complexity of the secure message distribution protocol may blow up by a factor of n. So in this case, the bound (4) becomes

$$O\left(n^2 \cdot (L + R) \cdot \log|\mathbb{A}| + \lambda \cdot n^3 \cdot \log n\right),$$

and choosing R as in (5), the bound (6) becomes

$$O\left(n^2 \cdot L \cdot \log|\mathbb{A}| + n^2 \cdot (\sigma + n) \cdot \frac{\log|\mathbb{A}|}{\log|\mathcal{E}|} + \lambda \cdot n^3 \cdot \log n\right).$$

Thus, all of our estimates above for amortized communication complexity per sharing get blown up by a factor of n.

5.3.5 Message complexity. The message complexity of Π_{avss1} on both the "happy path" and the "unhappy path", not including the random beacon, is $O(n^2)$. Here, the message complexity of a protocol is defined to be the total number of messages sent by honest parties (to either honest or corrupt parties) over the point-to-point channels. In practice, the message complexity of the random beacon is typically also $O(n^2)$.

6 Restricting the secrets to a subring

We assume here that a secret is encoded, as usual, as the constant term of a polynomial. As discussed in Section 2.4, our AVSS protocol may be used in an application where the secrets lie in a ring $\mathbb{S}$ that does not contain the appropriate evaluation coordinates, and we are forced to run our AVSS protocol in an extension ring $\mathbb{A}$ that does contain such coordinates. In this section, we give an AVSS protocol that enforces the restriction that the shared secret in fact lies in $\mathbb{S}$. Our protocol works when $\mathbb{S}$ and $\mathbb{A}$ are Galois rings, and does secret sharing over a related ring $\mathbb{A}'$. Our AVSS protocol that enforces this restriction makes use of a subprotocol for secret sharing over $\mathbb{A}'$. Our technique for ensuring inputs lie in $\mathbb{S}$ makes use of the checking technique of extending the p-adic precision one works with, which was first introduced in [CDE+18].

6.1 Auxiliary rings

We begin by describing the relationship between the various rings involved. Let $G(z) \in \mathbb{Z}[z]$ be a monic polynomial of degree $\epsilon \geq 1$. Let $F(y, z) \in \mathbb{Z}[y, z]$ be a bivariate polynomial of the form

$$F(y, z) = F_0(z) + F_1(z) \cdot y + \cdots + F_{\delta-1}(z) \cdot y^{\delta-1} + y^{\delta},$$

where $\delta \geq 1$. For each $m \geq 1$, define the rings

$$\mathbb{S}^{(m)} \coloneqq \mathbb{Z}[z]/(p^m, G(z)) \quad \text{and} \quad \mathbb{A}^{(m)} \coloneqq \mathbb{Z}[y, z]/(p^m, G(z), F(y, z)).$$

We naturally view $\mathbb{Z}/(p^m) \subseteq \mathbb{S}^{(m)} \subseteq \mathbb{A}^{(m)}$ as a tower of ring extensions, where $\mathbb{S}^{(m)}$ has degree ϵ over $\mathbb{Z}/(p^m)$ and $\mathbb{A}^{(m)}$ has degree δ over $\mathbb{S}^{(m)}$. Indeed, every element of $\mathbb{A}^{(m)}$ can be expressed uniquely as the image of a polynomial in $\mathbb{Z}[y, z]$ of the form $A_0(z) + A_1(z) \cdot y + \cdots + A_{\delta-1}(z) \cdot y^{\delta-1}$, where for $i = 0, \ldots, \delta - 1$, we have $\deg A_i < \epsilon$, and each coefficient of A_i lies in the interval $[0, p^m)$. The ring $\mathbb{S}^{(m)}$ corresponds to the subset of such polynomials of degree at most 0 in y. The ring $\mathbb{Z}/(p^m)$ corresponds to the subset of such polynomials of degree at most 0 in y and z.

We shall require that $\mathbb{S}^{(1)}$ and $\mathbb{A}^{(1)}$ are *fields*. This requirement ensures that $\mathbb{S}^{(m)}$ and $\mathbb{A}^{(m)}$ are *Galois rings*. Note that for $m' \geq m$, there is a natural map from $\mathbb{A}^{(m')}$ to $\mathbb{A}^{(m)}$, and the restriction of this map to $\mathbb{S}^{(m')}$ is the natural map from $\mathbb{S}^{(m')}$ to $\mathbb{S}^{(m)}$. The units in $\mathbb{A}^{(m)}$ are the elements whose images in $\mathbb{A}^{(1)}$ are nonzero (this follows from Hensel lifting).

We fix a sequence polynomials $E_1, \ldots, E_n \in \mathbb{Z}[y, z]$ whose images in $\mathbb{A}^{(1)}$ form an exceptional sequence. Note that for every $m \geq 1$, the images of these polynomials in $\mathbb{A}^{(m)}$ also form an exceptional sequence in $\mathbb{A}^{(m)}$.

Now fix an integer $k \geq 1$ and define

$$\mathbb{S} \coloneqq \mathbb{S}^{(k)} \quad \text{and} \quad \mathbb{A} \coloneqq \mathbb{A}^{(k)}.$$

Let $e_1, \ldots, e_n \in \mathbb{A}$ be the images of $E_1, \ldots, E_n$ in $\mathbb{A}$. Our ring of secrets will be $\mathbb{S}$. Our goal is to design a secret sharing protocol that can be used to share of a secret in $\mathbb{S}$, where the evaluation coordinates are $e_1, \ldots, e_n$, and so the shares lie in $\mathbb{A}$ even though the secret lies in $\mathbb{S}$. Such a protocol should provide all the usual guarantees of any secret sharing protocol, but should also enforce the restriction that the shared secret is in $\mathbb{S}$, even if the dealer is corrupt.

To do this, we will actually perform a secret sharing over another ring. Fix an integer $k' \geq k$ and define

$$\mathbb{S}' \coloneqq \mathbb{S}^{(k')} \quad \text{and} \quad \mathbb{A}' \coloneqq \mathbb{A}^{(k')}.$$

Let $e'_1, \ldots, e'_n \in \mathbb{A}'$ be the images of $E_1, \ldots, E_n$ in $\mathbb{A}'$. Let ϕ be the natural map from $\mathbb{A}'$ to $\mathbb{A}$. Observe that $\mathbb{S}'$ is a subring of $\phi^{-1}(\mathbb{S})$. The idea is that we will do the following steps:

1. Perform a sharing of a secret in $\mathbb{S}'$ with shares in $\mathbb{A}'$, with respect to the evaluation coordinates $e'_1, \ldots, e'_n$.
2. Perform a probabilistic check that ensures that the secret lies in $\phi^{-1}(\mathbb{S})$ with high probability.

After this, each party can locally apply ϕ to its share to obtain a sharing of a secret in $\mathbb{S}$ with shares in $\mathbb{A}$, with respect to the evaluation coordinates $e_1, \ldots, e_n$.

On the one hand, if the dealer is honest, in order to protect the privacy of the dealer's secret, it is essential that their secret s' lies in $\mathbb{S}'$. Of course, an honest dealer should really be starting out with a secret in $s \in \mathbb{S}$ and then choose $s' \in \mathbb{S}'$ as some (arbitrary) preimage of s under ϕ. On the other hand, if the dealer is corrupt, the protocol does not enforce the constraint that the dealer's secret lies in $\mathbb{S}'$, but only that it lies in $\phi^{-1}(\mathbb{S})$. In either case, after each party locally applies ϕ to its share, we end up with a sharing of a secret in $\mathbb{S}$.

6.2 Two special cases

We briefly sketch how the above general setting includes two important special cases.

$\mathbb{S}$ is a non-prime finite field: This corresponds to the setting where $k = 1$ and $\epsilon > 1$. In this case, $\mathbb{S} = \mathbb{Z}[z]/(p, G(z))$ is a finite field of cardinality $q = p^\epsilon$, and $\mathbb{A} = \mathbb{Z}[y, z]/(p, G(z), F(y, z))$ is an extension field of degree over δ over $\mathbb{F}_q$ (and so has $q^\delta = p^{\epsilon \cdot \delta}$ elements). We also have corresponding *rings* $\mathbb{S}' = \mathbb{Z}[z]/(p^{k'}, G(z))$ and $\mathbb{A}' = Z[y, z]/(p^{k'}, G(z), F(y, z))$. Note that even though $\mathbb{S}$ and $\mathbb{A}$ are fields, $\mathbb{S}'$ and $\mathbb{A}'$ will not be (assuming $k' > 1$).

$\mathbb{S}$ is of the form $\mathbb{Z}/(p^k)$: This corresponds to setting $G(z) \coloneqq z$, and using a polynomial of the form $F(y) \in \mathbb{Z}[y]$ in the role of $F(y, z)$. Then $\mathbb{S} = \mathbb{Z}/(p^k)$ and $\mathbb{A} = \mathbb{Z}[y]/(p^k, F(y))$. We also have corresponding rings $\mathbb{S}' = \mathbb{Z}/(p^{k'})$ and $\mathbb{A}' = \mathbb{Z}[y]/(p^{k'}, F(y))$.

6.3 The protocol

The basic idea is this. The dealer has polynomials $f_1, \ldots, f_L \in \mathbb{A}'[x]_{<d}$, where for each $\ell \in [L]$, the corresponding secret is the constant term $f_\ell(0)$, which lies in $\mathbb{S}'$. The dealer chooses a random "blinding" polynomial $g \in \mathbb{A}'[x]_{<d}$ also with $g(0) \in \mathbb{S}'$, and then runs an AVSS protocol on the polynomials $f_1, \ldots, f_L, g$. After this, a random beacon is used to generate a random "challenge"

$$\boldsymbol{\gamma} \coloneqq (\gamma_1, \ldots, \gamma_L) \in (\mathbb{Z}/(p^{k'}))^L.$$

The dealer then computes the "response" polynomial

$$h \leftarrow g + \sum_{\ell \in [L]} \gamma_\ell \cdot f_\ell, \tag{7}$$

which is also a polynomial in $\mathbb{A}'[x]_{<d}$ with $h(0) \in \mathbb{S}'$, and reliably broadcasts h. After receiving the polynomial h and verifying that it is of the correct form (i.e., of the right degree and with constant term in $\mathbb{S}'$), each party P_j verifies that h is locally correct based on its shares by checking that (7) holds at the evaluation coordinate e'_j. The parties then run a trivial voting protocol that will ensure that they only output their shares if at least $n - 2t \geq d$ parties have successfully performed this local check. This ensures that (7) holds. We then argue that if $f_{\ell^*}(0) \notin \phi^{-1}(\mathbb{S})$ for some $\ell^* \in [L]$, then the probability that $h(0) \in \mathbb{S}'$ for randomly chosen $\boldsymbol{\gamma}$ is at most $p^{k-k'-1}$. This implies that except with probability $p^{k-k'-1}$, we can be sure that the secret $f_\ell(0)$ lies in $\phi^{-1}(\mathbb{S})$ all $\ell \in [L]$.

Our protocol, which we call Π_{ravss1}, is presented in Fig. 12. It makes use of a repetition parameter R, so that the above probabilistic check is actually performed R times. It makes use of an $(n, d, L + R)$-AVSS subprotocol over $\mathbb{A}'$ with respect to $(e'_1, \ldots, e'_n)$. In the description of Π_{ravss1}, we invoke this as an ideal functionality $\mathcal{F}_{\text{avss}}$. Protocol Π_{ravss1} also makes use of

- A random beacon whose output space is defined to be

$$\left\{ \left\{ \gamma_\ell^{(r)} \right\}_{\ell \in [L], r \in [R]} \ : \ \gamma_\ell^{(r)} \in \mathbb{Z}/(p^{k'}) \right\},$$

  which is invoked as an ideal functionality $\mathcal{F}_{\text{Beacon}}$;
- A reliable broadcast subprotocol, which is invoked as an ideal functionality $\mathcal{F}_{\text{ReliableBroadcast}}$.

As we shall argue below, when the protocol produces an output, the shared secrets must lie in $\phi^{-1}(\mathbb{S})$ (with high probability). As mentioned above, each party can then locally apply ϕ to its shares to obtain sharings of secrets in $\mathbb{S}$.

Π_{ravss1}

// ***Dealer*** D ***with input*** $f_1, \ldots, f_L \in \mathbb{A}'[x]_{<d}$ with constant terms in $\mathbb{S}'$
for all $r \in [R]$: choose random $g^{(r)} \in \mathbb{A}'[x]_{<d}$
invoke operation Input($\{f_\ell\}_{\ell\in[L]}$, $\{g^{(r)}\}_{r\in[R]}$) on $\mathcal{F}_{\text{avss}}$
wait for $\mathcal{F}_{\text{Beacon}}$ to deliver a message Output($\{\gamma_\ell^{(r)}\}_{\ell\in[L],r\in[R]}$)
for all $r \in [R]$: compute $h^{(r)} \leftarrow g^{(r)} + \sum_{\ell\in[L]} \gamma_\ell^{(r)} \cdot f_\ell \in \mathbb{A}'[x]_{<d}$
invoke operation Input($\{h^{(r)}\}_{r\in[R]}$) on $\mathcal{F}_{\text{ReliableBroadcast}}$

// ***Receiving party*** P_j
wait for $\mathcal{F}_{\text{avss}}$ to deliver a message Output($\{v_{\ell,j}\}_{\ell\in[L]}$, $\{w_j^{(r)}\}_{r\in[R]}$)
invoke operation Input(init) on $\mathcal{F}_{\text{Beacon}}$
wait for $\mathcal{F}_{\text{Beacon}}$ to deliver a message Output($\{\gamma_\ell^{(r)}\}_{\ell\in[L],r\in[R]}$)
wait for $\mathcal{F}_{\text{ReliableBroadcast}}$ to deliver a message Output($\{h^{(r)}\}_{r\in[R]}$)
if for all $r \in [R]$:
  $h^{(r)} \in \mathbb{A}'[x]_{<d}$ with constant term in $\mathbb{S}'$ and
  $h^{(r)}(e'_j) = w_j^{(r)} + \sum_{\ell\in[L]} \gamma_\ell^{(r)} \cdot v_{\ell,j}$
then
  send a "vote" message to $P_1, \ldots, P_n$
  wait for "vote" messages from $n - t$ distinct parties
  output $\{v_{\ell,j}\}_{\ell\in[L]}$

Fig. 12. AVSS protocol for a dealer to provably enter values in $\phi^{-1}(\mathbb{S})$

6.4 Security analysis

The fact that Π_{ravss1} provides completeness follows from the completeness of its subprotocols. We shall show that assuming $p^{k-k'-1}$ is negligible, Π_{ravss1} securely realizes the ideal functionality $\mathcal{F}_{\text{ravss}}$ given in Fig. 13. To do this, we employ the following simple lemma.

Lemma 6.1. *Let p be a prime and let s be a positive integer. Let $a_1, \ldots, a_L$ be integers, not all zero mod p^s. Let r be the largest positive integer such that p^r divides a_ℓ for $\ell \in [L]$, so that $r < s$. Let b be an arbitrary integer. Let N be the number of integers $x_1, \ldots, x_L$ in the range $[0, p^s)$ that satisfy*

$$a_1 \cdot x_1 + \cdots a_L \cdot x_L + b \equiv 0 \pmod{p^s}. \tag{8}$$

Then $N/p^{L\cdot s} \le p^{r-s}$.

Proof. Without loss of generality, assume that $p^r \mid a_1$ but $p^{r+1} \nmid a_1$. We may assume $p^r \mid b$, as otherwise (8) has no solutions. In this case, (8) holds iff

$$(a_1/p^r) \cdot x_1 + \cdots (a_L/p^r) \cdot x_L + (b/p^r) \equiv 0 \pmod{p^{s-r}}.$$

Moreover, since a_1/p^r is not divisible by p, for every choice of $x_2, \ldots, x_L$, there is a unique choice of x_1 mod p^{s-r}, and so p^r choices for x_1 in the interval $[0, p^s)$. The lemma follows. □

This lemma says that if $x_1, \ldots, x_L$ are randomly chosen from the interval $[0, p^s)$, then the probability that (8) holds is at most p^{r-s}.

Theorem 6.1 (Security of Π_{ravss1}). *Assume $p^{(k-k'-1)\cdot R}$ is negligible. Then we have:*

$\mathcal{F}_{\text{ravss}}$

$\mathsf{Input}(f_1, \ldots, f_L)$: this operation is invoked once by the dealer D, who inputs polynomials $f_1, \ldots, f_L \in \mathbb{A}'[x]_{<d}$ $\mathbb{S}'$ to $\mathcal{F}_{\text{ravss}}$.
If D is honest, the constant terms are required to lie in $\mathbb{S}'$; however, if D corrupt, the constant terms are only required to lie in $\phi^{-1}(\mathbb{S})$.
In response, $\mathcal{F}_{\text{ravss}}$ sends the message $\mathsf{NotifyInput}()$ to the ideal-world adversary.

$\mathsf{RequestOutput}(j)$: after the input has been received, this operation may be invoked by the ideal-world adversary, who specifies $j \in [n]$. In response, $\mathcal{F}_{\text{ravss}}$ sends to P_j the message

$$\mathsf{Output}\Big(\{ f_\ell(e'_j) \}_{\ell \in [L]} \Big).$$

Fig. 13. The restricted AVSS Ideal Functionality (parameterized by n, d, L, R, D, p, k, k', F, G, $(E_1, \ldots, E_n)$, which define $\mathbb{S}$, $\mathbb{A}$, $\mathbb{S}'$, $\mathbb{A}'$, ϕ, and $e'_1, \ldots, e'_n$)

(i) *Π_{ravss1} securely realizes $\mathcal{F}_{\text{ravss}}$ in the $(\mathcal{F}_{\text{avss}}, \mathcal{F}_{\text{Beacon}}, \mathcal{F}_{\text{ReliableBroadcast}})$-hybrid model.*

(ii) *If Π_{ravss1} is instantiated with concrete protocols for $\mathcal{F}_{\text{avss}}$, $\mathcal{F}_{\text{Beacon}}$, and $\mathcal{F}_{\text{ReliableBroadcast}}$ that are secure (i.e., securely realize the corresponding functionality) and complete (i.e., satisfy the corresponding completeness property), then the resulting concrete protocol*
   (a) securely realizes $\mathcal{F}_{\text{ravss}}$, and
   (b) satisfies the AVSS completeness property.

Proof. We start with statement (i) of the theorem. To that end, we need to show that there is a simulator that interacts with $\mathcal{F}_{\text{ravss}}$ in the ideal world such that no environment can effectively distinguish the ideal world from the hybrid world.

If the dealer is honest, the proof reduces to showing that the values $h^{(r)}$ and $w_j^{(r)}$ for $j \in \mathcal{C}$ and $r \in [R]$ do not leak any extra information. This is a standard argument, based on the "random padding" supplied by the polynomials $g^{(r)}$. In more detail, the ideal functionality $\mathcal{F}_{\text{ravss}}$ gives the simulator the values $v_{\ell,j}$ for $\ell \in [L]$ and $j \in \mathcal{C}$. For each $r \in [r]$, the simulator then chooses $h^{(r)} \in \mathbb{A}'[x]_{<d}$ with constant term in $\mathbb{S}'$ at random and then computes

$$w_j^{(r)} \leftarrow h^{(r)}(e'_j) - \sum_{\ell \in [L]} \gamma_\ell^{(r)} \cdot v_{\ell,j} \quad (\text{for } j \in \mathcal{C}).$$

Note that the simulator can also generate the random beacon values $\gamma_\ell^{(r)}$ in advance of this computation.

Now consider case is that when the dealer is corrupt. The dealer must submit polynomials $\{f_\ell\}_{\ell \in [L]}$ and $\{g^{(r)}\}_{r \in [R]}$ of degree less than d to $\mathcal{F}_{\text{avss}}$ before the random beacon values $\gamma_\ell^{(r)}$ are revealed. Let s_ℓ denote the constant term of f_ℓ for $\ell \in [L]$ and let $s^{(r)}$ denote the constant term of $g^{(r)}$ for $r \in [R]$. If and when an honest party produces an output, the simulator needs to submit $\{f_\ell\}_{\ell \in [L]}$ to the ideal functionality $\mathcal{F}_{\text{ravss}}$, and so the simulation will fail if some s_ℓ is not in $\phi^{-1}(\mathbb{S})$. To bound the probability that this simulation fails, suppose that $s_{\ell^*} \notin \phi^{-1}(\mathbb{S})$ for some particular $\ell^* \in [L]$. To finish the proof of the theorem, it will suffice to show that for randomly chosen $\{\gamma_\ell^{(r)}\}_{\ell \in [L], r \in [R]}$ the probability that any honest party ever produces an output is at most $p^{(k-k'-1)\cdot R}$.

Suppose that some honest party produces an output. The honest party that produced an output received $n-t$ "vote" messages, which means that at least $n-2t \geq d$ honest parties performed their

local checks and these checks passed. This implies that each $h^{(r)}$ has the correct form (is of degree less than d with constant term in $\mathbb{S}'$), and that

$$h^{(r)} = g^{(r)} + \sum_{\ell \in [L]} \gamma_\ell^{(r)} \cdot f_\ell \quad \text{(for all } r \in [R]\text{)}. \tag{9}$$

This implies that

$$s^{(r)} + \sum_{\ell \in [L]} \gamma_\ell^{(r)} \cdot s_\ell \in \mathbb{S}' \quad \text{(for all } r \in [R]\text{)}. \tag{10}$$

So it suffices to show that (10) holds with probability at most $p^{(k-k'-1)\cdot R}$.

For $\ell \in [L]$, we can express s_ℓ uniquely as the image in $\mathbb{A}'$ of a polynomial in $\mathbb{Z}[y, z]$ of the form

$$\sum_{i=0}^{\delta-1} \sum_{j=0}^{\epsilon-1} a_{i,j,\ell} \cdot y^i \cdot z^j,$$

where each $a_{i,j,\ell}$ is an integer in the range $[0, p^{k'})$. The assumption that $s_{\ell^*} \notin \phi^{-1}(\mathbb{S})$ means that for some $i^* \geq 1$ and $j^* \geq 0$, we have $a_{i^*,j^*,\ell^*} \not\equiv 0 \pmod{p^k}$. For $r \in [R]$, we can similar express $s^{(r)}$ uniquely as the image in $\mathbb{A}'$ of a polynomial in $\mathbb{Z}[y, z]$ of the form

$$\sum_{i=0}^{\delta-1} \sum_{j=0}^{\epsilon-1} b_{i,j} \cdot y^i \cdot z^j,$$

For $\ell \in [L]$ and $r \in [R]$, we can view each $\gamma_\ell^{(r)}$ as the image in $\mathbb{Z}/(p^{k'})$ of a randomly chosen integer $x_\ell^{(r)}$ in the range $[0, p^{k'})$. But then by (10), we must have

$$b_{i^*,j^*} + \sum_{\ell \in [L]} a_{i^*,j^*,\ell} \cdot x_\ell^{(r)} \equiv 0 \pmod{p^{k'}} \quad \text{(for all } r \in [R]\text{)}. \tag{11}$$

But by Lemma 6.1, the congruence (11) holds with probability at most $p^{(k-k'-1)\cdot R}$.

That proves statement (i) of the theorem. Statement (ii)(a) is a direct consequence of statement (i) and the UC composition theorem. Statement (ii)(b) follows fairly easily from the security and completeness properties of the concrete subprotocols, and the logic of Π_{ravss1}. The argument in the case where the dealer is honest is entirely straightforward. The argument in the case where the dealer is corrupt hinges on the following observation. In the hybrid version of Π_{ravvs1}, as we argued above, if one honest party produces an output, then at least $n - 2t \geq d$ honest parties must have performed their local checks and these checks passed, and this implies that each $h^{(r)}$ has the correct form and (9) must hold. This means that when *any* honest party performs its local check, that check will pass and it will broadcast a "vote" message. Since this property always holds in the the hybrid version of Π_{ravvs1}, it must hold with overwhelming probability in the concrete version of Π_{ravvs1} as well. Therefore, in the concrete version of Π_{ravvs1}, if one honest party produces an output, then with overwhelming probability, all parties will eventually do so. □

After the proof of Theorem 5.1, we remarked that in some instantiations, we cannot justify the use of a random beacon that generates a short seed that is then stretched using a pseudorandom generator (as discussed in Section 3.1.2). That limitation does not apply here. Indeed, as discussed

in Section 3.1.2, to do this, it suffices that the failure event in Theorem 6.1 can be efficiently detected as a function of the output of G and data that is available to the adversary prior to revealing the beacon. The reader can verify that this is the case.

Note that the completeness argument at the end of the above proof would have been simpler if instead of the trivial one-round voting protocol, we used the one-sided voting protocol $\Pi_{\text{OneSidedVote}}$, which has two-rounds. Note also that if we instantiate $\mathcal{F}_{\text{avss}}$ with our protocol Π_{avss1}, which already performs a one-sided vote, the resulting protocol could be optimized by combining these two voting steps into just a single one-sided vote.

6.5 Communication complexity

We calculate the communication complexity of this protocol. We assume the AVSS protocol in Section 5 protocol is used for sharing over $\mathbb{A}'$ and we consider the amortized complexity on the "happy path" — but we could use any AVSS protocol that achieves linear amortized communication complexity of the "happy path". In this setting, the communication complexity (amortized, "happy path") is

$$O(n \cdot \log|\mathbb{A}'|).$$

The settings of the parameters R and k' affect both the communication complexity and the failure bound.

6.5.1 Setting $k' := k$. At one extreme, we could set $k' := k$. In this case, to achieve σ bits of security, we need to set the repetition parameter $R := \lceil \sigma \cdot \log_p 2 \rceil$. The main advantage of this parameter setting is that $\mathbb{A}' = \mathbb{A}$, and the amortized communication complexity remains the same as in Section 5.3. The main disadvantage of this setting is that the amortized computational complexity blows up by a factor of R.

6.5.2 Setting $R := 1$. At another extreme, we could set $R := 1$. In this case, to achieve σ bits of security, we need to set $k' := k - 1 + \lceil \sigma \cdot \log_p 2 \rceil$. Assuming $\mathbb{S}$ has degree ϵ over $\mathbb{Z}/(p^k)$ and $\mathbb{A}$ has degree δ over $\mathbb{S}$ the complexity (amortized, "happy path") is

$$O(n \cdot \delta \cdot (\log|\mathbb{S}| + \epsilon \cdot \sigma)).$$

Finite Field Case. Suppose $\mathbb{A}$ is a field extension of degree δ over the finite field $\mathbb{S} = \mathbb{F}_q$, where $q = p^\epsilon$, $\delta = O(\log_q n)$, and $k = 1$. Then the communication complexity (amortized, "happy path") is

$$O(n \cdot \log_q n \cdot (\log|\mathbb{S}| + \epsilon \cdot \sigma)),$$

which is

$$O(n \cdot \log n \cdot (1 + \sigma / \log p)).$$

Galois Ring Case. Suppose $\mathbb{S} = \mathbb{Z}/(p^k)$ and $\mathbb{A}$ is of degree $\delta = O(\log_p n)$ over $\mathbb{S}$, so $\epsilon = 1$. Then the communication complexity (amortized, "happy path") is

$$O(n \cdot \log_p n \cdot (\log|\mathbb{S}| + \sigma)),$$

which is

$$O(n \cdot \log n \cdot (k + \sigma / \log p)).$$

As a special case, suppose $p = 2$ and k is large enough so that 2^{-k} is negligible. Then by setting $k' := 2k$, we obtain $k+1$ bits of security, while both the amortized communication and computational complexity increase by just a small constant factor over the basic AVSS protocol.

7 Random oracle implementations

In this section, we present a variant of our new AVSS protocol Π_{avss1} (from Section 5), as well a variant of the AVSS protocol Π_{ravss1} (from Section 6, which restricts the dealer's secrets). These variants do not need a random beacon; rather, they require that we model a hash function as a random oracle in the security analysis. Besides eliminating the need for a random beacon subprotocol, these variants require significantly fewer rounds of communication.

7.1 A random oracle version of Π_{avss1}

At a very high level, protocol Π_{avss1} in Section 5 is based on the classical commit-challenge-response paradigm, which we can view as an interactive game between the dealer and a challenger:

1. The dealer sends a vector of messages $\mathbf{m} = (m_1, \ldots, m_n)$ to the challenger.
2. The challenger generates a random challenge $\omega \in \Omega$ and sends this to the dealer.
3. The dealer responds with a message m'.

In Π_{avss1}:

- the messages $m_1, \ldots, m_n$ encode the shares of the dealer's input polynomials, which are distributed using a secure message distribution subprotocol,
- the challenge is generated using a distributed random beacon subprotocol, and
- the response m' encodes the dealer's "response" polynomials, which are broadcast using a reliable broadcast subprotocol.

The main idea is to replace the distributed random beacon subprotocol by a hash function that is modeled as a random oracle. This way, the dealer can interact exclusively with the random oracle to generate a 3-move "conversation", consisting of $(\mathbf{m}, \omega, m')$, and then disseminate this conversation to all parties in a way that satisfies the privacy, correctness, and completeness properties of secure message distribution and reliable broadcast, and that includes support for the "forwarding mechanism" of secure message distribution.

7.1.1 A subprotocol for disseminating 3-move conversations. We show how to carry out this idea by sketching a protocol we call Π_{Dst3move}.

The dealer will first build a Merkle tree as shown in Fig. 14. It uses the same Merkle tree structure, rooted at r, used in our secure message distribution protocol $\Pi_{\text{SecMsgDst}}$ in Section 4.3 to encrypt and encode $\mathbf{m} = (m_1, \ldots, m_n)$, and the same Merkle tree structure, rooted at r', used in the compact reliable broadcast protocol $\Pi_{\text{CompactBroadcast}}$ in Section 3.2.2 to encode m'. In addition, the dealer creates a new Merkle tree with root $\hat{r}'$ whose children are r and r'. We assume that the hash that outputs the root r also outputs the challenge ω.

We assume that all hash functions queries are properly domain separated, meaning that we always include an input to the hash that identifies the individual instance of the protocol, including the identity of the dealer. We also assume that for a given protocol instance, different uses of the hash function are further domain separated. In particular, we assume that the hashes used to

– derive (r, ω),
– derive $\hat{r}'$,
– compute H in $\Pi_{\text{SecMsgDst}}$, and
– create all other nodes of Merkle trees,

are domain separated from each other. This domain separation greatly simplifies the analysis in the random oracle model.

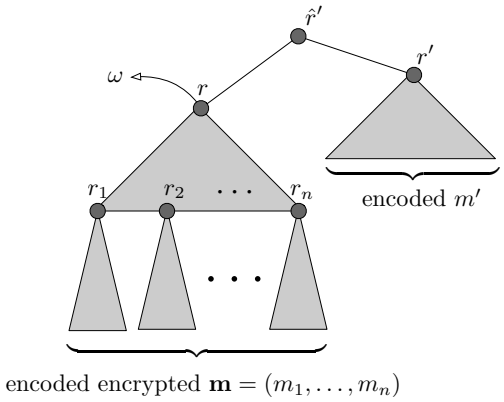

Fig. 14. Merkle tree structure for a 3-move conversation

Next, the dealer initiates a subprotocol to distribute each m_j to each P_j as in $\Pi_{\text{SecMsgDst}}$ and broadcast m' to all parties as in $\Pi_{\text{CompactBroadcast}}$. This subprotocol has the same *send-echo-vote* structure as in each of these protocols. However, in this subprotocol, all *echoing/voting* is done on the "global" root $\hat{r}'$ of the entire structure, so that just one round of *send-echo-vote* suffices to distribute the m_j's and to broadcast m'. In particular, the message *sent* by the dealer to a party P_j will include $\hat{r}'$, r, and r', as well as

– r_i and the jth validated path in the Merkle tree rooted at r_i for all $i \in [n]$, and
– the jth validated path in the Merkle tree rooted at r'.

In response, each party sends *echo* messages, where each *echo* message contains the root $\hat{r}'$, along with

– the validated path starting at r it would normally send in $\Pi_{\text{SecMsgDst}}$, and
– the validated path starting at r' it would normally send in $\Pi_{\text{CompactBroadcast}}$.

The forwarding mechanism is implemented in precisely the same way as in $\Pi_{\text{SecMsgDst}}$.

Note that in this construction, the dealer explicitly includes the hash values r and $\hat{r}'$ in the *send* messages, and the receiver verifies that these are consistent with $r_1, \ldots, r_n$ and r', rather than just computing r and $\hat{r}'$. This simplifies the analysis in the random oracle model, as we can assume that when the dealer is corrupt, it must have already invoked the random oracle to obtain r and $\hat{r}'$ explicitly before *sending* to an honest party (as otherwise that party will almost surely ignore the message).

In order to analyze the security of Π_{Dst3move}, we rely on the fact that for a corrupt dealer, we have the following *extractability* property: by observing the random oracle queries of the adversary,

a simulator can effectively determine the committed input $\mathbf{m} = (m_1, \ldots, m_n)$ before the random challenge ω is generated — note that some of the m_j's may be $\perp$ if the adversary has not made the necessary random oracle queries. We also rely on the fact that for an honest dealer, we have the following *equivocability* property: a simulator is free to program the random oracle (the one that outputs the random challenge) so that it outputs a given random challenge ω. To justify this, we have to ensure that the input to this hash has high entropy and so that adversary is unlikely to have queried the random oracle previously at this point; however, this will be the case, since in the protocol, the input to this hash query (the one that outputs (r, ω)) is ultimately derived from the outputs of the H hash function in $\Pi_{\text{SecMsgDst}}$, and these outputs are essentially random.

We also observe that an adversary can carry out a "grinding" attack. Indeed, an adversary can effectively:

- try many different message vectors, where for each such message vector $\mathbf{m}$, it builds a Merkle tree and derives the corresponding challenge ω, creating a partial conversation $(\mathbf{m}, \omega)$,
- pick one such partial conversation $(\mathbf{m}, \omega)$ that it likes,
- extend $(\mathbf{m}, \omega)$ with a message m' of its choice to obtain a full conversation $(\mathbf{m}, \omega, m')$, and finally
- disseminate the full conversation $(\mathbf{m}, \omega, m')$ to all parties.

We can characterize the security of protocol Π_{Dst3move} in terms of an ideal functionality $\mathcal{F}_{\text{Dst3move}}$. To this functionality, the dealer submits a message vector $\mathbf{m} = (m_1, \ldots, m_n)$ and obtains a random challenge ω, creating the partial conversation $(\mathbf{m}, \omega)$. After this, the dealer submits a message m', creating the full conversation $(\mathbf{m}, \omega, m')$. This full conversation is then disseminated to all parties following the semantics of $\mathcal{F}_{\text{SecMsgDst}}$ and $\mathcal{F}_{\text{ReliableBroadcast}}$ — in particular, for an honest dealer, the only information leaked to the ideal-world adversary is the challenge ω, the final message m', and (implicitly) the messages m_j belonging to corrupt parties P_j.

To model the above "grinding" attack, the functionality also allows a corrupt dealer to repeatedly submit a message vector $\mathbf{m}$, obtaining a challenge ω and creating the partial conversation $(\mathbf{m}, \omega)$. We call one such operation a **grind on** $\mathcal{F}_{\text{Dst5move}}$. At some point, the corrupt dealer may choose a partial extended conversation $(\mathbf{m}, \omega)$ and submit a message m', creating the full conversation $(\mathbf{m}, \omega, m')$, which is then disseminated to all parties following the semantics of $\mathcal{F}_{\text{SecMsgDst}}$ and $\mathcal{F}_{\text{ReliableBroadcast}}$.

We leave it to the reader to verify that Π_{Dst3move} securely realizes the ideal functionality $\mathcal{F}_{\text{Dst3move}}$. Moreover, in the security reduction, the number of grinds on $\mathcal{F}_{\text{Dst3move}}$ made by the simulator is bounded by the number of random oracle queries made by the real-world adversary.

We also leave it to the reader to verify that Π_{Dst3move} enjoys completeness properties analogous to those of $\Pi_{\text{SecMsgDst}}$ and $\Pi_{\text{ReliableBroadcast}}$.

7.1.2 An AVSS protocol. To build an AVSS protocol using Π_{Dst3move}, we use Π_{Dst3move} as a drop-in replacement in Π_{avss1} for $\mathcal{F}_{\text{SecMsgDst}}$, $\mathcal{F}_{\text{Beacon}}$, and $\mathcal{F}_{\text{ReliableBroadcast}}$. The only remaining steps of Π_{avss1} are local computations and $\mathcal{F}_{\text{OneSidedVote}}$. The result is a protocol we denote by Π_{avss2}.

We leave it to the reader to verify the following. Notation is as in Theorem 5.1.

Theorem 7.1 (Security of Π_{avss2}). *Assume $Q \cdot 2^n \cdot \chi(\mathbb{A}, \mathbb{B}, \Theta)^R$ is negligible, where Q is the number of grinds on $\mathcal{F}_{\text{Dst3move}}$ Then we have:*

(i) Π_{avss2} *securely realizes* $\mathcal{F}_{\text{avss}}$ *in the* $(\mathcal{F}_{\text{Dst3move}}, \mathcal{F}_{\text{OneSidedVote}})$*-hybrid model.*

(ii) If Π_{avss2} *is instantiated with concrete protocols for* $\mathcal{F}_{\text{Dst3move}}$ *and* $\mathcal{F}_{\text{OneSidedVote}}$*, that are secure (i.e., securely realize the corresponding functionality) and complete (i.e., satisfy the corresponding completeness property), then the resulting concrete protocol*

(a) securely realizes $\mathcal{F}_{\text{avss}}$*, and*

(b) satisfies the AVSS completeness property.

The communication complexity of Π_{avss2} is essentially the same as that of Π_{avss1}. However, in terms of rounds of communication, Π_{avss2} is significantly better. Indeed (on the happy path), if we instantiate Π_{avss2} with Π_{Dst3Move} and $\Pi_{\text{OneSidedVote}}$, we see that it needs 3 rounds for Π_{Dst3move} and 2 rounds for $\Pi_{\text{OneSidedVote}}$, for a total of just 5 rounds. In contrast, if we instantiate Π_{avss1} with $\Pi_{\text{SecMsgDst}}$, $\Pi_{\text{CompactBroadcast}}$, $\Pi_{\text{OneSidedVote}}$, and any 1-round protocol that realizes $\mathcal{F}_{\text{Beacon}}$, we that it needs 3 rounds for $\Pi_{\text{SecMsgDst}}$, 1 round for the beacon, 3 rounds for $\Pi_{\text{CompactBroadcast}}$, and 2 rounds for $\Pi_{\text{OneSidedVote}}$, for a total of 9 rounds.

We speculate that Π_{avss2} can be further optimized by folding the one-sided voting subprotocol into Π_{Dst3Move} in such a way that the total number of rounds is 4 rather than 5; however, we have not carried out the associated full design and analysis.

We also speculate that Π_{avss2} can be implemented in such a way that it is secure against adaptive corruptions in the random oracle model. To achieve this, it should be sufficient to use an *equivocable* symmetric encryption scheme as discussed in Section 4.3. However, we have not carried out the associated full design and analysis.

7.2 A random oracle version of Π_{ravss1}

Let us assume that we instantiate $\mathcal{F}_{\text{avss}}$ in protocol Π_{ravss1} with Π_{avss1}. The resulting protocol then has the following structure of a 5-move interactive game between the dealer and a challenger:

1. The dealer sends a vector of messages $\mathbf{m} = (m_1, \ldots, m_n)$ to the challenger.
2. The challenger generates a random challenge $\omega \in \Omega$ and sends this to the dealer.
3. The dealer sends a message m' to the challenger.
4. The challenger generates a random challenge $\omega' \in \Omega'$ and sends this to the dealer.
5. The dealer responds with a message m''.

Here, moves 1–3 are as in Π_{avss1}, while moves 3 and 4 correspond to the challenge-response phase of Π_{ravss1}. One might hope to reduce these 5 moves to just 3 moves, but we do not see a way to do that. Indeed, as remarked in Section 5.1.1, a corrupt dealer only commits to its input polynomials after move 3, and in order for the security proof of Theorem 6.1 to go through, we cannot afford to reveal the challenge ω' until that occurs.

As above in Section 7.1, the main idea is to replace the distributed random beacon subprotocols by hash functions that are modeled as random oracles — the dealer can interact exclusively with the random oracle to generate a 5-move "conversation", consisting of $(\mathbf{m}, \omega, m', \omega'', m'')$, and then disseminate this conversation to all parties. We sketch here a protocol Π_{Dst5move} that does exactly this.

The dealer will first build a Merkle tree as shown in Fig. 15. The structure is the same as in Fig. 14, except that we have added a Merkle subtree, rooted at r'', with the same structure as the compact reliable broadcast protocol $\Pi_{\text{CompactBroadcast}}$ in Section 3.2.2 to encode m'', and a "global" root $\hat{r}''$ with $\hat{r}'$ and r'' as children. We assume that the hash that outputs the root $\hat{r}'$ also

outputs the challenge ω'. We also assume all hashes are properly domain separated as discussed above Section 7.1, and in particular, the hashes used to derive (r, ω), $(\hat{r}', \omega')$, and $\hat{r}''$ are all domain separated from each other as well as from other use cases.

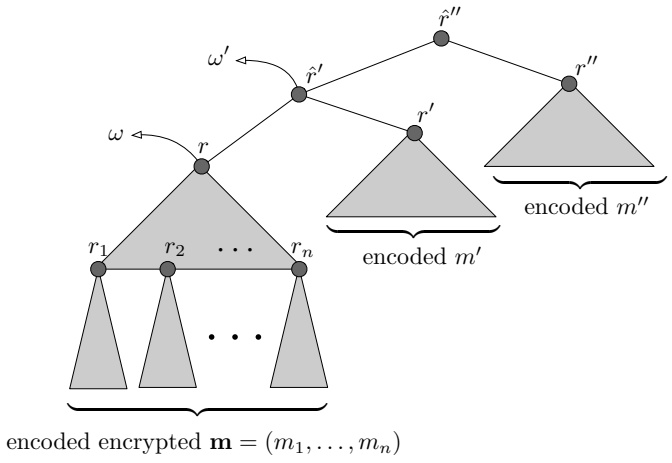

Fig. 15. Merkle tree structure for 5-move conversation

Next, the dealer initiates a subprotocol to distribute each m_j to each P_j as in $\Pi_{\text{SecMsgDst}}$, broadcast m' to all parties as in $\Pi_{\text{CompactBroadcast}}$, and broadcast m'' to all parties as in $\Pi_{\text{CompactBroadcast}}$. As above in Section 7.1, this subprotocol has a *send-echo-vote* structure, and all *echoing/voting* is done on the "global" root $\hat{r}''$, so that just one round of *send-echo-vote* suffices to distribute the m_j's and to broadcast m', m''. Analogously to what we did in Section 7.1, the dealer explicitly includes the hash values r, $\hat{r}'$, and $\hat{r}''$ in the *send* messages

We can characterize the security of protocol Π_{Dst5move} in terms of an ideal functionality $\mathcal{F}_{\text{Dst5move}}$. To this functionality, the dealer submits a message vector $\mathbf{m} = (m_1, \ldots, m_n)$ and obtains a random challenge ω, creating the partial conversation $(\mathbf{m}, \omega)$. After this, the dealer submits a message m' and obtains a random challenge ω', creating the extended conversation $(\mathbf{m}, \omega, m', \omega')$. Finally, the dealer submits m'', creating the full conversation $(\mathbf{m}, \omega, m', \omega', m'')$, which is then disseminated to all parties following the semantics of $\mathcal{F}_{\text{SecMsgDst}}$ and $\mathcal{F}_{\text{ReliableBroadcast}}$.

To model a "grinding" attack, the functionality also allows a corrupt dealer to repeatedly submit a message vector $\mathbf{m}$, obtaining a challenge ω and creating the partial conversation $(\mathbf{m}, \omega)$. The corrupt dealer may also repeatedly choose any such partial $(\mathbf{m}, \omega)$ conversation, submit a message m', obtaining a challenge ω' and creating the extended conversation $(\mathbf{m}, \omega, m', \omega')$. We call either one of the above operations a **grind on** $\mathcal{F}_{\text{Dst5move}}$. At some point, the corrupt dealer may choose an extended conversation $(\mathbf{m}, \omega, m', \omega')$ and submit a message m'', creating the full conversation $(\mathbf{m}, \omega, m', \omega', m'')$, which is then disseminated to all parties following the semantics of $\mathcal{F}_{\text{SecMsgDst}}$ and $\mathcal{F}_{\text{ReliableBroadcast}}$.

We leave it to the reader to verify that Π_{Dst5move} securely realizes the ideal functionality $\mathcal{F}_{\text{Dst5move}}$. Moreover, in the security reduction, the number of grinds on $\mathcal{F}_{\text{Dst5move}}$ made by the simulator is bounded by the number of random oracle queries made by the real-world adversary.

We also leave it to the reader to verify that Π_{Dst5move} enjoys completeness properties analogous to those of $\Pi_{\text{SecMsgDst}}$ and $\Pi_{\text{ReliableBroadcast}}$.

We now sketch how to build a variant of Π_{ravss1} protocol using Π_{Dst5move}. First, recall that we are assuming here that we instantiate $\mathcal{F}_{\text{avss}}$ in Π_{ravss1} with Π_{avss1}. Next, observe that both Π_{ravss1} and the subprotocol Π_{avss1} use a voting step to express whether they are happy with the information they have received. As noted after Theorem 6.1, we can safely combine these two voting steps into a single one-sided vote. That is, each party P_j receives $(m_j, \omega, m', \omega', m'')$, and checks if (m_j, ω, m') is valid according to Π_{avss1}, and checks if $(\bar{m}_j, \omega', m'')$ is valid according to Π_{ravss1} (here, $\bar{m}_j$ is the same as m_j, but with shares of blinding polynomials specific to Π_{avss1} filtered out). A single one-sided vote (i.e., $\Pi_{\text{OneSidedVote}}$) can be used to attest that both of these checks pass. We then use Π_{Dst5move} to replace all steps other than local computations and this one-sided voting step. The result is a protocol we denote by Π_{ravss2}.

We leave it to the reader to verify the following. Notation is as in Theorems 5.1 and 7.1, but where R is the repetition parameter for Π_{avss1} and R' is the repetition parameter for Π_{ravss1}:

Theorem 7.2 (Security of Π_{ravss2}). *Assume $Q \cdot \left(2^n \cdot \chi(\mathbb{A}, \mathbb{B}, \Theta)^R + p^{(k-k'-1)\cdot R'}\right)$ is negligible, where Q is the number of grinds on $\mathcal{F}_{\text{Dst5move}}$ Then we have:*

(i) Π_{ravss2} securely realizes $\mathcal{F}_{\text{ravss}}$ in the $(\mathcal{F}_{\text{Dst5move}}, \mathcal{F}_{\text{OneSidedVote}})$-hybrid model.

(ii) If Π_{ravss2} is instantiated with concrete protocols for $\mathcal{F}_{\text{Dst5move}}$ and $\mathcal{F}_{\text{OneSidedVote}}$, that are secure (i.e., securely realize the corresponding functionality) and complete (i.e., satisfy the corresponding completeness property), then the resulting concrete protocol

(a) securely realizes $\mathcal{F}_{\text{ravss}}$, and

(b) satisfies the AVSS completeness property.

The communication complexity of Π_{ravss2} is essentially the same as that of Π_{ravss1}. However, in terms of rounds of communication, Π_{ravss2} is significantly better. Indeed (on the happy path), it needs just 5 rounds of communication (the same as Π_{avss1}). In contrast, if we instantiate Π_{ravss1} with $\Pi_{\text{SecMsgDst}}$, $\Pi_{\text{CompactBroadcast}}$, $\Pi_{\text{OneSidedVote}}$, and any 1-round protocol that realizes $\mathcal{F}_{\text{Beacon}}$, we that it needs 3 rounds for $\Pi_{\text{SecMsgDst}}$ in Π_{avss1}, 1 round for the beacon in Π_{avss1}, 3 rounds for the $\Pi_{\text{CompactBroadcast}}$ in Π_{avss1}, 2 rounds for $\Pi_{\text{OneSidedVote}}$ in Π_{avss1}, 1 round for the beacon in Π_{ravss1}, 3 rounds for the $\Pi_{\text{CompactBroadcast}}$ in Π_{ravss1}, 1 round for the simplified one-sided vote in Π_{ravss1}, for a total of 14 rounds.

Just as for Π_{avss2}, we speculate that Π_{ravss2} can be further optimized so that the total number of rounds is 4 rather than 5. In addition, we also speculate that Π_{ravss2} can be implemented in such a way that it is secure against adaptive corruptions in the random oracle model.

Acknowledgements

The work of the first author was partially done while he was employed at DFINITY. The work of the second author was supported by CyberSecurity Research Flanders with reference number VR20192203, by the FWO under an Odysseus project GOH9718N. The second author would like to thank Jesper Buus Nielsen and Robin Jadoul for some discussions on various related topics whilst the work in this paper was being carried out.

References

ACD+19. M. Abspoel, R. Cramer, I. Damgård, D. Escudero, and C. Yuan. Efficient information-theoretic secure multiparty computation over $\mathbb{Z}/p^k\mathbb{Z}$ via galois rings. In D. Hofheinz and A. Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 471–501, Nuremberg, Germany, Dec. 1–5, 2019. Springer, Heidelberg, Germany.

ACD+20. M. Abspoel, R. Cramer, I. Damgård, D. Escudero, M. Rambaud, C. Xing, and C. Yuan. Asymptotically good multiplicative LSSS over Galois rings and applications to MPC over $\mathbb{Z}/p^k\mathbb{Z}$. In S. Moriai and H. Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 151–180, Daejeon, South Korea, Dec. 7–11, 2020. Springer, Heidelberg, Germany.

AJM+23. I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, and G. Stern. Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In H. Handschuh and A. Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 39–70, Santa Barbara, CA, USA, Aug. 20–24, 2023. Springer, Heidelberg, Germany.

BBB+23. A. Bandarupalli, A. Bhat, S. Bagchi, A. Kate, and M. Reiter. HashRand: Efficient asynchronous random beacon without threshold cryptographic setup. Cryptology ePrint Archive, Paper 2023/1755, 2023. https://eprint.iacr.org/2023/1755.

BCG93. M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In *25th Annual ACM Symposium on Theory of Computing*, pages 52–61, San Diego, CA, USA, May 16–18, 1993. ACM Press.

BGR98. M. Bellare, J. A. Garay, and T. Rabin. Batch verification with applications to cryptography and checking. In C. L. Lucchesi and A. V. Moura, editors, *LATIN 1998: Theoretical Informatics, 3rd Latin American Symposium*, volume 1380 of *Lecture Notes in Computer Science*, pages 170–191, Campinas, Brazil, Apr. 20–24, 1998. Springer, Heidelberg, Germany.

BLS01. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532, Gold Coast, Australia, Dec. 9–13, 2001. Springer, Heidelberg, Germany.

BLW08. D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In S. Jajodia and J. López, editors, *ESORICS 2008: 13th European Symposium on Research in Computer Security*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206, Málaga, Spain, Oct. 6–8, 2008. Springer, Heidelberg, Germany.

Bol03. A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Y. Desmedt, editor, *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46, Miami, FL, USA, Jan. 6–8, 2003. Springer, Heidelberg, Germany.

BR93. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, Nov. 3–5, 1993. ACM Press.

Bra87. G. Bracha. Asynchronous Byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, 1987.

BTH06. Z. Beerliová-Trubíniová and M. Hirt. Efficient multi-party computation with dispute control. In S. Halevi and T. Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 305–328, New York, NY, USA, Mar. 4–7, 2006. Springer, Heidelberg, Germany.

BTH08. Z. Beerliová-Trubíniová and M. Hirt. Perfectly-secure MPC with linear communication complexity. In R. Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 213–230, San Francisco, CA, USA, Mar. 19–21, 2008. Springer, Heidelberg, Germany.

Can00. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. https://eprint.iacr.org/2000/067.

CDE+18. R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing. SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 769–798, Santa Barbara, CA, USA, Aug. 19–23, 2018. Springer, Heidelberg, Germany.

CKL21. J. H. Cheon, D. Kim, and K. Lee. MHz2k: MPC from HE over $\mathbb{Z}_{2^k}$ with new packing, simpler reshare, and better ZKP. In T. Malkin and C. Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 426–456, Virtual Event, Aug. 16–20, 2021. Springer, Heidelberg, Germany.

CKPS01. C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. Cryptology ePrint Archive, Report 2001/006, 2001. https://eprint.iacr.org/2001/006.

Coh16. R. Cohen. Asynchronous secure multiparty computation in constant time. In C.-M. Cheng, K.-M. Chung, G. Persiano, and B.-Y. Yang, editors, *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 9615 of *Lecture Notes in Computer Science*, pages 183–207, Taipei, Taiwan, Mar. 6–9, 2016. Springer, Heidelberg, Germany.

CP17. A. Choudhury and A. Patra. An efficient framework for unconditionally secure multiparty computation. *IEEE Trans. Inf. Theory*, 63(1):428–468, 2017.

CP23. A. Choudhury and A. Patra. On the communication efficiency of statistically-secure asynchronous MPC with optimal resilience. *Journal of Cryptology*, 36:13, 2023.

CT05. C. Cachin and S. Tessaro. Asynchronous verifiable information dispersal. In P. Fraigniaud, editor, *Distributed Computing, 19th International Conference, DISC 2005, Cracow, Poland, September 26-29, 2005, Proceedings*, volume 3724 of *Lecture Notes in Computer Science*, pages 503–504. Springer, 2005.

DN07. I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In A. Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590, Santa Barbara, CA, USA, Aug. 19–23, 2007. Springer, Heidelberg, Germany.

DW20. S. Dolev and Z. Wang. SodsBC: Stream of distributed secrets for quantum-safe blockchain. Cryptology ePrint Archive, Report 2020/205, 2020. https://eprint.iacr.org/2020/205.

DWZ23. S. Duan, X. Wang, and H. Zhang. Practical signature-free asynchronous common subset in constant time. Cryptology ePrint Archive, Report 2023/154, 2023. https://eprint.iacr.org/2023/154.

DXR21. S. Das, Z. Xiang, and L. Ren. Asynchronous data dissemination and its applications. In G. Vigna and E. Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 2705–2721, Virtual Event, Republic of Korea, Nov. 15–19, 2021. ACM Press.

DXR22. S. Das, Z. Xiang, and L. Ren. Balanced quadratic reliable broadcast and improved asynchronous verifiable information dispersal. Cryptology ePrint Archive, Report 2022/052, 2022. https://eprint.iacr.org/2022/052.

EXY22. D. Escudero, C. Xing, and C. Yuan. More efficient dishonest majority secure computation over $\mathbb{Z}_{2^k}$ via galois rings. In Y. Dodis and T. Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 383–412, Santa Barbara, CA, USA, Aug. 15–18, 2022. Springer, Heidelberg, Germany.

Feh98. S. Fehr. Span programs over rings and how to share a secret from a module, 1998. MSc Thesis, ETH Zurich.

FY92. M. K. Franklin and M. Yung. Communication complexity of secure computation (extended abstract). In *24th Annual ACM Symposium on Theory of Computing*, pages 699–710, Victoria, BC, Canada, May 4–6, 1992. ACM Press.

GS22. J. Groth and V. Shoup. Design and analysis of a distributed ECDSA signing service. Cryptology ePrint Archive, Report 2022/506, 2022. https://eprint.iacr.org/2022/506.

HN06. M. Hirt and J. B. Nielsen. Robust multiparty computation with linear communication complexity. In C. Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 463–482, Santa Barbara, CA, USA, Aug. 20–24, 2006. Springer, Heidelberg, Germany.

HNP08. M. Hirt, J. B. Nielsen, and B. Przydatek. Asynchronous multi-party computation with quadratic communication. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *ICALP 2008: 35th International Colloquium on Automata, Languages and Programming, Part II*, volume 5126 of *Lecture Notes in Computer Science*, pages 473–485, Reykjavik, Iceland, July 7–11, 2008. Springer, Heidelberg, Germany.

HS15. D. Hofheinz and V. Shoup. GNUC: A new universal composability framework. *Journal of Cryptology*, 28(3):423–508, July 2015.

JSL22. R. Jadoul, N. P. Smart, and B. V. Leeuwen. MPC for Q_2 access structures over rings and fields. In R. AlTawy and A. Hülsing, editors, *SAC 2021: 28th Annual International Workshop on Selected Areas in Cryptography*, volume 13203 of *Lecture Notes in Computer Science*, pages 131–151, Virtual Event, Sept. 29 – Oct. 1, 2022. Springer, Heidelberg, Germany.

KMTZ13. J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Universally composable synchronous computation. In A. Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 477–498, Tokyo, Japan, Mar. 3–6, 2013. Springer, Heidelberg, Germany.

OSV20. E. Orsini, N. P. Smart, and F. Vercauteren. Overdrive2k: Efficient secure MPC over $\mathbb{Z}_{2^k}$ from somewhat homomorphic encryption. In S. Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, volume 12006 of

Lecture Notes in Computer Science, pages 254–283, San Francisco, CA, USA, Feb. 24–28, 2020. Springer, Heidelberg, Germany.

QBC13. G. Quintin, M. Barbier, and C. Chabot. On generalized Reed–Solomon codes over commutative and noncommutative rings. *IEEE Trans. Inf. Theory*, 59(9):5882–5897, 2013.

SJK+17. E. Syta, P. Jovanovic, E. Kokoris-Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy*, pages 444–460, San Jose, CA, USA, May 22–26, 2017. IEEE Computer Society Press.

YLF+21. T. Yurek, L. Luo, J. Fairoze, A. Kate, and A. Miller. hbACSS: How to robustly share many secrets. Cryptology ePrint Archive, Report 2021/159, 2021. https://eprint.iacr.org/2021/159.

YPA+21. L. Yang, S. J. Park, M. Alizadeh, S. Kannan, and D. Tse. DispersedLedger: High-throughput Byzantine consensus on variable bandwidth networks. *CoRR*, abs/2110.04371, 2021, 2110.04371. URL https://arxiv.org/abs/2110.04371.