Lightweight Asynchronous Verifiable Secret Sharing with Optimal Resilience

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

[1] DFINITY, Zurich, Switzerland.
[2] imec-COSIC, KU Leuven, Leuven, Belgium.
[3] Zama Inc., Paris, France.
victor.shoup@dfinity.org.
nigel.smart@kuleuven.be/nigel@zama.ai.

May 30, 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 honest parties initiate the protocol and 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 shares 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.[4] 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

[4] There is some inconsistency in the literature regarding this terminology. For example, what we call "batched" secret sharing is called "packed" secret sharing in [AJM+22].

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".

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 this distinction here. 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+22], 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] (see Section 4.3 for more details). In another protocol, we fully embrace the random oracle model, which allows for a simpler protocol (and one that we hypothesize is resistant to adaptive corruptions, rather than just static corruptions). 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 obviously 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+22] 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.[5] 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

[5] 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].

point is the underlying batched synchronous VSS protocol contained in [DN07]. At a high level, this protocol works in the following steps:

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 soundness. 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 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 develop in Section 6.

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 $n = 3 \cdot t + 1$, and we only

collect $n-t = 2\cdot t+1$ 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, and 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 get optimal resilience, but we also get 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.

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 $\{s_i\}_{i\in I}$ such that each $s_i \in \mathbb{A}$, and for all $i, j \in I$ with $i \neq j$, we have $s_i - s_j \in \mathbb{A}^*$. 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 if 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 also known as the *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 of our protocols are 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 deal polynomials $f_1, \ldots, f_L \in \mathbb{A}[x]_{<d}$ to $P_1, \ldots, P_n$ in such a way that the following conditions are satisfied:

Completeness: If one honest party outputs a value, then every honest party eventually outputs a value. Moreover, if D is honest, then every honest party eventually outputs a value.

Correctness: There exists polynomials $\hat{f}_1, \ldots, \hat{f}_L \in \mathbb{A}[x]_{<d}$, such that for any honest party P_j that outputs a value, that value is $\{\hat{f}_\ell(e_j)\}_{\ell=1}^L$. Moreover, if D is honest, then $\hat{f}_\ell = f_\ell$ for $\ell \in [L]$.

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.

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

Note that this ideal functionality does not capture at all the completeness property: this is a separate property, which means that if

- the dealer is honest or any honest party outputs a value,
- all honest parties have initiated the protocol, and
- all messages sent between honest parties have been delivered,

then all honest parties have output a value. One can show that like security, completeness is a composable property in the UC framework. That is, one can naturally define the notion of completeness for a hybrid protocol which makes use of ideal functionalities as subprotocols, and it is a simple exercise to show that if we replace these idealized subprotocols by complete, concrete protocols that securely realize these functionalities, the resulting protocol is a concrete protocol that is secure and complete. Completeness for such a hybrid protocol is the same as for a concrete protocol, except that in addition to the condition that all messages sent between honest parties have been delivered, we also add the condition that all RequestOutput messages for the ideal functionalities have been delivered — in fact, concrete protocols are really hybrid protocols with an appropriate secure messaging ideal functionality, and so this is actually the same condition.[6]

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

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 NotifyInput() to the ideal-world adversary.

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)

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

[6] In related work, [CP23] study AVSS and AMPC protocols in the UC framework, but make use of a formal notion of time [CP23] introduced in [KMTZ13]. We claim that this extra machinery is unnecessary.

this, the secret is essentially encodes 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 $n \geq d + 2 \cdot t$, 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 an 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 protocols 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 multinomial 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 multinomial 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.

3.1 Random Beacon

This is a protocol that reveals a value ω chosen at random from an **output space** Ω, in such a way that the following conditions are satisfied:

Completeness: If at least $t + 1$ honest parties initiate the protocol, every honest party eventually outputs a value.

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

Privacy: The value ω remains unpredictable at any time before at least one honest party initiates the protocol.

The random beacon is best defined in terms of an ideal functionality $\mathcal{F}_{\text{Beacon}}$, see Figure 2.

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

Input(): 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, 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)

As it will only be run a small number of times, a random beacon can be securely realized with a statistically secure protocol based on any AVSS protocol and any consensus protocol, using the standard technique of agreeing on a set of $t+1$ secret sharings, and then opening all of them and adding them up. This will not affect the amortized complexity of our batched AVSS protocol. That said, we will also give a simpler batched AVSS protocol that does not need a random beacon, but instead is analyzed in the random oracle model.

It is also possible, and perhaps more practical, to implement a random beacon using "heavyweight" cryptography. A standard way to do this is to use a $(t+1)$-out-of-n threshold BLS signature scheme [BLS01,Bol03,SJK+17]. Despite being based on "heavyweight" cryptography, this beacon is efficient enough for our purposes; however, using it requires assuming a hardness assumption which is not post-quantum secure.

3.2 Reliable broadcast

A **reliable broadcast** protocol allows a sender S to broadcast a single message m to $P_1, \ldots, P_n$ in such a way that the following conditions are satisfied:

Completeness: If one honest party outputs a message, then every honest party eventually outputs a message. Moreover, if S is honest, then every honest party eventually outputs a message.

Correctness: All honest parties that output a message output the same message. Moreover, if S is honest, that message is m.

We say that S **reliably broadcasts** m, and that a party **reliably receives** m. The correctness property can alternatively be better captured by an ideal functionality $\mathcal{F}_{\text{ReliableBroadcast}}$, given in Figure 3. Note that this ideal functionality does not capture at all the completeness property: this is a separate property. A simple protocol for this, called **Bracha Broadcast** [Bra87], is given in Figure 4.

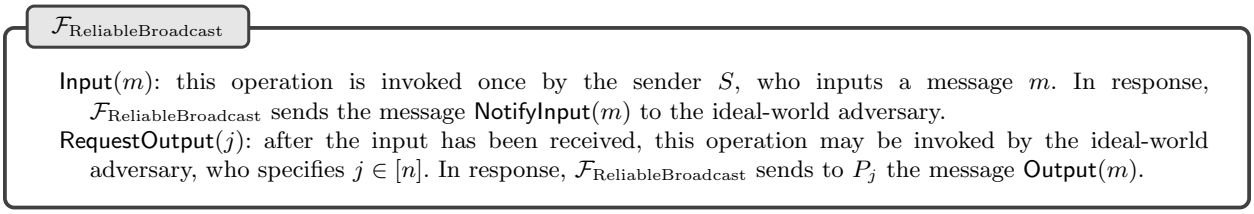

Fig. 3. The Reliable Broadcast Ideal Functionality (parameterized by S)

The communication complexity of this protocol is $O(n^2 \cdot |m|)$. However, protocols with better communication can be obtained by use of *erasure codes*. The basic idea of an erasure coded broadcast is as follows. Given a long message m, the sender S encodes the message using an $(n, n-2\cdot t)$-Reed-Solomon code, to obtain a vector of n fragments $(f_1, \ldots, f_n)$ from which m can be reconstructed from any subset of $n-2\cdot t$ fragments. Each fragment has size roughly $|m|/(n-2\cdot t)$ — so assuming $n > 3 \cdot t$, the size of each fragment is at most roughly $3 \cdot |m|/n$. 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 care must be

$\Pi_{\text{BrachaBroadcast}}$

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

// Receiving party P_j
acquired ← false
voted ← false

repeat forever:
    wait for either:
        not acquired and ∃m: received (send, m) from S:
            acquired ← true
            send (echo, m) to P_1, ..., P_n

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

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

        ∃m: received (vote, m) from n − t distinct parties:
            output m
            halt
```

Fig. 4. Bracha's Protocol for Reliable Broadcast

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. For somewhat shorter messages, a protocol such as that in [DXR21] may be used, which achieves a communication complexity of $O(n \cdot |m| + \lambda \cdot n^2)$. Note that while the protocol in [CT05] uses only the Reed-Solomon as an erasure code, the protocol in [DXR21] uses a the error correcting properties of Reed-Solomon to avoid the use of Merkle trees. One advantage of the scheme in [CT05] over that in [DXR21] is that the former has a more balanced communication pattern, which can be important to prevent bandwidth bottlenecks. The paper [DXR22] improves on [DXR21], obtaining the same communication complexity, but with a balanced communication pattern. Despite these improvements, [CT05] may still be preferable, especially when $|m| \gg \lambda \cdot n \cdot \log n$, as its computational complexity is somewhat lower.

3.3 Simple Asynchronous Agreement

A degenerate version of Bracha broadcast can be used as a simple asynchronous agreement primitive, see Figure 5. The key properties of this protocol are:

Completeness: If one honest party outputs done, then every honest party eventually outputs done. Moreover, if all honest parties initiate the protocol with input true, then every honest party eventually outputs done.

Correctness: If any honest party outputs done, then at least $n - 2 \cdot t$ honest parties initiated the protocol with input true.

The correctness property can alternatively be captured by an ideal functionality $\mathcal{F}_{\text{AsyncAgreement}}$, given in Figure 6. Note that this ideal functionality does not capture at all the completeness property: this is a separate property.

3.4 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 m_j is delivered to P_j. After this, party P_j may then choose to authentically forward its message m_j to other parties. This last property will be needed to deal with the "unhappy" path of our AVSS protocol. We also require that if the sender is honest, then the adversary learns nothing about message m_j unless P_j is corrupt or authentically forwards m_j to a corrupt party.

We can formulate the required properties more precisely as follows.

Distribution completeness: If one honest party securely receives a distributed message, then every honest party eventually does so. Moreover, if S is honest, then every honest party eventually securely receives a distributed message.

Distribution correctness: If S is honest and securely distributes $\mathbf{m} = (m_1, \ldots, m_n)$, then when an honest party P_j securely receives a distributed message, that message is m_j.

Forward completeness: If an honest party P_j authentically forwards its message to an honest party P_i, then eventually P_i authentically receives a forwarded message from P_j.

Forward correctness: All honest parties that authentically receive a forwarded message from some party P_j receive the same message. Moreover:

$\Pi_{\text{AsyncAgreement}}$

Each party P_j takes as input a boolean value $v_j \in \{\mathsf{true}, \mathsf{false}\}$ and runs as follows.

voted $\leftarrow$ false
if v_j then send (ok) to $P_1, \ldots, P_n$

repeat forever:
  wait for either:
    not *voted* and received (ok) from $n - t$ distinct parties:
      send (vote) to $P_1, \ldots, P_n$
      voted $\leftarrow$ *true*

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

    received (vote) from $n - t$ distinct parties:
      output done
      halt

Fig. 5. Degenerate Version of Bracha's Protocol for Asynchronous Agreement

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

Input(v_j): This operation may be invoked once by each party P_j, who inputs a boolean value v_j. In response, $\mathcal{F}_{\text{AsyncAgreement}}$ sends NotifyInput(j, v_j) to the ideal-world adversary.

RequestOutput(j): after $n - 2 \cdot t$ honest parties have input the value true, this operation may be invoked by the ideal-world adversary, who specifies $j \in [n]$. In response, $\mathcal{F}_{\text{AsyncAgreement}}$ sends to P_j the message Output(done).

Fig. 6. The Simple Asynchronous Agreement Ideal Functionality $\mathcal{F}_{\text{AsyncAgreement}}$

– if P_j is honest, then that message is the same distributed message that P_j securely received, and

if S is honest and securely distributes $\boldsymbol{m} = (m_1, \ldots, m_n)$, that message is m_j.

Privacy: If the sender is honest and no honest party P_j forwards its message m_j to any other party, then the adversary learns nothing about these messages m_j belonging to the honest P_j's.

Note that whenever a party either securely receives a distributed message or authentically receives a forwarded message, 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 P_j to authentically forward a *tag* along with the distributed message m_j. There is no requirement that these tags are consistent in any way (i.e., one party may receive an authentically forwarded message with one tag, while another may receive the same message with a different tag).

It is best to formulate the correctness and privacy properties for secure message distribution as an ideal functionality $\mathcal{F}_{\text{SecMessDist}}$, given in Figure 7. Note that this ideal functionality does not capture at all the completeness property: this is a separate property. Also note that besides correctness and privacy, a protocol that securely realizes this ideal functionality must also be *input extractable*, in the sense that if the sender is corrupt, a simulator (i.e., ideal-world adversary) can extract all of the messages $\mathbf{m} = (m_1, \ldots, m_n)$ before delivering any message to an honest party. This property will also be crucial in our AVSS protocol.

We note that this ideal functionality is stronger than we need, in the sense that we may assume that if the sender is honest, then 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. (It is possible to lift this constraint, however, in the random oracle model.)

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

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, $\mathcal{F}_{\text{SecMessDist}}$ 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, $\mathcal{F}_{\text{SecMessDist}}$ sends to P_j the message Output(distribute, m_j).

Input(forward, i, tag_{ji}): after P_j has received m_j, it may invoke this operation once per $i \in [n]$ (indicating that the message m_j should be forwarded to player P_i with tag tag_{ji}). In response, $\mathcal{F}_{\text{SecMessDist}}$ sends the message NotifyInput(forward, j, i, tag_{ji}) to the ideal-world adversary.

RequestOutput(forward, j, i): after P_j has invoked Input(forward, i, tag_{ji}), this operation may be invoked by the ideal-word adversary. In response, $\mathcal{F}_{\text{SecMessDist}}$ sends the message Output(forward, j, m_j, tag_{ji}) to P_i.

Fig. 7. The Ideal Functionality for Secure Message Delivery $\mathcal{F}_{\text{SecMessDist}}$ (parameterized by S)

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 4.3) for more details). The communication complexity of the distribution phase of our protocol is $O(|\mathbf{m}| + \lambda \cdot n^2 \cdot \log n)$. Later, if P_j authentically forwards

its message m_j to a single party, this contributes an additional $O(|m_j| + \lambda \cdot n \cdot \log n)$ 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 build the functionality $\mathcal{F}_{\text{SecMessDist}}$ from Figure 7.

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.

4.1 AVID: Asynchronous Verifiable Information Dispersal

Our secure message distribution is based on techniques used for *Asynchronous Verifiable Information Dispersal*, or *AVID*. Although we will not use an AVID protocol directly, we review the definition of AVID and the relevant implementation techniques.

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. An AVID protocol should satisfy the following properties:

Dispersal completeness: If one honest party completes the dispersal phase, then every honest party eventually completes the dispersal phase. Moreover, if the sender is honest, then every honest party eventually completes the dispersal phase.

Retrieval completeness: If the retrieval phase for an honest party P_j is initiated, then P_j eventually outputs a message.

Correctness: All honest parties that retrieve a message output the same message. Moreover, if the sender is honest, that message is m.

Clearly, any AVID protocol can be used to implement reliable broadcast. A simple and efficient AVID protocol is used in the DispersedLedger system [YPA+21]. That AVID protocol 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. Implementing reliable broadcast using this AVID protocol results in a protocol with the same communication complexity as that in [CT05]. Just as in [CT05], the factor $\log n$ arises from Merkle trees. The paper [DXR22] gives an AVID protocol with an improved communication complexity of $O(|m| + \lambda \cdot n)$ per download (but with somewhat greater computational complexity, as it relies heavily on error correction, whereas DispersedLedger's protocol does not).

For reference, we present the DispersedLedger protocol[7] in Figure 8. The dispersal phase is a simple variation of Bracha broadcast, which disperses the fragments of an erasure coding to the

[7] We actually present a somewhat simplified version of their protocol, in which the only "clients" that retrieve a message are the "servers" P_j.

$\Pi_{\text{DispersedLedger}}$

// Sender S with input m
encode m as $(f_1, \ldots, f_n)$ using an $(n, n-2t)$-Reed-Solomon code
compute the Merkle tree on $(f_1, \ldots, f_n)$ with root r
for $j \in [n]$: send $(\mathsf{disperse}, r, \pi_j, f_j)$ to P_j,
    where π_j is the validation path for f_j under r

// Dispersal phase for party P_j
acquired $\leftarrow$ false, *voted* $\leftarrow$ false, *result* $\leftarrow \perp$
$(r^*, \pi_j^*, f_j^*) \leftarrow (\perp, \perp, \perp)$

repeat forever:
    wait for either:
        not *acquired* and $\exists (r, \pi_j, f_j)$: received $(\mathsf{disperse}, r, \pi_j, f_j)$ from S:
            acquired $\leftarrow$ *true*
            if π_j is a correct validation path for f_j under r then
                $(r^*, \pi_j^*, f_j^*) \leftarrow (r, \pi_j, f_j)$
                send (echo, r) to $P_1, \ldots, P_n$

        not *voted* and $\exists r$: received (echo, r) from $n - t$ distinct parties:
            send (vote, r) to $P_1, \ldots, P_n$
            voted $\leftarrow$ *true*

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

        $\exists r$: received (vote, r) from $n - t$ distinct parties:
            if $r \neq r^*$ then $(r^*, \pi_j^*, f_j^*) \leftarrow (r, \perp, \perp)$
            result $\leftarrow (r^*, \pi_j^*, f_j^*)$
            terminate dispersal phase

// Retrieval logic for party P_j
obtain a pair (π_i^*, f_i^*) from each of $n - 2t$ distinct parties P_i,
    where each π_i^* is a correct validation path for f_i^* under r^*
reconstruct the message m' from the $n - 2t$ fragments $\{f_i\}_i$
compute the fragments $(f'_1, \ldots, f'_n)$ of message m'
compute the Merkle tree for $(f'_1, \ldots, f'_n)$ with root r'
if $r^* = r'$
    then output m'
    else output $\perp$

Fig. 8. DispersedLedger Protocol

parties. For the erasure code, we may use an $(n, n-2t)$-Reed-Solomon code, or any erasure code with the property that any $n-2t$ fragments can be used to reconstruct the original message.

Party P_j completes the dispersal phase when it sets a value $result = (r^*, \pi_j^*, f_j^*)$, and we say it *holds the root* r^* and if $(\pi_j^*, f_j^*) \neq (\perp, \perp)$, we say it *holds a fragment*. One can show that if any honest server completes the dispersal phase holding a root, then eventually all honest servers complete the dispersal phase holding the same root, and among these, at least $n-2t$ hold a fragment.

When the retrieval phase for a party P_j is initiated, it obtains a pair (π_i^*, f_i^*) from each of $n-2t$ distinct parties P_i. Each such pair must be valid, in the sense that π_i^* is a correct validation path for f_i^* under r^*. Party P_j then reconstructs a message m' from these $n-2t$ fragments, computes all the fragments $(f_1', \ldots, f_n')$ of this message m', and computes the Merkle tree for $(f_1', \ldots, f_n')$ with root r'. If $r^* = r'$, party P_j outputs m'; otherwise, it outputs $\perp$.

Note that regardless of whether the sender is honest or not, every party will output the same message (which may be $\perp$). Moreover, if the sender is honest, this output will be the same as the message uploaded (and not $\perp$). We have left out the details of how a retrieval is initiated. This is somewhat application specific, and the details are not important to us here.

One can verify that this AVID protocol 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.

4.2 Reliable Message Distribution

Recall the notion of a *secure message distribution* protocol, which we introduced in Section 3.4. As a stepping stone, we introduce the notion of a **reliable message distribution** protocol, which satisfies all of the properties of secure message distribution *except privacy*.

We can implement this using a variant of the DispersedLedger AVID protocol in which the sender sends each P_j the collection of values $\{(r_i, \pi_{ij}, f_{ij})\}_{i=1}^n$, who then forms a Merkle tree from $(r_1, \ldots, r_n)$ with root r. Here, each m_i is encoded as a vector of fragments $(f_{i1}, \ldots, f_{in})$, and these fragments form the leaves of a Merkle tree with root r_i. The "echo" and "vote" stages of the protocol use the value r. Once the "vote" stage of the protocol finishes, each party P_j retrieves m_j by obtaining from each server P_i that holds a fragment the values $\pi_j, r_j, \pi_{ji}, f_{ji}$, where π_j is the validation path for r_j under r.

The communication complexity of the distribution phase is $O(|\mathbf{m}| + \lambda \cdot n^2 \cdot \log n)$. Later, if P_j wants to authentically forward its message m_j to P_i, it sends the collection of values $\pi_j, r_j, \pi_{ji}, f_{ji}$ obtained above to P_i. This contributes $O(|m_j| + \lambda \cdot n \cdot \log n)$ to the total communication cost.

Note that when an honest party receives an authentically forwarded message from a corrupt party, this does not contribute anything to the communication complexity.

4.3 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**. Here, the sender S **securely distributes** a vector of keys $\boldsymbol{k} = (k_1, \ldots, k_n)$. Here, 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 $\boldsymbol{k}$ at the end of the protocol. The same properties above should hold, namely, distribution/forward completeness and correctness. In addition, the following property should hold:

Privacy: If the sender is honest and no honest party P_j forwards its key k_j to any other party, then the adversary learns nothing about these keys k_j belonging to the honest P_j's.

To implement such a scheme, we modify the reliable message distribution protocol above so that instead of encoding a key k_i using an erasure code, the sender proceeds as follows. For each $i \in [n]$, the sender chooses an random polynomial $f_i \in K_{<n-2t}$ for an appropriate (large) finite field K. Let $s_{ij} \coloneqq f_i(\eta_j)$ for $j \in [0..n]$. Here, $\{\eta_j\}_{j\in[0..n]}$ is some arbitrary collection of distinct coordinates in K. The key k_i is defined as $k_i \coloneqq H(s_{i0})$ for an appropriate cryptographic hash H. The sender builds a Merkle tree with root r_i and leaves $\{H(s_{ij})\}_{j=1}^n$.

To distribute the vector of keys $\boldsymbol{k} = (k_1, \ldots, k_n)$, the sender sends each P_j the collection of values $\{(r_i, \pi_{ij}, s_{ij})\}_{i=1}^n$. The rest of the protocol goes through in an analogous obvious way.

One can easily prove that this protocol satisfies our definition of a secure key distribution protocol, under the assumption that the hash function used to implement the Merkle trees is collision resistant, and the following assumption on the hash function H, which we might call the **linear hiding assumption**. This assumption is defined by a game in which the adversary first chooses distinct pairs $(a_1, b_1), \ldots, (a_m, b_m) \in K^2$, with each $a_i \neq 0$. The task of the adversary is to distinguish the distribution

$$(H(a_1 \cdot r + b_1), \ldots, H(a_m \cdot r + b_m)),$$

where $r \in K$ is randomly chosen, from the uniform distribution on $\mathcal{K}^m$, where $\mathcal{K}$ is the output space of H. 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 $m^2/|K|$ is negligible (just to avoid collisions on the inputs to H), and so it seems a reasonable assumption (a kind of "related key attack" assumption).

Assuming individual keys are of size $O(\lambda)$, the communication complexity of the distribution phase is $O(\lambda \cdot n^2 \cdot \log n)$. Later, if P_j wants to authentically forward its key k_j to P_i, this contributes $O(\lambda \cdot n \cdot \log n)$ to the communication complexity.

4.4 Building Secure Message Distribution

Finally we can build $\mathcal{F}_{\text{SecMessDist}}$ using the tools we have just defined. We just need to combine a secure key distribution protocol with a reliable message distribution protocol, where the latter is used to distribute a vector of ciphertexts, each encrypting a message under the corresponding key. In fact, we can combine the protocols into a single protocol where there is just a single root r that controls both the keys and the ciphertexts.

One can show that the protocol obtained by combining the secure key distribution and reliable message distribution protocols presented above securely realizes $\mathcal{F}_{\text{SecMessDist}}$ with respect to *restricted environments* that never request that an honest party authentically forwards its message if S is honest. This is sufficient for our application. The communication complexity of the distribution phase is $O(|\mathbf{m}| + \lambda \cdot n^2 \cdot \log n)$. Later, if P_j authentically forwards its message m_j to P_i, this contributes $O(|m_j| + \lambda \cdot n \cdot \log n)$ to the communication complexity.

5 Our AVSS protocols

We now present 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.

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 $i \neq j$. We have n parties $P_1, \ldots, P_n$, of which at most $t < n/3$ may be corrupt. Our protocol Π_{avss1}, see Figure 9, is an (n, d, L)-AVSS protocol over $\mathbb{A}$ (with respect to $\boldsymbol{e}$). The protocol requires $t < d \leq n - 2 \cdot t$.

Our AVSS protocol makes use of a variation of the probabilistic degree check from [DN07]. In addition to the ring $\mathbb{A}$, the protocol works with a ring extension $\mathbb{B}$ of $\mathbb{A}$; however, in some instantiations of the protocol, we may well have $\mathbb{A} = \mathbb{B}$. Our protocol is also parameterized in terms of a *repetition parameter* R; however, in some instantiations of the protocol, we may well have $R = 1$.

A key ingredient in our AVSS protocol is a subprotocol for secure message distribution protocol (see Section 3.4). Our AVSS protocol is described in terms of the corresponding ideal functionality $\mathcal{F}_{\text{SecMessDist}}$.

Our protocol also uses a random beacon (see Section 3.1) whose output space is of the form Θ^R, where $\Theta \subseteq \mathbb{B}^L$. In one instantiation of the protocol, we will use

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

where $\mathcal{E} \subseteq \mathbb{B}$. In another instantiation, we will use

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

So the random beacon will output a collection of R sequences $\{\boldsymbol{\theta}^{(r)}\}_{r \in [R]}$, where each $\boldsymbol{\theta}^{(r)}$ is randomly chosen from Θ. The advantage of using $\Theta = \Theta^{\mathcal{E}}_{\text{pow}}$ is that the random beacon really only needs to output a single elements $\theta \in \mathcal{E}$. The disadvantage of using $\Theta = \Theta^{\mathcal{E}}_{\text{pow}}$ is that the security of our protocol is a bit weaker. Our AVSS protocol is described in terms of the corresponding ideal functionality $\mathcal{F}_{\text{Beacon}}$ parameterized by the output space Θ^R.

Our protocol also makes use of a reliable broadcast protocol (see Section 3.2) and a simple asynchronous agreement protocol (see Section 3.3). Our AVSS protocol is described in terms of the corresponding ideal functionalities $\mathcal{F}_{\text{ReliableBroadcast}}$ and $\mathcal{F}_{\text{AsyncAgreement}}$.

5.1 Security analysis

In order to analyze the failure probability of this generic protocol, we need a 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 $\Theta = \Theta^{\mathcal{E}}_{\text{pow}}$, 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.

Π_{avss1}

The dealer $D \in \{P_1, \ldots, P_n\}$ has input $f_1, \ldots, f_L \in \mathbb{A}[x]_{<d}$.

1. Dealer D:
   (a) Choose random $g^{(r)} \in \mathbb{B}[x]_{<d}$ for $r \in [R]$.
   (b) Compute evaluations
   $$v_{\ell,j} \leftarrow f_\ell(e_j) \in \mathbb{A} \quad (\ell \in [L], j \in [n])$$
   and
   $$w_j^{(r)} \leftarrow g^{(r)}(e_j) \in \mathbb{B} \quad (r \in [R], j \in [n]),$$
   (c) Compute the messages
   $$m_j = \big(\{v_{\ell,j}\}_{\ell \in [L]},\ \{w_j^{(r)}\}_{r \in [R]} \big), \tag{1}$$
   (d) Invoke the operation $\mathsf{Input}(\mathsf{distribute}, \mathbf{m})$ on $\mathcal{F}_{\text{SecMessDist}}$, where $\mathbf{m} = (m_1, \ldots, m_n)$. *Note: if the sender is corrupt, some of these messages may be* $\perp$.
2. Each P_j:
   (a) Wait for $\mathcal{F}_{\text{SecMessDist}}$ to deliver the message $\mathsf{Output}(\mathsf{distribute}, m_j)$.
   (b) Invoke the operation $\mathsf{Input}()$ on $\mathcal{F}_{\text{Beacon}}$.
3. Each P_j: Wait for $\mathcal{F}_{\text{Beacon}}$ to deliver the message $\mathsf{Output}(\{\boldsymbol{\theta}^{(r)}\}_{r \in [R]})$. *For each* $r \in [R]$, $\boldsymbol{\theta}^{(r)}$ *is a random element of* $\Theta \subseteq \mathbb{B}^L$, *and we write*
   $$\boldsymbol{\theta}^{(r)} = (\theta_1^{(r)}, \ldots, \theta_L^{(r)}).$$
4. Dealer D:
   (a) Compute
   $$h^{(r)} \leftarrow g^{(r)} + \sum_{\ell \in [L]} \theta_\ell^{(r)} \cdot f_\ell \in \mathbb{B}[x]_{<d} \qquad (\text{for } r \in [R])$$
   (b) Invoke the operation $\mathsf{Input}(\{h^{(r)}\}_{r \in [R]})$ on $\mathcal{F}_{\text{ReliableBroadcast}}$.
5. Each P_j:
   (a) Wait for $\mathcal{F}_{\text{ReliableBroadcast}}$ to deliver the message $\mathsf{Output}(\{h^{(r)}\}_{r \in [R]})$.
   (b) Check that $m_j \neq \perp$, and so is of the form (1), and that
   $$h^{(r)}(e_j) = w_j^{(r)} + \sum_{\ell \in [L]} \theta_\ell^{(r)} \cdot v_{\ell,j} \qquad (\text{for } r \in [R]). \tag{2}$$
   If these checks pass, we say that P_j is *happy*, and set $happy_j \leftarrow \mathsf{true}$; otherwise, we say P_j is *unhappy*, and set $happy_j \leftarrow \mathsf{false}$.
   (c) Invoke the operation $\mathsf{Input}(happy_j)$ on $\mathcal{F}_{\text{AsyncAgreement}}$.
6. P_j: Wait for $\mathcal{F}_{\text{AsyncAgreement}}$ to deliver the message $\mathsf{Output}(\mathsf{done})$.
   If P_j is happy, then output $\{v_{\ell,j}\}_{\ell \in [L]}$, and also do the following:
   (a) Wait to receive a valid *complaint* from some player P_i, which is a message m_i securely forwarded from P_i (see details below). Such a complaint is validated by checking that P_i is unhappy, meaning that either $m_i = \perp$ or that equation (2), with j replaced by i, does not hold.
   (b) If the complaint is valid then broadcast an *assist*, which is done by P_j invoking $\mathsf{Input}(\mathsf{forward}, i, \mathsf{assist})$ on $\mathcal{F}_{\text{SecMessDist}}$ for all $i \in [n]$.
   Otherwise, P_j is unhappy and so we do the following.
   (a) Broadcast a *complaint*. This is done by invoking $\mathsf{Input}(\mathsf{forward}, i, \mathsf{complaint})$ on $\mathcal{F}_{\text{SecMessDist}}$ for all $i \in [n]$.
   (b) Now wait for $t+1$ valid *assists*. An assist from P_i is a message m_i securely forwarded by P_i (see details above). Such an assist is validated by checking that P_i is happy, meaning that $m_i \neq \perp$ and that equation (2), with j replaced by i, holds.
   (c) Once $t+1$ valid assists are obtained, party P_j can interpolate to obtain and output his correct shares.

Fig. 9. An AVSS protocol over $\mathbb{A}$

Theorem 5.1 (Security of Π_{avss1}). *Assuming $2^n \cdot \chi(\mathbb{A}, \mathbb{B}, \Theta)^R$ is negligible, Π_{avss1} securely and completely realizes $\mathcal{F}_{\text{avss}}$ in the $(\mathcal{F}_{\text{SecMessDist}}, \mathcal{F}_{\text{Beacon}}, \mathcal{F}_{\text{ReliableBroadcast}}, \mathcal{F}_{\text{AsyncAgreement}})$-hybrid model.*

Proof. Completeness is clear. We focus on proving 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.

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 "random padding" supplied by the 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 $n - 2 \cdot t$ honest parties is happy, and this subset may be determined *after* is 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 $\mathcal{F}_{\text{SecMesDist}}$, where each m_j is either $\perp$ or of the form (1), and which are committed before the random beacon is revealed. Here, we are 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 that an honest party first passes the wait condition in Step 6. 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 extracted input m_j, the random beacon value, and the polynomials $h^{(r)}$. This set includes *all* parties, both honest and dishonest. By the correctness property of the simple asynchronous agreement protocol, we have $|\mathcal{P}^* \cap \mathcal{H}| \geq n - 2 \cdot t \geq d$.

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 $\mathcal{P}^*$).

Claim 1: the simulation fails with probability at most $2^n \cdot \chi(\mathbb{A}, \mathbb{B}, \Theta)^R$. To prove Claim 1, let us first make a definition. Consider any point in time after the dealer has submitted its input vector $\boldsymbol{M} = (m_1, \ldots, m_n)$ to $\mathcal{F}_{\text{SecMessDist}}$, so the messages m_j for $j \in [n]$ are fixed. Let $\mathcal{P} \subseteq [n]$ with $n' := |\mathcal{P}| \geq d$ 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.

Claim 2: if $\mathcal{P}$ is not d-consistent, then the probability that P_j is happy for all $j \in \mathcal{P}$ is at most $\chi(\mathbb{A}, \mathbb{B}, \Theta)^R$. To prove Claim 2, suppose $\mathcal{P}$ is not d-consistent. 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 $C \in \mathbb{A}^{n' \times (n'-d)}$. 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 $\boldsymbol{v}_{\ell^*} \cdot C \neq 0$. Define the matrix $Q \in \mathbb{A}^{L \times n'}$ whose ℓ-th row is $\boldsymbol{v}_\ell$ for $\ell \in [L]$. So we have $Q \cdot C \in \mathbb{A}^{L \times (n'-d)}$ is nonzero matrix. For each $r \in [R]$, define $\boldsymbol{w}^{(r)} \coloneqq \{w_j^{(r)}\}_{j \in \mathcal{P}} \in \mathbb{B}^{n'}$. Consider the corresponding (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, if P_j is happy for all $j \in \mathcal{P}$, this means that for all $r \in [R]$, the vector $\boldsymbol{w}^{(r)} + \boldsymbol{\theta}^{(r)} \cdot Q$ lies in the extended Reed-Solomon code, which implies

$$\boldsymbol{w}^{(r)} \cdot C + \boldsymbol{\theta}^{(r)} \cdot Q \cdot C = 0. \tag{3}$$

Since $Q \cdot C$ is a nonzero matrix, we can choose one nonzero column of $Q \cdot C$, and (3) implies that for some fixed $b \in \mathbb{B}$ and fixed, nonzero $\boldsymbol{a} \in \mathbb{A}^L$, we have

$$b + \langle \boldsymbol{a}, \boldsymbol{\theta}^{(r)} \rangle = 0. \tag{4}$$

So for each $r \in [R]$, equation (4) holds with probability at most $\chi(\mathbb{A}, \mathbb{B}, \Theta)$, and repeating this R times gives the desired probability and proves Claim 2.

Returning to the proof of Claim 1: if the simulation fails, this implies that $\mathcal{P}^*$ is not d-consistent yet P_j is happy for all $j \in \mathcal{P}^*$. Now, even though the inputs (1) are chosen before the random beacon is revealed, the subset $\mathcal{P}^*$ may be chosen by the adversary after the random beacon is revealed. Claim 1 follows by applying the union bound to Claim 2. □

The above theorem establishes the protocol securely realizes the $\mathcal{F}_{\text{avss}}$ functionality. Completeness is straightforward to establish from the completeness of the subprotocols.

Note that the factor 2^n in the failure bound does not arise in the analysis of the corresponding protocol in [DN07]. This seems hard to circumvent, as in the asynchronous setting, the adversary can run the protocol with an arbitrarily chosen subset of happy honest parties. For modest sized n (like $n < 100$), this should be acceptable, using a larger extension $\mathbb{B}$, or larger value of R, as necessary.

Note that we get a somewhat better failure bound when we use $\Theta = \Theta_{\text{lin}}^{\mathcal{E}}$. However, in this instantiation, our random beacon has to output a vector $(\theta_1, \ldots, \theta_L) \in \mathcal{E}^L$. This could be implemented by running a fast PRG on a seed obtained from a random beacon. However, to justify this step, we have take account of the fact that the failure condition in Theorem 5.1, is based on the union bound, is not efficiently verifiable. Indeed, the adversary gets to choose the subset needed to break the protocol after seeing the seed, and there are exponentially many subsets to choose from. We can still justify this step by implementing the PRG using a hash function that we model as a random oracle. Indeed, this gives us an implementation of a random beacon with outputs in $\Theta_{\text{lin}}^{\mathcal{E}}$ that securely realizes $\mathcal{F}_{\text{Beacon}}$ in the random oracle model.

An example. Here is a simple example that shows why a factor of $2^{\Omega(n)}$ in the failure probability seems hard to avoid.

Let $n = 3 \cdot t + 1$ and $d = t + 1$. Suppose a corrupt dealer chooses polynomials $f_1 = x^{t+2} + x^{t+1}$ and $f_2 = x^{t+2}$. All other polynomials are of degree at most t. Now consider any set $\mathcal{P} \in [n]$ of size

$t+2$, and define

$$f_1^{(\mathcal{P})} \coloneqq f_1 \bmod \prod_{j \in \mathcal{P}} (x - e_j)$$

and

$$f_2^{(\mathcal{P})} \coloneqq f_2 \bmod \prod_{j \in \mathcal{P}} (x - e_j).$$

Then we have

$$f_1^{(\mathcal{P})} = (1 + T^{(\mathcal{P})}) \cdot x^{t+1} + \text{lower order terms}$$

and

$$f_2^{(\mathcal{P})} = T^{(\mathcal{P})} \cdot x^{t+1} + \text{lower order terms},$$

where

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

Assume $\Theta = \Theta_{\text{pow}}$. For a given θ, assuming $\theta \neq 0$ and $T^{(\mathcal{P})} \neq 0$, the adversary will break the protocol if

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

or in other words

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

So to break the protocol, the adversary has to find a subset $\mathcal{P} \subseteq \mathcal{H}$ of size $t+2$ that satisfies this equation.

To really get an effective attack that succeeds with probability at least $2^{\Omega(n)}/|\mathbb{B}|$, we need that the map $\mathcal{P} \mapsto T^{(\mathcal{P})}$ is nearly one-to-one and not too hard to invert. For example, if $|\mathbb{A}| > 2^n$ and the evaluation coordinates are $1, 2, \ldots, 2^{n-1}$, then both of these conditions are true. However, if the evaluating point are $1, 2, \ldots, n$, then this attack succeeds with probability only $O(n^2/|\mathbb{B}|)$, since the image if the map $\mathcal{P} \mapsto T^{(\mathcal{P})}$ is of size $O(n^2)$.

So even though this example has some limitations, it shows that getting a significantly better failure bound may be very challenging if not impossible.

5.2 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.

The Happy Path. Each message m_j input into the $\mathcal{F}_{\text{SecMessDist}}$ has size

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

Our protocol for secure message distribution, outlined in 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 $O(n\cdot R\cdot\log|\mathbb{A}|)$. If we implement this using, say, the protocol from [CT05], 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 $\mathcal{F}_{\text{AyncAgreement}}$ as in Section 3.3, which contributes $O(n^2)$ to the communication complexity. We shall ignore for now the communication complexity of the random beacon functionality. Thus the total communication complexity, ignoring the random beacon, is

$$O\left(n\cdot(L+nR)\cdot\log|\mathbb{A}| + \lambda\cdot n^2\cdot\log n\right). \tag{5}$$

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

$$R = \left\lceil \frac{\sigma+n}{\log_2|\mathcal{E}|} \right\rceil. \tag{6}$$

With this setting of R, our communication complexity bound (5) becomes

$$O\left(n\cdot L\cdot\log|\mathbb{A}| + n^2\cdot(\sigma+n)\cdot\frac{\log|\mathbb{A}|}{\log|\mathcal{E}|} + \lambda\cdot n^2\cdot\log n\right). \tag{7}$$

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, (7) 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 $\max\{n,\sigma\} \le \lambda$, 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. 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

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

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 (7) 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

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

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 (5) 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 (6), the bound (7) 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 Using a Random Oracle instead of a Random Beacon

We can simplify the protocol significantly by using a random oracle in place of a random beacon. In more detail, suppose that hash function used to implement the secure message distribution protocol is modeled as a random oracle. This includes the Merkle trees and as well as the use in the linear hiding assumption in Section 4.3. Intuitively, this makes the root of the top-level Merkle tree in the secure message distribution protocol an *extractable commitment*. We can also apply another

random oracle to this root to obtain the output of the random beacon. With this approach, the adversary must commit to the secure message distribution inputs before seeing the random oracle output, but he can "grind", trying several sets of inputs until he finds an input/beacon pair that he likes.

To formalize this, we can define an ideal functionality $\mathcal{F}^*_{\text{SecMessDist}}$ that combines that of secure message distribution and a random beacon. This is the same $\mathcal{F}_{\text{SecMessDist}}$, except as follows:

- As soon as a sender inputs $\mathbf{m}$, the functionality gives a random beacon value $\omega \in \Omega$ to the ideal-world adversary.
- The outputs to the parties in the distribution phase include the value $\omega \in \Omega$.
- If the sender is corrupt, the ideal-world adversary may alternatively specify the sender's input via a special *grinding interface*:
  - Before specifying an input, the ideal-world adversary may make several calls to the grinding interface of $\mathcal{F}^*_{\text{SecMessDist}}$.
  - In each such grinding call, the ideal-world adversary specifies an input $\mathbf{m}$, to which $\mathcal{F}^*_{\text{SecMessDist}}$ responds with a corresponding random beacon value $\omega \in \Omega$.
  - After making several such calls, the adversary may fix the sender's input by identifying one of the grinding calls, so that the protocol runs with the values $\mathbf{m}$ and ω from that call.

It is not hard to see that our protocol for secure message distribution securely realizes $\mathcal{F}^*_{\text{SecMessDist}}$. Moreover, the total number of grinding calls made by the simulator in the ideal world is bounded by the total number of calls to the random oracle used to implement the random beacon.

Note that in a system in which many instances of our AVSS protocol may run using the same hash function, it is essential that appropriate (and standard) "domain separation" techniques be used to ensure that each instance is effectively getting its own independent random oracles.

Let is call this variation Π_{avss2}. It is the same as Π_{avss1}, except that we use this combined secure message distribution plus random beacon subprotocol, which securely and completely realizes $\mathcal{F}^*_{\text{SecMessDist}}$ in the random oracle model. It is easy to adapt the proof of Theorem 5.1 to prove:

Theorem 5.2 (Security of Π_{avss2}). *Assuming and $Q \cdot 2^n \cdot \chi(\mathbb{A}, \mathbb{B}, \Theta)^R$ is negligible, where Q is the number of grinding calls, Π_{avss2} securely and completely realizes $\mathcal{F}_{\text{avss}}$ in the $(\mathcal{F}^*_{\text{SecMessDist}}, \mathcal{F}_{\text{ReliableBroadcast}})$-hybrid model. $(\mathcal{F}^*_{\text{SecMessDist}}, \mathcal{F}_{\text{ReliableBroadcast}}, \mathcal{F}_{\text{AsyncAgreement}})$-hybrid model.*

All of the issues and results discussed above, including the discussion on communication complexity, carry over here as well. We also hypothesize that Π_{avss2} can be implemented in such a way that it is secure against *adaptive* corruptions in the random oracle model — we expect a proof of this to be straightforward.

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 get 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 $g \in \mathbb{A}'[x]_{<d}$ 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 vector

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

The dealer then computes the polynomial

$$h \leftarrow g + \sum_{\ell \in [L]} \gamma_\ell \cdot f_\ell, \tag{8}$$

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 (8) holds at the evaluation coordinate e'_j. The parties then run a very simple agreement 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 each h was computed correctly. We then argue that if $f_{\ell^*}(0) \notin \phi^{-1}(\mathbb{S})$ for some $\ell^* \in [L]$, then with probability at most $p^{k-k'-1}$, for randomly chosen $\boldsymbol{\gamma}$, we have

$$g + \sum_{\ell \in [L]} \gamma_\ell \cdot f_\ell \in \mathbb{S}'.$$

This implies that except with probability $p^{k-k'-1}$, we can be sure that all $\ell \in [L]$, the secret $f_\ell(0)$ lies in $\phi^{-1}(\mathbb{S})$.

Our protocol, which we call Π_{ravss1}, is presented in Figure 10. 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 that returns for each $r \in [R]$ a random vector $\boldsymbol{\gamma}^{(r)} := (\gamma_1^{(r)}, \ldots, \gamma_L^{(r)}) \in (\mathbb{Z}/(p^{k'}))^L$, 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 line in $\phi^{-1}(\mathbb{S})$ (with high probability). As mentioned above, each party can then locally apply ϕ to its shares to get sharings of secrets in $\mathbb{S}$.

Π_{ravss1}

The dealer $D \in \{P_1, \ldots, P_n\}$ has input $f_1, \ldots, f_L \in \mathbb{A}'[x]_{<d}$ with constant terms in $\mathbb{S}'$.

1. Dealer D:
   (a) For each $r \in [R]$, choose a random polynomial $g^{(r)} \in \mathbb{A}'[x]_{<d}$ with constant term in $\mathbb{S}'$.
   (b) Invoke the operation $\mathsf{Input}(\ \{f_\ell\}_{\ell\in[L]},\ \{g^{(r)}\}_{r\in[R]})$ on $\mathcal{F}_{\text{avss}}$.
2. Each P_j:
   (a) Wait for $\mathcal{F}_{\text{avss}}$ to deliver the message $\mathsf{Output}(\ \{v_{\ell,j}\}_{\ell\in[L]},\ \{w_j^{(r)}\}_{r\in[R]}\ )$.
   (b) Invoke the operation $\mathsf{Input}()$ on $\mathcal{F}_{\text{Beacon}}$.
3. Each P_j: Wait for $\mathcal{F}_{\text{Beacon}}$ to deliver the message $\mathsf{Output}(\ \{\boldsymbol{\gamma}^{(r)}\}_{r\in[R]}\ )$, where $\boldsymbol{\gamma}^{(r)} = (\gamma_1^{(r)}, \ldots, \gamma_L^{(r)}) \in (\mathbb{Z}/(p^{k'}))^L$.
4. Dealer D:
   (a) For each $r \in [R]$, compute
   $$h^{(r)} \leftarrow g^{(r)} + \sum_{\ell\in[L]} \gamma_\ell^{(r)} \cdot f_\ell.$$
   (b) Invoke the operation $\mathsf{Input}(\ \{h^{(r)}\}_{r\in[R]}\ )$ on $\mathcal{F}_{\text{ReliableBroadcast}}$.
5. Each P_j:
   (a) Wait for $\mathcal{F}_{\text{ReliableBroadcast}}$ to deliver the message $\mathsf{Output}(\ \{h^{(r)}\}_{r\in[R]}\ )$.
   (b) For each $r \in [R]$, check that $h^{(r)} \in \mathbb{A}'[x]_{<d}$ with constant term in $\mathbb{S}'$, and that
   $$h^{(r)}(e'_j) = w_j^{(r)} + \sum_{\ell\in[L]} \gamma_\ell^{(r)} \cdot v_{\ell,j}.$$

   If these checks do not pass then *abort*. Otherwise:
   i. Send a "vote" message to all parties;
   ii. Wait for "vote" messages from $n - t$ distinct parties;
   iii. Output $\{v_{\ell,j}\}_{\ell\in[L]}$.

Fig. 10. 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 Figure 11. 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*

$$a_1 \cdot x_1 + \cdots a_L \cdot x_L + b \equiv 0 \pmod{p^s}. \tag{9}$$

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

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 (9) has no solutions. In this case, (9) 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 (9) holds is at most p^{r-s}.

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

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 NotifyInput() to the ideal-world adversary.

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

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

Fig. 11. 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 $e'_1, \ldots, e'_n$)

Theorem 6.1 (Security of Π_{ravss1}). *Assuming $p^{(k-k'-1)\cdot R}$ is negligible, Π_{ravss1} securely and completely realizes $\mathcal{F}_{\text{ravss}}$ in the $(\mathcal{F}_{\text{avss}}, \mathcal{F}_{\text{Beacon}}, \mathcal{F}_{\text{ReliableBroadcast}})$-hybrid model.*

Proof. For completeness, suppose that some honest party has produced an output and all relevant RequestOutput messages have been delivered. The honest party that produced an output received $n - t$ "vote" messages, which means that at least $n - 2t \geq d$ honest parties successfully performed their local checks. This implies that each $h^{(r)}$ has the correct form (is of degree less than d with constant term in $\mathbb{S}'$), and that

$$h^{(r)} = g^{(r)} + \sum_{\ell \in [L]} \gamma_\ell^{(r)} \cdot f_\ell \quad (\text{for all } r \in [r]). \tag{10}$$

But this implies that all honest parties have successfully performed their local checks and broadcast "vote" messages, and so every honest party has produced an output.

We now prove 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

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

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 $\{(\gamma_1^{(r)}, \ldots, \gamma_L^{(r)})\}_{r \in [R]}$ 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 $r \in [R]$. Suppose that $s_{\ell^*} \notin \phi^{-1}(\mathbb{S})$ for some $\ell^* \in [L]$. To finish the proof of the theorem, it will suffice to show that for randomly chosen $\{(\gamma_1^{(r)}, \ldots, \gamma_L^{(r)})\}_{r \in [R]}$, the probability that any honest party produces an output is at most $p^{(k-k'-1)\cdot R}$.

First observe that if any honest party produces an output, then as already discussed above, each $h^{(r)}$ has the correct form and (10) holds. This implies that

$$s^{(r)} + \sum_{\ell \in [L]} \gamma_\ell^{(r)} \cdot s_\ell \in \mathbb{S}' \quad (\text{for all } r \in [R]). \tag{11}$$

So it suffices to show that (11) 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 (11), we must have

$$b_{i^*,j^*} + \sum_{\ell \in [L]} a_{i^*,j^*,\ell} \cdot x_\ell^{(r)} \equiv 0 \pmod{p^{k'}} \quad (\text{for all } r \in [R]). \tag{12}$$

But by Lemma 6.1, the congruence (12) holds with probability at most $p^{(k-k'-1)\cdot R}$.

After the proof of Theorem 5.1, we remarked that in some instantiations, we cannot justifiably use a random beacon that generates a short seed that is then stretched using a PRG. That limitation does not apply here.

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.

Setting $k' \coloneqq k$. At one extreme, we could set $k' \coloneqq k$. In this case, to achieve σ bits of security, we need to set the repetition parameter $R \coloneqq \lceil \sigma \cdot \log_p 2 \rceil$. The main advantage of this parameter setting is that $\mathbb{A}' = \mathbb{A}$, and the amortized communication complexity remains the same as in Section 5.2. The main disadvantage of this setting is that the amortized computational complexity blows up by a factor of R.

Setting $R \coloneqq 1$. At another extreme, we could set $R \coloneqq 1$. In this case, to achieve σ bits of security, we need to set $k' \coloneqq k - 1 + \lceil \sigma \cdot \log_p 2 \rceil$. 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 $k' \coloneqq 2 \cdot k$, we get $k+1$ bits of security, while both the amortized communication and computational complexity increase by just a small constant factor over the basic AVSS protocol.

6.6 Using a Random Oracle instead of a Random Beacon

Analogous to what we did in Section 5.3, we can simplify the protocol significantly by using a random oracle in place of a random beacon.

Defining $\mathcal{F}^*_{\text{avss}}$. In more detail, let us start by defining an ideal functionality $\mathcal{F}^*_{\text{avss}}$ that combines that of AVSS and a random beacon. This is the same $\mathcal{F}_{\text{avss}}$, except as follows:

- As soon as a dealer inputs $(f_1, \ldots, f_L)$, the functionality gives a random beacon value $\omega \in \Omega$ to the ideal-world adversary.
- The outputs to the parties include the value $\omega \in \Omega$.
- If the dealer is corrupt, the ideal-world adversary may alternatively specify the dealer's input via a special *grinding interface*:
  - Before specifying an input, the ideal-world adversary may make several calls to the grinding interface of $\mathcal{F}^*_{\text{avss}}$.
  - In each such grinding call, the ideal-world adversary specifies an input $(f_1, \ldots, f_L)$, to which $\mathcal{F}^*_{\text{avss}}$ responds with a corresponding random beacon value $\omega \in \Omega$.
  - After making several such calls, the adversary may fix the dealer's input by identifying one of the grinding calls, so that the protocol runs with the values $(f_1, \ldots, f_L)$ and ω from that call.

The relationship between $\mathcal{F}_{\text{avss}}$ and $\mathcal{F}^*_{\text{avss}}$ is analogous to that between $\mathcal{F}_{\text{SecMessDist}}$ and $\mathcal{F}^*_{\text{SecMessDist}}$ in Section 5.3.

Implementing $\mathcal{F}^*_{\text{avss}}$. To securely realize $\mathcal{F}^*_{\text{avss}}$, we can modify our protocol Π_{avss2} in Section 5.3. Recall that Π_{avss2} uses $\mathcal{F}^*_{\text{SecMessDist}}$ to generate a random beacon value after the inputs $\mathbf{m} = (m_1, \ldots, m_n)$ to the secure message distribution protocol are fixed. We can extend the output space of this random beacon to include a sufficiently long random string ρ. Also recall that in Π^*_{avss2} (just like in Π_{avss2}), the dealer reliably broadcasts $\mathbf{h} = \{h^{(r)}\}_{r \in [R]}$. Observe that the values $\mathbf{m}$ and $\mathbf{h}$ together completely determine the effective inputs $(f_1, \ldots, f_L)$ to the AVSS ideal functionality.[8] Therefore, we modify Π_{avss2} so that it generates its own random beacon output by feeding ρ and $\mathbf{h}$ to a random oracle.

Let us denote by Π^*_{avss2} this modified version of Π^*_{avss2}. It is not hard to see that Π^*_{avss2} securely and completely realizes $\mathcal{F}^*_{\text{avss}}$, the total number of grinding calls made by the simulator in the ideal world is bounded by the total number of calls to the random oracle used to implement the random beacon.

Using $\mathcal{F}^*_{\text{avss}}$ *to implement* Π_{ravss2}. We can now modify protocol Π_{ravss1} to using $\mathcal{F}^*_{\text{avss}}$ in place of $\mathcal{F}_{\text{avss}}$ and a random beacon. Let is call this new protocol Π_{ravss2}. It is easy to adapt the proof of Theorem 6.1 to prove:

Theorem 6.2 (Security of Π_{ravss2}). *Assuming $Q \cdot p^{(k-k'-1)\cdot R}$ is negligible, where Q is the number of grinding calls, Π_{ravss2} securely and completely realizes $\mathcal{F}_{\text{ravss}}$ in the $(\mathcal{F}^*_{\text{avss}}, \mathcal{F}_{\text{ReliableBroadcast}})$-hybrid model.*

[8] In fact, the effective inputs $(f_1, \ldots, f_L)$ are not really determined at all until $\mathbf{h}$ is fixed. Indeed, a corrupt dealer could distribute arbitrary shares to the parties that are completely uncorrelated and do not lie on any particular low-degree polynomial. Only after the random beacon value is revealed, the dealer can choose an arbitrary subset of d honest parties to make happy and then choose $\mathbf{h}$ to make them so. The shares of these happy parties determine the polynomials $(f_1, \ldots, f_L)$. After this, these d honest parties together with t corrupt parties can make the protocol successfully produce an output, without the help of the unhappy parties. Later, the unhappy parties may complain and obtain their shares from the happy parties.

Acknowledgements

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+22. I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, and G. Stern. Bingo: Adaptively secure packed asynchronous verifiable secret sharing and asynchronous distributed key generation. Cryptology ePrint Archive, Report 2022/1759, 2022. `https://eprint.iacr.org/2022/1759`.

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.

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.

DXR21. S. Das, Z. Xiang, and L. Ren. Asynchronous data dissemination and its applications. Cryptology ePrint Archive, Report 2021/777, 2021. `https://eprint.iacr.org/2021/777`.

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`.

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.

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`.